ארץ עיר אחד על אחד

שם התלמיד: רון מיכלמן.

תעודת זהות של התלמיד: 216262469.

שם המנחה: יובל פיטוסי.

שם החלופה: הגנת סייבר ומערכות הפעלה במסגרת לימודי ההתמחות בהנדסת תוכנה 883589.

בית הספר: התיכון העירוני על שם בליך.

תאריך הגשה: 06/06/2024.

תוכן העניינים

מבוא

ייזום

מטרה ורציונל

מטרת הפרויקט הינה יצירת גרסה דיגיטלית של משחק הילדים המוכר "ארץ עיר", המוצר הגמור נותן שירות במבנה של שרת ולקוח על מנת לאפשר למספר רב של משתתפים לשחק אחד נגד השני במשחק במקביל, כאשר המשחק מבוסס על הגרסה האמיתית ובה יש להצליח להעלות כמה שיותר מילים מתאימות לקטגוריות על פי האות הראשונה במילה, וכאשר האות נבחרת בתחילת הסיבוב לשני המשתמשים, בנוסף לתחרות בין־אישית בין המשתתפים קיימת תחרות בין המשתתף להגבלת הזמן המוקצבת עליו בכל משחק.

סיבת בחירה בפרויקט זה

הבחירה בפרויקט באה מצורך סביבתי ששמתי לב אליו במשך שנות לימודיי, כמעט בכל מסגרת לימודית שהייתי בה התלמידים היו משחקים במשחק ארץ עיר ובגרסאות שלו בשיעורים ובהפסקות, לאחר הבנה שלא קיימות תוכנות המאפשרות לשחק במשחק זה בצורה דומה לחיים האמיתיים עלה בי הצורך לבנות משחק כזה בעצמי, כך אוכל לתת מענה לכל חבריי המעוניינים לשחק במשחק.

אתגרים צפויים

שני אתגרים מרכזיים ששמתי לב אליהם בתחילת הדרך, עוד מההנחיות לפרויקט, היו הצורך במימוש פרוטוקול תקשורת מלא ומאובטח וכן יצירת ממשק משתמש. יצירת פרוטוקול תקשורת מאובטח זוהי מטלה מסובכת הדורשת הרבה מאמצים על מנת לאפשר מערכת יציבה לאורך זמן שלא נופלת במתקפות וכן שלא מעלה שגיאות תחת אינסוף המקרים שעלולים לקרות בתקשורת ברשת. ממשק משתמש ובאופן ספציפי ממשק משתמש בשפת פייטון תמיד היו ידועים בתור הדברים המסובכים והמתישים ביותר שקיימים בתחום, עיצוב ממשק משתמש שמיש ויפה דורש השקעה רבה ולמידת ספריות ודרכי כתיבת קוד חדשות ומעניינות ששיפרו את יכולות פתירת הבעיות שלי.

הגדרת לקוח

המערכת ושימושה אינם מוגבלים בגיל או במטרות השימוש, כמימוש של משחק טריוויה הוא נועד לכל שכבות הגיל ולכל אדם בעל זיקה למשחקי הילדות, לדעתי האישית השימוש בתוכנה לגיל המבוגר יותר הוא דרך מבורך מכיוון וזהו משחק מבוסס טריוויה שיגרום לך לחשוב על קטגוריות ותבניות אשר לא חשבת עליהן בחייך. חוץ מהיתרונות הברורים שיש למשחק כזה לזיכרון, הוא משחק מהנה גם לילדים אשר לומדים מדינות וערי בירה וגם לילדים המבוגרים מהם שרק רוצים לנוח ולהנות במשחק תחרותי.

סקירת פתרונות קיימים

לאחר חיפוש ארוך גיליתי כי לא קיימות תוכנות או אתרים המאפשרים משחק ארץ עיר אחד על אחד בצורה תחרותית ונאמנה למקור המשחק, כן קיימים אתרים המציעים משחקים נגד הזמן או שמגלים לך את התשובות, אך לא מצאתי משחק עדכני בעל יכולת משחק גבוהה במיוחד.

סקירת טכנולוגיות בפרויקט

בפרויקט לא ממומשות אף סוגי טכנולוגיות חדשות, רוב צורות המימוש הן הסטנדרט בתחום, בין אם בהצפנה המשתמשת בהצפנת AES ולאחריה הצפנת RSA על המפתח של AES, בין אם צורת ניהול הלקוחות בעזרת פיצול ל־thread־ים ובין אם ממשק המשתמש שנוסה להיעשות ב־PYQT וכן ב־flask ולבסוף נעשה עם tkinter.

תיחום הפרויקט

כאמור מלעיל הפרויקט דן ומתעסק רבות ביכולות ההצפנה של המידע העובר בין השרת ללקוח, וכן בשימוש ב־thread־ים שונים לפעולות שונות והקשר ובכך גם ביחס במערכת ההפעלה ל־thread־ים שמטרתם ביצוע פעולות אשר חוסמות פעולות תקשורת אחרות ברשת.

אפיון

המערכת מורכבת משני חלקים נפרדים, צד השרת וצד הלקוח, מטרת צד השרת היא לעבד את בקשות הלקוח ולהגיב בהתאם בעוד מטרת הלקוח היא להריץ ממשק משתמש רציף המאפשר למשתמש שליטה בפונקציונליות השרת ללא גישה פיזית לקוד.

השרת

על מנת לאפשר למספר רב של משתתפים לשחק במקביל, השרת נדרש להשתמש ולזכור מספר רב של חיבורי socket עם הלקוחות השונים, כאשר לכל לקוח יש מספר פונקציונליות עיקריות. בכל תחילת שיחה עם לקוח השרת יתחיל את השיחה בעזרת שליחת הודעות בדומה לצורת three way handshake על מנת להעביר בין השרת והמשתמש את המפתח הפומבי אחד של השני בהצפנת RSA, מעבר לכך בכל הודעה מאותו שלב הוא ישלח את ההודעות בפרוטוקול משותף שנקבע בקוד ועל מנת לאפשר שליחת הודעות בכל אורך הוא מעביר את ההודעות עם הצפנה סימטרית AES כאשר מפתח ההצפנה מוצפן בעצמו בעזרת מפתח ה־RSA הפומבי של הצד השני. מעבר לפרטים הטכניים השרת מאפשר יכולת רישום וכניסה לחשבונות בהן הוא שומר את הניקוד ואת המידע של הלקוח תחת בסיס נתונים חיצוני מוגן בהצפנה, לבסוף השרת מאפשר ללקוח לחכות ללקוח נוסף שיתחבר על מנת לגשר ביניהם למשחק ארץ עיר בממשק המשותף. לבסוף מטרת השרת הינה לספוג כל ניסיון קריסה ופריצה, ובכך הוא משתמש במערכת exceptions המנתבת כל שגיאה לפי בחירת המשתמש בעת הרצת התוכנית עם דגל d- עבור מצב debug. השרת מאפשר קבלת שאילתות מהלקוח וכן בעצמו מאפשר שליחת מידע בהתאם לכל שאילתה וזיהוי שגיאה בתשובת השרת והלקוח בהתאם. בנוסף לכל אלה על השרת לוודא תשובות בעת קבלת תשובות ממשחק בין שני משתמשים, כך עליו לגשת למקומות בהם שמורים התשובות של ארץ עיר ולוודא מולם את נכונותם.

לקוח

הלקוח נדרש לשתי פעולות, תקשורת עם השרת וכן הרצת ממשק משתמש באותו הזמן, תחילה הלקוח כמו השרת מייצר מפתחת הצפנה בשיטת RSA ואת שאר ההודעות מעביר בפרוטוקול המצויין עם ההצפנה הסימטרית כמו השרת, במקביל לזאת הלקוח מאפשר למשתמש שימוש בממשק בצורת חלון במערכת ההפעלה אשר בכל פעולה הדורשת את פעילות השרת תפתח thread נוסף לפי צורך המאפשר לו להפעיל במקביל את הממשק ואת התקשורת עם השרת. מעבר לפרטים הטכניים של צורת ההרצה, הלקוח יכול לעשות שאילתות התחברות, רישום, בקשת נתוני משתמש, בקשת יציאה, בקשת התחלת משחק, התחלת שיחה חדשה. בנוסף

ליכולות הממשק של המשתמש הוא יכול לרוץ גם ללא פתיחת הממשק בצורה מוגבלת על ידי פקודות החיבור, רישום וכל הפקודות עד לרגע תחילת המשחק.

בדיקות

בדיקות לקוד נעשות כמובן פר צורך של השרת והלקוח ומודולים ופונקציות נפרדות, הקוד בנוי בצורה מודולרית שמאפשרת גמישות בפיתוח ובניהול וזיהוי שגיאות ובכך כל חלק מרכזי בקוד נבדק בצורה נפרדת. על הקוד נעשו בדיקות יציבות תקשורת שרת־לקוח, יציבות הפרוטוקול וצורת ההצפנות, דימוי ריצה של פונקציות במערכת סגורה בה ניתן היה לבצע בדיקות, לדוגמה הרצת מבנה הנתונים על ידי המודול הנפרד המיועד לו ובדיקה כללית של כל פונקציה כפרט, לדוגמה פונקציית זיהוי לגיטימציה של אימייל.

לוחות זמנים

על מנת לבצע הליך פיתוח מקצועי, נדרשתי לעמוד בלוחות הזמנים שהקדשתי עבור עצמי,

משימה	**תאריך מתוכנן**	**תאריך בפועל**
חקירת פרויקטים דומים בשוק.	.01.02.2024	.01.02.2024
חקירת פרוטוקולי רשת מתאימים לתוכנה.	.08.02.2024	.06.02.2024
ניסיון התחלה בעזרת flask. לא צלח עקב אי עמידה בתנאי הפרויקט.		
בניית פרוטוקול רשת מתאים.	.18.02.2024	.09.02.2024
בניית צידי שרת ולקוח בסיסיים לקבלת ולשליחת מסרים.	.20.02.2024	.16.02.2024
בניית בסיס נתונים וחיבורו לשרת.	.21.02.2024	.19.02.2024
ניסיון בניית הצפנה מבוססת RSA. לא צלח היות RSA לא מקבל כל אורך של הודעה. לבסוף צלח עם שילוב של RSA ושל AES.		
בניית מערכת הצפנה לפרוטוקול הרשת.	.01.03.2024	.05.03.2024
ניסיון יצירת GUI עם PyQt וכן עם תוכנות RAD. לא צלח, לבסוף הוחלט על tkinter.		
יצירת מערכת GUI בסיסית על מנת להמשיך בקידוד התקשורת.	.01.04.2024	.01.04.2024

חיבור מערכות GUI ללקוח.	.06.04.2024	.05.04.2024
בניית שרת מתקדם בעל יכולת משחק מלאה.	.01.05.2024	.01.05.2024
הצפנת בסיס הנתונים.	.02.05.2024	.02.05.2024
עיצוב GUI.	.09.05.2024	.09.05.2024

סיכונים

על מנת להוריד את הסיכונים האפשריים ביצירת התוכנה נלקח זמן backup למקרה ופונקציונליות מסויימת אינה מתפקדת כראוי, בנוסף ניהול שגיאות תקין ואמין וכן שימוש ב־GitHub על מנת לשמור שינויים בקוד על מנת שיהיה גיבוי לקוד ששונה ויכל להישבר לאחר שינוי.

שגיאות

כנאמר מלעיל, השרת והלקוח משתמשים במערכת שגיאות ובדיקות המכסה כמעט כל שגיאה, וכל אחת שלא כוסתה תעלה כבעיה גנרית מצד השרת והמשתמש על מנת לאפשר לשניהם במידת היכולת להמשיך את התקשרות. לשרת וכן ללקוח יש אפשרות הרצה d- עבור debug המאפשרת הדפסה מלאה של השגיאות כפי שהן ללא התערבות של מערכת תפיסת השגיאות במידת הצורך. השגיאות שכוסו במידה פרטית (על ידי מודול מיוחד שמיועד לשגיאות האלו, מודול internal_exception.py) הן,

1. שגיאת תחילת הרצה, בהרצת התוכנה היא מקבלת ארגומנטים מה־shell, ושגיאה בארגומנטים המועברים תיתן שגיאה אישית של "Invalid arguments.".
2. ניסיון שליחה של הודעה שאינה עומדת בפרוטוקול,
   א. אם הפקודה לא בפרוטוקול תעלה שגיאת "Command not defined in the protocol.".
   ב. אם הפקודה לא קיבלה את הארגומנטים המתאימים תעלה שגיאה "Invalid number of parameters for the command.".
3. בעיות חיבור לשרת,
   א. אם השרת לא עולה בצורה נורמלית תעלה השגיאה "Please check if a server is already running or use a valid ip.".
   ב. אם הלקוח לא מצליח להתחבר לשרת תעלה השגיאה "Please check if a server is already running or use a valid ip"
4. אם יש בעיית חיבור עם הלקוח מצד השרת או יציאה לא מצופה על ידיו תעלה שגיאה בשרת "Server has stopped working due to an error.".
5. אם הלקוח שלח פקודה לא חוקית תעלה בשרת שגיאת "Command given isn't valid.".
6. אם השרת שלח פקודה לא חוקית תעלה בלקוח שגיאת "Command given isn't valid".
7. אם הלקוח שלח פקודה שהשרת לא מצפה לה אך היא חוקית "Command not meant for server.".

8. אם השרת שלח פקודה שהלקוח לא מצפה לה אך היא חוקית "Command not meant for client.".
9. אם עולה שגיאה בתחילת התקשורת בהעברת המפתחות,
   א. אם הפקודה לא תקינה או לא מצופה מהלקוח נקבל "Command given isn't valid.".
   ב. אם הפקודה לא תקינה או לא מצופה מהשרת נקבל "Command given isn't valid.".
   ג. אם חלה שגיאה כללית בחלק זה תעלה בלקוח שגיאה "Handshake failed. Closing connection.".
10. בזמן שהלקוח מחכה למשחק הוא עלולה לעלות שגיאת "Failed to send or get command from the server.".
11. אם בממשק עולה בעיה בהצגת הדפים השונים ביניהם המשתמש עובר תעלה שגיאה "Page % not found.".

מעבר לכל טיפולי השגיאות האלו, במידה וקיימת בעיה בתקשורת השרת יאפשר ללקוח ולעצמו 10 ניסיונות (מספר דיפולטיבי, קבוע בפרוטוקול שניתן לשנות לפי איכות הרשת) להעברת ההודעה המקורית בצורה תקינה אחרת תעלה שגיאת בעיית תקשורת ובקשת התנתקות על מנת לאפשר לצדדים לייצב את הרשת לפני חזרה בשנית לקשר.

תיאור

בשני צדדי המערכת קיימות מספר רב של יכולות אשר אותו צד אמור לבצע,

1. שם היכולת: הרצה במצב debug.
   - מהות: מצב ריצה עבור אנשי בעלי עניין בבעיות העולות בהרצה.
   - נדרש:
     - יכולת ניתוב של כל השגיאות לאיזור מרכזי המנהל אותן בהתאם לבחירת המשתמש.
     - יצירת שגיאות אישיות לכל שגיאת מערכת אפשרית המטפלת באופן יחידני בשגיאה בהתאם לבקשת המשתמש.
     - הדפסת השגיאה בהתאם לרצון המשתמש או ביצוע raise עם עצירת התוכנה.
2. שם היכולת: הקמת תשתית התקשורת.
   - מהות: העברת מפתחות פומביים של השרת והלקוח על מנת לאפשר הצפנת תקשורת.
   - נדרש:
     - יצירת מפתח RSA פרטי ופומבי על ידי השרת אישי לכל משתמש חדש.
     - העברת המפתח הפומבי דרך הרשת אל הלקוח.
     - יצירת מפתח RSA פרטי ופומבי על ידי הלקוח.
     - העברת המפתח הפומבי דרך הרשת אל הלקוח.
     - קבלת תשובה חיובית או שלילית לביצוע הפועלה מהשרת.

3. שם יכולת: הצפנת המידע המועבר בתקשורת.
   - מהות: הצפנת ההודעות העוברות בתקשורת לאחר הקמת התשתית.
   - נדרש:
     - יצירת מפתח AES (הצפנה סימטרית) על ידי שולח ההודעה.
     - הצפנת המידע הרצוי על ידי מפתח ה־AES והצפנת המפתח של AES בעזרת המפתח הפומבי RSA של הצד השני.
     - העברת ההודעה המוצפנת אל הצד השני.
     - פיענוח מפתח ה־AES על ידי מפתח ה־RSA הפרטי.
     - פיענוח ההודעה המוצפנת על ידי מפתח ה־AES.
     - העברת הודעת המשך או תגובה בחזרה באותה שיטה.
4. שם היכולת: הרשמה למערכת.
   - מהות: רישום משתמש חדש במערכת – קליטת שם משתמש, סיסמה נבחרת, ואימייל לאימות.
   - נדרש:
     - מסך בממשק המשתמש על מנת לאפשר הרשמה.
     - יכולת קליטת הנתונים על ידי הלקוח.
     - בדיקת תקינות של האימייל המועבר.
     - העברה בפרוטוקול התקשורת.
     - הצפנה של המידע לשימור בבסיס נתונים.
     - קבלת תשובה חיובית או שלילית לביצוע הפעולה מהשרת.
     - הצגת התשובה למשתמש בממשק.
5. שם היכולת: חיבור למערכת.
   - מהות: חיבור משתמש חוזר למערכת – קליטת שם משתמש וסיסמה.
   - נדרש:
     - מסך בממשק המשתמש על מנת לאפשר התחברות.
     - יכולת קליטת הנתונים על ידי הלקוח.
     - בדיקת תקינות של הסיסמה והשם משתמש (חסרי רווחים).
     - העברה בפרוטוקול התקשורת.
     - השוואה למידע המוצפן בבסיס הנתונים בצד השרת.
     - קבלת תשובה חיובית או שלילית לביצוע הפעולה מהשרת.
     - הצגת התשובה למשתמש בממשק.
6. שם היכולת: יציאה מהמערכת.

- מהות: בקשת יציאה מהשרת.
- נדרש:
  - כפתור בכל דפי הממשק עליו כתוב Exit.
  - יכולת הבנה של לחיצת הכפתור על ידי הלקוח.
  - שליחת בקשת יציאה לשרת.
  - קבלת תשובה חיובית או שלילית.
  - סגירת החיבור עם הלקוח מצד השרת ללא פגיעה בכל שאר המשתמשים.

7. שם היכולת: בקשת מידע של המשתמש.
   - מהות: על מנת להשיג את הניקוד של הלקוח.
   - נדרש:
     - שליחת בקשת יציאה לשרת.
     - הבנה על ידי השרת מהו הלקוח המדובר.
     - הוצאת המידע מבסיס הנתונים.
     - שליחת תשובה חיובית או שלילית עם המידע במידת היכולת.
8. שם היכולת: בקשת התחלת משחק חדש.
   - מהות: בקשת התחלת משחק על ידי הלקוח.
   - נדרש:
     - כפתור בממשק המשתמש על מנת לאפשר בקשה למשחק חדש.
     - יכולת הבנת הלחיצה על ידי הלקוח.
     - פתיחת thread חדש על מנת להעביר את הבקשה וקבלת תשובה במקביל ליכולת שליטה בממשק על ידי המשתמש.
     - העברת הבקשה עם מידע הלקוח בפרוטוקול התקשורת.
     - הוספת הלקוח לתור המחכה להתחברות משתמש נוסף.
     - קבלת תשובה חיובית או שלילית לביצוע הפעולה מהשרת בעת התחברות משתמש נוסף המבקש לשחק.
     - עצירת ה־thread בצד הלקוח והעברת המידע לממשק.
     - הצגת התשובה למשתמש בממשק והתחלת המשחק במידת היכולת.
9. שם היכולת: הגשת תשובות למשחק.
   - מהות: הגשת התשובות לבדיקה על ידי השרת.
   - נדרש:
     - כפתור בממשק המשתמש על מנת לאפשר שליחת התשובות.

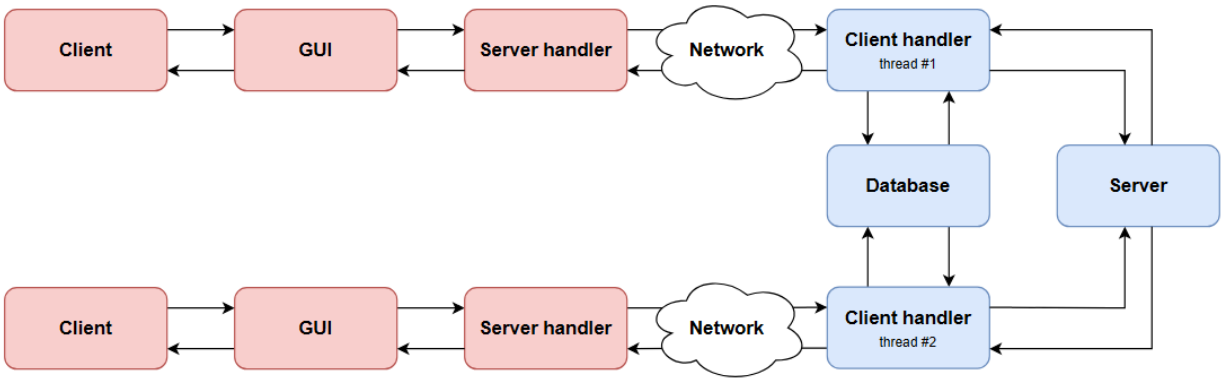
- יכולת הבנת הלחיצה על ידי הלקוח.
- פתיחת thread חדש על מנת להעביר את הבקשה וקבלת תשובה במקביל ליכולת שליטה בממשק על ידי המשתמש.
- העברת הבקשה עם מידע הלקוח בפרוטוקול התקשורת.
- בדיקת תשובות הלקוח אל מול בסיסי הנתונים המיועדים לפתרונות.
- קבלת תשובה חיובית או שלילית לתקינות התשובות על ידי המשתמש ששלח אותן בעוד שליחת הודעת הפסד ועצירת משחק למשתמש השני.
- עדכון הניקוד של שני המשתמשים בצד השרת על ידי גישות לבסיס הנתונים.

מבנה / ארכיטקטורה

ארכיטקטורה וזרימת מידע

זרימת מידע

נתחיל בדיאגרמה המציגה את התהליכים מבחינת high level של הארכיטקטורה המצופה,



זרימת המידע מבחינת פקודות ספציפיות נכנסת למספר קטגוריות, כמובן על כל פקודה והעברת מסרים עלולה לחזור תשובת Error כמתואר בפרוטוקול התקשורת.

עבור פקודת התחלת התקשורת,

1. השרת מקבל חיבור חדש מלקוח, ומעביר אותו לטיפול על ידי client handler.
2. מערכת client handler מייצרת מפתח RSA חדש פרטי ופומבי לשימוש.
3. מערכת client handler שולח פקודת Hello לא מוצפנת עם ארגומנט יחיד, ה־public key שלו בהצפנת RSA.
4. בלקוח מערכת server handler מקבלת את הפקודה, אם יש שגיאה מחזירה Error.
5. מערכת server handler מייצרת מפתח RSA חדש פרטי ופומבי לשימוש.
6. מערכת server handler שולח פקודת Hello לא מוצפנת עם ארגומנט יחיד, ה־public key שלו בהצפנת RSA.
7. בשרת מערכת client handler מקבלת את הפקודה, אם יש שגיאה מחזירה Error.
8. מערכת client handler שולח פקודת Success לא מוצפנת

עבור פקודות הרשמה, התחברות וקבלת מידע בנוגע למשתמש הליך המידע הוא כמפורט,

1. המשתמש הולך בממשק לעמוד התוכן הרצוי (עמוד Login או Signup).
2. המשתמש מקליד את פרטיו ועושה submit.
3. ה־GUI פונה אל המערכת server handler ושולחת פקודת Login (לדוגמה) על פי המצוין בפרוטוקול כולל כל ההצפנות שיפורטו.
4. בשרת המערכת client handler מקבלת את הפקודה ופונה אל בסיס הנתונים.

5. בסיס הנתונים בודק אצלו האם הנתונים אמיתיים, אם כן מחזיר אחורה Success אחרת Fail.
6. הפעם מערכת client handler שולחת את התוצאה חזרה למערכת server handler.
7. המערכת server handler מעבירה למערכת GUI את המידע והיא מציגה זאת על מסך המשתמש.

דרך התחלת משחק, המשתמש חייב להיות מחובר לחשבון!

1. המשתמש הולך בממשק לעמוד "התחלת משחק".
2. ה־GUI פונה אל המערכת server handler דרך thread חדש ושולחת פקודת Waiting על פי המצוין בפרוטוקול כולל כל ההצפנות שיפורטו.
3. בשרת המערכת client handler מקבלת את הפקודה ופונה אל ה־server.
4. השרת בודק האם קיים משתמש אחר שמחכה למשחק, עד שלא קיים כזה המערכת שמה את ה־thread במצב wait.
5. כאשר משתמש חדש מחפש משחק מערכת השרת מעירה את ה־thread הקפוא ומחליפה ביניהם את הפרטים על מנת שיוכלו לשחק.
6. מערכת client handler שולחת פקודת Match על מנת להודיע ללקוח שהמשחק אושר.
7. מערכת server handler מודיע למערכת GUI שמשחק עומד להתחיל עם כל הפרטים.
8. מערכת GUI מציגה ללקוח את נתוני המשחק ומעבירה אותו מעמוד שבו הוא מחכה לעמוד המשחק.

פרוטוקול תקשורת

תחילה נסביר מהו תוכן ההודעה שנשלח תחת הפרוטוקול, לאחר מכן נדבר על מגבלות עליו, ולבסוף על צורת השליחה הסופית. כרגע התקשורת היא מעל פורט 9960 מהיותו חסר שימוש רגולרי (בר שינוי כמו כל קבועי הפרוטוקול על ידי מחלקת הפרוטוקול).

בפרוטוקול יש מגוון פקודות המסוכמות בטבלה הבאה,

שם הפקודה	מטרת הפקודה
ERROR	שגיאה בקבלת הודעה.
HELLO public_key	לתחילת קשר, מצורף המפתח הפומבי בהצפנת RSA.
EXIT	בקשת יציאה מהקשר, נשלח על ידי הלקוח.
SUCCESS	הפעולה בוצעה בהצלחה.

הפעולה לא בוצעה ואף נכשלה.	**FAIL**
בקשת התחברות עם פרטי המשתמש המתחבר.	**LOGIN** username password
בקשת יצירת חשבון עם פרטי יצירת החשבון.	**SIGNUP** username password email
בקשה לקבלת מידע על המשתמש המחובר.	**REQIN**
תשובה לבקשת מידע, מכיל את האימייל, כמות הניצחונות וההפסדים.	**RESIN** email wins losses
בקשה להתחלת משחק מצד הלקוח.	**WAITING**
קבלת יריב למשחק מצד השרת, שם היריב הוא שם המשתמש של היריב והאות היא האות הנבחרת למשחק הארץ עיר.	**MATCH** opponent_username letter
קבלת התשובות מצד הלקוח אל השרת של משחק הארץ עיר, התשובות מתאימות לשאלות במשחק.	**ANSWERS** country city animal plant boy girl

נשים לב כי קיימת בעיה בשימוש פשוט עם RSA, ההודעות מוגבלות לאורך המפתח הנבחר, ומפתח גדול יותר לוקח משמעותית יותר זמן ומשאבים לייצור, הדרך בה פרוטוקול זה עוקפת זאת. בין כל לקוח לשרת נוצרים מפתחות RSA לשימוש ביניהם, כאשר כל הודעה מוצפנת בצורה סימטרית על ידי מפתח AES הנשלח עם ההודעה לאחר שהוא עצמו מוצפן על ידי ה־RSA, כך אנו יודעים מראש את גודל הטקסט שצריך הצפנה תחת RSA וכן ניתן להצפין כל אורך הודעה עם AES.

בפרט צורת השליחה הסופית של הודעה בפרוטוקול תיראה כך,

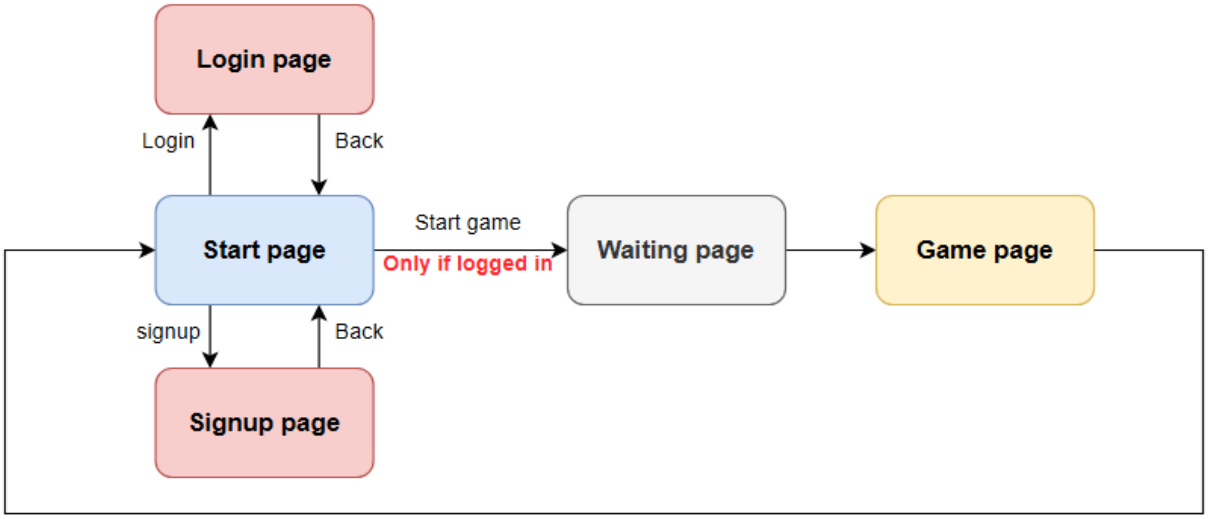
RSA(AES_key) + size_length + message_size + AES(message)

כאשר AES_key הוא מפתח פרוטוקול AES אשר לאחר הפעלת RSA שדה זה בגודל 64 בתים. שדה size_length הוא באורך 4 בתים והוא אומר מהו האורך של שדה message_size. שדה message_size הוא

שדה המתאר את אורך AES(message). השדה AES(message) מתאר את ההודעה לאחר הפעלת message עליה.

מסכי המערכת

צורת זרימת הלקוח בממשק היא כזאת,



מבני נתונים

כאמור בתוכנה היה שימוש במבנה נתונים מסוג SQL הניתן דרך הספריה sqlite3 בשפה python. בסיס הנתונים הוא לוקאלי והוא בעל השדות הבאים תחת הטבלה users,

- שדה username, סוג Text, זהו שדה ראשי אשר דרכו מבוצע זיהוי המשתמש.
- שדה password, סוג Text, השדה מכיל את סיסמת המשתמש.
- שדה email, סוג Text, השדה מכיל את אימייל המשתמש.
- שדה wins, סוג Integer בעל ערך דיפולטיבי 0, שומר את כמות הניצחונות של המשתמש.
- שדה losses, סוג Integer בעל ערך דיפולטיבי 0, שומר את כמות ההפסדים של המשתמש.

טכנולוגיה

ספריות ושפת תכנות

התוכנה כולה נבנתה בשפת python תחת שימוש במספר ספריות,

- ספריות כלולות ב־python,
  - ספריית socket, על מנת ליצור חיבור socket בין השרת והלקוח ב־python.

- ספריית os, על מנת להשתמש ביכולת ה־random שלה.
- ספריית threading, על מנת לפצל את ביצוע התוכנה ל־thread־ים שונים שיעשו פעולות שונות במקביל.
- ספריית enum, על מנת לייצר מחלקה של פקודות מותרות בפרוטוקול.
- ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
- ספריית re, על מנת לוודא regex של אימייל.

- ספריות שאינן כלולות ב־python,
  - ספריית rsa, על מנת לייצר מפתחות, הצפנה ופענוח RSA כנכתב מלעיל אנו משתמשים בספרייה חיצונית.
  - ספריית Cryptodome, על מנת לייצר מפתחות, הצפנה ופענוח AES נשתמש במודול Cryptodome.Cipher המאפשר שימוש בהצפנה זו.
  - ספריית uuid, על מנת לייצר אובייקט בתים רנדומליים עבור nonce של הצפנת AES.
  - ספריית sqlite3, על מנת ליצור ולהשתמש בבסיס נתונים SQL לוקאלי.
  - ספריית tkinter, על מנת לייצר ממשק משתמש כ־window של מערכת ההפעלה.

סביבת עבודה

כאמור מלעיל כל הקוד בוצע בשפת python 3.12 ובעזרת משאבים וספריות בה, אם כך הקוד עובד בכל מערכת הפעלה עליה python רץ באופן טבעי. בפיתוח השתמשי במערכת הפעלה Windows11 וכן בתוכנות העריכה PyCharm premium עם Visual Studio code.

הצפנה

בפרויקט זה נעשה שימוש בשתי דרכי הצפנה שונות, כמפורט בפרוטוקול.

הצפנת RSA, הינה שיטת הצפנה אסימטרית, כלומר בעלת מפתח פומבי ופרטי שונים, אשר משתמשת במפתח הפרטי שלה על מנת לפענח קודים אשר הוצפנו על ידי המפתח הציבורי שלה. הדבר החשוב באלגוריתם זה הוא היות המפתח הפרטי שמור רק לעצמך, והאלגוריתם מתבסס על כך שלא ניתן לחשב באופן יעיל את המפתח הפרטי על ידי המפתח הציבורי בעולם פרה־קוונטי.

הצפנת EAS, הינה שיטת הצפנה סימטרית, כלומר בעלת מפתח אחד הנועד גם לפענוח וגם להצפנת מידע. לעומת הצפנת RSA הצפנה זו עובדת על כל אורך של טקסט והיא מתבססת על כך שלא ניתן לחשב באופן יעיל את המפתח של ההצפנה על ידי ההודעה המוצפנת על ידיו.

חולשות ואיומים

במהלך הפרויקט נעשו מאמצים רבים למנוע איומים על המערכת. תחילה צורת השימוש בספריית sqlite3 ב־python חסינה מפני תקיפות sql injection, וידוא תהליך ה־login נעשה מול מבנה נתונים מוצפן, בקוד הקיים לא יכולות להיות התקפות DDOS או DOS מכיוון וצורת התקשורת עם השרת היא הודעה הודעה וכן יש זמן

מוקצב לכל לקוח בין ביצוע פעולות שאם הוא לא מקיים אותו הוא נחשב idle והקשר איתו מתנתק. כמו כן מבחינת התעבורה כנאמר בפרוטוקול יש שימוש בהצפנה המוגדרת בלחיצת יד משולשת, RSA וכן בפרוטוקול AES בעלי מפתחות גדולים אשר אינם פריצים בעולם פרה־קוונטי בזמן סביר.

מימוש הפרויקט

פירוט מימוש

מחלקת AESCipher בקובץ aes_cipher.py

מטרת המחלקה היא יצירת אובייקט היכול לפענח ולהצפין על פי אלגוריתם AES ותוך כדי לקבוע קבועים גלובליים למטרה זו.

- מודולים,
  - ספריית os, על מנת להגריל מספר בתים מסוים.
  - ספריית uuid, על מנת להגדיר nonce רנדומלי לפרוטוקול AES.
  - ספריית Cryptodome.Cipher.AES, על מנת להביא מימוש של פרוטוקול AES.
- תכונות,
  - תכונה padding_character, מסוג char, זהו התו איתו יעשה padding מהיות AES מקבל רק טקסט באורך כפולות של 16.
  - תכונה AES_key_length, מסוג int, אורך המפתח של פרוטוקול AES בבתים.

```
def __init__(self, key: bytes | None = None, nonce: bytes | None = None):
    """
    Constructor for AESCipher class.
    :param key: the key to use for the AES cipher. If None, a random key will
be generated.
    :param nonce: the nonce to use for the AES cipher.
    """
def encrypt_message(self, msg: str) -> bytes:
    """
    Encrypt a message using AES algorithm.
    :param msg: the message to encrypt.
    :return: the encrypted message.
    """
def decrypt_message(self, msg: bytes) -> str:
    """
    Decrypt a message using AES algorithm.
    :param msg: the message to decrypt.
    :return: the decrypted message.
    """
```

מחלקת CommandName בקובץ command.py

מטרת המחלקה היא יצירת אובייקטים שאפשר לגשת אליהם דרך השם שלהם וגם ערכם על מנת לאפשר ניהול קל של פקודות ושמירה מהירה ויעילה בזיכרון. המחלקה יורשת ממחלקת enum.

- מודולים,
  - ספריית enum, על מנת לאפשר למחלקה לשמור רק ערכים אפשריים למחלקה זו.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
- תכונות,
  - תכונה ERROR, מסמל error message בפרוטוקול.
  - תכונה HELLO, מסמל hello message בפרוטוקול.
  - תכונה EXIT, מסמל exit message בפרוטוקול.
  - תכונה SUCCESS, מסמל success message בפרוטוקול.
  - תכונה FAIL, מסמל fail message בפרוטוקול.
  - תכונה LOGIN, מסמל login message בפרוטוקול.
  - תכונה SIGNUP, מסמל signup message בפרוטוקול.
  - תכונה INFO_REQUEST, מסמל info request message בפרוטוקול.
  - תכונה INFO_RESPONSE, מסמל info response message בפרוטוקול.
  - תכונה WAITING, מסמל waiting message בפרוטוקול.
  - תכונה MATCH, מסמל match message בפרוטוקול.
  - תכונה ANSWES, מסמל answers message בפרוטוקול.

מחלקת Command בקובץ command.py

מטרת המחלקה היא יצירת פקודה בעלת ארגומנטים.

- מודולים,
  - ספריית enum, על מנת לאפשר למחלקה לשמור רק ערכים אפשריים למחלקה זו.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
- תכונות,
  - תכונה params_per_command, מסוג dict[CommandName, int], מילון המתאר כמה ארגומנטים כל סוג פקודה לוקח.

```
def __init__(self, *args: str | bytes | CommandName):
    """
    Constructor for Command class, calls other 'constructors'.
```

```
    :param args: the command and parameters passed in the socket.
    """
def __init_one(self, data: str | bytes):
    """
    Constructor for Command class.
    :param data: the command and parameters passed in the socket.
    """
@staticmethod
def extract_words(data: str) -> list[str]:
    """
    Extract the command and parameters from a valid request. For example, for
'DIR path' the returned value is ['DIR', path].
    :param data: the command and parameters passed in the socket.
    :returns: a list of the command and its parameters.
    """
```

מחלקת InternalException בקובץ internal_exception.py

מטרת המחלקה היא יצירת אובייקט שתופס error על מנת לבצע איתו לבקשת המשתמש, הדפסה בפורמט מיוחד או ללא הדפסה כלל והתעלמות. המחלקה יורשת ממחלקת Exception.

- תכונות,
  - תכונה debug, מסמל האם האדם בחר להריץ במצב debug את התוכנה ולפי זאת להראות את השגיאות בתצורה כזאת או אחרת.

```
def __init__(self, message, e: Exception = None):
    """
    Constructor for InternalException class.
    :param message: message to print when exception is raised.
    """
```

מחלקת Protocol בקובץ protocol.py

מטרת המחלקה היא לטפל בקבלת משתמשים חדשים וקישוריות ביניהם במשחקים.

- מודולים,
  - ספריית socket, על מנת ליצור תקשורת.
  - ספריית rsa, על מנת להשתמש בפרוטוקול הצפנה RSA.
  - מחלקת Command, על מנת להשתמש בקונבנציית הפקודות שנבחרה.

- מחלקת AESCipher, על מנת להשתמש בפרוטוקול הצפנה AES.

- תכונות,
  - תכונה LENGTH_FIELD_SIZE, מסוג int, אורך החלק שאומר את גודל החלק של הגודל של המידע בפרוטוקול.
  - תכונה PORT, מסוג int, אומר את ה־port של התקשורת.
  - תכונה ERROR_LIMIT, מסוג int, כמות השגיאות ברצף בתעבורה לפני שמבוצע ניתוק בין השרת ללקוח.
  - תכונה RSA_KEY_SIZE, מסוג int, גודל מפתח בפרוטוקול RSA בביטים.

```
@staticmethod
def create_msg(cmd: Command, public_key: rsa.PublicKey | None = None) ->
bytes:
    """
    Create a message according to the protocol.
    :param cmd: the command to send.
    :param public_key: the public key to encrypt the message with, or None if
shouldn't be encrypted.
    :returns: the message to send.
    """
@staticmethod
def get_msg(sock: socket.socket, private_encrypt: rsa.key.PrivateKey | None =
None) -> tuple[bool, Command | None]:
    """
    Extract message from protocol, without the length field.
    :param sock: active socket to receive from.
    :param private_encrypt: if the data is encrypted will contain the private
RSA key.
    :returns: a bool representing if the data and protocol are valid. And a
Command instance or null, depending on the validity.
    """
```

מחלקת Server בקובץ server.py

מטרת המחלקה היא ליצור פרוטוקול אחיד בין השרת והלקוח.

- מודולים,
  - ספריית socket, על מנת ליצור תקשורת.
  - ספריית threading, על מנת לפתוח פונקציות במקביל על thread־ים שונים.

- ספריית rsa, על מנת להשתמש בפרוטוקול הצפנה RSA.
- מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
- מחלקת ClientHandler, על מנת לטפל בכל משתמש בנפרד על thread נפרד.
- מחלקת Protocol, על מנת לאפשר שימוש בפרוטוקול.
- מחלקת Database, על מנת לאפשר גישה לבסיס הנתונים.

```
def __init__(self, ip_address: str, port: int = Protocol.PORT):
    """
    Constractor for Server class, creates a socket with parameters given.
    :param ip_address: the ip address of the server.
    :param port: the port of the server. Default is as in the mutual
protocol, not a specific case used port.
    """
def stop_server(self):
    """
    Method to stop the server.
    """
def listen_for_exit(self):
    """
    Listen for user input to stop the server.
    """
def main(self):
    """
    The main function of the server, handles the server connections and
threads.
    """
```

מחלקת ClientHandler בקובץ client_handler.py
מטרת המחלקה היא לטפל במשתמש יחיד.

- מודולים,
  - ספריית socket, על מנת ליצור תקשורת.
  - ספריית random, על מנת לבחור אות רנדומלית ליצירת משחק.
  - ספריית threading, על מנת לפתוח פונקציות במקביל על thread־ים שונים.
  - ספריית rsa, על מנת להשתמש בפרוטוקול הצפנה RSA.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
  - מחלקות Command, על מנת להשתמש בקונבנציית הפקודות שנבחרה.

- מחלקת ClientHandler, על מנת לטפל בכל משתמש בנפרד על thread נפרד.
- מחלקת Protocol, על מנת לאפשר שימוש בפרוטוקול.
- מחלקת Database, על מנת לאפשר גישה לבסיס הנתונים.

- תכונות,
  - תכונה hebrew_alphabet, מסוג list[str], אותיות האלפבית העברי.

```
def __init__(self, client_socket: socket.socket, client_address: tuple[str,
int], server: "Server"):
    """
    Constractor for ClientHandler class, initializes the client values.
    :param client_socket: the socket of the client.
    :param client_address: the address of the client.
    :param server: the parent server object.
    """
def login_request(self, username: str, password: str) -> Command:
    """
    Check if the user login info is valid in the database.
    :param username: the username of the client.
    :param password: the password of the client.
    :returns: the response command to the client.
    """
def signup_request(self, username: str, password: str, email: str) ->
Command:
    """
    Check if the user login info are valid in the database, if they are add
them to the database.
    :param username: the username of the client.
    :param password: the password of the client.
    :param email: the email of the client.
    :returns: the response command to the client.
    """
def info_request(self) -> Command:
    """
    Get the info of the user.
    :returns: the response command to the client.
    """
def random_letter_for_match(self) -> str:
    """
```

```
    Get a random hebrew letter for the match.
    :returns: a random hebrew letter.
    """
def wait_for_match(self) -> Command:
    """
    Wait for a user pair for a match.
    """
def handle_request(self, validity: bool, cmd: Command, prev_cmd: Command) ->
Command:
    """
    Handles the request from the client.
    :param validity: the validity of the command.
    :param cmd: the command to handle.
    :param prev_cmd: the last command sent from server.
    :returns: the response command to the client.
    """
def three_way_handshake(self) -> rsa.PublicKey:
    """
    Perform a three-way handshake with the client.
    :return: True if the handshake was successful, False otherwise.
    """
def handle_client(self) -> None:
    """
    Handles the sending and receiving from the client.
    """
```

מחלקת Database בקובץ database.py

מטרת המחלקה היא לטפל בקשר עם בסיס הנתונים.

- מודולים,
  - ספריית re, על מנת ליצור regex.
  - ספריית sqlite3, על מנת ליצור בסיס נתונים SQL.

```
def __init__(self, database_file: str = 'Server/Database/database.db'):
    """
    Constructor for the Database class.
    :param database_file: the file path of the database.
    """
```

```
def __str__(self):
    """
    String representation of the Database class.
    """
def __del__(self):
    """
    Destructor for the Database class, closes the connection to the database.
    """
@staticmethod
def is_valid_email(email: str) -> bool:
    """
    Check if the email is a valid format.
    """
def is_valid_user(self, username: str, password: str) -> bool:
    """
    Check if a user exists in the database.
    :param username: the username of the user.
    :param password: the password of the user.
    :return: True if the user exists, False otherwise.
    """
def add_user(self, username: str, password: str, email: str) -> bool:
    """
    Add a new user to the database.
    :param username: the username of the user.
    :param password: the password of the user.
    :param email: the email of the user.
    :return: True if the user was added successfully, False otherwise.
    """
def get_mail(self, username: str) -> str:
    """
    Get the email of a user.
    :param username: the username of the user.
    :return: the email of the user.
    """
def get_score(self, username: str) -> tuple[int, int]:
    """
    Get the score of a user, tuple of wins and losses.
    :param username: the username of the user.
```

```
    :return: the score of the user.
    """
def set_score(self, username: str, score: tuple[int, int]) -> bool:
    """
    Set the score of a user.
    :param username: the username of the user.
    :param score: the score of the user.
    :return: True if the score was set successfully, False otherwise.
    """
```

מחלקת Client בקובץ client.py

מטרת המחלקה היא לטפל בקשר עם השרת מצד המשתמש.

- מודולים,
  - ספריית socket, על מנת ליצור תקשורת.
  - ספריית rsa, על מנת להשתמש בפרוטוקול הצפנה RSA.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
  - מחלקות Command, על מנת להשתמש בקונבנציית הפקודות שנבחרה.
  - מחלקת Protocol, על מנת לאפשר שימוש בפרוטוקול.
  - מחלקת Window, על מנת ליצור ממשק GUI.
- תכונות,
  - תכונת username, מסוג None | str, שם המשתמש של הלקוח.

```
def __init__(self, ip_address: str, port: int = Protocol.PORT):
    """
    Constractor for Client class, creates a socket with parameters given.
    :param ip_address: the ip address of the server.
    :param port: the port of the server. Default is as in the mutual
protocol, not a specific case used port.
    """
@staticmethod
def get_command_from_user() -> Command:
    """
    Get a command from the user, used for no GUI.
    :return: the command to send to the server.
    """
```

```
@staticmethod
def handle_server_response(validity: bool, cmd: Command, last_command:
Command) -> Command:
    """
    Handle the request from the client, used for no GUI.
    Has limited commands available, Login, Signup, Exit, Error.
    :param validity: the validity of the command.
    :param cmd: the command to handle.
    :param last_command: the last command that the user has sent to the
server.
    :returns: the response command to the client.
    """
def three_way_handshake(self) -> rsa.PublicKey:
    """
    Perform a three-way handshake with the client.
    :return: True if the handshake was successful, False otherwise.
    """
def main_user_input(self) -> None:
    """
    Used as the main function of the client when no GUI is needed,
responsible for the communication with the server.
    """
def main(self):
    """
    Main function of the client with GUI.
    """
def send_and_get(self, cmd: Command) -> tuple[bool, Command]:
    """
    Send a command to the server and get a response.
    :param cmd: the command to send to the server.
    :return: the validity of the command and the command itself.
    """
```

מחלקת GamePage בקובץ game_page

מטרת המחלקה היא עמוד המשחק בממשק משתמש. המחלקה יורשת ממחלקת PageTemplate.

- מודולים,

- ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
- ספרייה tkinter, על מנת לייצר ממשק משתמש.
- מחלקת PageTemplate, על מנת לירוש את התבנית הכללית של עמוד בממשק.

```
def __init__(self, window: "Window"):
    """
    Initialize the game page.
    :param window: the window of the application.
    """
def show_self(self) -> None:
    """
    Show the frame.
    """
def place_widgets(self) -> None:
    """
    Place the widgets in the frame.
    """
def dec_timer(self) -> None:
    """
    Decrease the timer by 1 second.
    """
def start_timer(self) -> None:
    """
    Start the timer.
    """
```

מחלקת LoginPage בקובץ login_page

מטרת המחלקה היא עמוד ההתחברות בממשק משתמש. המחלקה יורשת ממחלקת PageTemplate.

- מודולים,
  - ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.
  - מחלקת PageTemplate, על מנת לירוש את התבנית הכללית של עמוד בממשק.
  - מחלקות Command, על מנת להשתמש בקונבנציית הפקודות שנבחרה.

```
def __init__(self, window: "Window") -> None:
    """
    Initialize the login page.
    :param window: The window of the application.
    """
def place_widgets(self) -> None:
    """
    Place the widgets in the frame.
    """
def lock_entries(self):
    """
    Lock the entries while used for logging.
    """
def unlock_entries(self):
    """
    Unlock the entries while used for logging.
    """
def submit_login_info(self):
    """
    Submit the login information to the server.
    """
def open_start_game_button(self):
    """
    Open the start game button for the client after logging in.
    """
```

מחלקת PageTemplate בקובץ page_template

מטרת המחלקה היא תבנית כללית לעמוד בממשק משתמש. המחלקה יורשת ממחלקת tk.Canvas.

- מודולים,
  - ספריית __future__.annotations, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.

```
def __init__(self, window: "Window"):
    """
    Template to create a new page in the application.
```

```
    :param window: the window parent of the frame.
    """
def show_self(self) -> None:
    """
    Show the frame.
    """
def unshow_self(self):
    """
    Unshow the frame.
    """
def place_widgets(self) -> None:
    """
    Place the widgets in the frame.
    """
def exit_event(self):
    """
    If got here, then the client wishes to close the game.
    """
```

מחלקת SignupPage בקובץ signup_page

מטרת המחלקה היא עמוד הרשמה בממשק משתמש. המחלקה יורשת ממחלקת PageTemplate.

- מודולים,
  - ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.
  - מחלקת PageTemplate, על מנת לירוש את התבנית הכללית של עמוד בממשק.
  - מחלקות Command, על מנת להשתמש בקונבנציית הפקודות שנבחרה.

```
def __init__(self, window: "Window") -> None:
    """
    Initialize the signup page.
    :param window: The window of the application.
    """
def place_widgets(self) -> None:
    """
```

```
    Place the widgets in the frame.
    """
def lock_entries(self):
    """
    Lock the entries while used for signuping.
    """
def unlock_entries(self):
    """
    Unlock the entries while used for signuping.
    """
def submit_signup_info(self):
    """
    Submit the signup information to the server.
    """
def open_start_game_button(self):
    """
    Open the start game button for the client after logging in.
    """
```

מחלקת StartPage בקובץ start_page

מטרת המחלקה היא העמוד הראשי בממשק משתמש. המחלקה יורשת ממחלקת PageTemplate.

- מודולים,
  - ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.
  - מחלקת PageTemplate, על מנת לירוש את התבנית הכללית של עמוד בממשק.

```
def __init__(self, window: "Window") -> None:
    """
    Initialize the login page.
    :param window: The window of the application.
    """
def place_widgets(self) -> None:
    """
    Place the widgets in the frame.
    """
```

מחלקת WaitingPage בקובץ waiting_page
מטרת המחלקה היא העמוד שבוא מחכים למשחק בממשק משתמש. המחלקה יורשת ממחלקת PageTemplate.

- מודולים,
  - ספריית annotations.__future__, על מנת לאפשר type hints שאחרת היו זקוקים לשימוש בשגיאה circular imports.
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.
  - מחלקת PageTemplate, על מנת לירוש את התבנית הכללית של עמוד בממשק.
  - ספריית threading, על מנת לפתוח פונקציות במקביל על thread־ים שונים.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.

```
def __init__(self, window: "Window"):
    """
    Initialize the waiting page.
    :param window: the main window of the application.
    """
def show_self(self) -> None:
    """
    Show the frame.
    """
def process_response(self):
    """
    Process the response from the server.
    """
def ask_to_play(self) -> None:
    """
    Ask the server to play.
    """
def place_widgets(self) -> None:
    """
    Place the widgets in the frame.
    """
```

מחלקת Window בקובץ window
מטרת המחלקה היא להריץ חלון של הממשק, המחלקה יורשת ממחלקת tk.Tk.

- מודולים,
  - ספרייה tkinter, על מנת לייצר ממשק משתמש.
  - מחלקת InternalException, על מנת לאפשר הרמת error בצורה שבה המשתמש בוחר.
  - ספרייה typing.Callable, על מנת לעשות type hints שהן פונקציות.
  - ספרייה PIL, על מנת לייבא תמונת רקע לממשק.
  - ספרייה GamePage, על מנת להראות את העמוד במידת הצורך.
  - ספרייה LoginPage, על מנת להראות את העמוד במידת הצורך.
  - ספרייה PageTemplate, על מנת להשתמש בתור type hint לעמוד.
  - ספרייה SignupPage, על מנת להראות את העמוד במידת הצורך.
  - ספרייה StartPage, על מנת להראות את העמוד במידת הצורך.
  - ספרייה WaitingPage, על מנת להראות את העמוד במידת הצורך.
- תכונות,
  - תכונת window_size, מסוג tuple[int, int], מגדיר את גודל החלון.
  - תכונת background_image_path, מסוג str, המסלול לתמונת הרקע בקבצים.
  - תכונת pages, מסוג list[PageTemplate], מכיל את כל העמודים החוקיים להעביר ביניהם.

```
def __init__(self, client: "Client"):
    """
    Initialize the main window of the application.
    """
def initialize_image(self) -> ImageTk.PhotoImage:
    """
    Initialize the background image of the window.
    :return: the background image of the window.
    """
def show_page(self, page_name: str) -> Callable[[], None]:
    """
    Show the page with the given name.
    :param page_name: the page name of the page to show.
    :return: the function to show the page, to give button command to call.
    """
def inner_show_page() -> None:
    """
    Show the page with the given name.
    """
```

```
def initialize_pages(self) -> None:
    """
    Initialize the pages of the application.
    """
```

מסמך בדיקות

בדיקות לקוד נעשות כמובן פר צורך של השרת והלקוח ומודולים ופונקציות נפרדות, הקוד בנוי בצורה מודולרית שמאפשרת גמישות בפיתוח ובניהול וזיהוי שגיאות ובכך כל חלק מרכזי בקוד נבדק בצורה נפרדת. על הקוד נעשו בדיקות יציבות תקשורת שרת־לקוח, יציבות הפרוטוקול וצורת ההצפנות, דימוי ריצה של פונקציות במערכת סגורה בה ניתן היה לבצע בדיקות, לדוגמה הרצת מבנה הנתונים על ידי המודול הנפרד המיועד לו ובדיקה כללית של כל פונקציה כפרט, לדוגמה פונקציית זיהוי לגיטימציה של אימייל.

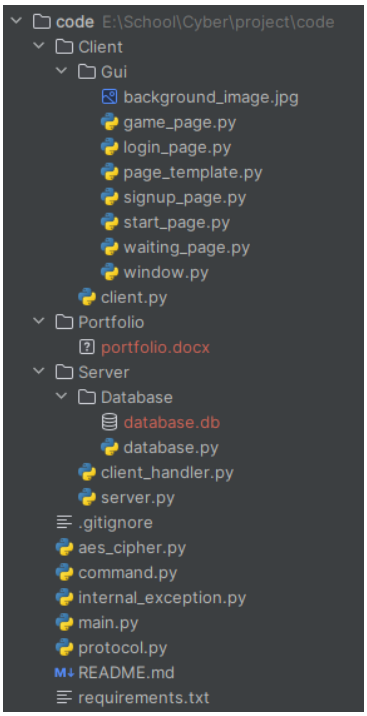
על מנת לוודא שהקוד תקין נעשו מספר בדיקות במהלך שלבי הפיתוח השונים,

- בדיקת פעילות שרת,
  - יצירת קשר עם השרת בעזרת לקוח מדומה אשר שולח את כל רצף הפקודות האפשריות וכל שגיאה מצידו ורואה כיצד השרת מגיב בהתאם.
  - הכל עבד נהדר בעקבות צורת בניית הקוד.
- בדיקת פעילות לקוח,
  - יצירת קשר עם הלקוח בעזרת שרת מדומה אשר שולח את כל רצף התגובות האפשריות וכל שגיאה מצידו ורואה כיצד הלקוח מציב בהתאם.
  - הכל עובד נהדר בעקבות צורת בניית הקוד.
- בדיקות בסיס נתונים,
  - יצירת בסיס נתונים מדומה והרצת כל הפונקציונליות האפשריות בו כולל שגיאות זמן ריצה אפשריות.
  - הכל עובד נהדר בעקבות צורת בניית הקוד.
- בדיקות הצפנה תעברותית,
  - יצירת לקוח ושרת מדומים אשר שולחים ביניהם פקודות חסרות משמעות על מנת לראות האם תהליך ההצפנה תקין משני הכיוונים.
  - גיליתי מספר פעמים שהפרוטוקול נשבר במקרי קצה, לכן נדרש היה תיקון לסוגייה וטופלה.
- בדיקות ממשק,
  - יצירת לקוח מדומה אשר מריץ ממשק באשר אני אנסה כל קומבינציה אפשרית של הכנסות של קלטים ועמודים שונים על מנת לראות את התגובה.
  - הכל עובד נהדר בעקבות צורת בניית הקוד.

מדריך למשתמש

עץ קבצים

בפרויקט הייתה כתיבת קוד מודולרית ככל הניתן, וניתן לשים לב לזה בעץ הקבצים,



נעבור ונסביר קובץ קובץ,

- תיקייה Client, כל הפעולות המיועדות ללקוח.
  - תיקייה Gui, כל פעולות הממשק ועיצובו.
    - קובץ background_image, תמונת הרקע של הממשק.
    - קובץ game_page, עמוד המשחק עצמו בממשק.
    - קובץ login_page, עמוד ההתחברות בממשק.
    - קובץ page_template, מחלקת העמוד ממנה כל שאר העמודים יורשים את תכונותיהם ואת צורת היצירה וההצגה שלהם.
    - קובץ signup_page, עמוד ההרשמה בממשק.
    - קובץ start_page, עמוד ההתחלה של הממשק.
    - קובץ waiting_page, העמוד בו מחכים למשחק בממשק.
    - קובץ window, עמוד המגדיר את הממשק עצמו ומחליף בין הדפים.
  - קובץ client, מנהל את הקשר בין הלקוח לשרת וכן מריץ את הלולאה הראשית של הלקוח הכוללת את הממשק.

- תיקייה Portfolio, תיק פרוייקט.
  - קובץ portfolio, קובץ זה.
- תיקייה Server, כל הפעולות המיועדות לשרת.
  - תיקייה Database, כל פעולות בסיס הנתונים.
    - קובץ database.db, קובץ מבנה הנתונים עצמו ששומר את כל הנתונים.
    - קובץ database.py, קובץ שמתקשר בין מבנה הנתונים למערכת השרת ב־python.
  - קובץ client_handle, הקובץ המטפל בלקוח באופן אישי על thread נפרד.
  - קובץ server, מקבל לקוחות חדשים ושם אותם עם client_handler משלהם.
- קובץ aes_cipher, קובץ פעולות ההצפנה של אלגוריתם AES.
- קובץ command, מכיל בתוכו את רשימת הפקודות החוקיות עם הנתונים שלהן.
- קובץ internal_exception, הקובץ האחראי לכל השגיאות והניתוב שלהם בפרויקט.
- קובץ main, קובץ ההרצאה המתחיל את התוכנה בהתאם לארגומנטים.
- קובץ protocol, קובץ המתאר את פרוטוקול התקשורת וכל יצירת וקריאת ההודעות בתקשורת.
- קובץ requirements, קובץ כל הספריות הדרושות להרצת הפרויקט.

קונפיגורציית ריצה

על מנת להריץ את התוכנה יש להשתמש בגרסת python 3.12, אין להניח כי התוכנה תרוץ על גרסאות ישנות יותר, מעבר לכך אין צורך באף כלי נוסף. כל הוראות השימוש מעבר לצורת הרצת הקובץ נמצאות בממשק המשתמש עבור הלקוח, בעוד שהשרת בעל יכולת shell יחידה בזמן ריצה, והיא שכתיבת EXIT תוביל לסגירת השרת, למרות ששיטות כמו alt+break יעבדו כמו כן באפקטיביות.

הרצת הלקוח

על מנת להריץ את הלקוח עם חיבור לשרת הנמצא בכתובת 0.0.0.0 יש לכתוב את הפקודה הבאה ב־shell,

```
python main.py client 0.0.0.0
```

וכמובן לשנות את הכתוב לכתובת השרת בהתאם.

בנוסף לצורת הריצה הרגילה קיימת ריצה בצורת debug המאפשרת לראות את כל השגיאות שנוצרו,

```
python main.py -d client 0.0.0.0
```

וכן בצורה ניסיונית בלבד, על מנת להריץ ללא ממשק משתמש ניתן להריץ דרך PyCharm את הקובץ Client/client.py כאשר הפקודות האפשריות הן כל הפקודות עד כדי שלב התחלת המשחק בתוכנה.

הרצת השרת

על מנת להריץ את השרת יש לכתוב את הפקודה הבאה ב־shell,

```
python main.py server
```

וכמובן לשנות את הכתוב לכתובת השרת בהתאם.

בנוסף לצורת הריצה הרגילה קיימת ריצה בצורת debug המאפשרת לראות את כל השגיאות שנוצרו,

```
python main.py -d server
```

רפלקציה / סיכום אישי

דעתי על העבודה שנויה במחלוקת. מאוד נהניתי לראות כיצד החומר הנלמד במשך השנים האחרונות בבית הספר הפך מחומר תאורטי לחומר פרקטי, בנוסף היה מעניין ללכלך קצת עם הידיים עם פרויקט אמיתי שמעניין אותי לשם שינוי. נדהמתי מכמות הלמידה שנדרשה לי על מנת להצליח לעבוד על הפרויקט כראוי, למדתי לעצב גרפית עם ספריות שונות ומגוונות בפייתון שזהו תחום שלא ציפיתי לעשות, ולמדתי על מגבלות של הצפנות מסוימות על פני אחרות וכיצד להגן מפני מתקפות כאלו ואחרות. כנראה שהייתי משנה את הפרויקט להיות בעל משתמש web אך לצערי לא נראה שזה עמד בצורה כל כך תקינה בדרישות הפרויקט שניתנו, כמו כן הייתי משקיע יותר בגרפיקה מקצועית אחרי שהבנתי כמה זמן דבר שנראה כל כך פשוט כמו זה לוקח באמת. הדבר שנהניתי באמת להתעסק בו מבחינת הלמידה הוא צורת השימוש ב־thread־ים בפייתון, זהו נושא שתמיד עניין אותי להבין וללמוד לעומק אך אף פעם לא יצא לי ללמוד אותו כמו שצריך ובטח שלא פיזית לעבוד איתו בצורה שדורשת ממני הבנה כזו או אחרת. החיסרון היחיד שאני חוויתי בפרויקט הוא תיק הפרויקט, לדעתי הדרישות בתיק הפרויקט דרושות שיפור על מנת לאפשר לתלמידים שעמלו כל כך קשה על פרויקט תכנותי מאט מקום לקוד ולא רק לדברים הפחות חשובים בפועל כמו תיק פרויקט. לדעתי האישית הוספת דוקומנטציה אמיתית כמו Javadoc או pydoc הייתה מעלה את רמת העבודה פי כמה וכמה ומלמדת את התלמידים צורת כתיבת קוד נכונה ולא צורת כתיבת מסמכים ארוכים כמו במדעי הרוח. בפרט החלק של דוקומנטציה בתיק הפרויקט לקח זמן רב שחבל שלקח בזמן שהקוד מלא בדוקומנטציה מכף רגל ועד ראש שלא היה ניתן להשתמש עקב דרישות פורמט יחודיות בתיק הפרויקט. סך הכל אבל נהניתי ולמדתי המון על הרבה מאוד תחומים בפרויקט ומעבר לכל שמחתי להיות חלק מהמגמה המדהימה הזאת עם התלמידים וצוות המורים המופלא שעזר בכל שיעור ובכל פנייה.

נספחים

קוד הפרויקט המלא מצורף כאן,

קובץ aes_cipher.py

```
import os
import uuid

from Cryptodome.Cipher import AES


class AESCipher:
    """
    AESCipher class to encrypt and decrypt messages of any length using AES
algorithm.
    Based on, https://gist.github.com/syedrakib/d71c463fc61852b8d366.
    """

    padding_character: str = "{"
    AES_key_length: int = 16  # AES key length in bytes

    def __init__(self, key: bytes | None = None, nonce: bytes | None = None):
        """
        Constructor for AESCipher class.
        :param key: the key to use for the AES cipher. If None, a random key
will be generated.
        :param nonce: the nonce to use for the AES cipher.
        """

        if key is None:
            self.key: bytes = os.urandom(self.AES_key_length)
        else:
            self.key: bytes = key

        if nonce is None:
            self.nonce: bytes = uuid.uuid4().bytes
        else:
            self.nonce: bytes = nonce

    def encrypt_message(self, msg: str) -> bytes:
        """
        Encrypt a message using AES algorithm.
        :param msg: the message to encrypt.
        :return: the encrypted message.
        """

        cipher = AES.new(self.key, AES.MODE_EAX, nonce=self.nonce)
        padded_private_msg = msg + (self.padding_character * ((16 - len(msg))
% 16))  # AES works only on blocks of 16 bytes
        encrypted_msg = cipher.encrypt(padded_private_msg.encode("utf-8"))
        return encrypted_msg

    def decrypt_message(self, msg: bytes) -> str:
        """
        Decrypt a message using AES algorithm.
```

```
        :param msg: the message to decrypt.
        :return: the decrypted message.
        """

        cipher = AES.new(self.key, AES.MODE_EAX, nonce=self.nonce)
        decrypted_msg = cipher.decrypt(msg)
        unpadded_private_msg =
decrypted_msg.decode().rstrip(self.padding_character)
        return unpadded_private_msg


if __name__ == "__main__":
    private_msg = "This is4אזל51 a private message"
    aes = AESCipher()
    encrypted = aes.encrypt_message(private_msg)
    decrypted = aes.decrypt_message(encrypted)

    print(f"Secret Key: {private_msg} - {len(private_msg)}")
    print(f"Encrypted Msg: {encrypted} - {len(encrypted)}")
    print(f"Decrypted Msg: {decrypted} - {len(decrypted)}")
```

קובץ command.py

```
from enum import Enum

from internal_exception import InternalException


class CommandName(Enum):
    """
    An enum class that represents the command names in the protocol.
    """

    ERROR: str = "ERROR"  # error message.
    HELLO: str = "HELLO"  # hello message.
    EXIT: str = "EXIT"  # exit from the application.
    SUCCESS: str = "SUCCESS"  # if command needs conformation
    FAIL: str = "FAIL"  # if command need conformation.

    LOGIN: str = "LOGIN"  # login to a user.
    SIGNUP: str = "SIGNUP"  # sign up a new user.

    INFO_REQUEST: str = "REQIN"  # request for information.
    INFO_RESPONSE: str = "RESIN"  # response for information.

    WAITING: str = "WAITING"  # waiting for a game.
    MATCH: str = "MATCH"  # match with another player.
    ANSWERS: str = "ANSWERS"  # answers to the questions.


class Command:
    """
    A class that represents a command in the protocol. It is used to parse
the command and its parameters from the socket.
    """
```

```
    params_per_command: dict[CommandName, int] = {  # number of parameters
expected for each command
        CommandName.ERROR: 0,
        CommandName.HELLO: 1,
        CommandName.EXIT: 0,
        CommandName.SUCCESS: 0,
        CommandName.FAIL: 0,
        CommandName.LOGIN: 2,
        CommandName.SIGNUP: 3,
        CommandName.INFO_REQUEST: 0,
        CommandName.INFO_RESPONSE: 3,
        CommandName.WAITING: 0,
        CommandName.MATCH: 2
    }

    def __init__(self, *args: str | bytes | CommandName):
        """
        Constructor for Command class, calls other 'constructors'.
        :param args: the command and parameters passed in the socket.
        """

        if len(args) == 1:
            self.__init_one(args[0])
        else:
            self.__init_one(" ".join(args))

    def __init_one(self, data: str | bytes):
        """
        Constructor for Command class.
        :param data: the command and parameters passed in the socket.
        """

        if isinstance(data, bytes):  # if the data is bytes, decode it
            data: str = data.decode()
        words: list[str] = self.extract_words(data)  # extract the command
and parameters

        if words[0] not in CommandName:  # check if the command is defined in
the protocol
            raise InternalException("Command not defined in the protocol")
        self.command: CommandName = CommandName(words[0])

        if len(words) - 1 != Command.params_per_command[self.command]:  #
check if the number of parameters is valid
            raise InternalException("Invalid number of parameters for the
command")
        self.args: list[str] = words[1:]

    def __str__(self) -> str:
        return f"{self.command.value} {' '.join(self.args)}"

    def __eq__(self, other):
        return (self.command == other.command) and (self.args == other.args)

    @staticmethod
    def extract_words(data: str) -> list[str]:
```

```
        """
        Extract the command and parameters from a valid request. For example,
for 'DIR path' the returned value is ['DIR', path].
        :param data: the command and parameters passed in the socket.
        :returns: a list of the command and its parameters.
        """

        data: str = data.strip()  # trim whitespaces at the ends
        return data.split(" ")
```

קובץ internal_exception.py

```
class InternalException(Exception):
    """
    An exception class that is used for internal exceptions.
    """

    debug = False  # debug mode

    def __init__(self, message, e: Exception = None):
        """
        Constructor for InternalException class.
        :param message: message to print when exception is raised.
        """
        # Call the base class constructor
        super().__init__(message)

        self.message = message
        self.real = e

    def __str__(self):
        """
        String representation of the exception.
        :returns: the message of the exception.
        """
        return self.message


def handel_exceptions(ex: Exception) -> None:
    """
    Handle exceptions that occur in the main function.
    :param ex: the exception that occurred.
    """

    if isinstance(ex, InternalException):
        if InternalException.debug:
            print(ex.real)
        else:
            print("\nEXCEPTION: " + str(ex))
    else:
        if InternalException.debug:
            print(ex)
        else:
            print("\nEXCEPTION: An error occurred.")
```

קובץ main.py

```
import sys

import internal_exception
from Client.client import Client
from internal_exception import InternalException
from protocol import Protocol
from Server.server import Server


def start_client(ip: str) -> None:  # python main.py client 127.0.0.1
    """
    Start the client with the given ip.
    :param ip: the ip to connect to.
    """

    client = Client(ip, Protocol.PORT)
    client.main()

def start_server() -> None:  # python main.py server
    """
    Start the server.
    """

    server = Server("0.0.0.0", Protocol.PORT)
    server.main()

def main(args: list[str]):
    try:
        if args[0] == "-d":  # set debug mode
            InternalException.debug = True
            args = args[1:]
        else:
            sys.tracebacklimit = 0  # make it not show tree of exceptions

        if (args[0] == "client") and (len(args) == 2):  # client mode
            start_client(args[1])
        elif (args[0] == "server") and (len(args) == 1):  # server mode
            start_server()
        else:
            raise InternalException("Invalid arguments.")

    except Exception as ex:
        internal_exception.handel_exceptions(ex)


if __name__ == "__main__":
    main(sys.argv[1:])
```

קובץ protocol.py

```
import socket
import rsa
```

```
from command import Command
from aes_cipher import AESCipher


class Protocol:
    """
    A class that represents the protocol of the server-client communication.
    """

    LENGTH_FIELD_SIZE: int = 4  # the size of the length field in the
protocol
    PORT: int = 9960  # the port of the server
    ERROR_LIMIT: int = 10  # limit of sequential errors
    RSA_KEY_SIZE: int = 512  # the size of the RSA key in bits

    @staticmethod
    def create_msg(cmd: Command, public_key: rsa.PublicKey | None = None) ->
bytes:
        """
        Create a message according to the protocol.
        :param cmd: the command to send.
        :param public_key: the public key to encrypt the message with, or
None if shouldn't be encrypted.
        :returns: the message to send.
        """

        # create the full command with the command name and the arguments
        full_command: str = " ".join([cmd.command.value] + cmd.args)

        # create encoded command
        command_bytes: bytes = full_command.encode()

        # make the message content encrypted with AES
        if public_key is not None:
            aes = AESCipher()
            result_aes: bytes = aes.encrypt_message(full_command)  # encode
the message with AES
            command_bytes = result_aes  # send the AES encode key encoded
with RSA with the AES encoded message

        # add the length of data size & the data size to the data
        data_size: bytes = str(len(command_bytes)).encode()
        data_size_length: bytes =
str(len(data_size)).zfill(Protocol.LENGTH_FIELD_SIZE).encode()

        result: bytes = data_size_length + data_size + command_bytes

        # add the AES key to the message
        if public_key is not None:
            key_rsa: bytes = rsa.encrypt(aes.key, public_key)  # encode the
AES key with RSA
            nonce_rsa: bytes = rsa.encrypt(aes.nonce, public_key)  # encode
the AES nonce with RSA
            result = key_rsa + nonce_rsa + result

        return result
```

```
    @staticmethod
    def get_msg(sock: socket.socket, private_encrypt: rsa.key.PrivateKey |
None = None) -> tuple[bool, Command | None]:
        """
        Extract message from protocol, without the length field.
        :param sock: active socket to receive from.
        :param private_encrypt: if the data is encrypted will contain the
private RSA key.
        :returns: a bool representing if the data and protocol are valid. And
a Command instance or null, depending on the validity.
        """

        try:
            if private_encrypt is not None:
                aes_key_encoded = sock.recv(int(Protocol.RSA_KEY_SIZE / 8))
# convert bit to byte
                aes_nonce_encoded = sock.recv(int(Protocol.RSA_KEY_SIZE / 8))
# uuid is always 16 bytes
                aes_key = rsa.decrypt(aes_key_encoded, private_encrypt)
                aes_nonce = rsa.decrypt(aes_nonce_encoded, private_encrypt)

            # read the length of size header the message
            length: str = sock.recv(Protocol.LENGTH_FIELD_SIZE).decode()
            if not length.isdigit():
                return False, None

            # read message size and make sure it's an integer
            size: str = sock.recv(int(length)).decode()
            if not size.isdigit():
                return False, None

            # read message and return
            left: int = int(size)
            message: bytes | str = sock.recv(left)

            if private_encrypt is not None:
                message = AESCipher(aes_key,
aes_nonce).decrypt_message(message)

            return True, Command(message)

        except:
            return False, None


if __name__ == "__main__":
    public_key, private_key = rsa.newkeys(Protocol.RSA_KEY_SIZE)
    print(len(rsa.encrypt("dgd11g".encode(), public_key)))

    print(type(public_key.n))

    a = Protocol.create_msg(Command("LOGIN user pass"), public_key)
    print(a)
```

קובץ server.py

```
import socket
import threading
import rsa

import internal_exception
from Server.client_handler import ClientHandler
from internal_exception import InternalException
from protocol import Protocol
from Server.Database.database import Database


class Server:
    """
    The server class, handles the server connections side of the application.
    """

    def __init__(self, ip_address: str, port: int = Protocol.PORT):
        """
        Constractor for Server class, creates a socket with parameters given.
        :param ip_address: the ip address of the server.
        :param port: the port of the server. Default is as in the mutual
protocol, not a specific case used port.
        """

        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        try:
            # allow other sockets to bind to this port
            self.server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
            self.server_socket.bind((ip_address, port))
        except Exception as e:
            raise InternalException("Please check if a server is already
running or use a valid ip", e)

        self.server_socket.listen()
        print("Server is up and running!")
        print(f"Listening on {ip_address}:{port}")

        # threads list to keep track of all clients
        self.users: dict[ClientHandler, threading.Thread] = {}

        # users that wait for a match
        self.waiting_users: list[ClientHandler] = []
        # condition for thread to wait for a match
        self.waiting_users_condition = threading.Condition()

        # create a database object
        self.database: Database = Database()

        # generate public and private keys for communication
        self.public_key, self.private_key =
rsa.newkeys(Protocol.RSA_KEY_SIZE)
```

```
    def stop_server(self):
        """
        Method to stop the server.
        """

        print("Server is shutting down...")
        for user in self.users:
            self.users[user].join()
        self.server_socket.close()

    def listen_for_exit(self):
        """
        Listen for user input to stop the server.
        """

        while True:
            user_input = input()
            if user_input == "EXIT":
                self.stop_server()

    def main(self):
        """
        The main function of the server, handles the server connections and
threads.
        """

        try:
            # Start a thread that listens for user input to stop the server
            stop_thread = threading.Thread(target=self.listen_for_exit,
daemon=True)
            stop_thread.start()

            while True:
                client_socket, client_address = self.server_socket.accept()
                print(f"Connection from {client_address}")

                # start a thread for the client
                try:
                    user = ClientHandler(client_socket, client_address, self)
                    t = threading.Thread(target=user.handle_client)
                    t.start()
                    self.users[user] = t
                except InternalException as e:
                    internal_exception.handel_exceptions(e)
                    client_socket.close()

        except Exception as e:
            raise InternalException("Server has stopped working due to an
error.", e)
        finally:
            self.stop_server()
```

קובץ client_handler.py

```
from __future__ import annotations

import random
import socket
import threading

import rsa

from Server.Database.database import Database
from command import Command, CommandName
import internal_exception
from internal_exception import InternalException
from protocol import Protocol


class ClientHandler:
    """
    The client handler class, handles the server client communication side of
the server.
    """

    hebrew_alphabet = ['א', 'ב', 'ג', 'ד', 'ה', 'ו', 'ז', 'ח', 'ט', 'י', 'כ',
'ל', 'מ', 'נ', 'ס', 'ע', 'פ', 'צ', 'ק',
                       'ר', 'ש', 'ת']

    def __init__(self, client_socket: socket.socket, client_address:
tuple[str, int], server: "Server"):
        """
        Constractor for ClientHandler class, initializes the client values.
        :param client_socket: the socket of the client.
        :param client_address: the address of the client.
        :param server: the parent server object.
        """

        print(f"Handling connection with {client_address}")

        self.client_socket = client_socket
        self.client_address = client_address
        self.server = server

        # used for identification of the user
        self.username: str | None = None

        # used ONLY for the client handler of the match
        self.opponent: ClientHandler | None = None
        self.letter: str | None = None

    def login_request(self, username: str, password: str) -> Command:
        """
        Check if the user login info is valid in the database.
        :param username: the username of the client.
        :param password: the password of the client.
        :returns: the response command to the client.
        """
```

```
        if self.server.database.is_valid_user(username, password):
            self.username = username
            return Command(CommandName.SUCCESS.value)
        return Command(CommandName.FAIL.value)

    def signup_request(self, username: str, password: str, email: str) ->
Command:
        """
        Check if the user login info are valid in the database, if they are
add them to the database.
        :param username: the username of the client.
        :param password: the password of the client.
        :param email: the email of the client.
        :returns: the response command to the client.
        """

        if self.server.database.add_user(username, password, email):
            self.username = username
            return Command(CommandName.SUCCESS.value)
        return Command(CommandName.FAIL.value)

    def info_request(self) -> Command:
        """
        Get the info of the user.
        :returns: the response command to the client.
        """

        if self.username is None:
            return Command(CommandName.FAIL.value)
        else:
            user_mail = self.server.database.get_mail(self.username)
            user_score = self.server.database.get_score(self.username)
        return Command(CommandName.INFO_RESPONSE.value, self.username,
user_mail, user_score)

    def random_letter_for_match(self) -> str:
        """
        Get a random hebrew letter for the match.
        :returns: a random hebrew letter.
        """

        return random.choice(self.hebrew_alphabet)

    def wait_for_match(self) -> Command:
        """
        Wait for a user pair for a match.
        """

        if self.username is None:
            return Command(CommandName.FAIL.value)

        waited: bool = False  # if the user was in the waiting list

        with self.server.waiting_users_condition:
            if not self.server.waiting_users:
                # If the waiting_users list is empty, append yourself and go
to sleep
```

```
                self.server.waiting_users.append(self)
                waited = True
                self.server.waiting_users_condition.wait()

            if not waited:
                # If there is a thread in the list, check if it's alive
                partner = self.server.waiting_users.pop(0)
                if self.server.users[partner].is_alive():

                    # choose a random letter for the match
                    self.letter = self.random_letter_for_match()
                    partner.letter = self.letter

                    # set the opponent for both users
                    partner.opponent = self
                    self.opponent = partner

                    # notify the opponent that you are his partner
                    self.server.waiting_users_condition.notify()
                else:
                    # If the thread is dead, go recursively to the bottom of
the list
                    return self.wait_for_match()
        return Command(CommandName.MATCH.value, self.opponent.username,
self.letter)


    def handle_request(self, validity: bool, cmd: Command, prev_cmd: Command)
-> Command:
        """
        Handles the request from the client.
        :param validity: the validity of the command.
        :param cmd: the command to handle.
        :param prev_cmd: the last command sent from server.
        :returns: the response command to the client.
        """

        if not validity:
            raise InternalException("Command given isn't valid.")

        try:
            match cmd.command:
                case CommandName.ERROR:
                    return prev_cmd
                case CommandName.EXIT:
                    return Command(CommandName.SUCCESS.value)
                case CommandName.LOGIN:
                    return self.login_request(*cmd.args)
                case CommandName.SIGNUP:
                    return self.signup_request(*cmd.args)
                case CommandName.INFO_REQUEST:
                    return self.info_request()
                case CommandName.WAITING:
                    return self.wait_for_match()
                case _:
                    raise InternalException("Command not meant for server.")
        except Exception as e:  # if there is a problem in Command class,
```

```
move it upwards
            raise e

    def three_way_handshake(self) -> rsa.PublicKey:
        """
        Perform a three-way handshake with the client.
        :return: True if the handshake was successful, False otherwise.
        """

        # send HELLO with public key to client.
        response = Protocol.create_msg(Command(CommandName.HELLO.value,
str(self.server.public_key.n)))
        self.client_socket.send(response)

        # get HELLO with public key from client.
        validity, cmd = Protocol.get_msg(self.client_socket)
        if not validity:
            raise InternalException("Command given isn't valid.")
        elif cmd.command != CommandName.HELLO:
            raise InternalException("Command given isn't valid.")

        client_public_key = rsa.PublicKey(int(cmd.args[0]), 65537)

        # send SUCCESS to client.
        response = Protocol.create_msg(Command(CommandName.SUCCESS.value))
        self.client_socket.send(response)

        return client_public_key


    def handle_client(self) -> None:
        """
        Handles the sending and receiving from the client.
        """

        # start communication by exchanging public keys
        try:
            client_public_key: rsa.PublicKey | None =
self.three_way_handshake()
        except:
            self.client_socket.close()
            print(f"Closing connection with {self.client_address}")
            return

        # if the client sends an error message, we need to remember what we
sent last
        prev_response_command: Command = Command(CommandName.ERROR.value)

        # sequentially error counter
        errors: int = 0

        # handle requests until user asks to exit
        while True:
            validity, cmd = Protocol.get_msg(self.client_socket,
self.server.private_key)

            try:  # handle the request, if not valid will send an exception
```

```
                response_command: Command = self.handle_request(validity,
cmd, prev_response_command)
                print(f"Received: {cmd.command.value} with args {cmd.args}")
                print(f"Responding with: {response_command.command.value}")
            except:
                response_command: Command = Command(CommandName.ERROR.value)

            # increment the number of consecutive errors or reset it
            if prev_response_command == response_command:
                errors += 1
            else:
                errors = 0

            # if a lot of errors sequentially, then something went wrong,
terminating the connection
            if errors >= Protocol.ERROR_LIMIT:
                break

            # renew the prev command to be the last one
            prev_response_command = response_command

            # create the final message to return to the client.
            response = Protocol.create_msg(response_command,
client_public_key)
            try:
                self.client_socket.send(response)
            except Exception as e:
                internal_exception.handel_exceptions(e)

            if (cmd is not None) and (cmd.command == CommandName.EXIT):
                break

        self.client_socket.close()
        print(f"Closing connection with {self.client_address}")
```

database.py קובץ

```
import re
import sqlite3


class Database:
    """
    Database class for handling sqlite3 database operations.
    """

    def __init__(self, database_file: str = 'Server/Database/database.db'):
        """
        Constructor for the Database class.
        :param database_file: the file path of the database.
        """
        # connect to the database that will be created in this folder
        self.conn = sqlite3.connect(database_file, check_same_thread=False)

        self.cursor = self.conn.cursor()
```

```
        # set a new table for the users information
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS users (
            username TEXT PRIMARY KEY,
            password TEXT,
            email TEXT,
            wins INTEGER DEFAULT 0,
            losses INTEGER DEFAULT 0
        );""")

        self.conn.commit()

    def __str__(self):
        """
        String representation of the Database class.
        """

        # get all users from table
        self.cursor.execute("SELECT * FROM users")
        items = self.cursor.fetchall()

        return "\n".join([str(item) for item in items])

    def __del__(self):
        """
        Destructor for the Database class, closes the connection to the
database.
        """

        self.conn.close()

    @staticmethod
    def is_valid_email(email: str) -> bool:
        """
        Check if the email is a valid format.
        """

        # Regular expression for validating an Email
        regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

        # If the string matches the regex, it is a valid email
        if re.match(regex, email):
            return True
        else:
            return False

    def is_valid_user(self, username: str, password: str) -> bool:
        """
        Check if a user exists in the database.
        :param username: the username of the user.
        :param password: the password of the user.
        :return: True if the user exists, False otherwise.
        """

        try:
            # check if the user exists
            self.cursor.execute("SELECT * FROM users WHERE username = ? AND
```

```
password = ?;", (username, password))
            user = self.cursor.fetchone()
            return user is not None
        except:
            return False

    def add_user(self, username: str, password: str, email: str) -> bool:
        """
        Add a new user to the database.
        :param username: the username of the user.
        :param password: the password of the user.
        :param email: the email of the user.
        :return: True if the user was added successfully, False otherwise.
        """

        # check if the email is valid
        if not self.is_valid_email(email):
            return False

        try:
            # create a new user in the database
            self.cursor.execute("INSERT INTO users (username, password,
email) VALUES (?, ?, ?);",
                                (username, password, email))

            # commit changes to the database
            self.conn.commit()
            return True
        except:
            return False

    def get_mail(self, username: str) -> str:
        """
        Get the email of a user.
        :param username: the username of the user.
        :return: the email of the user.
        """

        try:
            # get the email of the user
            self.cursor.execute("SELECT email FROM users WHERE username =
?;", (username,))
            email = self.cursor.fetchone()
            return email[0]
        except:
            return ""

    def get_score(self, username: str) -> tuple[int, int]:
        """
        Get the score of a user, tuple of wins and losses.
        :param username: the username of the user.
        :return: the score of the user.
        """

        try:
            # get the score of the user
            self.cursor.execute("SELECT wins, losses FROM users WHERE
```

```
username = ?;", (username,))
            score = self.cursor.fetchone()

            # extract first and only user found.
            return score[0]
        except:
            return -1, -1

    def set_score(self, username: str, score: tuple[int, int]) -> bool:
        """
        Set the score of a user.
        :param username: the username of the user.
        :param score: the score of the user.
        :return: True if the score was set successfully, False otherwise.
        """

        if (score[0] < 0) or (score[1] < 0):
            return False

        try:
            # set the score of the user
            self.cursor.execute("UPDATE users SET wins = ?, losses = ? WHERE
username = ?;",
                                (score[0], score[1], username))

            # commit changes to the database
            self.conn.commit()
            return True
        except:
            return False


if __name__ == '__main__':
    db = Database('database.db')
    print(db.add_user("user1", "password1", "ron@gmail.com"))
    print(db.add_user("user8", "nigga!", "bulbul@yahoo.org"))
    print(db.add_user("user69", "nigga?!", "bulbul"))
    print(db)
    del db

    print(Database.is_valid_email("w@gmail.com"))
    print(Database.is_valid_email("lol@yahho.org"))
    print(Database.is_valid_email("@edu.co.il"))
    print(Database.is_valid_email("mikhelman.ron@israel.giv.il"))
    print(Database.is_valid_email("mikhelman.ron@edu"))
    print(Database.is_valid_email("shlomi.eat123@edu.co.il"))
    print(Database.is_valid_email("wgmail.com"))
    print(Database.is_valid_email("shlomi.eat123@edu.co2.il"))
    print(Database.is_valid_email("shlomi.eat123@edu.co2.il18"))
```

קובץ client.py

```
import socket

import rsa
```

```
from command import Command, CommandName
from internal_exception import InternalException
from protocol import Protocol
from Client.Gui.window import Window


class Client:
    """
    The client class, responsible for the client side of the application.
    """

    username: str | None = None  # client's username if logged to the server

    def __init__(self, ip_address: str, port: int = Protocol.PORT):
        """
        Constractor for Client class, creates a socket with parameters given.
        :param ip_address: the ip address of the server.
        :param port: the port of the server. Default is as in the mutual
protocol, not a specific case used port.
        """

        print("Connecting to server...")
        print(f"Connecting to {ip_address}:{port}")
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        try:
            self.server_socket.connect((ip_address, port))
        except Exception as e:
            raise InternalException("Please check if a server is already
running or use a valid ip.", e)

        # generate public and private keys for communication
        self.public_key, self.private_key =
rsa.newkeys(Protocol.RSA_KEY_SIZE)

        self.server_public_key: rsa.PublicKey | None = None

    @staticmethod
    def get_command_from_user() -> Command:
        """
        Get a command from the user, used for no GUI.
        :return: the command to send to the server.
        """

        command: Command = Command(CommandName.ERROR.value)  # initialize
with error command

        legal_command = False
        while not legal_command:
            try:
                command_str = input("Enter command: ")
                command: Command = Command(command_str)
                legal_command = True
            except:
                print("Invalid command. Please try again.")
```

```
        return command

    @staticmethod
    def handle_server_response(validity: bool, cmd: Command, last_command:
Command) -> Command:
        """
        Handle the request from the client, used for no GUI.
        Has limited commands available, Login, Signup, Exit, Error.
        :param validity: the validity of the command.
        :param cmd: the command to handle.
        :param last_command: the last command that the user has sent to the
server.
        :returns: the response command to the client.
        """

        if not validity:
            raise InternalException("Command given isn't valid.")

        try:
            match cmd.command:
                case CommandName.ERROR:
                    return Client.get_command_from_user()
                case CommandName.SUCCESS:
                    return Client.get_command_from_user()
                case CommandName.FAIL:
                    return Client.get_command_from_user()
                case _:
                    raise InternalException("Command not meant for client.")
        except Exception as e:  # if there is a problem in Command class,
move it upwards
            raise e

    def three_way_handshake(self) -> rsa.PublicKey:
        """
        Perform a three-way handshake with the client.
        :return: True if the handshake was successful, False otherwise.
        """

        # get HELLO with public key from server.
        validity, cmd = Protocol.get_msg(self.server_socket)
        if not validity:
            raise InternalException("Command given isn't valid.")
        elif cmd.command != CommandName.HELLO:
            raise InternalException("Command given isn't valid.")

        server_public_key = rsa.PublicKey(int(cmd.args[0]), 65537)

        # send HELLO with public key to server.
        response = Protocol.create_msg(Command(CommandName.HELLO.value,
str(self.public_key.n)))
        self.server_socket.send(response)

        # get SUCCESS from server.
        validity, cmd = Protocol.get_msg(self.server_socket)
        if not validity:
            raise InternalException("Command given isn't valid.")
```

```
        elif cmd.command != CommandName.SUCCESS:
            raise InternalException("Command given isn't valid.")

        return server_public_key

    def main_user_input(self) -> None:
        """
        Used as the main function of the client when no GUI is needed,
responsible for the communication with the server.
        """

        # start communication by exchanging public keys
        try:
            server_public_key: rsa.PublicKey | None =
self.three_way_handshake()
        except:
            self.server_socket.close()
            raise InternalException("Handshake failed. Closing connection.")

        # the last command that the user has sent to the server.
        sent_command: Command = self.get_command_from_user()
        request = Protocol.create_msg(sent_command, server_public_key)

        self.server_socket.send(request)

        while True:
            validity, cmd = Protocol.get_msg(self.server_socket,
self.private_key)

            try:  # handle the response, if not valid will send an exception
                print(f"Received: {cmd.command.value} with args {cmd.args}")
                response_command: Command =
self.handle_server_response(validity, cmd, sent_command)
            except:
                response_command: Command = Command(CommandName.ERROR.value)

            # update the last command sent to the server.
            sent_command = response_command

            # create the final message to return to the client.
            response = Protocol.create_msg(response_command,
server_public_key)
            self.server_socket.send(response)

    def main(self):
        """
        Main function of the client with GUI.
        """

        # start communication by exchanging public keys
        try:
            self.server_public_key = self.three_way_handshake()
        except:
            self.server_socket.close()
            raise InternalException("Handshake failed. Closing connection.")

        # create the main GUI window
```

```
        window: Window = Window(self)

    def send_and_get(self, cmd: Command) -> tuple[bool, Command]:
        """
        Send a command to the server and get a response.
        :param cmd: the command to send to the server.
        :return: the validity of the command and the command itself.
        """

        sent_command: Command = cmd
        request = Protocol.create_msg(sent_command, self.server_public_key)
        self.server_socket.send(request)
        print("Sent command to server.")

        validity, cmd = Protocol.get_msg(self.server_socket,
self.private_key)

        return validity, cmd


if __name__ == "__main__":
    client = Client("127.0.0.1")
    client.main_user_input()
```

game_page קובץ

```
from __future__ import annotations

from tkinter import ttk

from Client.Gui.page_template import PageTemplate


class GamePage(PageTemplate):
    """
    Game page of the application.
    """

    def __init__(self, window: "Window"):
        """
        Initialize the game page.
        :param window: the window of the application.
        """
        super().__init__(window)

        # Timer variables
        self.timer_seconds: int = 0
        self.timer_minutes: int = 1
        self.timer_text: str = "%02d:%02d" % (self.timer_minutes,
self.timer_seconds)
        self.timer_carry_on: bool = True
        self.timer_label: ttk.Label | None = None

    def show_self(self) -> None:
        """
```

```
        Show the frame.
        """
        super().show_self()

        self.start_timer()

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """

        self.timer_label = ttk.Label(self, text=self.timer_text)
        self.timer_label.place(x=800, y=400, width=50, height=50)

        start_button = ttk.Button(self, text="Back",
command=self.window.show_page("StartPage"))
        start_button.place(x=800, y=200, width=50, height=50)

        login_button = ttk.Button(self, text="Login",
command=self.window.show_page("LoginPage"))
        login_button.place(x=300, y=200, width=50, height=50)

        signup_button = ttk.Button(self, text="Resign")
        signup_button.place(x=400, y=200, width=50, height=50)

        exit_button = ttk.Button(self, text="Exit", command=self.exit_event)
        exit_button.place(x=500, y=200, width=50, height=50)

    def dec_timer(self) -> None:
        """
        Decrease the timer by 1 second.
        """

        self.timer_seconds -= 1

        if self.timer_seconds == -1:
            self.timer_seconds = 59
            self.timer_minutes -= 1

        if self.timer_minutes == -1:
            return

        # change timer display text
        self.timer_text = "%02d:%02d" % (self.timer_seconds,
self.timer_seconds)
        self.timer_label.configure(text=self.timer_text)

        # schedule next update 1 second later
        self.window.after(1000, self.update)

    def start_timer(self) -> None:
        """
        Start the timer.
        """

        self.window.after(1000, self.update)
```

קובץ login_page

```
from __future__ import annotations

import tkinter as tk
from tkinter import ttk

from Client.Gui.page_template import PageTemplate
from command import Command, CommandName


class LoginPage(PageTemplate):
    """
    Login page of the application.
    """

    def __init__(self, window: "Window") -> None:
        """
        Initialize the login page.
        :param window: The window of the application.
        """
        super().__init__(window)

        self.username: tk.StringVar = tk.StringVar()
        self.password: tk.StringVar = tk.StringVar()

        self.username_entry: ttk.Entry = ttk.Entry(self,
textvariable=self.username)
        self.password_entry: ttk.Entry = ttk.Entry(self,
textvariable=self.password)

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """

        start_button = ttk.Button(self, text="Back",
command=self.window.show_page("StartPage"))
        start_button.place(x=800, y=200, width=50, height=50)

        login_button = ttk.Button(self, text="Signup",
command=self.window.show_page("SignupPage"))
        login_button.place(x=300, y=200, width=50, height=50)

        signup_button = ttk.Button(self, text="Submit",
command=self.submit_login_info)
        signup_button.place(x=400, y=200, width=50, height=50)

        exit_button = ttk.Button(self, text="Exit", command=self.exit_event)
        exit_button.place(x=500, y=200, width=50, height=50)

        # place entries
        self.username_entry.place(x=400, y=300, width=200, height=50)
        self.password_entry.place(x=400, y=400, width=200, height=50)

    def lock_entries(self):
        """
```

```
        Lock the entries while used for logging.
        """

        # lock the entries to prevent user from changing them
        self.username_entry.config(state="disabled")
        self.password_entry.config(state="disabled")

        # lock the entries in the signup page

self.window.page_instances["SignupPage"].username_entry.config(state="disable
d")

self.window.page_instances["SignupPage"].password_entry.config(state="disable
d")

self.window.page_instances["SignupPage"].email_entry.config(state="disabled")

    def unlock_entries(self):
        """
        Unlock the entries while used for logging.
        """

        # unlock the entries to allow user to change them
        self.username_entry.config(state="enabled")
        self.password_entry.config(state="enabled")

        # unlock the entries in the signup page

self.window.page_instances["SignupPage"].username_entry.config(state="enabled
")

self.window.page_instances["SignupPage"].password_entry.config(state="enabled
")

self.window.page_instances["SignupPage"].email_entry.config(state="enabled")

    def submit_login_info(self):
        """
        Submit the login information to the server.
        """

        # lock the entries to prevent user from changing them
        self.lock_entries()

        # get the information from the entries
        username_to_submit: str = self.username.get()
        password_to_submit: str = self.password.get()

        # check if the user is already logged in
        if not self.window.client.username is None:
            # TODO
            return

        # check if the data is valid
        try:
            cmd: Command = Command("LOGIN", username_to_submit,
password_to_submit)
```

```
        except:
            # TODO
            self.unlock_entries()
            return

        # send the command to the server and get the response
        validity, response = self.window.client.send_and_get(cmd)
        if (not validity) or (response.command != CommandName.SUCCESS):
            # TODO
            self.unlock_entries()
            return
        else:
            self.window.client.username = username_to_submit
            self.open_start_game_button()
            return

    def open_start_game_button(self):
        """
        Open the start game button for the client after logging in.
        """


self.window.page_instances["StartPage"].start_button.config(state="enabled")
```

קובץ page_template

```
from __future__ import annotations

import tkinter as tk
from tkinter import ttk


class PageTemplate(tk.Canvas):
    """
    Template to create a new page in the application.
    """

    def __init__(self, window: "Window"):
        """
        Template to create a new page in the application.
        :param window: the window parent of the frame.
        """
        super().__init__(window, width=window.window_size[0],
height=window.window_size[1])

        self.window: "Window" = window

    def show_self(self) -> None:
        """
        Show the frame.
        """

        # Load the background image and resize the image to fit the window
        self.create_image(0, 0, image=self.window.image, anchor="nw")
```

```
        self.place_widgets()

        self.pack(fill="both", expand=True)

    def unshow_self(self):
        """
        Unshow the frame.
        """

        self.pack_forget()

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """

        pass

    def exit_event(self):
        """
        If got here, then the client wishes to close the game.
        """

        self.window.destroy()
```

קובץ signup_page

```
from __future__ import annotations

import tkinter as tk
from tkinter import ttk

from Client.Gui.page_template import PageTemplate
from command import Command, CommandName


class SignupPage(PageTemplate):
    """
    Signup page of the application.
    """

    def __init__(self, window: "Window") -> None:
        """
        Initialize the signup page.
        :param window: The window of the application.
        """
        super().__init__(window)

        self.username: tk.StringVar = tk.StringVar()
        self.password: tk.StringVar = tk.StringVar()
        self.email: tk.StringVar = tk.StringVar()

        self.username_entry: ttk.Entry = ttk.Entry(self,
textvariable=self.username)
        self.password_entry: ttk.Entry = ttk.Entry(self,
```

```
textvariable=self.password)
        self.email_entry: ttk.Entry = ttk.Entry(self,
textvariable=self.email)

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """

        start_button = ttk.Button(self, text="Back",
command=self.window.show_page("StartPage"))
        start_button.place(x=800, y=200, width=50, height=50)

        login_button = ttk.Button(self, text="Login",
command=self.window.show_page("LoginPage"))
        login_button.place(x=300, y=200, width=50, height=50)

        signup_button = ttk.Button(self, text="Submit",
command=self.submit_signup_info)
        signup_button.place(x=400, y=200, width=50, height=50)

        exit_button = ttk.Button(self, text="Exit", command=self.exit_event)
        exit_button.place(x=500, y=200, width=50, height=50)

        # place entries
        self.username_entry.place(x=400, y=300, width=200, height=50)
        self.password_entry.place(x=400, y=400, width=200, height=50)
        self.email_entry.place(x=400, y=500, width=200, height=50)

    def lock_entries(self):
        """
        Lock the entries while used for signuping.
        """

        # lock the entries to prevent user from changing them
        self.username_entry.config(state="disabled")
        self.password_entry.config(state="disabled")
        self.email_entry.config(state="disabled")

        # lock the entries in the login page

self.window.page_instances["LoginPage"].username_entry.config(state="disabled
")

self.window.page_instances["LoginPage"].password_entry.config(state="disabled
")

    def unlock_entries(self):
        """
        Unlock the entries while used for signuping.
        """

        # unlock the entries to allow user to change them
        self.username_entry.config(state="enabled")
        self.password_entry.config(state="enabled")
        self.email_entry.config(state="enabled")
```

```
        # unlock the entries in the login page

self.window.page_instances["LoginPage"].username_entry.config(state="enabled"
)

self.window.page_instances["LoginPage"].password_entry.config(state="enabled"
)

    def submit_signup_info(self):
        """
        Submit the signup information to the server.
        """

        # lock the entries to prevent user from changing them
        self.lock_entries()

        # get the information from the entries
        username_to_submit: str = self.username.get()
        password_to_submit: str = self.password.get()
        email_to_submit: str = self.email.get()

        # check if the user is already logged in
        if not self.window.client.username is None:
            # TODO
            return

        # check if the data is valid
        try:
            cmd: Command = Command("SIGNUP", username_to_submit,
password_to_submit, email_to_submit)
        except:
            # TODO
            self.unlock_entries()
            return

        # send the command to the server and get the response
        validity, response = self.window.client.send_and_get(cmd)
        if (not validity) or (response.command != CommandName.SUCCESS):
            # TODO
            self.unlock_entries()
            return
        else:
            self.window.client.username = username_to_submit
            self.open_start_game_button()
            return

    def open_start_game_button(self):
        """
        Open the start game button for the client after logging in.
        """


self.window.page_instances["StartPage"].start_button.config(state="enabled")
```

קובץ start_page

```
from __future__ import annotations

import tkinter as tk
from tkinter import ttk

from Client.Gui.page_template import PageTemplate


class StartPage(PageTemplate):
    """
    Start page of the application.
    """

    def __init__(self, window: "Window") -> None:
        """
        Initialize the login page.
        :param window: The window of the application.
        """
        super().__init__(window)

        self.start_button = ttk.Button(self, text="Start game",
state="disabled", command=self.window.show_page("WaitingPage"))

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """

        self.start_button.place(x=800, y=200, width=50, height=50)

        login_button = ttk.Button(self, text="Login",
command=self.window.show_page("LoginPage"))
        login_button.place(x=300, y=200, width=50, height=50)

        signup_button = ttk.Button(self, text="Signup",
command=self.window.show_page("SignupPage"))
        signup_button.place(x=400, y=200, width=50, height=50)

        exit_button = ttk.Button(self, text="Exit", command=self.exit_event)
        exit_button.place(x=500, y=200, width=50, height=50)
```

קובץ waiting_page

```
from __future__ import annotations

import threading
from tkinter import ttk

from Client.Gui.page_template import PageTemplate
from command import Command, CommandName
from internal_exception import InternalException
```

```
class WaitingPage(PageTemplate):
    """
    Waiting page of the application.
    """

    def __init__(self, window: "Window"):
        """
        Initialize the waiting page.
        :param window: the main window of the application.
        """
        super().__init__(window)
        self.letter: str | None = None
        self.opponent_username: str | None = None

    def show_self(self) -> None:
        """
        Show the frame.
        """
        super().show_self()

        self.ask_to_play()

    def process_response(self):
        """
        Process the response from the server.
        """

        validity, response =
self.window.client.send_and_get(Command(CommandName.WAITING.value))
        if (not validity) or (response.command != CommandName.MATCH):
            raise InternalException("Failed to send or get command from the
server.")
        else:
            self.opponent_username = response.args[0]
            self.letter = response.args[1]

            print(f"Matched with {self.opponent_username} and got letter
{self.letter}.")
            self.window.show_page("GamePage")()

    def ask_to_play(self) -> None:
        """
        Ask the server to play.
        """

        try:
            # send the command to the server and get the response
            t = threading.Thread(target=self.process_response)
            t.start()
        except Exception as e:
            print(e)

    def place_widgets(self) -> None:
        """
        Place the widgets in the frame.
        """
```

```
        exit_button = ttk.Button(self, text="Exit", command=self.exit_event)
        exit_button.place(x=500, y=200, width=50, height=50)
```

קובץ window

```
from typing import Callable
import tkinter as tk

from PIL import Image, ImageTk

from Client.Gui.game_page import GamePage
from Client.Gui.login_page import LoginPage
from Client.Gui.page_template import PageTemplate
from Client.Gui.signup_page import SignupPage
from Client.Gui.start_page import StartPage
from Client.Gui.waiting_page import WaitingPage
from internal_exception import InternalException


class Window(tk.Tk):
    """
    The main window of the application.
    """

    window_size: tuple[int, int] = (1200, 800)
    background_image_path: str = "./Client/Gui/background_image.jpg"
    pages = [StartPage, LoginPage, SignupPage, WaitingPage, GamePage]

    showing_page: str | None = None

    def __init__(self, client: "Client"):
        """
        Initialize the main window of the application.
        """
        super().__init__()

        self.client: "Client" = client

        # set title and window size
        self.title("Categories game")
        self.geometry(f"{self.window_size[0]}x{self.window_size[1]}")
        self.resizable(False, False)

        self.image = self.initialize_image()

        # pages in the application
        self.page_instances: dict[str, PageTemplate] = {}
        self.initialize_pages()

        # Start the main loop of tkinter
        self.mainloop()

    def initialize_image(self) -> ImageTk.PhotoImage:
        """
        Initialize the background image of the window.
```

```
        :return: the background image of the window.
        """
        bg_image_raw = Image.open(self.background_image_path)
        bg_image_resized = bg_image_raw.resize(self.window_size,
Image.Resampling.LANCZOS)
        bg_image: ImageTk.PhotoImage = ImageTk.PhotoImage(bg_image_resized)
        return bg_image

    def show_page(self, page_name: str) -> Callable[[], None]:
        """
        Show the page with the given name.
        :param page_name: the page name of the page to show.
        :return: the function to show the page, to give button command to
call.
        """

        def inner_show_page() -> None:
            """
            Show the page with the given name.
            """

            # page not found
            if page_name not in self.page_instances:
                raise InternalException(f"Page {page_name} not found.")

            page = self.page_instances[page_name]

            # hide the current page
            if (self.showing_page != page_name) and (not self.showing_page is
None):
                self.page_instances[self.showing_page].unshow_self()

            # show the new page
            page.show_self()
            self.showing_page = page_name

        return inner_show_page

    def initialize_pages(self) -> None:
        """
        Initialize the pages of the application.
        """

        for P in self.pages:
            page = P(self)  # create instance of the page
            self.page_instances[P.__name__] = page

        # show the start page
        self.show_page("StartPage")()


if __name__ == "__main__":
    Window()
```