צ'אט אונליין

דניאל וולקוביץ'



# תוכן עניינים

[תוכן עניינים 2](#_Toc111972333)

[מבוא 3](#_Toc111972334)

[מדריך למשתמש 3](#_Toc111972335)

[מידע בסיסי: 3](#_Toc111972336)

[אמצעי הקלט אצל הלקוח 4](#_Toc111972337)

[סביבת העבודה 5](#_Toc111972338)

[סביבת פיתוח 5](#_Toc111972339)

[סביבת הרצה 5](#_Toc111972340)

[מבנה התוכנה 6](#_Toc111972341)

[תרשים ארכיטקטורה תוכנה (UML Component diagram) 6](#_Toc111972342)

[תרשים מבנה תוכנה של ChatServer Component (UML class diagram) 6](#_Toc111972343)

[תרשים מבנה תוכנה של ChatClient Component (UML class diagram) 7](#_Toc111972344)

[בחירת פרוטוקול TCP(רמה 4) 7](#_Toc111972345)

[פרוטוקול הצ'אט 7](#_Toc111972346)

[תקשורת עם השרת 7](#_Toc111972347)

[פורמט הפרוטוקול 8](#_Toc111972348)

[תרשים הפרוטוקול: אתחול התקשרות (UML Sequence Diagram) 9](#_Toc111972349)

[תרשים הפרוטוקול: שליחת הודעות בצ'אט (UML Sequence Diagram) 9](#_Toc111972350)

[הודעות על PORT 5000 10](#_Toc111972351)

[תיאור פונקציות ומחלקות עיקריות בשרת 17](#_Toc111972352)

[מחלקות: 17](#_Toc111972353)

[Handle 17](#_Toc111972354)

[Message 17](#_Toc111972355)

[Handle message 17](#_Toc111972356)

[תיאור פונקציות עיקריות בלקוח 17](#_Toc111972357)

[Sign\_up\_message, sign\_in\_meassage, log\_out\_message 17](#_Toc111972358)

[view\_connected\_list\_message\_and\_choose\_chat 18](#_Toc111972359)

[Send\_message 18](#_Toc111972360)

[Receive\_message 18](#_Toc111972361)

[תיאור בעיות ואתגרים והתמודדות איתן 18](#_Toc111972362)

[הדפסה בצד הלקוח 18](#_Toc111972363)

[תקשורת הצ'אט בין השרת ללקוח 18](#_Toc111972364)

[הצעות להרחבה עתידית 19](#_Toc111972365)

[אפשרות להעלות קבצים לצ'אט 19](#_Toc111972366)

[קבלת אישורי קריאת הודעות בין המשתמשים 19](#_Toc111972367)

[הצפנת ההודעות שעוברות בין המשתמשים 19](#_Toc111972368)

[קבוצות 19](#_Toc111972369)

[מנהלים לקבוצות 19](#_Toc111972370)

[טעינת היסטוריית צ'אט 19](#_Toc111972371)

[יצירת חברים 19](#_Toc111972372)

[שמירת חשבונות המשתמשים לקובץ 20](#_Toc111972373)

[נספחים 20](#_Toc111972374)

[הצעת ההגשה המקורית 20](#_Toc111972375)

[דרישות הפונקציונליות המקוריות שרת ולקוח 20](#_Toc111972376)

# מבוא

התוכנית נכתבה בשפתpython כחלק מפרויקט סיום שנה בתוכנית "גבהים – מצוינות בסייבר", והיא עוסקת במימוש החומר שנלמד בהרצאות בתחום הרשתות.

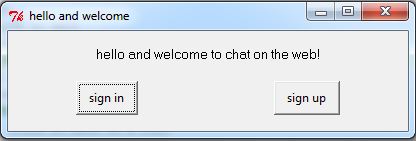
הפרויקט מהווה תוכנת צ'אט, אשר מאפשרת ללקוחות מרובים ליצור קשר אחד עם השני. זאת על ידי כתיבת צדדי שרת ולקוח ותוך יישום תחומים הכוללים תקשורת, הצפנה ו-GUI.

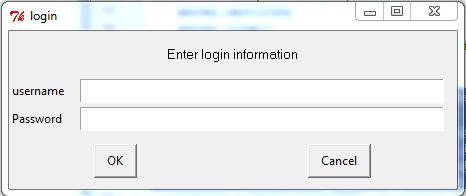
מסמך זה יתאר מדריך למשתמש, את סביבת הפיתוח, מבנה התוכנית, והבעיות איתן התמודדתי במהלך הפיתוח.

# מדריך למשתמש

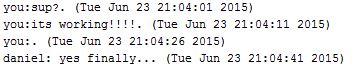
## מידע בסיסי:

1. בתוכנת הצ'אט, הלקוחות (המשתמשים) ינהלו דו שיח על גבי הרשת דרך שרת ייעודי.
2. על מנת להשתמש בצ'אט, כל לקוח חייב לבצע כניסה לחשבון צ'אט שנמצא בשרת. הכניסה תתבצע על ידי שימוש בשם משתמש וסיסמא. משתמש שאין לו חשבון חייב להירשם פעם ראשונה.
3. המשתמשים יכולים לצפות ברשימת המשתמשים המחוברים לשרת באותה עת.
4. משתמש יכול לבדוק אם ממתינות לו הודעות חדשות בשרת.
5. השרת מנהל מספר צ'אטים במקביל.
6. המשתמש יכול לנהל מספר צ'אטים במקביל.
7. השרת תומך בהתנתקות פתאומית של הלקוח.
8. קליטת המסרים מן המשתמש על ידי תוכנית הלקוח (client) מתבצעת בחלון המיועד לכך, לעומת זאת ההדפסה מתבצעת גם בקונסול interpreter וגם בעזרת החלון.
9. המשתמש יגיב להודעות הקופצות בעזרת הקלדה או בעזרת לחיצה על כפתורים שיופיעו על גבי ההודעות הקופצות.
10. הצ'אט מסתיים כאשר המשתמש סוגר את התוכנית או כאשר הוא מחליט לצאת מחשבונו.

דוגמא להודעה קופצת בעלת כפתורים:

דוגמא להודעה קופצת בעלת שדה קבלת קלט מהמקלדת:

דוגמא לשיחת צ'אט:



## אמצעי הקלט אצל הלקוח

להלן פירוט אמצעי הקלט:

1. הלחצן השמאלי של העכבר: בחלונות הקופצים העכבר ישמש ללחיצה על כפתורים.
2. Q או q: מקש יציאה מן הצ'אט.
3. המילה "CHANGE": לאחר שהמשתמש התחיל צ'אט עם משתמש אחר הוא יכול להחליף את בן שיחו לצ'אט בעזרת הקלדת המילה הנ"ל.
4. כל שאר המקשים יתפקדו כמו בכל תוכנה רגילה.



יציאה

מהצ'אט

החלפת שותף לצ'אט בעזרת הקלדת "CHANGE"

# סביבת העבודה

## סביבת פיתוח

שפת הפיתוח היא python2.7 (שפה עילית).

סביבת הפיתוח הינה pycharm community edition 3.4.1.

התוכנה מספקת סביבת עבודה משולבת (IDE), הכוללת:

* עורך מסמכים.
* דיבאגר.
* טרמינל מובנה.
* מנהל מסדי נתונים.

בעזרת התוכנה ניתן לכתוב בקלות קטעי קוד שלמים ב Python. התוכנה מוצאת שגיאות ובעיות, ועוזרת לתקן אותן בקלות. כמו כן ניתן להריץ בקלות את התוכנית מתוך התוכנה ללא שימוש ב cmd.

בצד השרת נעשה שימוש במודולים המובנים הבאים: SocketServer, socket, errno and Queue.

בצד הלקוח נעשה שימוש במודולים המובנים: socket, sys ,time thread, ובמודול המובנה: ) easygui המודול מאפשר יצירת GUI).

## סביבת הרצה

סביבת ההרצה היא חלון ריצה של התוכנה ב-1pycharm community edition 3.4. שמריצה את התוכנית על גבי השפה python2.7.

היתרון של python הוא שניתן לכתוב בעזרתה קטעי קוד ארוכים במהירות, וניתן אחר כך לקרוא ולהבין אותם בקלות. בנוסף ניתן להריץ בסביבת ההרצה קבצי py באופן מהיר.

# מבנה התוכנה

## תרשים ארכיטקטורה תוכנה (UML Component diagram)



## תרשים מבנה תוכנה של ChatServer Component (UML class diagram)

## תרשים מבנה תוכנה של ChatClient Component (UML class diagram)

## בחירת פרוטוקול TCP(רמה 4)

בתוכנית בחרתי להשתמש בפרוטוקול TCP, מכיוון שתוכניות מסוג צ'אט זקוקות לאמינות. קרי, כל ההודעות בין המשתמשים חייבות להגיע אחד לשני בסדר הנכון על מנת שצ'אט יתפקד כצ'אט יעיל ואמין. לכן שימוש בפרוטוקל UDP בפרויקט יהווה מימוש לא מתאים.

## פרוטוקול הצ'אט

### תקשורת עם השרת

* הפרוטוקול יהיה מעל **IP/TCP**.
* מספר ה-PORT של השרת יהיה 5000 (השרת ירוץ על 192.168.30.50)
* על כל PACKET של הלקוח, תינתןPACKET תשובה ע"י השרת.
* תשובת השרת, תשלח ל-PORT ול-IP שממנו התקבלה הבקשה.

### פורמט הפרוטוקול

* כל הודעה תכיל צמדים מסוג של VALUE=**KEY**
* כל צמד יופרד ע"י התו '|' (PIPE)

לדוגמא:

**MSG**=GAME\_CREATE|**GAME\_NAME**=Yair’s War|**PLAYERS\_COUNT**=4

* כל רשימה תוקף ע"י סוגריים מרובעים: [ ], ופסיקים יפרידו בין איברי הרשימה.

לדוגמא:

**MSG**=GAMES\_LIST\_REPSONSE|**GAME\_NAME\_LIST**=[Yair’s War, Hagai’s Game, Shaul’s bloodbath]

* לכל הודעה ישנו KEY בשם **MSG**, שמכיל את סוג ההודעה.
* כל תשובה מהשרת תכיל את סוג ההודעה המקורית שנשלחה ע"י הלקוח בתוספת RESPONSE\_

לדוגמא, על הודעה MOVE\_GAME, השרת יענה עם הודעת RESPONSE\_MOVE\_GAME

* התווים **, | =** **] [** (פסיק, PIPE, שווה, פתיחת וסגירת סוגריים מרובעים) אינם תווים חוקיים למעט במקומות שהוגדרו לעיל.
* הן הלקוח והן השרת, **יתעלמו** מכל צמד **KEY**=VALUE, שהם אינם מכירים.
* במקרה של שגיאה שהשרת מצליח לזהות את מקורה, השרת יוסיף מפתח בשם **TEXT**=, עם תיאור חופשי של הבעיה בה הוא נתקל.
* כאשר השרת יזהה בעיה אך ללא הצלחה בניתוח הסיבה לבעיה, השרת יוסיף מפתח בשם **STATUS**=, שיכיל את ההודעה הבאה: FAILED\_UNKNOWN\_PROBLEM. במקרה כזה הלקוח ידפיס הודעה כי יש ליצור קשר עם בעלי השרת ויעצור את התוכנית.

### תרשים הפרוטוקול: אתחול התקשרות (UML Sequence Diagram)

### תרשים הפרוטוקול: שליחת הודעות בצ'אט (UML Sequence Diagram)



### הודעות על PORT 5000

להלן הגדרת כלל הודעות האפשריות במסגרת התוכנית:

#### Get version

|  |  |
| --- | --- |
| Get server version  Client -> Server  Server always responds with GET\_VERSION\_RESPONSE MSG. | |
| MSG | GET\_VERSION |

|  |  |
| --- | --- |
| Game server version Response  Server -> Client  Response to Game List | |
| MSG | **GET\_VERSION\_RESPONSE** |
| VERSION | <Integer>  Version number |
| STATUS | SUCCESS |

#### Sign up

|  |  |
| --- | --- |
| Creates a new user account in the server.  (user will be signed in automatically after creating an account)  Client -> Server  Server always responds with SIGN\_UP\_RESPONSE MSG. | |
| MSG | SIGN\_UP |
| USER\_NAME | <string>  The username of the user being created |
| PASSWORD | <string>  The password of the user being created |

Example: **MSG**=SIGN\_UP|**USER\_NAME**=shaked|**PASSWORD**=167daniel|**USER\_ADRESS**=(“127.0.0.1”, 6789)

|  |  |
| --- | --- |
| Sign up Response  Server -> Client | |
| MSG | **GAME\_CREATE\_RESPONSE** |
| STATUS | One of [SUCCESS, FAILED\_ALREADY\_EXISTS [ |

#### Sign in

|  |  |
| --- | --- |
| Signing in whit a user account that already exists  Client -> Server  Server always responds with SIGN\_IN\_RESPONSE MSG. | |
| MSG | SIGN\_IN |
| USER\_NAME | <string>  The username of the user being signed in |
| PASSWORD | <string>  The password of the user being signed in |

Example: **MSG**=SIGN\_IN|**USER\_NAME**=shaked|**PASSWORD**=167daniel|**USER\_ADRESS**=(“127.0.0.1”, 6789)

|  |  |
| --- | --- |
| Sign in response  Server -> Client | |
| MSG | **SIGN\_IN\_RESPONSE** |
| STATUS | One of [SUCCESS, FAILED\_PASSWORD\_INCORRECT, USER\_NAME\_UNRECOGNIZED] |

#### Log out

|  |  |
| --- | --- |
| Logging out from the server  Client -> Server  Server always responds with LOG\_OUT\_RESPONSE MSG. | |
| MSG | LOG\_OUT |
| USER\_NAME | <string>  The username of the user being created |

Example: **MSG**=LOG\_OUT|**USER\_NAME**=shaked

|  |  |
| --- | --- |
| Log out Response  Server -> Client | |
| MSG | **SIGN\_IN\_RESPONSE** |
| STATUS | SUCCESS |

#### View connected list

|  |  |
| --- | --- |
| View a list of all connected users and groups  Client -> Server  Server always responds with VIEW\_CONNECTED\_LIST \_RESPONSE MSG. | |
| MSG | VIEW\_CONNECTED\_LIST |
| USER\_NAME | <string>  The username of the user being created |

Example: **MSG**=VIEW\_CONNECTED\_LIST|**USER\_NAME**=shaked

|  |  |
| --- | --- |
| View connected list response  Server -> Client | |
| MSG | **VIEW\_CONNECTED\_LIST \_RESPONSE** |
| CONNECTED\_USERS\_LIST | <list>  A list of the currently connected users |
| STATUS | One of [SUCCESS, FAILED\_UNKNOWN\_PROBLEM] |

#### Send message

|  |  |
| --- | --- |
| Sending a chat message to another person who is connected to the server.  Client -> Server  Server always responds with SEND\_MESSAGE\_RESPONSE MSG. | |
| MSG | SEND\_MESSAGE |
| SENDER\_USER\_NAME | <string>  The username of the client that wants to communicate. |
| RECEIVER\_USER\_NAME | <string>  The username of the person that the client wants to communicate with |
| CONTENT | The content of the message |
| TIME | The time the message was sent |

Example: **MSG**=**SEND\_MESSAGE**|**SENDER\_USER\_NAME**=shaked**|RECEIVER\_USER\_NAME**=ron**|TIME=**Sunday 13:00

|  |  |
| --- | --- |
| Send message response  Server -> Client | |
| MSG | SEND\_MESSAGE\_RESPONSE |
| STATUS | SUCCESS |

#### Receive message

|  |  |
| --- | --- |
| Receiving a chat message from the server.  Server -> Client  Client always responds with RECEIVE\_MESSAGE\_RESPONSE MSG. | |
| MSG | RECEIVE\_MESSAGE |
| USER\_NAME | <string>  The username of the user being signed in |

Example: **MSG**= **RECEIVE\_MESSAGE|USER\_NAME**=shaked

|  |  |
| --- | --- |
| Receive message response  Client -> Server | |
| MSG | **RECEIVE\_MESSAGE\_RESPONSE** |
| INBOX\_STATUS | <string>  One of [NO\_MESSAGES, MORE\_AWAITING\_MESSAGES, NO\_MORE\_MESSAGES] |
| SENDER\_USER\_NAME | <string>  The username of the client that wants to communicate. |
| CONTENT | <string>  The content of the message |
| TIME | <string>  The time the message was sent to the server |
| STATUS | SUCCESS |

## תיאור פונקציות ומחלקות עיקריות בשרת

### מחלקות:

#### RequestHandler

המחלקה מממשת את פעולת ה SocketServer ומנהלת את יצירת הקשר עם הלקוח. המחלקה משתמשת ב threads כך שכל לקוח מנוהל על ידי thread משלו.

פעולות במחלקה:

### Handle

* תקשורת השרת עם הלקוח במהלך הצ'אט מתנהלת בפונקציה זו.

#### User

המחלקה מכילה את כלל המידע על כל משתמש ומשתמש.

#### User\_list

המחלקה מכילה רשימה של משתמשים.

פעולות במחלקה:

#### Sign\_up, sign\_in, log\_out

הפונקציות מנהלות את ההרשמה, את הכניסה ואת היציאה של המשתמשים מחשבונותיהם.

#### Check\_inbox

הפונקציה בודקת האם מחכות למשתמש הודעות חדשות כלשהן, ואם קיימות כאלה היא מחזירה את ההודעה הישנה ביותר שהתקבלה ללקוח.

#### put\_message\_in\_receiver\_inbox

הפונקציה דוחפת את ההודעה לתיבת ההודעות הנכנסות של המשתמש הרצוי.

פעולות מחוץ למחלקה:

### Message

המחלקה מכילה את כל המרכיבים של הודעת צ'אט.

### Handle message

הפונקציה מקבלת את ההודעה שנשלחה מהמשתמש ומכווינה אותה לפונקציה שתטפל בה.

## תיאור פונקציות עיקריות בלקוח

### Sign\_up\_message, sign\_in\_meassage, log\_out\_message

הפונקציות מטפלות בכניסה ,הרשמה ויציאה של המשתמש מהשרת, כאשר הפונקציה log\_out\_message גם סוגרת את תוכנית הלקוח.

### view\_connected\_list\_message\_and\_choose\_chat

הפונקציה מבקשת מהשרת לשלוח את רשימת המשתמשים שמחוברים אל השרת באותו רגע.

### Send\_message

הפונקציה מטפלת בקבלת הודעות מן המשתמש ושליחתם אל השרת (הפונקציה עובדת במקביל ל-(receive\_message.

### Receive\_message

הפונקציה בודקת האם יש בשרת הודעות חדשות הממתינות למשתמש (הפונקציה עובדת במקביל ל-(send\_message.

# תיאור בעיות ואתגרים והתמודדות איתן

## הדפסה בצד הלקוח

* תיאור הבעיה:

הבעיה בצד הלקוח: לא היה ברור כיצד ניתן להדפיס את הודעות הצ'אט שהתקבלו מן השרת ובו זמנית לקבל קלט מן המשתמש עבור ההודעות שברצונו לשלוח. הרי במידה שנשתמש בקונסול הרגיל, אם תבוצע ללקוח הדפסת הודעת צ'אט שהגיעה מן השרת תוך כדי שהלקוח נמצא בתהליך כתיבת הודעה מקבילה, ההדפסה תקטע את הכתיבה של המשתמש והוא יאלץ להמשיך את כתיבתו בשורה חדשה. אומנם הוא לא יאלץ לכתוב הכל מחדש, אך עדיין זה מבלבל להמשיך את אותה כתיבה בשורה חדשה.

* התמודדות / פתרון:

שימוש ב easygui, מודול חיצוני. הרעיון היה לפתוח חלון מיוחד לכתיבת הודעות בדיוק כמו בתוכנות צ'אט אמתיות. כל ההודעות והמסרים מודפסות בעזרת החלונות של easygui, בעוד כל התכתבות הצ'אט מודפסת בחלון הקונסול, ה-cmd.

## תקשורת הצ'אט בין השרת ללקוח

* תיאור הבעיה:

הבעיה בצד השרת: כיצד תתנהל התקשורת בין השרת ללקוח בקבלת ושליחת הודעות הצ'אט. הרי במקביל לכך שהלקוח צריך לקבל את ההודעות שנשלחו אליו, הוא גם שולח לשרת הודעות צ'אט שנכתבו ע"י הלקוח עצמו. בנוסף, אם השרת ישלח את ההודעה שקיבל למשתמש אחר ויעביר אותה ישירות ללקוח אזי צד הלקוח ידרש להמתין להודעה (כלומר להאזין על פורט) ולכך דרוש תזמון.

* התמודדות / פתרון:

לאחר הרבה ניסיונות ומחשבה הצלחתי למצוא פתרון פשוט ויעיל יחסית.

בצד השרת: כאשר השרת מקבל הודעת צ'אט ממשתמש, הוא אינו מעביר את ההודעה ללקוח המיועד באופן ישיר, אלא מניח את ההודעה בתיבת הודעות נכנסות (inbox שממומש עלי ידי תור). בצורה כזו הלקוח יקבל את ההודעות רק כשהוא מוכן לכך, ומבקש לקבל הודעות נכנסות ולא יקבל את ההודעה הנכנסת מן השרת ב "הפתעה".

בצד הלקוח: הלקוח משתמש בשני threads: אחד כדי לשלוח הודעות צ'אט ואחד כדי לקבל אותם מן השרת. בצורה זו הלקוח מסוגל במקביל לקבל את ההודעות שמחכות לו בשרת וגם לשלוח הודעות למשתמשים אחרים דרך השרת.

# הצעות להרחבה עתידית

## אפשרות להעלות קבצים לצ'אט

* הסבר: המשתמש יוכל להעלות קבצים לצ'אט ולשלוח אותם למשתמשים אחרים.
* מטרה: להפוך את הצ'אט לרב שימושי, כך שיוכל לשמש ליותר מ "רק" התכתבויות.
* הצעת יישום תמציתית: שימוש במודול PIL להעברת קבצים.

## קבלת אישורי קריאת הודעות בין המשתמשים

* הסבר: כל משתמש יקבל התראה המאשרת שהנמען אליו נשלחה הודעה כלשהי, אכן קרא אותה.
* מטרה: לאפשר למשתמש לדעת בזמ"א שההודעה שהוא שלח אכן נקראה.
* הצעת יישום תמציתית: קביעת הודעה בפרוטוקול שהלקוח ישלח כאשר ההודעה שקיבל נראתה על ידי המשתמש (קרי הודפסה).

## הצפנת ההודעות שעוברות בין המשתמשים

* הסבר: להוסיף אלמנט של אבטחה לפרויקט.
* מטרה: להפוך את הצ'אט למאובטח.
* הצעת יישום תמציתית: הצפנת ההודעות בעזרת SSL.

## קבוצות

* הסבר: יצירת קבוצות צ'אט.
* מטרה: המשתמש יוכל לדבר עם מספר משתמשים בו זמנית וההודעות ישלחו לכל החברים בקבוצה.
* הצעת יישום תמציתית: תור שיכיל איבר מסוג קבוצה ובתוך הקבוצה יהיו רשומים איברים מסוג USER.

## מנהלים לקבוצות

* הסבר: יהיו משתמשים שיוסמכו כמנהלים של קבוצות והם יוכלו לצרף משתמשים לקבוצה ולהסיר ממנה משתמשים.
* מטרה: שמירת סדר וארגון בקבוצות צ'אט שונות שייפתחו.
* הצעת יישום תמציתית: יצירת מערך שיכיל את המנהלים וקביעת הודעות בפרוטוקול הנוגעות לסמכויות המנהל.

## טעינת היסטוריית צ'אט

* הסבר: השרת ישמור האת היסטוריית ההודעות לכל משתמש וברגע שהוא יתחבר ההודעות יטענו ההודעות למסך ההדפסה של המשתמש.
* מטרה: שמירת ההיסטוריה תאפשר למשתמש לעיין בשיחות עבר, כך יוכל להיזכר בדברים ששכח וברצונו לקרוא מחדש.
* הצעת יישום תמציתית: שימוש בתור שיהיה בתוך המחלקה USER וישמור כל הודעה שנשלחת למשתמש.

## יצירת חברים

* הסבר: המשתמש יוכל לבחור משתמשים ולסמן אותם כחבריו.
* מטרה: לאפשר למשתמש לכתוב הודעות לחבריו באופן פשוט יותר, כך חבריו יוכלו לבלוט ברשימת המשתמשים המחוברים.
* הצעת יישום תמציתית: שימוש במערך שיהיה במחלקה USER וישמור את החברים של כל משתמש ומשתמש.

.

## שמירת חשבונות המשתמשים לקובץ

* הסבר: חשבונות המשתמשים יישמרו לקובץ חיצוני ובלתי תלוי בתוכנית, בתצורה הנוכחית במידה שהשרת נכבה, המשתמשים יימחקו – זו חולשה משמעותית.
* מטרה: להעניק לפרויקט פן של שרידות
* הצעת יישום תמציתית: כתיבת פרטי המשתמשים לקובץ txt, ואבטחתו ע"י שמירת סיסמאות בעזרת HASH.

# נספחים

## הצעת ההגשה המקורית



## דרישות הפונקציונליות המקוריות שרת ולקוח

