פרויקט גמר בסייבר

שם הפרויקט

**שם התלמיד: קובי סוננברג**

**ת"ז: 318134186**

**מנחים: מיכאל צ'רנובילסקי**

**בית הספר: תיכון חדש - הרצליה**

**מרץ 2016**

**תוכן עניינים**

תוכן

[תקציר כולל ורציונל הפרויקט 3](#_Toc447155080)

[מבוא ורקע כללי 4](#_Toc447155081)

[תיאור המוצר המוגמר 5](#_Toc447155082)

[שפת התכנות וסביבת העבודה 6](#_Toc447155083)

[תיאור אלגוריתמים קיימים 8](#_Toc447155084)

[הפתרון הנבחר 10](#_Toc447155085)

[פיתוח הפתרון בשכלול הקוד עם שפת התכנות 11](#_Toc447155086)

[תיאור המודולים המרכיבים את המוצר 17](#_Toc447155087)

[פירוט מבני הנתונים 20](#_Toc447155088)

[פירוט מאגרי המידע 21](#_Toc447155089)

[מדריך התקנה למשתמש 22](#_Toc447155090)

[תיאור הממשק למשתמש 23](#_Toc447155091)

[מבט אישי על העבודה ותהליך הפיתוח 25](#_Toc447155092)

[ביבליוגרפיה 26](#_Toc447155093)

[**קוד התוכנה** 27](#_Toc447155094)

[**צד השרת:** 27](#_Toc447155095)

# תקציר כולל ורציונל הפרויקט

בעת שימוש במחשבים ברשת מחשבים (במקום העבודה בעיקר) עולה לפעמים הבעיה שאנשים לא רצויים יוכלו לפתוח את הקבצים (האישיים לפעמים) שהמשתמש כתב, במקום עבודה זה יכול להיות דבר רע מאוד משום שאדם ללא הרשאה או סיווג מסוים יוכל להיחשף לתוכן אשר לא מתאים שיראה.

התוכנה שלי מתמודדת עם הבעיה המתוארת, וההשראה באה מניסיוני האישי, לעיתים, כאשר הייתי עובד על מחשב בבית הספר (לאו דווקא במקצוע המחשבים או הסייבר) הייתי שומר את הקובץ בבית הספר וכאשר הייתי חוזר לעבוד על הקובץ זמן מה לאחר מכן גיליתי שמישהו נכנס לי לקובץ ושינה לי דברים. התוכנה שלי מונעת את זה ולא נותנת אפשרות לאנשים שלא יכולים לראות את הקובץ לראות, או לשנות את הקובץ. בנוסף לכך, התוכנה שלי מאפשרת העאלה של כל הקובץ לענן, כך שאני יוכל לגשת אליו גם מהמחשב בבית, בהנחה שהתוכנה פועלת גם אצלי.

# מבוא ורקע כללי

הפרויקט שלי מבוסס על יצירת סוג קובץ חדש אשר מכיל כותרות (headers) חדשים שאני החלטתי עליהם וצירופם לתוכן המוצפן של הקובץ המקורי כך שלא יוכלו לקרוא את מה שכתבתי.

הנושא המרכזי של הפרויקט שלי הוא עבודה עם קבצים, כתיבה לתוכם, קריאה מהם וכן הצפנה ופיענוח.

לפרויקט יש מספר מרכיבים עיקריים ולהלן סקירת רקע לגביהם:

* מרכיב עיקרי בפרויקט שלי הוא **תקשורת**. תקשורת בעולם המחשבים מבוססת על Sockets, שהם נקודות קצה עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים. רוב הסוקטים כיום מבוססים על ה- Internet Protocol (IP).

בפרויקט שלי התקשורת בין השרת ללקוחות מתבצעת באמצעות Sockets כאשר בשכבת התעבורה יש שימוש בפרוטוקול ה- TCP, שייחודו בכך שהוא מוודא שכל הנתונים שנשלחו התקבלו ובסדר הנכון, ואחרת שולח אותם שוב. בפרויקט שלי יש חשיבות לכך שלא יהיה איבוד מידע ופרוטוקול ה- TCP/IPמתאים בדיוק למטרה זו. השימוש בSocket קורה הרבה בפרויקט:

* תקשורת בין הממשק הגרפי למנועים האחוריים.
* תקשורת בין המנועים לבין קבצי התקשורת שבין כל לקוח לשרת.
* תקשורת בין לקוח לשרת.
* הפרויקט משלב עבודה עם **מסד נתונים**. הנתונים שהתוכנה אוספת לגבי כל אחד ממחשבי הרשת נשמרים במסד הנתונים, שהעבודה איתו היא בטכנולוגיית ה- SQLite, שמממשת את רוב הסטנדרט של SQL (שהיא השפה המקובלת לטיפול ועיבוד מידע בבסיסי נתונים). להבדיל מרוב המערכות לניהול בסיסי נתונים, SQLite אינה תהליך עצמאי נפרד, אלא ספריה בתוך התהליך באופן אינטגרלי. ‏ זוהי בחירה פופולרית כאשר מפתחים בסיס נתונים מוטמע עבור צד לקוח כשירות ניהול מידע מקומי. מסד הנתונים שאותו מנהלת התוכנה שלי שומר מידע לשימוש מקומי בלבד, ולכן בחרתי להשתמש ב- SQLite שלא דורש הרצה של SQL Server. בנוסף, המנועים של הפרויקט נכתבו בפיית'ון והעבודה בין פיית'ון לSQLite מאוד פשוטה ונוחה.
* מרכיב חשוב נוסף בפרויקט שלי הוא הצפנה, משום שהעבודה מכילה תקשורת מוצפנת בין הלקוח לשרת, שכן מועבר מידע על המשתמשים (המתחברים), פרטים אלו הכוללים סיסמאות ושמות משתמש רגישים שכן אם יגלו אותם, יוכלו להיכנס ולפתוח קבצים מהחשבון שלהם.

# תיאור המוצר המוגמר

שם המוצר: שם המוצר.

המוצר מוודא שהאנשים בעלי המסכות הנכונה יוכלו לפתוח קבצים מסויימים, וגם כאן מפריד בין אנשים שיוכלים לערוך את הקובץ לבין אנשים שיכולים רק לקרוא אותו.

המטרה המרכזית של המוצר היא מניעת חשיפת מידע רגיש לגורמים לא מורשים.

השאלות שהמערכת עונה עליהן:

* **כיצד להגן על המידע הרגיש מפני גורמים לא מורשים?** התוכנה מצפינה את כל התקשורת בין מחשבי הרשת לשרת, כך שגורם לא מורשה לא יוכל להפיק שום תועלת מהאזנה למידע הנשלח, הוא לא יוכל לגלות אילו תהליכים רצים על מחשבי החברה וכמה משאבים הם צורכים.
* **כיצד לעבוד בו-זמנית עם כל המחשבים ברשת?** כאשר מחשבי הרשת מתחברים לשרת, הוא פותח Thread נפרד לכל אחד ומסוגל לתקשר במקביל עם כולם, לקבל מהם מידע ולשלוח להם פקודות, ולכל מחשב ברשת יש בצד השרת מסד נתונים נפרד שבו מאוחסן כל המידע שנאסף עליו.
* **כיצד למנוע את חשיפת הקבצים?** כאשר משתמש בוחר לנעול קובץ נעשה שימוש מיוחד בפרטי הקובץ ובפרטים שנבחרים ע"י (למי הקובץ ננעל למשל), שבא לידי ביטוי בהכנסתו לשיטת הצפנת קבצים ייחודית הדומה במקצת לChecksum ברשתות תקשורת (כמו בפרוטוקול TCP)

# שפת התכנות וסביבת העבודה

הפרויקט נכתב ב- Python וב- C#. כל הליבה של הפרויקט נכתבה ב- Python בשל הפשטות והנוחות של השפה, וה- GUI של הפרויקט נכתב ב- C# כי היה נוח מאוד לכתוב GUI ב- C# בעזרת ה- Visual Studio ואז לחבר אותו לליבה שנכתבה ב- Python בעזרת socket.

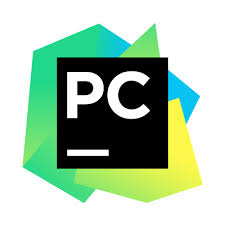
סביבת העבודה עם Python הייתה PyCharm (גרסה 5.0.4) שהיא תוכנה מאוד מקצועית ונוחה לעבודה על פרויקטים מרובי מודולים וקבצים. היא שימשה גם לפיתוח הקוד ועבודה במקביל עם כל המודולים השונים, וגם להרצתו בשביל בדיקות, השתמשתי בPython גרסה 2.7 .

סביבת העבודה עם C# הייתה Microsoft Visual Studio 2010 ששימשה לכתיבת ה- GUI באמצעות גרירת פקדים מה- toolbox, וה- debugger המובנה שימש לבדיקות. הבחירה בסביבת עבודה זו הייתה משום שה- Visual Designer המובנה היה פתרון מאוד נוח לכתיבת ה- GUI של הפרויקט, ובאופן כללי ה- Visual Studio הוא IDE טוב מאוד לעבודה ב- C#.

סביבת העבודה עם מסד הנתונים הייתה SQLite, שהיא ספרייה מוטמעת (כחלק בלתי נפרד) בתוך התהליך של התוכנה, שמאפשרת את ניהול מסדי הנתונים המקומיים בצד השרת.

כדי לראות בצורה ויזואלית את מסד הנתונים בחרתי להשתמש במצג של DB browser for sqlite.

בנוסף, את ההתקדמות של הפרויקט שמרתי באמצעות GitHub והשתמשתי בתוכנה GitHub for Desktop.



ניסוח וניתוח הבעיה האלגוריתמית

במהלך פיתוח המוצר נתקלתי במספר בעיות אלגוריתמיות:

* **מימוש שרת מולטי-קליינט:**

הבעיה הראשון שעמדה לפניי בפיתוח המוצר היא יצירת שרת מולטי-קליינט מולטי-ת'רד. כלומר, הייתי צריך לממש שרת שמתחברים אליו מספר לקוחות בו-זמנית והוא מסוגל לדבר עם כולם. בתוך בעיה אלגוריתמית זו ישנן מספר תתי-בעיות:

* **פיתוח פרוטוקול משלי:** השרת צריך לדבר עם כל הלקוחות בצורה פשוטה, מהירה, יעילה ובטוחה ולשם כך הייתי צריך לפתח את פרוטוקול התקשורת המתאים ביותר.
* **מימוש אבטחת מידע:** כדי למנוע מהמידע לעבור בין השרת ללקוחות באופן חשוף הייתי צריך לממש אבטחת מידע, כלומר להוסיף לפרוטוקול שלי הצפנה. הבעיה האלגוריתמית היא שליחה של מידע מוצפן בפרוטוקול ופענוח שלו בצד השני של הקו, וכן החלפת המפתחות.
* **חיבור עם ה- GUI :** השרת צריך להתחבר ל- GUI לשם הצגת מידע והתראות למשתמש וכן לתת להם לבצע פעולות רבות, (למנהל יש יותר אפשרויות משאר המשתמשים).
* **שינוי הקובץ לקובץ מבוטח:**

משום שאני משנה את סוג הקובץ, הייתי צריך לראות בדיוק איך בונים קבצים, וכן לראות מה הדרך הכי נכונה לכתוב לתוכם מידע ולהצפין את התוכן.

* **פיתוח ושמירת נתוני הקבצים ומזהי המשתמשים:**

כדי שהמשתמשים יוכלו לנעול ולפתוח מספר רב של קבצים, הייתי צריך למצוא דרך אשר שומרת את כל נתוני הקבצים במקום מסוים, וכן לפתח בצורה מסויימת כך שהכל יהיה ייחודי.

# תיאור אלגוריתמים קיימים

* **מימוש שרת מולטי-קליינט:**

המימוש המקובל של תקשורת בין מחשבים הוא באמצעות Sockets, ואולם ישנם מספר סוגים:

* Raw Socket – אין שכבת תעבורה (Transport), זהו Socket בסיסי שפועל בשכבת ה- Internet ויש גישה ישירה ל- packet headers.
* Datagram Socket – מכונה גם connectionless, משתמש בפרוטוקול UDP בשכבת התעבורה ומשמש להעברת מידע במהירות רבה תוך ויתור על הבטחת data integrity.
* Stream Socket – מכונה גם connection-oriented, משתמש בפרוטוקול TCP בשכבת התעבורה ומשמש להעברת מידע תוך וידוא שכל הפקטות מגיעות ליעדן ומגיעות בסדר הנכון.

כמו כן, למימוש השרת כמולטי-קליינט שתומך בתקשורת עם מספר לקוחות בו-זמנית יש שתי שיטות המוכרות לי:

* באמצעות ספריית select – ב- Python יש ספרייה בשם select שבתוכה יש פונקציה עם אותו שם, והיא יכולה לקבל רשימה של Sockets שמהם נרצה לקבל מידע ורשימה של Sockets שנרצה לשלוח דרכם מידע, והיא מחזירה רשימה של Sockets שמוכנים לקריאת מידע מתוכם ורשימה של Sockets שמוכנים לשליחת מידע דרכם לצד השני.
* באמצעות multi-threading – יהיה Socket מאזין שיאזין ללקוחות חדשים שמתחברים ב- Thread הראשי (בלולאה אינסופית), ובכל פעם שמתחבר לקוח חדש ייפתח עבורו Thread חדש ובו תתנהל התקשורת עם לקוח זה. בצורה כזו השרת ידבר עם כל לקוח ב- Thread נפרד וכך יוכל לדבר עם כולם בו-זמנית. ספריית threading של Python מאפשרת לעשות זאת.

לשם אבטחת מידע קיימים אלגוריתמי הצפנה רבים, למשל בהצפנה אסימטרית יש את El-Gamal ואת RSA, ובהצפנה סימטרית יש DES ו- AES (וכמובן עוד רבים אחרים).

* **מימוש שמירת הקובץ בצורה מאובטחת:**

בעזרת הספריה המובנית בפיית'ון ctypes אשר נותנת אפשרות להשתמש באובייטקטים של שפת התכנות C, מצאתי את הדרך הטובה ביותר לכתוב מידע חדש לפי רצוני לקבצים הננעלים וכן לקבצים הנפתחים.

* **פיתוח ושמירת נתוני הקבצים ומזהי המשתמשים:**

הדרשה למזהה יחודיי באורך שנקבע על ידי (8 ספרות) כמזהה למשתמש וכן כמזהה לקובץ, ובאורך של 16 ספרות כמפתח הקובץ נותנת את האפשרות הטובה ביותר אשר מאזנת בין מידע ארוך מידי לבין מספר אפשרי רב של איברים שכן באמצעות הספרייה random של פיית'ון הרכבתי מחרוזות של ספרות ששימשו כמזהים וכמפתחות, משום שבחרתי לקבוע אוצם בצורה רנדומלי זה נותן את האפשרות שיהיו הרבה מאוד מזהים שונים, ייחודיים. את המידע הזה אני שומר במסד נתונים שהגישה אליו נעשית באמצעות השפה SQL, אך גם כאן ישנן אפשרויות רבות למסדי הנתונים ובחרתי לעבוד עם SQLite בגלל הפשטות והנוחות של העבודה עם פיית'ון.

# הפתרון הנבחר

* **מימוש שרת מולטי-קליינט:**

את השרת מולטי-קליינט מימשתי באמצעות Stream Socket (בפרוטוקול TCP) שמאזין ב- Thread הראשי, ובכל פעם שמתחבר לקוח, נפתח ללקוח הזה Thread נפרד משלו ונפתח Socket שדרכו מתבצעת כל התקשורת עם לקוח זה. בצורה כזאת השרת מסוגל לדבר בו-זמנית עם מספר לקוחות בת'רדים נפרדים וגם להמשיך להאזין לקבלת חיבורים חדשים.

* **פיתוח פרוטוקול משלי ומימוש אבטחת מידע:** פיתחתי פרוטוקול משלי בשביל התקשורת בין השרת ללקוח. הפרוטוקול משתמש בשיטת כתיבת הודעות ייחודית כאשר בראש ההודעה נכתב המטרה של ההודעה ולאחר מכן #, אחריו מופיעים כל מיני נתונים. לדוגמה:
  + - כאשר לקוח מנסה להתחבר נשלחת הודעה מהלקוח לשרת אשר נראית כך:

Login#username#password\_hash

כאשר ההודעה מגיעה לשרת השרת מפריד בין חלקי ההודעה ובאמצעות מילת המפתח (מילת הפקודה, בדוגמה זו-Login) הוא יודע מה לעשות עם הפרטים שאחרי (שליחתם לפונקציית בדיקה במסד הנתונים).

התקשורת בפרוטוקול משתמשת בהצפנה, כאשר תהליך החלפת המפתחות בתחילת התקשורת בין השרת ללקוח מתנהל כך:

הלקוח הוא זה אשר פונה לשרת, וכאן הם שולחים אחד לשני הודעה אשר מצביעה על תחיל התקשורת ("hello"), כאשר שלב זה מסתיים החלפת המפתחות מתחילה.

השרת שולח תחילה את המפתח הציבורי שלו, ואז מקבל את המפתח הפרטי של הלקוח, ומחכה לקבל את הhash של המפתח הפרטי שלו כדי לאמת שבאמת התקבל כמו שצריך. לאחר מכן השרת מקבל מהלקוח הודעה מוצפנת בעזרת המפתח הציבורי של השרת hash של המפתח הסימטרי (בשיטת AES) וכן את המפתח הסימטרי עצמו. הhash מתקבל כחתימה דיגיטלית כדי לוודא שבאמת השרת הלקוח הוא זה ששלח את המפתח הסימטרי.

* **חיבור עם ה- GUI :** החיבור מתבצע באמצעות Socket מקומי.
* **מימוש שמירת הקובץ בצורה מאובטחת:**

כדי להשיג את השמירה המאובטחת ביותר שלא ניתנת להצפנה התוכנה משתמשת בכל הפרטים של שמירת הקובץ (המזהים של המשתמשים שמורשים לפתוח, אם הקובץ פתוח לקריאה בלבד או גם לעריכה- בכל הheaderים המשתנים).את כל הנתונים האלה שמרתי כמחרוזת עליה עשיתי hash, את המחזרוזת של הhash צמצמתי לאורך 16, והשתמשתי במחרוזת זו כחלק מייצור המפתח הסימטרי הייחודי של הקובץ הננעל. בצורה זו גם אם גורם עוין לא מורשה ינסה להוסיף את עצמו לקובץ כדי שיוכל לראות את מה שנכתב, הקובץ לא יפתח לו כי הוא לא הגיע למפתח הנכון, כלומר, הhash משמש כchecksum, אם הheaders ישתנו אז גם הhash ישתנה ולעולם המערכת לא תגיע למפתח הנכון ובכך הקובץ לא ייפתח. בנוסף, בתהליך הייצור של המפתח הסימטרי מכניסים גם מחזרוזת רנדומלית שנשמרת במסד הנתונים בצד השרת וזה מגדיל את הביטחון על הקובץ כי צריך בדיוק את שני המחרוזות האלו כדי להגיע למפתח המדויק המשמש לפתיחת הקובץ. שיטה זו נקראת auxiliary , (השיטה הוצעה ע"י גיא, עוזר ההוראה של הכיתה), השיטה בעצם אומרת שיש אלגוריתם מסוים(במקרה שלי האלגורית'ם ליצירת מפתח AES), אל האלגוריתם הזה מכניסים שני עצמים שונים המשפיעים בצורה ישירה על יצירת המפתח הסופי בצורה כזו שכל פעם שיוכנסו בדיוק המחרוזות המתאימות ובדיוק לתוך האלגוריתם הספציפי, נקבל מפתח נכון.

**האלגוריתם ליצירת מפתח סימטרי**

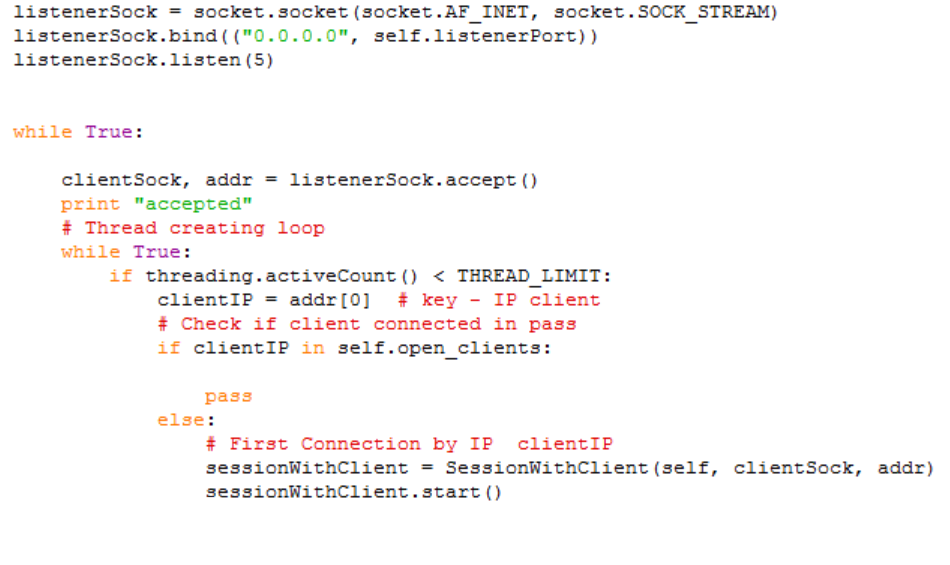
**AES**

Header's hash

Original key from DB

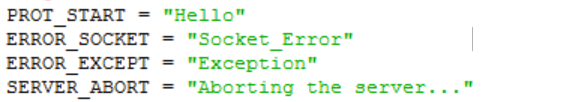
* **פיתוח ושמירת נתוני הקבצים ומזהי המשתמשים:** בעזרת פונקציות מובנות שבונות מחרוזות רנדומליות בפיית'ון ובC# הרכבתי מחרוזות זיהוי באורכים מתאימים, בנוסף החלטתי שכמפתח לכל משתמש או לכל קובץ אשתמש במזהה הרנדומלי שהוגרל לו, ולא בשם הקובץ למשל. את המידע הזה הכנסתי למסד הנתונים שם הם גם משמשים כערכי מפתח (.(primary keysפיתוח הפתרון בשכלול הקוד עם שפת התכנות
* **מימוש שרת מולטי-קליינט:**

הבסיס לשרת המולטי-קליינט הוא:

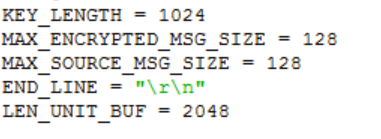
****

ה- listenersocket מאזין ובכל פעם שמתחבר לקוח חדש הוא יוצר בשבילו socket ואני פותח בשבילו Thread במחלקה sessionWithClient

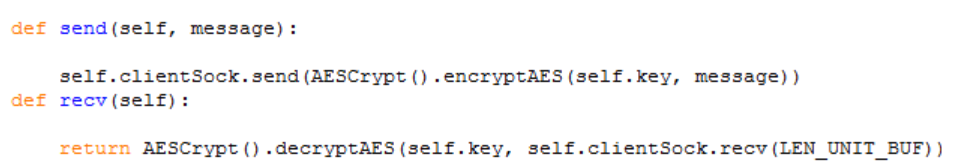
* **פיתוח פרוטוקול משלי ומימוש אבטחת מידע:**

קבעתי הערות שיישלחו מהשרת כקבועים.

מתוך :ServerCrypto.py

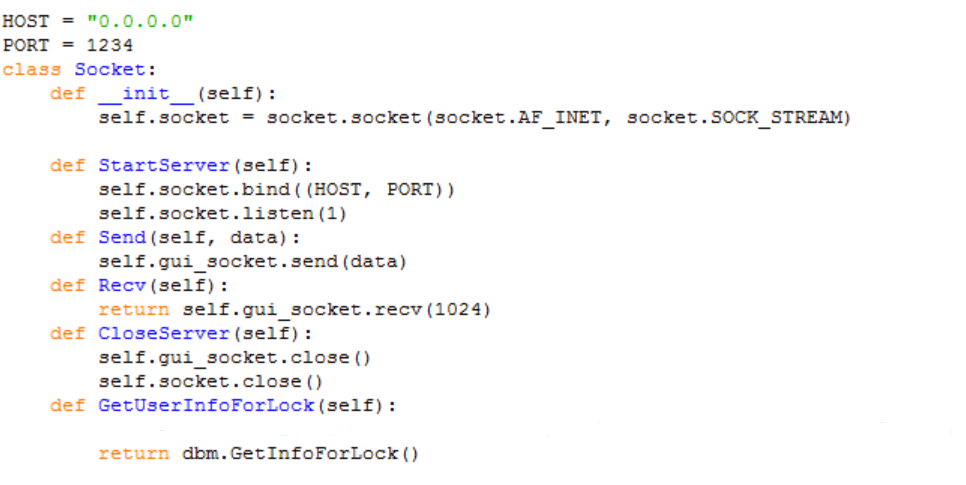


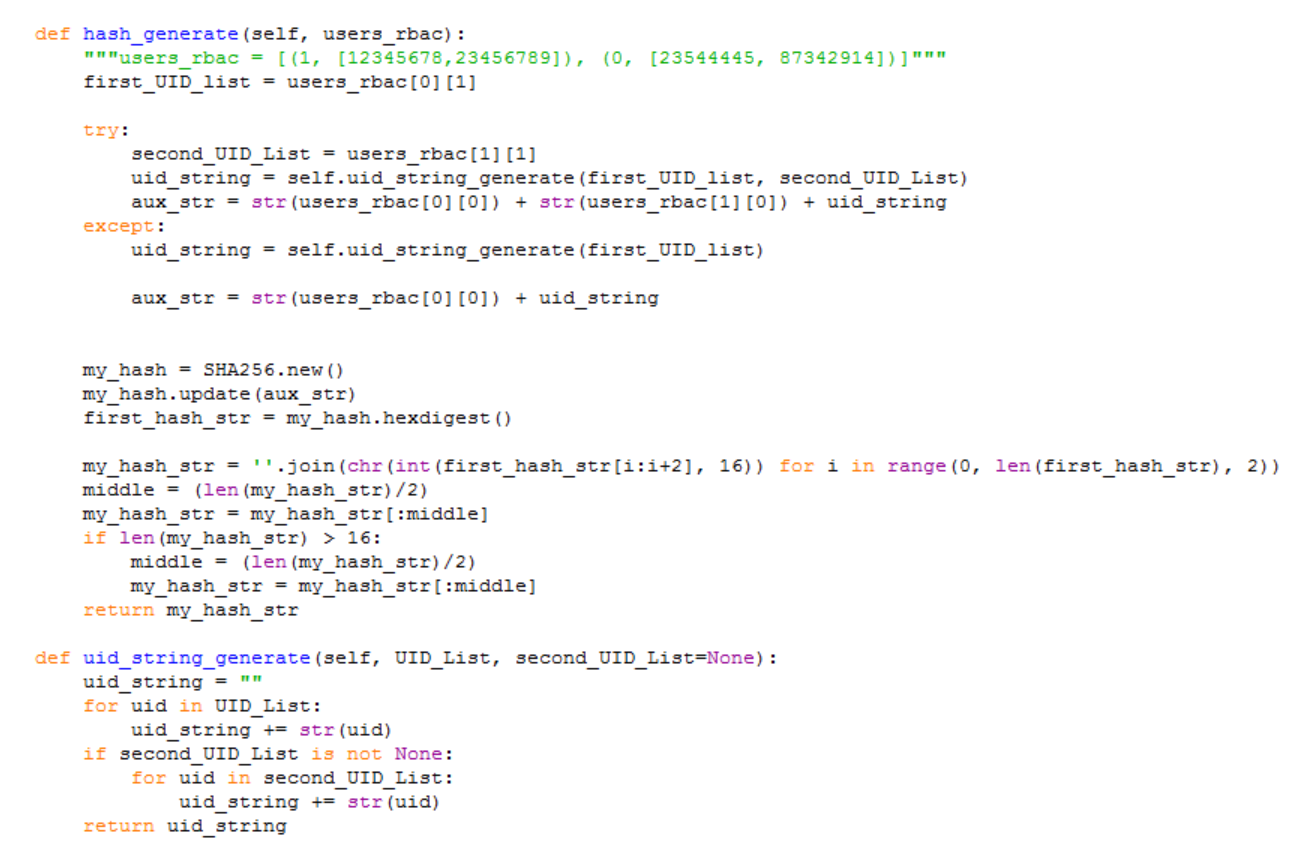
כתבתי פונקציות לשליחה וקבלה של מידע בהתאם לפרוטוקול שלי. יש פונקציות לשליחה וקבלה של מידע כולל ההצפנה:

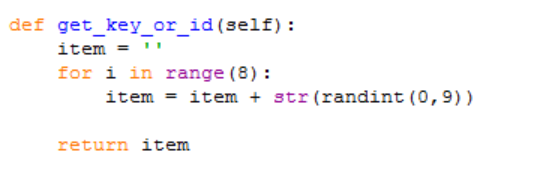


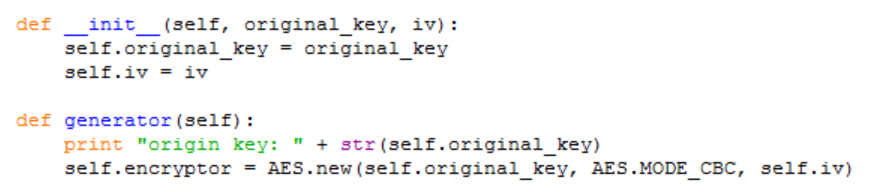
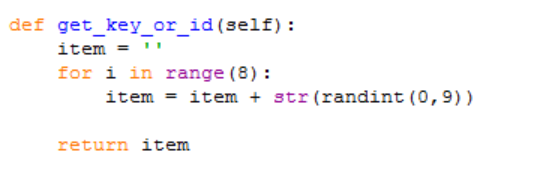
* **חיבור עם ה- GUI :**

החיבור ל- GUI מתבצע באמצעות Socket רגיל.



* **שמירת הקובץ בצורה מאובטחת:**
* יצירת הhash:
* יצירת מזהה ייחודי רנדומלי:



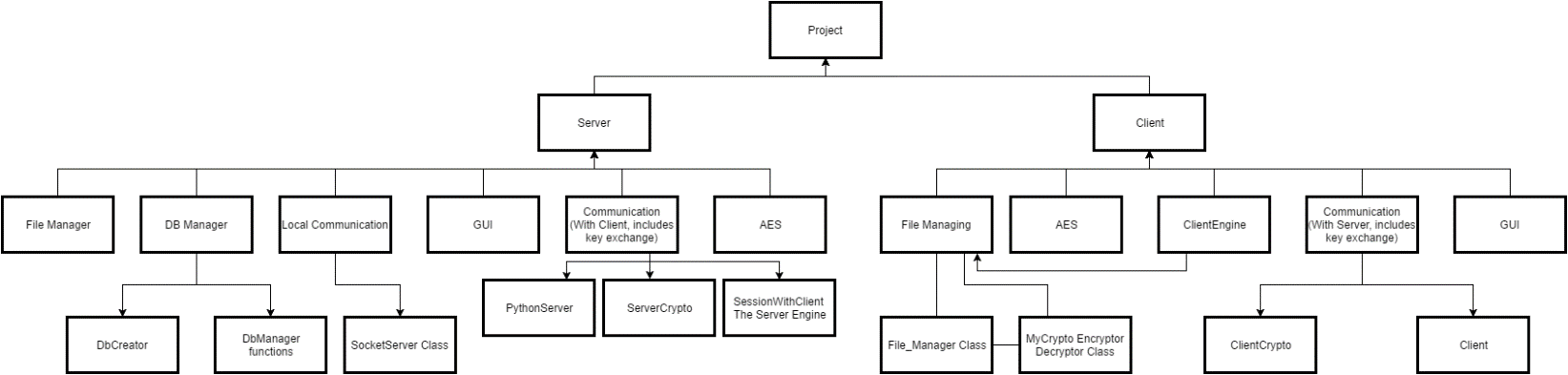
* קבלת המחרוזות ויצירת המפתח:
* **פיתוח ושמירת נתוני הקבצים ומזהי המשתמשים:**
* יצירת מזהה ייחודי רנדומלי: (אותה פונקציה משמשת ליצירת המפתח)
* הגדרת ערכי מפתח במסד הנתונים:

השורה הראשונה מגדירה את מספר הזהות שהוגרל לכל משתמש כעמודת המפתח של טבלת הנתונים של המשתמשים, בעוד שמספרי הזהות של הקצבים מוגדרים שערכי המפתח של טבלת הקבצים.



# 

# תיאור המודולים המרכיבים את המוצר

עץ המודולים המרכיבים את המוצר:

המודולים המרכזיים הם:

* **PythonServer.py, Client.py** – המודול של השרת והלקוח. השרת מאזין ב- socket בפורט מסוים, והלקוחות מתחברים אליו באמצעות ה- socket. לכל לקוח השרת פותח Thread נפרד ששומר את נתוני אותו לקוח. התקשורת ביניהם היא בפרוטוקול מותאם אישית, כאשר בהתחלה מתבצע תהליך של החלפת מפתחות, ואז מתחילה העברת המידע, למטרת התחברות או נעילה או פתיחה של קובץ, כל התקשורת מוצפנת ע"י מודול ההצפנה.
* **GUI**- המודול של הממשק הגרפי, לשרת וללקוח יש ממשקים שונים אך אחריותם זהה, הצגת מידע למשתמש, ייעול תהליך הנעילה, התחברות נוחה, הרשמה קלה. המודול מתקשר עם המנועים האחוריים בעזרת socket. בצד השרת – SocketServer.py, ובצד הלקוח ClientEngine.py .
* **בצד לקוח:**
* **ClientCrypto.py**- האחראי על החלפת המפתחות עם השרת.
* **File\_Manager.py**- המודול האחראי על ההתעסקות עם הקצבים, מודול זה כותב לקובץ, קורא אותו, שולח את התוכן להצפנה או לפיענוח, ואוסף את הנתונים לייצור המפתח.
* **DriveManager**.py- מודול האחראי על ניהול הענן, כולל פונקציות המעלות ומורידות את הקבצים.
* **בצד שרת:**
* **ServerCrypto.py**- האחראי על החלפת המפתחות עם הלקוח.
* **SessionWithClient.py**- האחראי על זרימת התקשורת עם הלקוח, לאחר ההתחברות שנעשה בPythonServer.py כל לקוח מקבל thread וממשיך לתקשר במודול זה.
* **DbManager.py**- המודול בצד השרת אחראי על העבודה עם מסד הנתונים. מודול זה כולל מספר פונקציות קלות להפעלה ונוחות לשימוש. מודול זה עושה שימוש במודול sqlite3 המובנה בפיית'ון.

עצי תתי-המודולים של מודול ה- GUI:

**צד השרת:**

תתי-המודולים הם:

* FirstPage.cs- הטופס הראשון שנועד להירשם או להתחברות, החלטתי שהכי בטוח והכי נכון שרק במחשב של השרת תהיה אפשרות להירשם בתוכנה ולכן יש בעמוד הפתיחה הירשמות וכניסה למשתמשים.
* SaveFile.cs- הטופס השני שמגיעים אליו לאחר ההתחברות, טופס זה מכיל שני דפים, האחד להגדרות משתמש (שינוי פרטים, מחיקת משתמש), והשני משמש לנעילה ולפתיחה של קבצים.
* SocketClient.cs- מחלקה האחראית על התקשורת בין הממשק הגרפי לבין המנוע האחורי (באמצעות socket מחובר לSocketServer.py).

**צד הלקוח:**

תתי-המודולים הם:

* FirstPage.cs- הטופס הראשון שנועד להתחברות, החלטתי שהכי בטוח והכי נכון שרק במחשב של השרת תהיה אפשרות להירשם בתוכנה ולכן יש בעמוד הפתיחה כניסה למשתמשים בלבד.
* SaveFile.cs- הטופס השני שמגיעים אליו לאחר ההתחברות, טופס זה דף אחד אשר משמש לנעילה ולפתיחה של קבצים.
* SocketClient.cs- מחלקה האחראית על התקשורת בין הממשק הגרפי לבין המנוע האחורי (באמצעות socket מחובר לClientEngine.py).

# פירוט מבני הנתונים

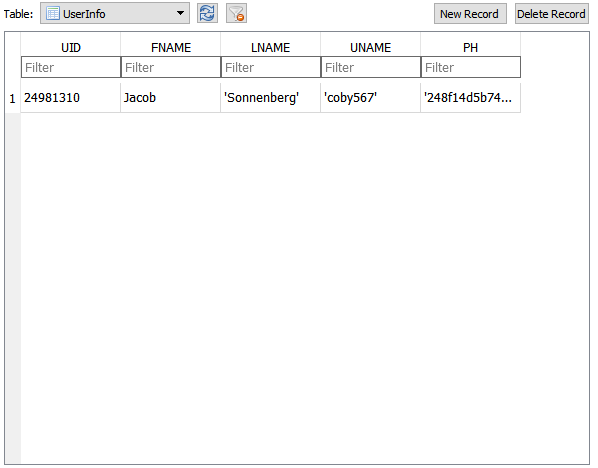
|  |  |  |  |
| --- | --- | --- | --- |
| **מודול** | **שם המבנה** | **סוג** | **תפקיד** |
| File\_Manager.py | FILE\_TYPE\_CODE\_DICTIONARY | מילון – string : int | נותן לכל סיומת של קובץ מספר סידורי מסויים, שאח"כ נשמר בheaderים של הקובץ הנעול, משום שבמהלך הנעילה אני משנה גם את סיומת הקובץ. |
| UID\_List | רשימת מחרוזות | רשימה שכוללת את כל המספרים המזהים הצומדים להרשאה מסויימת (rbac). |
| file\_header | Structure של ctypes. מכיל c\_ubyte, c\_uint | קובי את הheader. שיטה לכתיבה בתוך הקובץ, באמצעות המבנה הזה אני מרכז את הנתונים ומכתיב אוצם לתוך הקובץ בבת אחת, או שאני באמצעות זה קורא את הנתונים מהקובץ. |
| rbac | string | תו אחד המצביע על הרשאה מסויימת. המספר מייצג את ההרשאה כאשר 0 זה יכול לערוך ו-1 זה לקריאה בלבד. |
| users\_rbac | רשימה של טופלים המכילים:  (int, list[strings]) | רשימה שמכילה שני טופלים בשביל האפשרות של שתי קבוצות בעלות הרשאות שונות לקובץ אחד. |
| ClientCrypto.py | Self.private\_key | זוג מפתחות RSA | מפתח פרטי ומפתח ציבורי המשמשים להחלפת המפתחות. |
| ServerCrypto.py | Client\_sym\_key\_original | String | מחרוזת שתהפוך למפתח AES הסימטרי |
| Self.private\_key | זוג מפתחות RSA | מפתח פרטי ומפתח ציבורי המשמשים להחלפת המפתחות. |

# פירוט מאגרי המידע

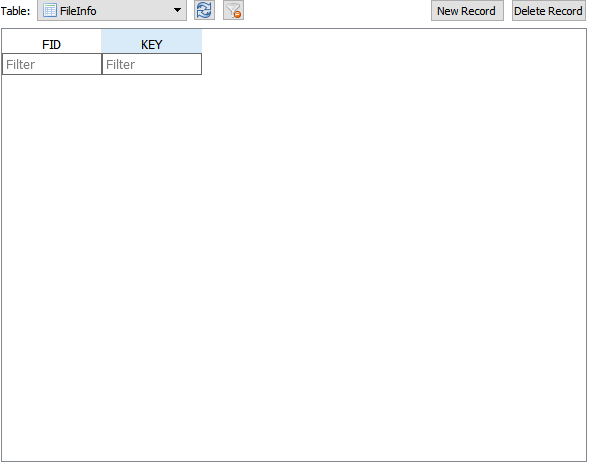
* **מסד הנתונים:**

התוכנה יוצרת מסד נותנים בעל שתי טבלאות, טבלה אחת לנתוני המשתמשים ואחת לנתוני הקבצים.

* טבלה ראשונה - טבלת המשתמשים UserInfo אשר שומרת מספר מזהה, שם פרטי, שם משפחה, שם משתמש וhash של הסיסמה:



* טבלה שנייה – טבלת מידע הקבצים FileInfo אשר שומרת מספר מזהה ומפתח ייחודי:



# 

# מדריך התקנה למשתמש

אילוצים ודרישות מערכת

* התוכנה פותחה עבור מערכת ההפעלה Windows. היא נבדקה ונוסתה בגרסת **Windows 7** בלבד, ועל כן השימוש בה בגרסה אחרת של Windows הוא באחריות המשתמש בלבד.
* יש לוודא שמותקן בכל מחשב קצה ובשרת **Python 2.7 (32/64-bit)**, ושמותקנת הספריה **PyCrypto** עבור גרסה זו של Python. בביליוגרפיה נמצאים לינקים להורדת Python ולהתקנת הספריות.
* כמו כן, צריכה להיות מותקנת במחשב השרת **.NET Framework 4.0** או גרסה עדכנית יותר (התוכנה נבדקה בגרסאות **4.0** ו- **4.5**).
* לבסוף, יש לוודא שקיים חיבור בין מחשבי הקצה למחשב עליו יותקן השרת (ובפרט יש לסדר את הגדרות ה- Firewalls שקיימים ברשת הפנימית כך שיאפשרו לתוכנה ליצור חיבורים).

התקנת והרצת התוכנה

* צד שרת:

1. יש להוריד את תיקיית **Server** למחשב השרת.
2. כדי להריץ את השרת, כשרת ללקוחות ממחשבים אחרים, יש להפעיל את הקובץ PythonServer.py, ואם רוצים להריץ את השרת (כדי לנעול/לפתוח קבצים, לרשום משתמשים או למחוק ולערוך נתונים), יש להריץ את הקובץ Server.sln שבתיקייה ServerSideGui וכן את הקובץ SocketServer.py
3. יש לוודא שנפתחת תיקייה בשם TheDrive , אם היא לא נפחתח יש ליצור תיקייה חדשה בתיקיית הServer ולקרוא לה TheDrive.

* צד לקוח:

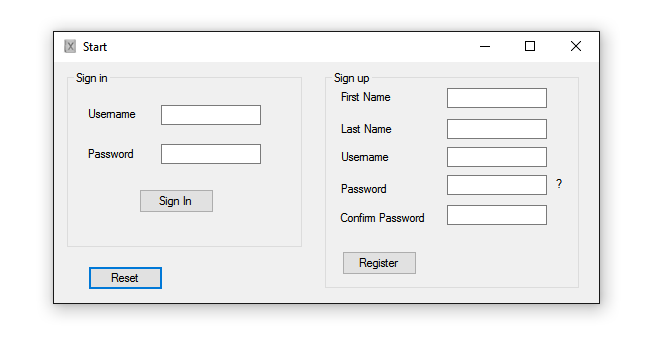
1. יש להוריד את תיקיית **Client** לכל אחד ממחשבי הקצה שרוצים להריץ עליהם את התוכנה.
2. כדי להריץ את התוכנה על מחשבי הקצה, יש ללחוץ לחיצה כפולה על הקובץ **Client.py** וכן את הקובץ Server.sln שבתוך תיקיית ,ClientSideGuiבמקרה שתקפוץ הודעת אזהרה מ- Windows Firewall יש לאשר אותה.

# תיאור הממשק למשתמש

הממשק הגרפי אצל **מנהל הרשת (צד השרת)** לאחר הרצת התוכנה מתחלק ל-2 חלקים:

1. **FirstPage**

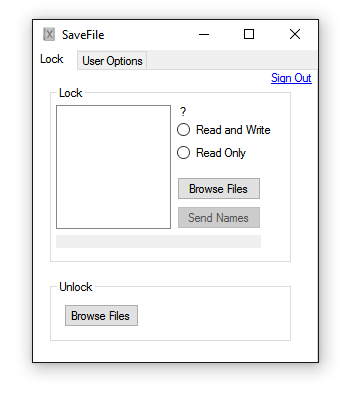
חלון זה הוא חלון ההתחברות וההרשמה, קיבלתי החלטה לאפשר הרשמה רק מהמחשב של המנהל מסיבות של ביטחון, שכן המערכת חייבת להיות סגורה (הצטרפות עצמאית לא אפשרית), חלון זה מחכה להכנסת הפרטים ואז ממשיך הלאה לפי הבקשה.



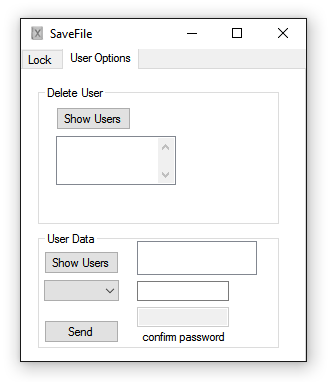
1. **SaveFile**

מגיעים לחלון זה אם המשתמש הצליח להתחבר, כמנהל הרשת, חלון זה מתחלק לשניים:

* + Lock- החלק הזה בחלון זהה גם אצל הלקוח, חלק זה הוא האחראי על נעילה והצפנה של הקבצים הרצויים. בעצם לאחר עריכת קובץ כלשהו ניתן להריץ את המערכת ובעזרתה לבחור את הקובץ שהמשתמש רוצה לנעול (או לפתוח).



* + Settings- החלק השני, קיים רק בצד השרת, יש אפשרות למחוק משתמשים וכן לשנות נתונים שלהם (את הסיסמה, את השם הפרטי וכן את שם המשתמש).
  + Drive- החלק השלישי, קיים בצד הלקוח וכולל חלק שמראה מה הקבצים הניתנים להורדה וכן נותן את האפשרות להעלות קובץ לשרת.



הממשק הגרפי אצל **הלקוח** לאחר הרצת התוכנה מתחלק ל-2 חלקים:

1. **FirstPage**

חלון זה הוא חלון ההתחברות, קיבלתי החלטה לאפשר הרשמה רק מהמחשב של המנהל מסיבות של ביטחון, שכן המערכת חייבת להיות סגורה (הצטרפות עצמאית לא אפשרית), חלון זה מחכה להכנסת הפרטים ואז ממשיך הלאה לפי הבקשה.

1. **SaveFile**

מגיעים לחלון זה אם המשתמש הצליח להתחבר. גם בצד השרת יש חלון כזה, חלק זה הוא האחראי על נעילה והצפנה של הקבצים הרצויים. בעצם לאחר עריכת קובץ כלשהו ניתן להריץ את המערכת ובעזרתה לבחור את הקובץ שהמשתמש רוצה לנעול (או לפתוח).

# מבט אישי על העבודה ותהליך הפיתוח

העבודה הזו הייתה הפעם הראשונה שבאמת התנסיתי עם שימוש בעיצוב גרפי ונהניתי לראות כמה שזה יכול להיות קל ומהנה. בנוסף זו הייתה הפעם הראשונה שעבדתי עם sqlite3 (עבדתי עם access בעבר) ושמחתי לגלות כמה שזה יותר פשוט. בפרויקט זה, בפעם הראשונה נדרשתי לעשות תכנון לוגי מראש, ולמרות הקושי שבפעם הראשונה הבנתי עד כמה זה עזר לי.

האתגרים והקשיים שנתקלתי בהם במהלך פיתוח הפרויקט הם:

* כתיבה לקבצים- היה קשה למצוא את הדרך לכתוב לתוך הקבצים משום שרציתי לכתוב ממש בתים פנימה, לאחר התחבטויות עם כל מיני שיטות החלטתי על לכתוב את כל הכותרת של הקובץ (הheader) באמצעות המודולctypes של פיית'ון.
* לפי דעתי הקושי הכי גדול שלי היה חיבור כל הפרויקט למוצר אחד גדול, זה בעצם הפרויקט הראשון בו אני כותב כל כך הרבה קבצים שאמורים לעבוד יחד ובאמצע קצת איבדתי את עצמי עם כל המודולים והמחלוקות, אך עזרה קטנה מהמורה שלי מיכאל הצליחה לסדר לי את מה שהתקשיתי בו והצלחתי לחבר הכל יחד.

הערכת הפתרון לעומת התכנון:

* בתכנון הראשוני התכוונתי לנעול את האפשרות לעשות אפילו העתק-הדבק על קובץ שפתחתי והוא ניתן לי לקריאה בלבד, אך בשל מורכבות הדבר (התעסקות עם הclipboard) לא הספקתי לעשות את זה.
* המוצר המוגמר יחסית תואם את התכנון הראשוני, ואם היה לי יותר זמן הייתי מוסיף דברים נוספים.
* ההמלצות לשיפורו הן אם כן:
* לנעול את האפשרות לעשות העתק-הדבק, כלומר, שימוש בclipboard בזמן שקבצים מסוימים נפתחים.
* הייתי רוצה להוסיף מעין ענן בו משתמשים יוכלו להוריד את הקבצים שהם מורשים לגשת אליהם גם אם הם לא קיבלו את הקובץ פיזית למחשב שלהם.
* בנוסף, אין ספק שיש מקום לשיפור המוצר מבחינת אבטחת מידע ומימוש פרוטוקול יותר מקצועי, יעיל ומאובטח.

# ביבליוגרפיה

1. הפורום הגדול ביותר לשאלות ותשובות בנושאי תכנות:  
   [http://stackoverflow.com](http://stackoverflow.com/)
2. MSDN, הרשת הרשמית של Microsoft למפתחים:  
   <http://msdn.microsoft.com/library>
3. התיעוד הרשמי של Python:  
   [https://docs.python.org/2.7](https://docs.python.org/2.7/)
4. הוויקי הרשמית של Python:  
   [https://wiki.python.org/moin](https://wiki.python.org/moin/)
5. להורדת Python 2.7 (32-bit):

<https://www.python.org/ftp/python/2.7/python-2.7.msi>

1. להתקנת ספריית PyCrypto בשביל Python 2.7 (32-bit):

<http://www.voidspace.org.uk/downloads/pycrypto26/pycrypto-2.6.win32-py2.7.exe>

**קוד התוכנה**

**צד השרת:**

