<VIT>

מסמך אפיון

<ינון ריבקין >

<גרסה 2>

<19.5.2022>

היסטוריית גרסאות המסמך

תאריך	גרסה	תקציר השינויים
8.4.2022	1	התחלה של מסמך אפיון, חלק מהפרטים עדיין חסרים.
19.5.2022	2	סיום מסמך אפיון. הפרויקט גמור.

1. הקדמה

חלק זה הינו הקשדמה של מסמך אפיון לפרויקט הגמר שלי בסייבר.

1.1 מטרה

מטרת מסמך האפיון הינו לספק הסבר מפורט לגבי המוצר, פרטים חשובים לגביו כולל מטרות, מיהו קהל היעד, מה הן מטרות המוצר ועוד.

1.2 המוצר

- *שם המוצר: VIT*
- *המוצר נותן אפשרות לטפל במספר רחב של מחשבים דרך חלוקה שלהם לקבוצות ופניה אליהם באמצעות פקודות פשוטות.*
- *מטרת המוצר הוא לתת גישה לאיש IT לכלל המחשבים ולטפל בהם ביעילות.ההיתרונות של מוצר זה הם שהוא מייעל את העבודה של איש IT בכך שהוא חוסך זמן. החיסרון של פרויקט זה הוא בכך שבמקרה שצריך לטפל בהרבה מחשבים המחשב של איש ה IT צריך להיות חזק ומהיר בכדי שתוכנה תעבוד מהר.*

1.3 קישור למסמכים קודמים

הגדרות:

מסד נתונים - הוא אמצעי המשמש לאחסון מסודר של נתונים במחשב, לשם אחזורם ועיבודם.

תוספים (אוספי תוספים, אוסף) - תוספים או אוספי תוספים או אוסף הם כינוי לשמות קבוצות השיוך שבהם נמצאים המחשבים, כל מחשב יכול להיות שייך לכמה תוספים.

כתובת ip - היא תווית מספרית (לדוגמה 192.168.70.10) המשמשת לזיהוי נקודות קצה ברשתות תקשורת שבהן משתמשים בפרוטוקול התקשורת IP, כגון רשת האינטרנט. נקודת קצה יכולה להיות מחשב, מדפסת, כונן ועוד

פורט -הוא תהליך שדרכו יכולות תוכנות להעביר נתונים. הפורט הינו מספר זיהוי של תכנית המחשב.

שרת- לקוח- השרת הוא תוכנה פסיבית, המאזינה לרשת ומחכה לקבל בקשות. הלקוח לעומתו בדרך כלל מהווה את ממשק המשתמש – הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו.

סוקטים - הוא נקודת קצה (endpoint) עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים.

הליך - הליך מאפשר לספק למשתמש במערכת ההפעלה מהירות תגובה ורציפות פעולה כאשר התהליך (יישום) מבצע כמה משימות במקביל.

איש IT - איש האחראי על התאמת מערכות מחשוב לעסקים, שרתים, תוכנות אנטי וירוס, תוכנות גיבוי, מערכות שונות הנדרשות לקיום הארגון וכלים טכנולוגיים שונים.

מסד נתונים של mongodb - מסד זה הוא בסיס הנתונים המוביל בעולם בקטגוריית NoSQL, ובין חמשת המובילים בכל הקטגוריות. בסיס הנתונים נשען על מבנה של מסמך בניגוד לבסיסי נתונים טבלאיים העובדים מעל טבלאות מקושרות. מבנה המסמכים עובד מעל מימוש של JSON הנקרא על ידי MongoDB BSON.

1.4 תקציר.

הסבר מה מופיע בהמשך המסמך:
הסבר על המוצר, מטרותיו, כיצד יפעל, כיצד יש להשתמש בו, מיהו קהל היעד, מימושים בשוק הקיים אשר משמשים לאותה המטרה.

2. תיאור כללי

החלק הזה מתאר את הגורמים העיקריים המשפיעים על המוצר ועל דרישותיו. חלק הזה לא מפרט דרישות ספציפיות אלא רק עוזר להבין את הדרישות.

2.1 פונקציונליות

התוכנה נותנת אפשרות לבצע פקודות אצל מחשבים מסוימים בקלות, ניתן לפנות למחשבים ספציפיים ולקבוצות של מחשבים וגם ניתן לפנות לכמה קבוצות ולכמה כתובות ספציפיים.
התוכנה מכילה ארבעה רכיבים עיקריים:

תוכנת המשתמש - תוכנת המשתמש לוקחת קלטים מהמשתמש, מעבירה אותם לשרת ומציגה את התשובה של השרת למשתמש

שרת - השרת מנהל את התקשורת עם הלקוחות ועם תוכנת המשתמש. השרת מקבל פקודה מתוכנת המשתמש ומעביר אותה ללקוחות הרלוונטים. השרת גם מוסיף לקוחות חדשים למסד הנתונים. השרת ותוכנת המשתמש נמצאים על אותו מחשב אך מורצים בנפרד.

לקוח - הלקוח הוא התוכנה הנמצאת על כל המחשבים שאיש ה IT צריך לתחזק. תוכנת הלקוח מקבלת פקודות מהשרת ומבצעת. לאחר ביצוע הפעולה היא מחזירה תשובה לשרת.

מסד הנתונים - מסד הנתונים אינו תוכנה שפועלת בעצמה אלה מחלקה שכל מי שצריך לקבל מידע או להוסיף ולמחוק מידע משתמש בה. מחלקה זאת מתחברת למסד הנתונים ומאפשרת בעזרת פונקציות לערוך את מסד הנתונים.

2.2 קהל היעד

קהל היעד של פרויקט זה הם חברות וארגונים שיש להם הרבה מחשבים. האנשים אשר יפעילו את התוכנה הם אנשי IT של אותה חברה .אנשים אלה עוברים הכשרות ולרוב בעלי ידע טכנולוגי רחב. ולכן התוכנה אינה חייבת להיות פשוטה לשימוש, אל אף שלדעתי התוכנה היינה מובנת וקלה להבנה וללמידה.

2.3 אילוצים עיקריים

המערכת שעליה תרוץ השרת צריכה להיות מערכת המאפשרת שימוש בהליכים רבים ובחיבור לרשת מקומית, כמו כן רצוי שהיא תהיה מהירה כדי שהיא תחסוך זמן לאיש ה IT כאשר הוא מחכה לתשובת הלקוחות. השרת צריך גם תמיד לרוץ. באותה מערכת צריך לרוץ גם מסד הנתונים שתמיד צריך להיות מופעל ותוכנת המשתמש שניתן לסגור אותה ולהתחבר מחדש לשרת.

המערכת שעליה תרוץ תוכנת הלקוח צריכה להיות מחוברת לרשת הפנימית שהשרת נמצא בה. תוכנת הלקוח צריכה לרוץ תמיד. המערכת צריכה לתמוך בהרצה של הליכים.

2.4 הנחות ותלויות

רצוי שהרשת הפנימית בה התוכנה נמצאת תתמוך הכתובות סטטים על אף שהתוכנה יכולה לפעול ולעבוד גם בלי זה. הפיכת הרשת המקומית לסטטית תיתן אפשרות לעקוב איפה כל מחשב נמצא (פיזית) וכך במיקרה שהארגון או החברה רוצה לשנות או להחליף מחשבים בין חדרים ניתן לדעת איזה לאיזה אוסף צריך להעביר את המחשבים. נוסף לכך התוכנה עובדת על סביבה של ווינדוס.

3. דרישות מפורטות

החלק העיקרי והחשוב ביותר של מסמך האפיון. פרק זה יכיל את הדרישות המפורטות מהמערכת שינחו את מעצבי המערכת, המפתחים, והבודקים בהמשך. כל דרישה שתפורט בחלק זה צריכה להיות:

3.1 דרישות פונקציונליות

תיאור של פיצ'רים ספציפיים במערכת.

חשוב לומר שכל מידע שעובר בין התוכנות דרך סוקטים צריך להיות מותאם לפרוטוקול ייחודי, הפרוטוקול בנוי מהדר שמראה את גודל המידע ומהמידע עצמו. קבלת המידע נעשית על ידי פונקציה שמקבלת את המידע לפי ההדר ויש לשמור שהעברה של מידע תעשה לפי הפרוטוקול.

קיימים ארבעה רכיבים עיקריים ולכל אחד פיצרים משלו:

תוכנת המשתמש:

קבלת מידע מהמשתמש והצגה של פלטים למשתמש:

ממשק המשתמש מנוהל על ידי הליך והיא אחראית עם קבלת מידע מהמשתמש והצגה של מידע למשתמש.הממשק הוא ממשק מסוג command line . ממשק זה צריך להיות נוח לשימוש ולהציג את המידע בצורה מובנת.

תקשורת עם השרת:

תוכנת המשתמש מתחברת לשרת דרך סוקטים. התקשורת עם השרת הוא דרך localhost שזה אומר שהשרת ותוכנת המשתמש נמצאים על אותו מחשב והמידע עובר דרך מערכת ההפעלה. התקשורת עם השרת מתבצע על יד הליך ייעודי שדואג שתהיה תקשורת רציפה עם השרת. הליך זה אחראי גם על קבלת מידע והחזרה שלו לממשק המשתמש.

טיפול במידע שבא מהשרת:

טיפול במידע שבא מהשרת נעשה עם ידי פונקציה ייעודית אשר מופעלת על ידי ההליך האחראי על ממשק המשתמש. המידע שבא מהשרת הוא בנוי מרשימה המכילה טאפלים שבהם מאוחסן כתובת הלקוח וההודעה שלו. הבעיה היא שכאשר אנו מקבלים את המידע מהסוקט הוא מתקבל כמחרוזת ולכן יש פיצ'ר האחראי על הפיכת המחרוזת לרשימה המכילה טאפלים (המבנה שהשרת שלח במקור) ועל הפיכת הרשימה למחרוזת שבה מסודר המידע האופן מובן, הפיצר מחזיר את המחרוזת.

<u>תוכנת השרת:</u>

<u>תקשורת עם המשתמש:</u>
תוכנת המשתמש מתחברת לשרת דרך סוקטים. התקשורת עם השרת הוא דרך localhost שזה אומר שהשרת ותוכנת המשתמש נמצאים על אותו מחשב והמידע עובר דרך מערכת ההפעלה. חלק זה מנוהל על ידי הליך וכך השרת ותוכנת המשתמש מקיימים תקשורת רציפה. השרת מקבל מידע מהמשתמש ושולח אותו לפיצר ייעודי שאחראי על המשך הטיפול בבקשת המשתמש ולאחר קבלת התשובה מהפיצר היא מחזירה את התשובה למשתמש.

<u>טיפול בבקשה שבאה מהמשתמש, מציאת הלקוחות הרצויים וטיפול בתשובה שלהם:</u>

פיצר זה מקבל את הפקודה של המשתמש ומפרק אותה למרכיבייה העיקריים. הוא מוצא את הלקוחות שאליהם המשתמש רוצה לשלוח פקודות ומעביר להם את הפקודה. את התשובת הלקוחות ואת שמם הוא שומר בתוך רשימה אותה הוא מחזיר.

<u>תקשורת עם הלקוח:</u>

כל לקוח חדש שמתחבר לשרת התקשורת איתו מתנהלת דרך הליך ייעודי חדש והליך זה נוסף לרשימה של הליכים פעילים. פיצ'ר זה מקבל את המידע שהמשתמש שלח לשרת ושולח אותו ללקוח. הפיצר מחזיר את תשובת הלקוח.

<u>תוכנת הלקוח:</u>

<u>תקשורת עם השרת:</u>

הלקוח מקבל מידע מהשרת ומעביר אותו לפיצר האחראי על טיפול הבקשה, לאחר מכן הוא מחזיר את תשובת הפיצר.

<u>טיפול בבקשה של השרת:</u>

פיצ'ר זה אחראי על טיפול בבקשת השרת. השרת שולח פקודות ללקוח ופיצ'ר זה מבצע אותם. לאחר ביצוע הפקודה הוא שולח תשובה לשרת לפי הפקודה אותה הוא שלח (לכל פקודה תשובה שונה)

מסד הנתונים

התקשרות עם מסד הנתונים נעשה על ידי אובייקט ייעודי וכל רכיב שצריך לתקשר עם המסד משתמש באובייקט זה. ההתנהלות של אובייקט זה היא פשוט וכולל שלושה פיצירים עיקריים:

קבלת מידע ממסד הנתונים:

פיצ'ר זה מקבל מהמסד את המידע אותו אנו רוצים. המידע אותו בדרך כלל אנו רוצים זה שמות כל האוספים הקיימים במסד וכל כתובות ה ip הנמצאות אצל אוסף מסויים. קבלת המידע נעשה על ידי שאילתות ספציפיות אותם האובייקט שולח למסד. פיצ'ר זה מחזיר את תשובת מסד הנתונים.

הוספה למסד הנתונים:

פיצ'ר זה אחראי על הוספה של נתונים חדשים למסד. קיימים שני סוגי נתונים שאותם אנו יכולים להוסיף: אוסף חדש וכתובת ip. בפיצר זה אנו משתמשים גם כדי להוסיף כתובת ip מאוסף אחד לאוסף אחר. פיצ'ר זה מחזיר הודעה שאומרת כי הכתובת הוספה או שמסיבות מסוימות היא לא הוספה.

מחיקה ממסד הנתונים:

פיצ'ר זה אחראי על מחיקת מידע ממסד הנתונים. המידע אותו אנו יכולים למחוק הוא או אוסף מסויים או כתובת ip מתוך אוסף. קיימת אפשרות גם למחוק כתובת ip מכל האוספים. פיצ'ר זה מחזיר אם הוא מחק או לא.

3.2 דרישות של ממשקים חיצוניים

3.2.1 ממשקי משתמש (למשל: GUI)

קיימים שתי ממשקי משתמש. האחד הוא ממשק מסוג command line שדרכו איש ה IT רושם את הפקודות ורואה את תשובת המחשבים על ממשק זה להיות נוח ומהיר, והשני הוא ממשק גרפי שדרכו ניתן לערוך את מסד הנתונים וממשק זה צריך להיות כמה שיותר פשוט והשליטה בו תהיה בעיקר על ידי לחיצות עכבר.

3.2.2 ממשקי חומרה

מבחינת החומרה הדרישה המרכזית היא שכלל המחשבים יתמכו בהליכים. חוץ מזה מומלץ שהמחשב של איש ה IT יהיה מחשב מהיר כדי שהשרת יכול לטפל בתשובת הלקוחות במהירות.

3.2.3 ממשקי תוכנה (אם התוכנה מחולקת לכמה תוכנות שונות שצריכות לתקשר בניהן)

התוכנה עצמה מחולקת לארבעה תוכנות עם דרישות.

מסד הנתונים - מסד הנתונים הוא מסוג mongodb ויש להוריד אותו למחשב של איש ה IT, מסד הנתונים חייב תמיד להיות מופעל.

שאר הממשקים (שרת, לקוח ותוכנת המשתמש) מתקשרים ביניהם (דרך השרת) דרך סוקטים, התקשורת נעשית באמצעות פרוטוקול מיוחד. יש לשמור שהמידע עובר בסוקטים יהיה לפי הפרוטוקול.

תוכנת השרת והלקוח חייבים תמיד להיות מופעלים בעוד שתוכנת המשתמש יכולה להיסגר ולהיפתח.

3.2.4 ממשקי תקשורת (למשל תקשורת בין שרת ללקוחות)

התקשרות בין כלל התוכנות (למעט מסד הנתונים) הוא דרך סוקטים בפרוטוקול TCP.
חשוב לדעת כי המידע עצמו העובר בין השרת, לקוח ותוכנת המשתמש נעשית לפי פרוטוקול מיוחד. הפרוטוקול בנוי מהדר שהוא ארבעה בייטים הנמצאים בתחילת ההודעה שמראים את גודל ההודעה ומההודעה עצמה. כל הודעה העוברת דרך הסוקטים צריכה להיות בהתאם לפרוטוקול הייחודי.

התקשורת בין תוכנת המשתמש לשרת אינו עובר דרך הרשת הפנימית מאחר ושניהן נמצאות על אותו מחשב, ולכן מערכת ההפעלה אחראית על העברת המידע בין התוכנות.

3.3 דרישות לא פונקציונליות

דרישות מהמערכת שלא מתבטאות בפיצ'ר ספציפי או בתהליך ספציפי שמתרחש במערכת אבל משפיעות על אופן עיצובה ומימושה, לדוגמא:

3.3.1 דרישות ביצועים (performance)

על המערכת להיות מהירה כדי לחסוך זמן. צריך לדאוג שהאופן שבו השרת מוצא לקוחות ומקבל את התשובות שלהם יהיה יעיל ומהיר.

3.3.2 דרישות מהימנות (reliability)

על כל חלקי המערכת לעבוד ללא אפשרות לקריסה, על מנת שלא יתקל המשתמש בבעיות. כלל חלקי המערכת צריכים להיות מתוחזקים ואמינים מבחינת ריצה על מנת שלא תהיה שום בעיה. המערכת צריכה לפעול תוך אמינות ויעילות רבה.

3.3.3 דרישות זמינות (Availability)

על השרת, הלקוח ומסד הנתונים לפעול כל הזמן, על מנת לספק זמינות מתמדת לאיש ה IT באופן מיידי. על המערכת להיות נגישה ופשוטה לשימוש, על המערכת להשתמש לדאוג שכלל הפיצ'רים של המערכת יהיו פשוטים להבנה.

3.3.4 דרישות אבטחה (security)

העברת המידע בין התוכנות צריכה להיות בטוחה כדי שמידע לא ידלוף וגם צריך שתהיה גישה למסד הנתונים רק למי שהוא איש IT .

3.3.5 דרישות תחזוקה (maintainability)

על המחשבים בארגון לעבוד כמו שצריך כדי שהתוכנה לא תקרוס. צריך לדאוג המסד הנתונים תמיד עובד.

3.3.6 דרישות ניידות (portability)

אין.

3.4 דרישות בסיס נתונים.

בסיס הנתונים הוא מסוג mongodb. בסיס נתונים זה משתמש בפורמט מיוחד שנקרא BSON ששומר מידע לפי אוספים. המידע בכל מסמך (בתוך כל אוסף יש מסמך שבו נשמר המידע עצמו) מסודר לפי מפתח וערך (כמו במילון). הנתונים הנשמרים בבסיס הנתונים הם כתובות ip של הלקוחות המחוברים לשרת. כתובות ip מסודרים בבסיס הנתונים בתוך אוספים כשאר כל אוסף מייצג את הקבוצה אליה המחשב שייך. קבוצה זו יכולה להיות למשל, המחשבים במעבדת פיזיקה או המחשבים של המורים.מאחר וכל מה שאנו שומרים הם כתובות ip, איננו צריכים שטח אחסון גדול. חשוב לציין כי בסיס הנתונים נמצא על המחשב של איש IT ורק למחשב זה יש גישה אליו ולכן המידע שבבסיס הנתונים נשאר היינו חסוי. וגם, הצפנה ופענוח של המידע יכול לגזול הרבה זמן ולהוריד מיעילות המערכת. עקב סיבות אלה אין צורך בהצפנת המידע שבבסיס הנתונים.

3.5 דרישות נוספות
קבצים שיש להוריד ולהריץ:

MongoDB Community Server - https://www.mongodb.com/try/download/community

במחשב של איש ה IT יש לדאוג שהספריות הבאות נמצאות:

```
import pymongo
 import ipaddress
import base64
import os
import socket
import threading
import msvcrt
```

במחשב של הלקוח צריכים להיות הספריות החיצוניות הבאות:

```
import socket
import threading
import pyautogui
import glob
import os.path
import shutil
import subprocess
import base64
```

** צריך גם להוריד את ספריית pillow אצל שניהם*

VIT

מסמך עיצוב

ינון ריבקין

<גרסה2 >

<18.5.2022>

היסטוריית גרסאות המסמך

תאריך	גרסה	תקציר השינויים
16.1.2022	1	הפרוייקט עדיין לא יציב וחלקים גדולים בו עדיין לא מוכנים
18.5.2022	2	הפרוייקט גמור

1. הקדמה

החלק הזה ישמש להצגה כללית של מסמך העיצוב. הסבר כללי, וקישור למסמכים קודמים כמו מסמך האפיון עליו אנו מתבססים

1.1 מטרה

מטרת הפרויקט הוא לעזור לאנשי IT לטפל במספר גדול של מחשבים באמצעות פקודות פשוטות, התוכנה תאפשר פנייה לקבוצות של מחשבים. למשל, לכל המחשבים שנמצאים בחדר מדעי המחשב או לאלה שבפיזיקה או גם לאלה שבחדר פיזיקה וגם לאלה שבחדר ביולוגיה. הוא מיועד לאנשים שמבינים במחשבים ובפרט אנשי IT

1.2 המוצר

- *שם המוצר: VIT*
- *המוצר נותן אפשרות לטפל במספר רחב של מחשבים דרך חלוקה שלהם לקבוצות ופנייה אליהם באמצעות פקודות פשוטות.*
- *מטרת המוצר הוא לתת גישה לאיש IT לכלל המחשבים ולטפל בהם לפי קבוצות וכך לחסוך לאיש ה IT זמן.*

1.3 קישור למסמכים קודמים

קישור למצגת של הפרויקט:
https://docs.google.com/presentation/d/15eyAhPA5dRZBf4w4KLGcZ-w4W0zQ9tawEA-syu8G858/edit?usp=sharing

1.4 הגדרות

הגדרות:

מסד נתונים - הוא אמצעי המשמש לאחסון מסודר של נתונים במחשב, לשם אחזורם ועיבודם.

תוספים (אוספי תוספים, אוסף) - תוספים או אוספי תוספים או אוסף הם כינוי לשמות קבוצות השיוך שבהם נמצאים המחשבים, כל מחשב יכול להיות שייך לכמה תוספים.

כתובת ip - היא תווית מספרית (לדוגמה 192.168.70.10) המשמשת לזיהוי נקודות קצה ברשתות תקשורת שבהן משתמשים בפרוטוקול התקשורת IP, כגון רשת האינטרנט. נקודת קצה יכולה להיות מחשב, מדפסת, כונן ועוד

פורט -הוא תהליך שדרכו יכולות תוכנות להעביר נתונים. הפורט הינו מספר זיהוי של תכנית המחשב.

שרת- לקוח- השרת הוא תוכנה פסיבית, המאזינה לרשת ומחכה לקבל בקשות. הלקוח לעומתו בדרך כלל מהווה את ממשק המשתמש – הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו.

סוקטים - הוא נקודת קצה (endpoint) עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים.

הליך - הליך מאפשר לספק למשתמש במערכת ההפעלה מהירות תגובה ורציפות פעולה כאשר התהליך (יישום) מבצע כמה משימות במקביל.

איש IT - איש האחראי על התאמת מערכות מחשוב לעסקים, שרתים, תוכנות אנטי וירוס, תוכנות גיבוי, מערכות שונות הדרושות לקיום הארגון וכלים טכנולוגיים שונים.

מסד נתונים של mongodb - מסד זה הוא בסיס הנתונים המוביל בעולם בקטגוריית NoSQL, ובין חמשת המובילים בכל הקטגוריות. בסיס הנתונים נשען על מבנה של מסמך בניגוד לבסיסי נתונים טבלאיים העובדים מעל טבלאות מקושרות. מבנה המסמכים עובד מעל מימוש של JSON הנקרא על ידי BSON MongoDB.

2. ארכיטקטורת המערכת

חלק זה כולל את תיאור מבנה המערכת ופירוט המודולים השונים בה

* חובה להוריד ולהריץ מסד נתונים של mongodb

2.1 מבט על

כל הפרויקט כתוב ב python. בפרויקט זה קיימים שישה צדדים: צד של לקוח שהוא המחשבים הנשלטים אשר מקבלים פקודות מהשרת, מריצים אותם ומחזירים תשובה לשרת. צד של שרת שהוא אחראי על העברת המסרים בין התוכנה של המשתמש ללקוחות הרצויים וגם הוא שומר ולוקח מידע ממסד הנתונים. צד התוכנה של איש ה IT (המשתמש) שבו נשלחים הפקודות לשרת ומשם ללקוחות הרצויים. צד המסד הנתונים שהוא מסד נתונים של Mongodb המאפשר שמירה של מידע לא לפי טבלה אלה לפי אוספים, הוא פשוט וקל לשימוש.צד הפרוטוקול שבעזרתו כל השרת ותוכנה של המשתמש מנהלות את התקשורת עם תוכנות אחרות וגם עם הפרוטוקול ניתן לשמור ולקבל מידע ממסד הנתונים, קיימת גרסה "פשוטה" של הפרוטוקול המכילה רק מה שנחוץ ללקוח ובו הלקוח משתמש. ולבסוף הצד של המערכת לשליטה במסד הנתונים שזוהי מערכת GUI (ממשק משתמש גרפי) אשר מאפשרת לשליטה נוחה במסד הנתונים.

השרת והלקוח הם בסקוטים מאחר וזו דרך יעילה וטובה להעביר מידע וגם התקשורת בין התוכנה של המשתמש לשרת היא באמצעות סוקטים. נוסף לכך, קיים בסיס נתונים שבו שמורים הכתובות של המחשבים והשיוך שלהם לקבוצה. הסיבה לכך שצריך היא מפני שאנו רוצים לשמור את הכתובות של המחשבים בצורה כזאת שיהיה קל למצוא את המחשבים ויהיה קל לשנות את השיוך שלהם לפי הצורך.

עם התוכנה של המשתמש רושמים את הפקודות, חשוב לזכור לרשום לאיזה תוסף או כתובת ip ברצוננו לשלוח את הפקודה. ניתן לרשום כמה תוספים וכתובות ip.

2.2 פירוט רכיבי המערכת

כאן יופיע פירוט לכל רכיב (ניתן לעשות באיטרציות – כל פעם עבור הרכיבים הרלוונטיים, לפני תהליך הפיתוח שלהם)

בסעיף זה יש לפרט את המבנה הפנימי של כל מודול/רכיב - כלומר:

הפרוטוקול:

הפרוטוקול הוא המרכיב הכי חשוב מאחר וכמעט כל הרכיבים האחרים משתמשים בו ולכן אציג אותו קודם. רכיב זה אחראי על ניהול התקשורת בין הרכיבים המשתמשים בו וגם על שמירת וקבלת נתונים ממסד הנתונים. חשוב
לציין כי יש גרסה פשוטה של הפרוטוקול המכילה רק את שתי הפונקציות האחרונות שיוזכרו פה.

המאפיינים של הרכיב:

```
mydb = data_base.DataBase()
command_list = ["state", "delete", "execute", "copy", "dir", "help", "take_screenshot", "send_photo", "find_path"]
```

פירוט על הפונקציות הנמצאות בפרוטוקול:

def add_ip_to_db(ip):

פונקציה זו מאפשר להוסיף כתובת ip לשרת. הכתובת נשמרת באוסף שנקרא "free-" ובו נשמרים כל הכתובות. והיא מחזירה True כדי להראות שהיא סיימה.

def additive_list():
פונקציה המחזירה על כל שמות אוספי "התוספים" הנמצאים במסד הנתונים.

def ip_list(col):
פונקציה זו מחזירה ממסד הנתונים כל את כתובות ה ip הנמצאות באוסף הנתון.

def state():
פונקציה זו מחזירה מחרוזת שבה נמצאים כל אוסף וכתובות ה Ip הנמצאים בו.

def validate_ip_address(address):
פונקציה שבודקת אם כתובת ip נמצאת בתוך מסד הנתונים.

def is_additive(additives):
פונקציה פנימית שמשתמשים בה בפונקציה command_split והיא בודקת אם כל תוסף שנמצא ברשימה של התוספים (אותה היא מקבלת) מתאים כדי להיחשב כתוסף.

def is_add_an_ip(address):
פונקציה שנמצאת בתוך הפונקציה is_additive ותפקידה הוא לבדוק
אם התוסף (אותו היא מקבלת) הוא כתובת ip אפשרית ואם כן אז הוא תקני.

def command_split(command):
פונקציה המקבלת פקודה שלמה ומפרקת אותו לגורמים החשובים. ולבסוף היא מחזירה כל גורם לבד.

def help_func():
פונקציה המחזירה מחרוזת ובה כל הפקודות שהמשתמש יכול להשתמש בהם
וגם את כל התוספים הקיימים במסד הנתונים.

def check_param_is_additive(params):
פונקציה זו בודקת אם כל חלק ברשימה שניתן לה נמצא במסד הנתונים.

def check_additive(add):
השימוש בפונקציה זו מתבצע על ידי הפונקציה check_command שמשתמשת בה כדי לבדוק
אם תוסף בחלק של התוספים הוא תקני. פונקציה זו משתמש גם בפונקציה
check_param_is_additive

check command(command, param, add):
הפונקציה מרכזית שבודקת אם הפקודה ששלח המשתמש היא תקנית על כל חלקיה.
פונקציה זו משתמשת גם בפונקציה check_additive.

def create_msg(data):
פונקציה זו היא הכרחית בתהליך התקשורת בין הרכיבים. היא אחראית על הוספת הדר המראה את גודל
ההודעה שברצונו לשלוח וזה הכרחי כדי שהצד המקבל יקבל את כל ההודעה בדיוק.

def get_msg(my_socket):
פונקציה האחראית על קבלה של מידע מהסוקט ופענוח שלו.

המסד הנתונים:

מסד הנתונים הוא מסד מסוג mongodb. והוא נמצא על המחשב של של איש ה IT.
ברכיב של מסד הנתונים קיימת מחלקה אחת .
class Database:

מחלקה זו אחראית על התחברות למסד הנתונים ומכילה מתודות לעריכה של מסד הנתונים, שליפה, מחיקה, הוספה וכו'. דרכה ניתן לערוך את מסד הנתונים בקלות רבה.

תכונות המחלקה:

```
self._myclient = pymongo.MongoClient("mongodb://localhost:27017/")
# creating database
self._mydb = self._myclient["mydatabase"]
# the dictionary is created in order to have easy access to all collections
self._addcol_dic = {}
# getting all collections from database and storing it in dictionary
for i in self._mydb.list_collection_names():
    self._addcol_dic[i] = self._mydb.get_collection(i)
```

כל התכונות הן מסוג private כך שאין גישה אליהן מחוץ לקובץ.

הפונקציות של המחלקה:

אפיון הפונקציה	שם הפונקציה והפרמטרים
משמשת כדי להתחבר למסד, יצירה של מאגר נתונים ושמירה של כל האוספים הקיימים במאגר במילון.	def __init__(self):
בודקת אם התוסף קיים ברשימת אוספי התוספים הקיימת במסד.	def check_addcol(self, add):
מעדכנת את המילון ומחזירה את את המפתחות שלו.	def get_all_addcol(self):
מקבלת שם של אוסף ומוחקת אותו מהמסד.	def delete_col(self, add):
מקבלת כתובת ip ושם של תוסף ומוסיפה את הכתובת לאוסף של התוסף.	def add_ip_to_addcol(self, add, ip):
לוקחת שם של תוסף ומחזירה את כל הכתובות שנמצאות באוסף של אותו תוסף.	def get_all_ip_from_addcol(self, add):
לוקחת כתובת IP ומוחקת כל קיום שלו במסד.	def delete_ip_from_data_base(self, ip):
מקבל כתובת IP ומוסיפה אותה לאוסף הכללי.	def add_ip_to_freecol(self, ip):
מקבלת כתובת IP ושם של תוסף ומוחקת את כתובת ה IP מהאוסף של התוסף.	def delete_ip_from_addcol(self, add, ip):
מקבלת שם של תוסף ומוסיפה אותו למסד הנתונים.	def add_addcol(self, add):

השרת:

לשרת יש שתי ערוצי תקשורת שאותם הוא מנל, הראשונה הוא שרת הליכים שזה אומר שלכל לקוח חדש שמתחבר השרת יוצר הליך שמנהל את התקשורת עם הלקוח וכך השרת יכול לתקשר עם מספר גדול של לקוחות במקביל. התקשורת עם כל לקוח מתבצעת על פורט 8080. השרת שומר כל כתובת של מחשב במסד הנתונים ואת ההליכים בתוך רשימה של הליכים פעילים. נוסף לכך, השרת

מנהל תקשורת גם עם התוכנה של המשתמש וגם בשבילה הוא יוצר הליך האחראי על התקשורת איתו. התקשורת עם התוכנה של משתמש מתבצעת על פורט 8820 ובכתובת "127.0.0.1".

<u>המחלקות של השרת:</u>

class ClientThread(threading.Thread):

המאפיינים של המחלקה:

```
threading.Thread.__init__(self)
self._client_socket = clientSocket
# sets name to ip address only.
self.name = clientAddress[0]
protocol.add_ip_to_db(self.name)
self.msg = ""
self.response = ""
self.event = threading.Event()
self.lock = threading.Lock()
self.photo_path = r""
```

הפונקציות של המחלקה:

אפיון הפונקציה	שם הפונקציה והפרמטרים
אחראית על אתחול המאפיינים ועל הוספת כתובת הלקוח למסד הנתונים	def __init__(self, clientAddress, clientSocket):

def get_name(self):	מחזירה את המאפיין name
def set_photo_path(self,photo_path):	מקבלת את מיקום התיקיה שבה נשמרים תמונות ומציבה אותו ב self.photo_path
def get_photo_path(self):	מחזירה את הכתובת שבה נשמרים תמונות
def set_msg(self, msg):	מקבלת הודעה ומציבה אותו במאפיין self.msg
def get_msg(self):	מחזיר את self.msg
def set_response(self,response):	מקבל תשובה ומציב אותה ב self.response
def get_response(self):	מחזיר את self.response
def send_photo_handler(self):	אחראי על קבלת התמונה מהלקוח ועל שמירה שלו בקובץ המתאים
def req_handeler(self, command):	מקבל פקודה ודואג שהיא תשלח ללקוח ומחזיר את התשובה של הלקוח
def run(self):	אחראי על שליחה של המידע ללקוח ועל קבל של מידע מהלקוח

class user_socket(threading.Thread):

המחלקה אחראית על התקשורת עם התוכנה של המשתמש, על קבלת מידע מהמשתמש ועל החזרת תשובה אליו. וגם היא אחראית על מציאת הלקוחות שאליהם הפקודה צריכה להגיע.

המאפיינים שלה:

```
threading.Thread.__init__(self)
self.IP = "127.0.0.1"
self.PORT = 8820
self.Sphoto_path = r""
self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.server_sock.bind((self.IP, self.PORT))
```

הפונקציות שלה:

אפיון הפונקציה	שם הפונקציה והפרמטרים
אחראית על אתחול המאפיינים ויצירת סוקט	def __init__(self):
מקבלת רשימה של תוספים והליך ובודקת אם ההליך הנתון שייך לאחד מן התוספים הרצויים	def check_thread_group(self, additives, thread):
מקבלת פקודה ואחראית שהפקודה תשלח להליכים הרצויים. מחזירה רשימה של תשובות של כל ההליכים שהפקודה הגיעה אליהם.	def user_handler(self, command):
יוצר תיקיה שבה ישמרו התמונות שהתקבלו מלקוחות.	def create_sphoto_path(self):
הפונקציה הראשית האחראית על התקשורת עם תוכנת המשתמש (קבלה ושליחה של מידע)	def run(self):

כמו כן קיים חלק בשרת שלא נמצא בתוך מחלקה:

def main():

פונקציה זו אחראית על יצירת השרת שאליו הלקוח מתחבר, יצירת הליך (ClientThread) שאחראי על התקשורת עם אותו לקוח וגם יצירת האובייקט שאחראי על התקשורת עם התוכנה של המשתמש.

<u>הלקוח:</u>

הלקוח אחראי על התחברות עם השרת, קבלת פקודות מהשרת וביצוען. חשוב לזכור שצריך לרשום את הכתובת של השרת באופן ידני.

המחלקה של הלקוח:

class Client:

המאפיינים של המחלקה:

```
self.IP = SERVER_IP
self.PORT = 8080
self.photo_path = r""
self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

הפונקציות של המחלקה:

אפיון הפונקציה	שם הפונקציה והפרמטרים
אחראית על אתחול המאפיינים ויצירת סוקט	def __init__(self):
מקבלת פקודה ובודקת אם היא תקנית ובמיקרה שיש כתובת של קובץ בפקודה, היא בודקת אם הכתובת תקנית.	def check_server_request(command):
אחראית על ביצוע בפקודות ומחזירה תשובה שהפעולה בוצעה או לא. (יהיה פירוט על איך צריך לכתוב כל פקודה נכון בחלק 3)	def handle_server_request(self, command, params):
יוצר תיקיה שבה ישמרו התמונות שנלקחו.	def create_dir(self):
הפונקציה הראשית האחראית על התקשורת עם השרת (קבלה ושליחה של מידע)	def main(self):

תוכנת המשתמש:

תוכנת המשתמש היינה הדרך של איש IT לשלוח פקודות לכלל המחשבים במערכת.
לתוכנה זו שני חלקים חשובים, הראשונה היא מחלקה שאחראית על התקשורת עם השרת והשניה היא כמה פונקציות האחראיות על קבלת קלט והחזרת פלט למשתמש.

ראשית המחלקה האחראית על התקשורת עם השרת.

class UserSocket(threading.Thread):

המאפיינים של המחלקה:

```
threading.Thread.__init__(self)
self.IP = "127.0.0.1"
self.PORT = 8820
self.client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.msg = ""
self.response = ""
self.lock = threading.Lock()
self.event = threading.Event()
```

הפונקציות של המחלקה:

אפיון הפונקציה	שם הפונקציה והפרמטרים
אחראית על אתחול המאפיינים ויצירת סוקט	def __init__(self):
מקבלת הודעה ומציבה אותו במאפיין self.msg	def set_msg(self, msg):
מחזיר את self.msg	def get_msg(self):
מקבל תשובה ומציב אותה ב self.response	def set_response(self,response):
מחזיר את self.response	def get_response(self):
מקבל פקודה ודואג שהיא תשלח לשרת ומחזיר את התשובה של השרת	def handler(self, command):
אחראי על שליחה של המידע לשרת ועל קבל של מידע מהשרת	def run(self):

קיימות פונקציות שלא נמצאות בתוך מחלקה והן אחראיות על קבלת מידע מהמשתמש והצגה של פלטים. התוכנה מרגישה כמו cmd.

def ui(thread):

פונקציה זו מקבלת את ההליך האחראי על התקשורת עם השרת.
פונקציה זו אחראית על קבלת מידע מהמשתמש והצגה של התשובות המגיעות מהשרת.
היא משתמשת בפונקציה (**send_command(command, thread** כדי לנהל את השליחה של הפקודות.

def send_command(command, thread):

הפונקציה מקבלת את הפקודה שרשם המשתמש ואת ההליך האחראי על התקשורת עם השרת.
פונקציה זו שולחת להליך האחראי על התקשורת עם השרת את הפקודה, לוקחת את התשובה של השרת ומטפלת בה כדי שיהיה ניתן להציג אותה בצורה ברורה. ומחזירה התשובה של השרת המטופלת.

def treat_data(data):

בפונקציה זו הפונקציה **send_command** משתמשת כדי לטפל במידע המגיע מהשרת.

def main():

פונקציה זו אחראית על יצירת ההליך האחראי על התקשור עם השרת ועל יצירת הליך האחראי על פונקצית ()ui ובכללי על החלק של קבלת מהמשתמש ועיבוד של התשובה מהשרת.

המערכת הגרפית לעריכת מסד הנתונים:

מערכת זו נותנת גישה נוחה למסד הנתונים והיא מאוד נוחה לשימוש.
מערכת זו פותחה באמצעות tkinter בחלק של ממשק המשתמש (חלק 4) אראה איך היא נראית ואסביר איך היא בנויה לעומק.

כל שינוי שהמשתמש עושה דרך המערכת נשמר במסד הנתונים.

אפרט על הפונקציות השונות בה:

class Button(tk.Button):

מחלקה זו היא עריכה של המחלקה המקורית של כפתור. הדבר היחידי שהוספתי הוא מאפיין שנקרא hidden שאומר אם לחצו על הכפתור.

def update_scroll_region():

פונקציה זו מעדכנת את פס הגלילה בכל פעם שמוסיפים אוסף חדש.

def open_addpopup():

פותח חלונית שבה אפשר לבחור באוסף.
פונקציה זו משתמשת בשתי הפונקציות הבאות כדי שהחלונית תיסגר כאשר נבחר באוסף ונלחץ הכפתור :

def set_col_name(lst_box):

פונקציה זו אחראית לקבלת הבחירה של האוסף מהמשתמש.

def open_addpopup_click(lst_box, top):

פונקציה זו אחראית על השמדת החלונית ברגע שנבחר אוסף (כאשר הפונקציה set_col_name מחזירה True).

def add_ip_to_col():

פונקציה זו אחראית על הוספה של ip לאוסף אותו נבחר.
פונקציה זו עושה שימוש בפונקציה **open_addpopup** כדי לבחור אוסף.

def delete_click():

פונקציה זו אחראית על מחיקת ה ip אותו המשתמש בחר מרשימת ה ip
של האוסף שאותו בחר.
במיקרה שהאוסף ממנו אנו מוחקים את הכתובת הוא "free-" (התוסף שבו נמצאים כל הכתובות)
אז הפונקציה קוראת לפונקציה אחרת כדי שכל קיום של הכתובת, גם באוספים אחרים, תימחק.
הפונקציה שבה היא משתמשת במקרה הזה היא:

def delete_ip_all_lists(ip):

פונקציה זו מקבלת כתובת ip ומוחקת אותו מרשימת הכתובות של כל אוסף (במידה והוא קיים באותו אוסף)

def delete_col_click():

פונקציה זו תפקידה הוא למחוק אוסף. חשוב לציין שאי אפשר למחוק את האוסף "free-" (התוסף שבו נמצאים כל הכתובות)
פונקציה זו עושה שימוש בפונקציה **open_addpopup** כדי לבחור אוסף.

def refresh_freecol():

בכל פעם שכפתור האוסף של free- נלחץ רשימת הכתובות שלו מתעדכנת כך שניתן לראות כתובות חדשות שהצטרפו.

def on_click(btn):

זו פונקצית ברירת המחדל של כל כפתורי האוספים. פונקציה זו אחראית לכך שכל פעם שנלחץ על כפתור של אוסף הכפתור ישאר לחוץ ורשימת הכתובות שלו תוצג וכן גם הכפתורים המאשרים מחיקה והעברה של כתובות יופיעו במקום הכפתורים של הוספה ומחיקת אוספים.

def add_name_button():

מאפשר הוספה של כפתורי אוספים חדשים שאליהם ניתן להוסיף כתובות. כל עד האוסף החדש נשאר ריק המסד הנתונים לא שומר אותו. את השם של האוסף החדש יש לרשום בתיבת המלל שנמצאת מעל לכפתור הוספה של אוספים. כדי שהשם של האוסף יהיה תקין חייב הוא חייב להתחיל ב "-" אחרת הוא לא יתווסף.

def add_col_from_db():

פונקצייה זו יוצרת כפתורי אוספים לפי האוספים הקיימים במסד הנתונים ויוצרת רשימת כתובות לפי הכתובות השמורות לאותו אוסף במסד הנתונים.

תומונות עם הסברים באנגלית של שאר הקוד שלא נמצא בפונקציה:

```
# creating the root window
window = tk.Tk()
# dictonarys for accessing needed objects
btn_frame_dic = {}
lstbox_dic = {}
button_dic = {}
# db is needed in order to access the database
db = data_base.DataBase()

# those Vars hold the the button that was selected
# and collection that was selected (from the popup window)
selected_btn_name = tk.StringVar()
selected_col_name = tk.StringVar()
selected_col_name.set("")
# creating the main frames that divide the window to constant parts and will host other frames
frm_upleft = tk.Frame(window, width=400, height=150, borderwidth=5)
frm_downleft = tk.Frame(window, relief="sunken", width=400, height=300, borderwidth=5)
frm_right = tk.Frame(master=window, relief="ridge", width=750, height=300, borderwidth=5)

# the canvas hosts the collection buttons and displays them
cvs_buttons = tk.Canvas(frm_upleft, width=400, height=50)
horibar = tk.Scrollbar(frm_upleft)
frm_canvas = tk.Frame(cvs_buttons, width=400, height=50)

# addcol_frm is hosted in the "right frame" and contain the widgets for adding and deleting collections
addcol_frm = tk.Frame(frm_right, width=750, height=300)
del_col_btn = tk.Button(master=frm_right, text="delete collection", width=20, command=delete_col_click)
lbl_addname = tk.Label(master=addcol_frm, text="add collection")
ent_addname = tk.Entry(master=addcol_frm)
btn_add = Button(master=addcol_frm, text="Add", command=add_name_button)
```

```python

# the opt_frm is hosted in the "right frame" and holds the widgets needed to delete and add an ip
# it appears when a collection button is pressed
opt_frm = tk.Frame(frm_right, width=750, height=300)
btn_addip = tk.Button(master=opt_frm, text="Add ip", command=add_ip_to_col)
btn_del = tk.Button(master=opt_frm, text="delete ip", command=delete_click)

# main frames griding
frm_upleft.grid(column=0, row=0, columnspan=2, sticky="sewn")
frm_downleft.grid(column=0, row=1, columnspan=2, rowspan=2, sticky="nsew")
frm_right.grid(column=2, row=0, rowspan=3, sticky="nsew")
addcol_frm.grid(column=0, row=0, rowspan=3, sticky="nsew")

# setting the size and setting for the collection button's canvas and scrollbar
cvs_buttons.config(width=400, height=50)
cvs_buttons.config(xscrollcommand=horibar.set)
horibar.config(orient=tk.HORIZONTAL, command=cvs_buttons.xview)
cvs_buttons.grid(column=0, row=0, columnspan=2, sticky="sewn")
horibar.grid(column=0, row=1, columnspan=2, sticky="ew")
cvs_buttons.create_window(0, 0, window=frm_canvas, anchor=tk.NW)

# once the canvas is ready it adds all collection from database to gui
add_col_from_db()

# griding all addcol_frm's widgets
del_col_btn.grid(column=0, row=0, sticky="new")
lbl_addname.grid(column=0, row=1, sticky="s")
ent_addname.grid(column=0, row=2, sticky="new", pady=10)
btn_add.grid(column=0, row=3, sticky="nesw")

# griding all opt_frm widgets
btn_addip.grid(column=0, row=1, sticky="nesw")
btn_del.grid(column=0, row=2, sticky="nesw")
```

```
# configuring all row and columns of frames
frm_right.columnconfigure(0, weight=1)
frm_right.rowconfigure(0, weight=1)
frm_right.rowconfigure(1, weight=1)
frm_right.rowconfigure(2, weight=1)

addcol_frm.columnconfigure(0, weight=1)
addcol_frm.rowconfigure(0, weight=1)
addcol_frm.rowconfigure(1, weight=1)
addcol_frm.rowconfigure(2, weight=1)

opt_frm.columnconfigure(0, weight=1)
opt_frm.rowconfigure(0, weight=1)
opt_frm.rowconfigure(1, weight=1)
opt_frm.rowconfigure(2, weight=1)

frm_upleft.columnconfigure(0, weight=1)
frm_upleft.columnconfigure(1, weight=1)
frm_upleft.rowconfigure(0, weight=1)
frm_upleft.rowconfigure(1, weight=1)

frm_downleft.columnconfigure(0, weight=1)
frm_downleft.columnconfigure(1, weight=1)
frm_downleft.rowconfigure(0, weight=1)
frm_downleft.rowconfigure(1, weight=1)

window.columnconfigure(0, weight=1)
window.columnconfigure(1, weight=1)
window.columnconfigure(2, weight=1)
window.rowconfigure(0, weight=1)
window.rowconfigure(1, weight=1)
window.rowconfigure(2, weight=1)
```

```
368
369     frm_canvas.update()
370     cvs_buttons.update()
371     window.mainloop()
372
```

2.3 דיון בנושא העיצוב הנבחר

בדיון זה אעסוק בעיצוב של של השרת, לקוח,תוכנת המשתמש , ממסד הנתונים ובמערכת הגרפית לעריכה של מסד הנתונים. כמו כן אדבר על למה בחרתי לנהל את התקשורת בסוקטים ולכתוב את הפרויקט ב python

ראשית העיצוב של השרת.

השרת שבו אנו משתמשים הוא שרת הליכים, משמע שכל תקשורת עם לקוח מנוהל על ידי הליך ייעודי חדש וכך ניתן לנהל תקשורת עם כמות גדולה של לקוחות (כמספר ההליכים שניתן לפתוח במחשב בו השרת נמצא). כמו כן, קיים הליך נוסף שאחראי על התקשורת עם תוכנת המשתמש, הליך זה אחראי על קבלת מידע מהמשתמש והעברה שלו להליכים הרצויים והוא גם מחזיר את התשובה של הלקוח למשתמש. קיימים יתרונות וחסרונות לשיטת התקשורת עם הלקוחות שבה אני משתמש, היתרונות הם שהשרת יכול לתקשר עם מספר גדול של לקוחות, ושנית הוא שניתן לשמור כל הליך ברשימה וכך ניתן למצוא בקלות את ההליכים הרצויים. החסרונות הם: שיחסית מסובך לדאוג לכך ששתי הלכים (ההליך שאחראי על התקשורת עם הלקוח וההליך שאחראי על קבלת מידע מהמשתמש) יתנהלו בסינכרון ויחכו עד שכל אחד מהם יסיים את העבודה וכמות גדולה של הליכים יכולה לגרום למחשב לעבוד מאוד קשה ולכן הוא יהיה איטי.
התוכנה של השרת בנוייה ככה שהיא צריכה להיות מופעלת כל הזמן מאחר ואנו רוצים שיהיה לאיש ה IT גישה תמידית לכל המחשבים.

העיצוב של הלקוח.

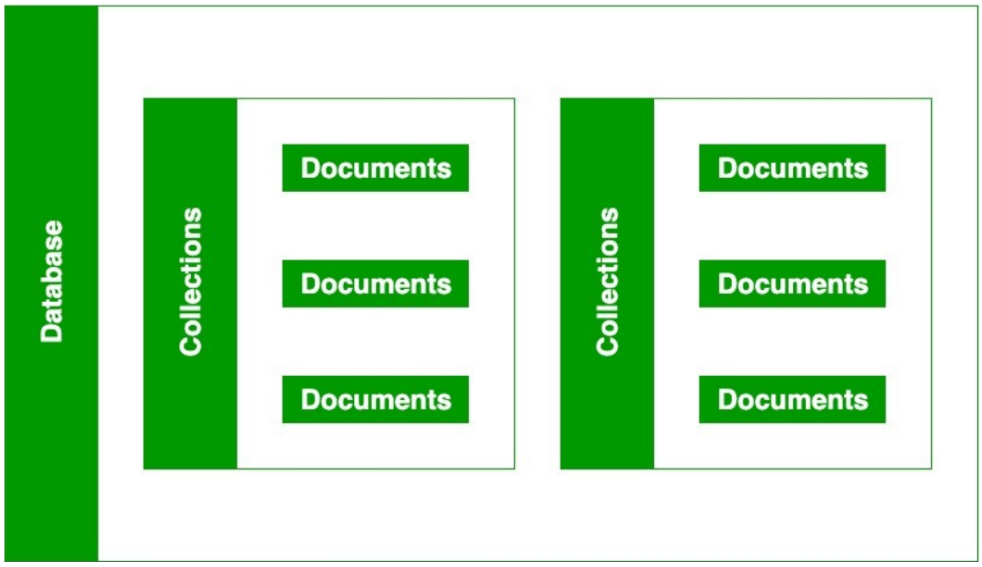
העיצוב של הלקוח הוא יחסית פשוט, כל מה שהוא עושה זה להתחבר לשרת, להקשיב להודעות שמגיעות מהשרת, לפעול לפי ההודעות בהתאם ולהחזיר תשובה לשרת.
הלקוח צריך להיות תמיד מופעל מאחר וברצוננו שתמיד תהיה לאיש ה IT גישה אליו.

תוכנת המשתמש

תוכנת המשתמש היא התוכנה באמצעות איש ה IT יכול לרשום פקודות שישלחו לשרת ומשם ללקוחות המיועדים. לתוכנה זו שני חלקים, חלק ראשון הוא מחלקה שהיא יורשת ממחלקת ההליכים, מחלקה זו אחראית על התקשורת עם השרת.תוכנת המשתמש והשרת נמצאים על אותו מחשב והם מתקשרים דרך localhost (תקשורת פנימית בין תהליכים הנמצאים על אותו מחשב, תקשורת זו עוברת דרך מערכת ההפעלה)ץ חלק שני הוא ממשק המשתמש שדרכו ניתן לרשום פקודות, ממשק זה מתנהל בצורה של comman line (כמו ב CMD). שיטה זו נבחרה מאחר ורציתי לאתגר את עצמי ולנסות ליצור משהו שדומה ל CM, בפועל היה ניתן להשתמש ב input וזה היה אולי חוסך טיפה עבודה. כמו כן אחד החסרונות של השיטה שלי היא שלא ניתן להעתיק ולהדביק דברין לתוך התוכנה וחייבים לרשום הכל מאפס ולכן היא אולי פחות יעילה.

מסד הנתונים.

מסד הנתונים בו אני משתמש הוא mongodb. אשר שומר נתונים לא בצורה של טבלה אלה בדרך מיוחדת שנקראת bson. מסד זה עובד בצורה כזאת: תחילה יש את מאגר המידע שאותו אנו יוצרים, מאגר זה מכיל בתוכו אוספים ובכל אוסף נמצאים אובייקטים הדומים לאובייקט מילון ובוא נמצאים המידע שאותו אנו שומרים. תמונה הממחישה זאת:



בתוכנה של מסד הנתונים קיימת מחלקה האחראית על עריכה של מסד הנתונים לפי צרכינו. את מחלקה זו בניתי כך שתיתן מענה לכל הצרכים הנדרשים לנו ונוכל להשתמש בה גם כדי לפתור צרכים עתידיים.
השתמשתי ב mongodb מאחר והמידע היחידי שאני שומר הוא כתובות IP ופחות מעניין אותו קבלה של מידע ספציפי אלה קבלה של אוספים ולכן בחרתי בשיטה זו מאחר שהיא פשוטה מאוד, מדרישה התעסקות מינימלית ועונה על הדרישות של הפרויקט.

המערכת הגרפית לעריכה של מסד הנתונים.

מערכת זו בעלת ממשק משתמש גרפי ומאפשרת עריכה פשוטה של מסד הנתונים. את המערכת בניתי עם tkinter שהיא סביבה פשוטה ליצירת ממשקים גרפים. היתרונות של tkinter הם שניתן ללמוד את הסביבה מאוד מהר, היא מאוד מובנת וישירה. החסרונות שלה הם שלפחות בגרסת הבסיס שלה יש מעט widgets שניתן להתעסק איתם, הממשקים בה נראים מיושנים, שיפור הנראות שלה מדריש הרבה התעסקות. את מערכת זו יצרתי ככה שאת רוב הפעולות ניתן לעשות רק עם העכבר ולא צריך לרשום דברים, חוץ מהוספה של אוסף חדש שבמיקרה זה צריך לרשום את שמו של האוסף. לדעתי המערכת שיצרתי היא ברורה וקלה לשימוש. בעיה אחת בה היא שכאשר מגדילים את החלונית הממשק מתעוות טיפה (הוא עדיין עובד כמו שצריך אבל הוא לא נראה טוב).

דברים כללים:

עכשיו אתייחס ללמה בחרתי להשתמש בסוקטים ולכתוב את התוכנה ב python.

ראשית בחרתי להשתמש בסוקטים מאחר ואני יודע להשתמש בהם טוב, התקשורת בסוקטים מתנהל בעזרת פרוטוקול שבנוי מהדר, ההדר הוא ארבעה ביטים שמראים את גודל ההודעה, וההודעה עצמה עקב זה שההדר הוא ארבעה בייטים הגודל המקסימלי שההודעה יוכלה להיות היא 9999 בייטים.
קיים קובץ שבו יש פונקציות עזר ובין היתר פונקציות שאחראיות על יצירת הדר להודעה וקבלת מידע מהסוקט לפי הפרוטוקול, כל תוכנה שמעבירה מידע דרך סוקטים משתמשת בקובץ הנ"ל.

את כל הפרויקט כתבתי ב python מאחר ואני יודע להשתמש בה הייטב. שפה זו נותנת אפשרויות רבות ונוחות להתעסקות עם מידע, היא קלה לשימוש ומאוד קל להבין קוד שכתוב בה. עם python תהליך כתיבת הקוד הוא הרבה יותר מהיר ביחס לשפות אחרות והיא השפה שאותה למדנו בשיעורי סייבר. ולכן בחרתי להשתמש בה.

3. עיצוב נתונים ופרוטוקולים

כאן יופיע תיעוד של מבני נתונים שונים / פרוטוקולים המשמשים אותנו במערכת.

פרוטוקול תקשורת

בפרויקט, התוכנות שמקיימות תקשורת ביניהן עושות זאת דרך סוקטים באמצעות פרוטוקול TCP. אנו משתמשים ב TCP מאחר והוא פרוטוקול בטוח המאפשר ווידו הגעה של המידע.

את שליחת המידע אנו עושים דרך פרוטוקול ייעודי שאיתו ניתן להבדיל בין הודעה להודעה. הפרוטוקול בנוי בצורה הבאה

הדר:
ארבעה בייטים שנמצאים בתחיל ההודעה אשר מראים מה גודל ההודעה. (כל כל הודעה יכולה להיות בגודל מקסימלי של 9999 בייטים).

התוכן:
ההודעה עצמה אותה אנו שולחים.

דוגמא לאיך ההודעה נראית אחרי הפרוטוקול:

“0015find_path -free”

כאשר אנו רוצים לקבל את ההודעה, הפונקציה האחראית על קבלת ההודעה קודם קולטת את ההדר ולפיו היא יודעת את גודל המידע שאותו היא רוצה לקלוט.

הסברים על איך לרשום נכון כל פקודה ומה כל פקודה מחזירה:

**** בפקודות הבאות חשוב לרשום בסוף כל פקודה את התוסף או את כתובת ה ip של המחשבים שאליהם הפקודה תגיע. ניתן לרשום כמה תוספים ולרשום גם תוספים וגם כתובות ip.***

FIND_PATH

כדי להריץ נכון את find_path צריך לרשום את הפקודה באותיות קטנות. לדוגמא:

find_path -free

הפקודה תחזיר את ה PATH של קובץ הלקוח במחשב.

DIR

כדי להריץ נכון את הפקודה dir יש לרשום את הפקודה באותיות קטנות,

את המיקום של התיקיה אותה אנו רוצים לסרוק וכמובן כתובת ip של המחשב ספציפי (ניתן לפנות לתוסף אבל במקרה כזה רק המחשבים שיש להם את המיקום הזה יחזירו תשובה והשאר יחזירו שאין להם תיקייה כזאת) . לדוגמה:

dir C:\Users\Innon\Downloads 10.0.0.5

או

dir C:\Users\Innon\Downloads -free

הפקודה תציג את מיקומם של הקבצים שנמצאים בתוך התיקייה. חשוב לדעת שקבצים בעלי שמות עבריים עלולים לגרום לבעיות.

* הפקודה לא יכולה להעביר מעל ל 9999 בייטים עקב המגבלה של הפרוטוקול. * לא ניתן לקרוא תיקיות שבשמם יש רווח כמו "file new"

DELETE

כדי להריץ את הפקודה delete צריך לרשום את הפקודה באותיות קטנות ואת המיקום של הקובץ אותו מוחקים. לדוגמא:

delete C:\Users\Innon\Downloads\a.txt 10.0.0.5

תחזיר הפקודה C:\Users\Innon\Downloads\a.txt was deleted

COPY

כדי להריץ את הפקודה copy צריך לרשום את הפקודה באותיות קטנות ואת המיקום של הקובץ שממנו רוצים להעתיק את התוכן ומיקום של קובץ שאליו רוצים להעתיק את המידע. לדוגמה:

copy C:\Users\Innon\Downloads\a.txt C:\Users\Innon\Downloads\b.txt 10.0.0.5

הפקודה מעתיקה את התוכן של הקובץ הראשון אל הקובץ השני. אם הקובץ השני לא קיים היא תצור קובץ חדש עם התוכן של הקובץ הראשון

הפקודה תחזיר
C:\Users\Innon\Downloads\a.txt was copied to C:\Users\Innon\Downloads\b.txt

EXECUTE

כדי להריץ את הפקודה execute צריך לרשום את הפקודה באותיות קטנות ואת המיקום של הקובץ שאותו מריצים. לדוגמה:

execute C:\WINDOWS\system32\notepad.exe -free

הפקודה פותחת הליך שאחראי על הרצת הקובץ, בזמן שהקובץ פתוח ניתן להמשיך לשלוח פקודות.

תחזיר הפקודה C:\WINDOWS\system32\notepad.exe was executed

TAKE_SCREENSHOT

כדי להריץ את הפקודה take_screenshot צריך לרשום את הפקודה באותיות קטנות וחשוב לרשום את הקו התחתון בין שני המילים. לדוגמה:

take_screenshot -free

הפקודה מצלמת צילום מסך אצל הלקוח ושומרת את התמונה במקום שהוצב .

הפקודה תחזיר

המקום * screen shot have been taken and been saved at C:\Users\Innon\Pictures\Screenshots\screenShot.jpg

PATH המוחזר הוא בהתאם ל למקום שבו נוצרה התיקייה.

SEND_PHOTO

כדי להריץ את הפקודה send_photo צריך לרשום את הפקודה באותיות קטנות וחשוב לרשום את הקו התחתון בין שני המילים. לדוגמה:

send_photo -free

לפקודה זו שני חלקים אחד אצל השרת ואחד אצל הלקוח. הלקוח שולח לשרת את הגודל ואת המידע של התמונה והשרת מקבל את המידע ושומר אותו בקובץ חדש

הפקודה תחזיר img was received successfully אם הכל עבר חלק ואם המידע לא עבר טוב היא תחזיר img data wasn't sent correctly

<u>הפקודות שאינן מצריכות שום תוסף:</u>

HELP

את הפקודה רושמים באותיות קטנות. לדוגמא

help

הפקודה מחזירה את כל הפקודות שניתן לשלוח וגם את כל התוספים הקיימים במסד הנתוני שאליהם ניתן לפנות.

STATE

את הפקודה רושמים באותיות קטנות. לדוגמא

state

הפקודה תחזיר את כל התוספים שיש במסד ואת כל כתובות ה ip שיש להם.

EXIT

כדי להריץ את הפקודה exit צריך לרשום את הפקודה באותיות קטנות. לא להוסיף תוסף מאחר

והפקודה לא נשלחת ללקוח . לדוגמה:

exit

הפקודה סוגרת את התקשורת בין התוכנה של המשתמש לשרת

הפקודה תחזיר user closes connection

מסד הנתונים:

כדי לשמור את הנתונים בצורה פשוטה ויעילה בחרתי להשתמש ב mongodb.
מסד זה שומר מידע לפי אוספים ולכן הוא מאוד יעיל עבור הפרוייקט. הדרך שבה **אני** שומר את המידע היא כזאת:

mydatabase.collection:


```
{
  "_id": {
    "$oid": "627e4a9e2d8cf74797e97255"
  },
  "ip": "10.0.0.10"
}
```

```
{
  "_id": {
    "$oid": "628655ac092fbfc71f3b353a"
  },
  "ip": "10.0.0.12"
}
```

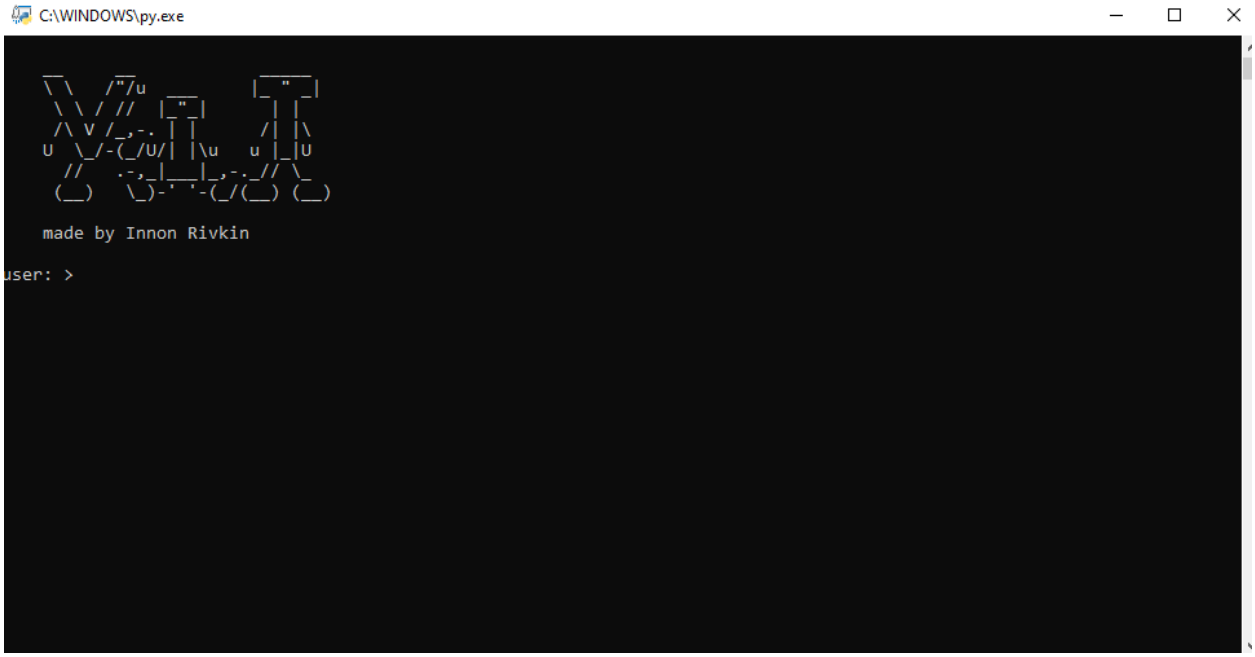

במסד אני שומר כל כתובת ip ב query משלה. ה query דומה למילון ובו אנו יכולים לשמור מידע לפי מפתחות. לכל אובייקט query יש id שהוא תעודת הזהות של האובייקט הייחודית רק לו, אך אני לא עושה שימוש בו מאחר ואני כמעט ולא מתעסק עם מידע ספציפי אל עם אוספים.

כל פעם שלקוח חדש מתחבר אני שומר את הכתובת שלו בתוך אוסף שנקרא free-, אוסף זה מכיל את כל כתובות ה ip במסד ובמקרים שבהם אנו רוצים לפנו לכתובת ספציפית אנו בודקים אם היא קיימת באוסף הזה.

4. ממשק משתמש

כאן יופיע פירוט ממשקי המשתמשים וכן תמונות שלהם. אני גם אוסיף הסבר על הממשק הגרפי ועל איך להשתמש בו.

תחילה, ממשק המשתמש נמצא במחשב של השרת שהוא המחשב של איש ה IT , הממשק הראשי הוא מסוג COMAND LINE ובוא נרשמים הסקריפים. להלן דוגמה של הרצה של פקודת find_path (שעובדת):



C:\WINDOWS\py.exe

```
   __     __           _____
   \ \   /"/u  ___     |_ " _|
    \ \ / //  |_"_|      | |
    /\ V /_,-. | |      /| |\
   U  \_/-(_/U/| |\u   u |_|U
     //   .-,_|___|_,-._// \_
    (__)   \_)-' '-(_/(__) (__)

   made by Innon Rivkin

user: >  find_path -free
```

ui_class.py - Shortcut

```
   __     __           _____
   \ \   /"/u  ___     |_ " _|
    \ \ / //  |_"_|      | |
    /\ V /_,-. | |      /| |\
   U  \_/-(_/U/| |\u   u |_|U
     //   .-,_|___|_,-._// \_
    (__)   \_)-' '-(_/(__) (__)

   made by Innon Rivkin

user: >  find_path -free
'10.0.0.14': 'C:\Users\Innon\Desktop\big_project\src'
'10.0.0.16': 'C:\Users\innon\OneDrive\Desktop\big_project\src'


user: >
```

ui_class.py - Shortcut

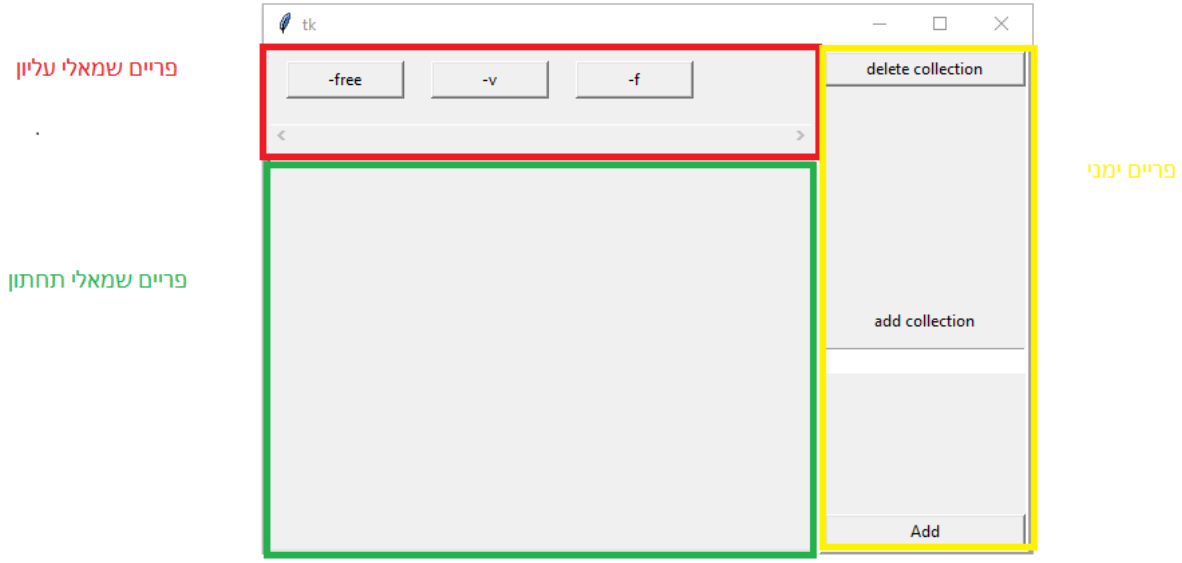
```
made by Innon Rivkin

user: >  find_path -free
'10.0.0.14': 'C:\Users\Innon\Desktop\big_project\src'
'10.0.0.16': 'C:\Users\innon\OneDrive\Desktop\big_project\src'


user: >  exit
exiting
```

הסבר מפורט על המערכת הגרפית לעריכת מסד הנתונים

אז ראשית אסביר איך המערכת בנויה. אז החלון הראשי (root window) מחולק לשלושה פריימים ראשיים:

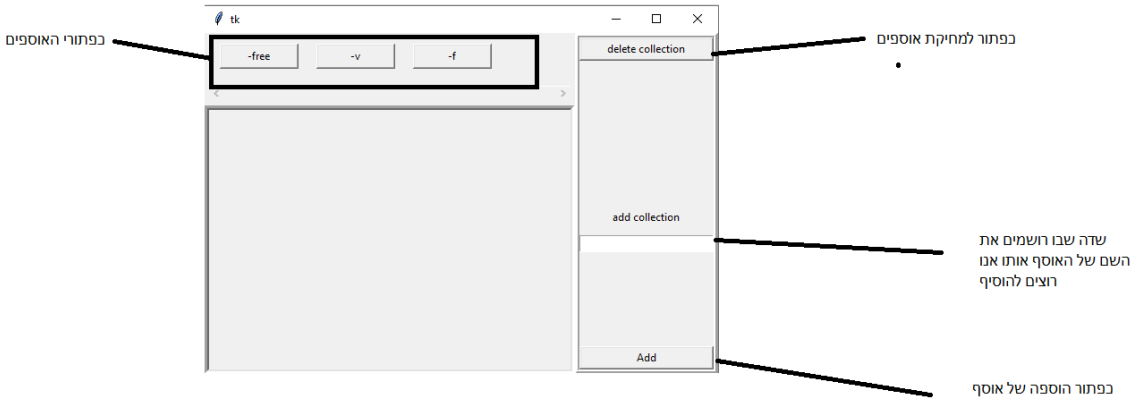


חלוקה זו נועדה כדי לתחום את החלון ולאפשר הצבה של פריימים מסויימים בתוך התחומים האלה (לפי הצורך).

בעיקרון מה כל פריים יש לו תפקיד. בפריים הימני מופיעים רק אפשרויות לעריכה של אוספים והוא משתנה במקרה שכפתור של אוסף נלחץ. הפריים השמאלי עליון אינו משתנה כמעט, רק במקרה שנוסף כפתור חדש אז מופיע שם עוד כפתור. הפריים שמאלי תחתון או שלא מופיע כלום או במקרה שכפתור של אוסף נלחץ אז מופיע שם התוכן של האוסף.

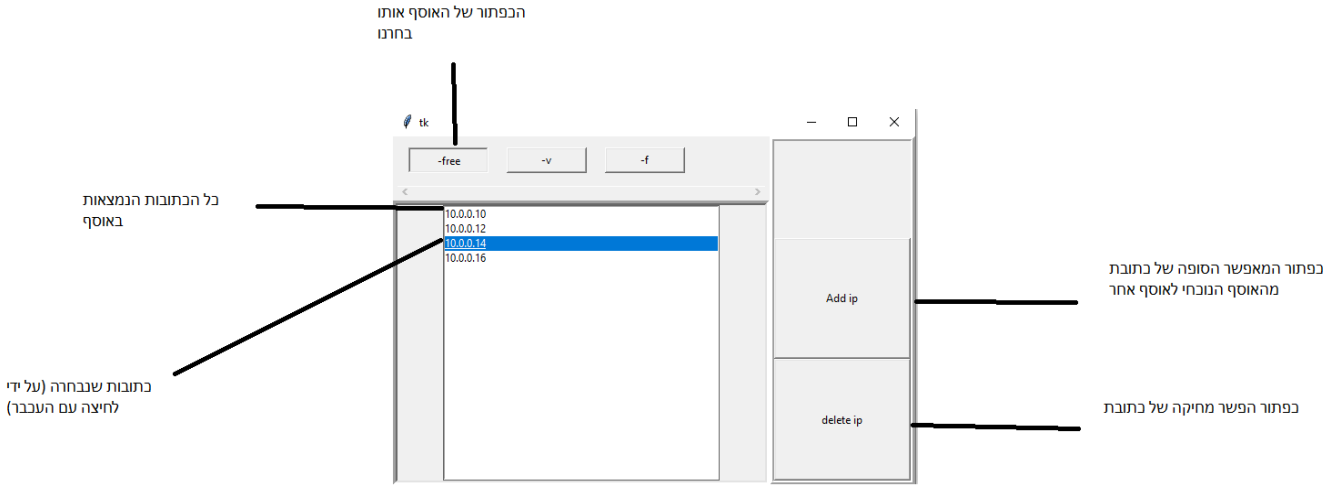
קיימים שתי מצבים לחלון הראשי :

מצב עריכת האוספים:
מצב זה מאפשר מחיקה והוספה של אוספים חדשים.אנו רואים את מצב זה כאשר אף כפתור של אוסף לא לחוץ. תמונה עם פירוט:

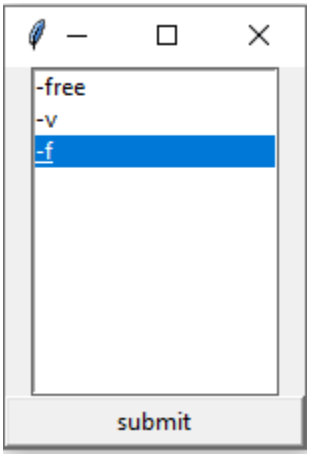


מצב עריכה של כתובות (מחיקה הוספה)

מצב זה מאפשר מחיקה או הוספה של כתובות מתוך אוספים. במצב זה אנו רואים את כל הכתובו הנמצאות באוסף שבחרנו. כדי לראות את מצב זה אנו צריכים ללחוץ על כפתור אוסף.



במקרה שצריך לבחור באוסף מסוים (למשל כשאנחנו רוצים למחוק אוסף או להעביר כתובת מאוסף לאוסף) החלונית הבאה תופיעה:



בחלונית זו ניתן לראות את הרשימה של כל האוספים הקיימים וכפתור submit. מתוך הרשימה של האוספים המשתמש צריך לבחור באוסף כלשהו על ידי לחיצה עם העכבר על האוסף (כפי שניתן לראות בתמונה האוסף "f-" נבחר) ולאחר בחרת האוסף יש ללחוץ על כפתור submit. לאחר לחיצה על הכפתור החלונית תיסגר. במיקרה שבו אנו מתחרטים צריך לבחור ב "free-" ולשלוח

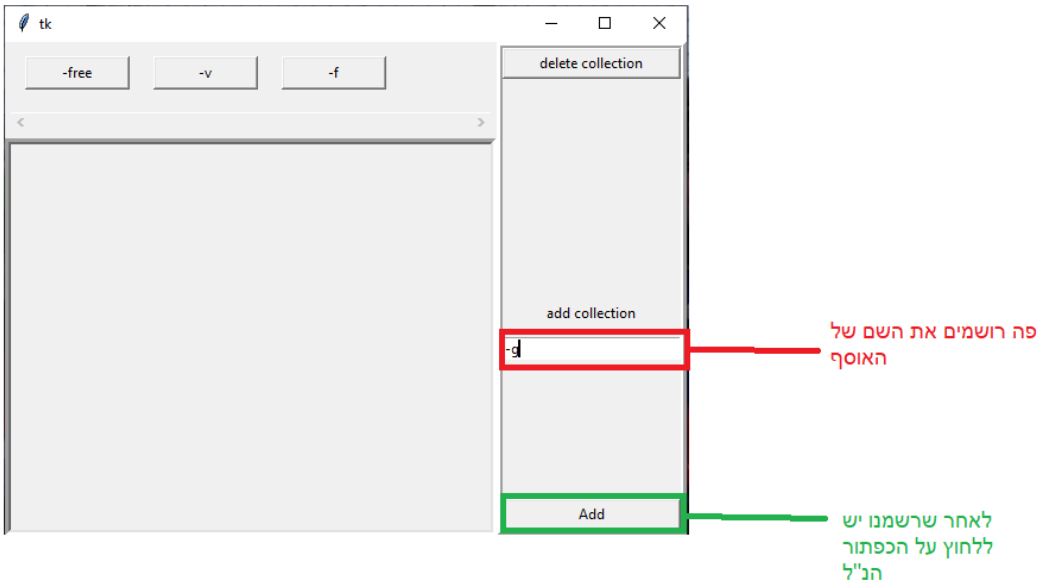
5. נספחים

מדריך למשתמש במערכת הגרפית לעריכת מסד הנתונים:

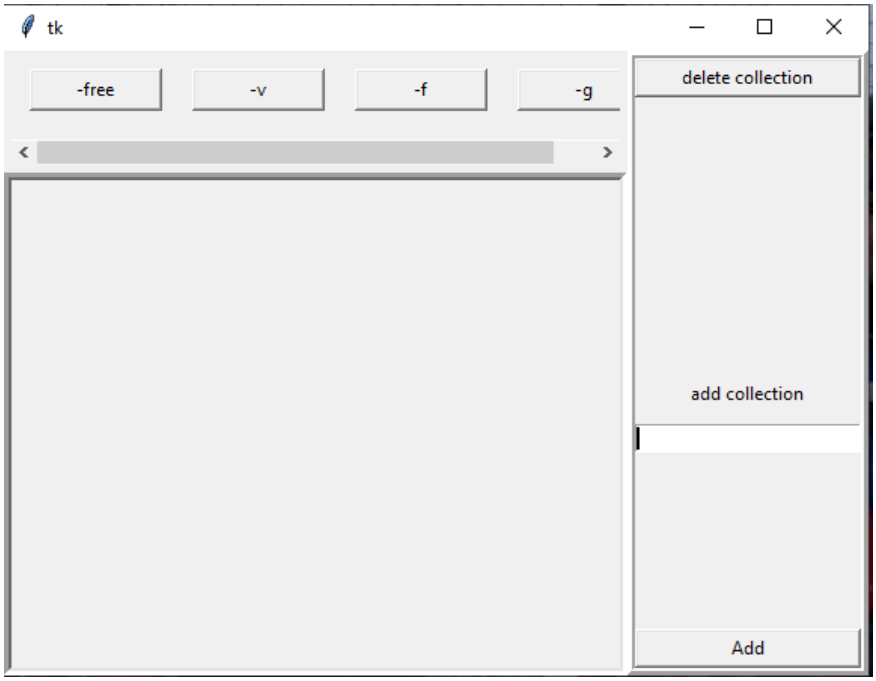
הוספה של אוסף:

יש לרשום את השם של האוסף אותו אנו רוצים להוסיף בתיבת המלל שנמצאת מתחת לכיתוב "add collection". רצוי ששם זה יהיה קצר והוא חייב להתחיל עם "-" אחרת הוא לא יקלט, אסור שיהיה בו רווחים בין מילים או אותיות מאחר וזה עלול לעשות בעיות. אוסף חדש לא נשמר במסד הנתונים כל עוד הוא ריק.

המחשה:



התוצאה:

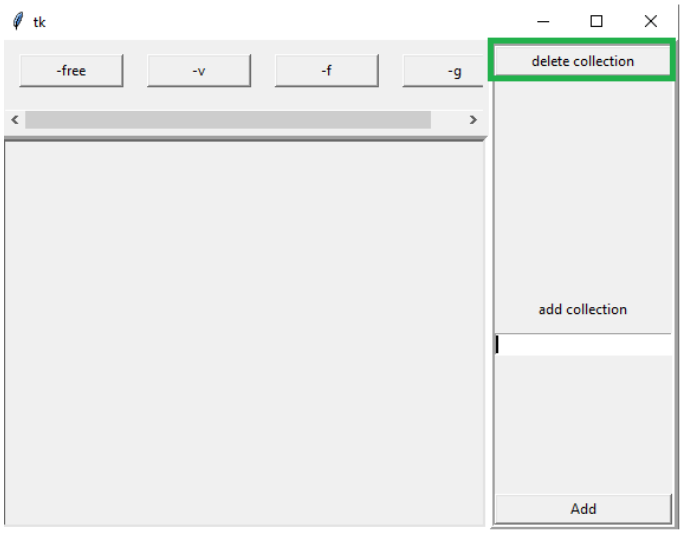


שימו לב שנוצר כפתור בשם "g-" ושנוצר לנו פס גלילה כדי שיהיה ניתן להגיע אליו. פס הגלילה הוא שימושי במקרה שיש הרבה כפתורים.

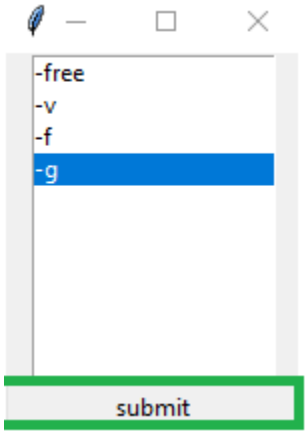
מחיקה של אוסף:

דוגמא ויזואלית למחיקה של האוסף "g-" עם הסברים.

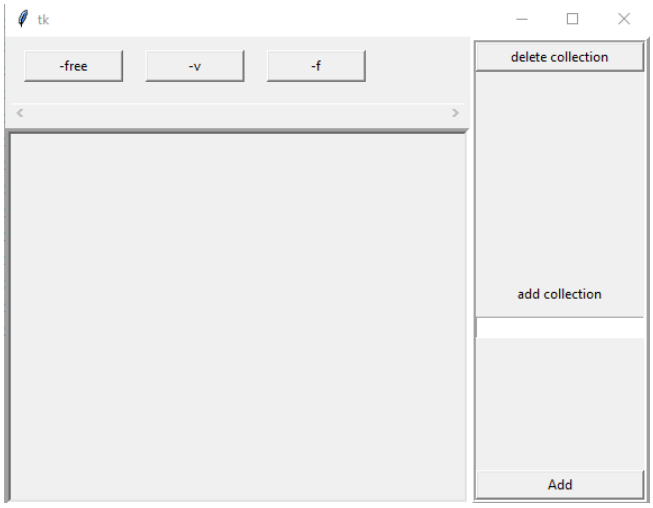
1.לחיצה על הכפתור "delete collection":



2. החלונית לבחירת אוסף מופיעה, בוחרים אוסף ולוחצים על submit:



. החלונית לבחירת אוספים נסגרת והכפתור של האוסף "g-" נמחק:

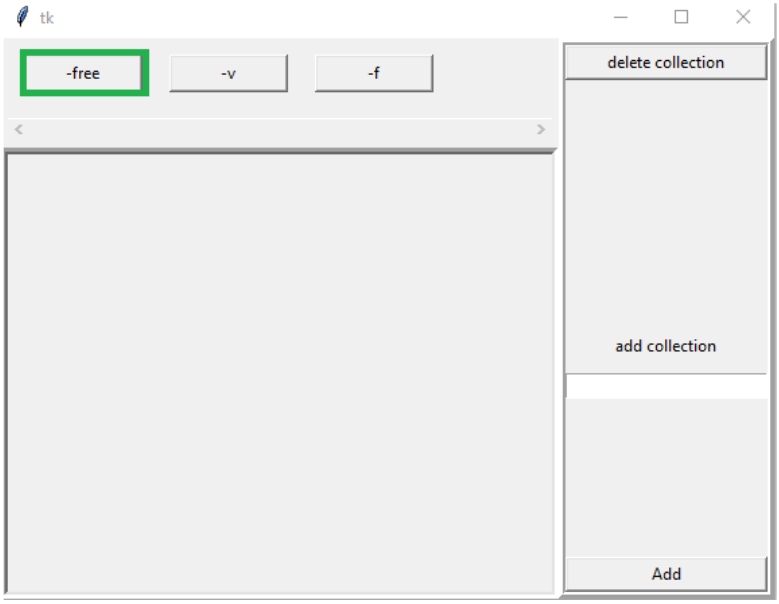


**** לא ניתן למחוק את האוסף "free-" מאחר והוא תמיד צריך להיות קיים.***

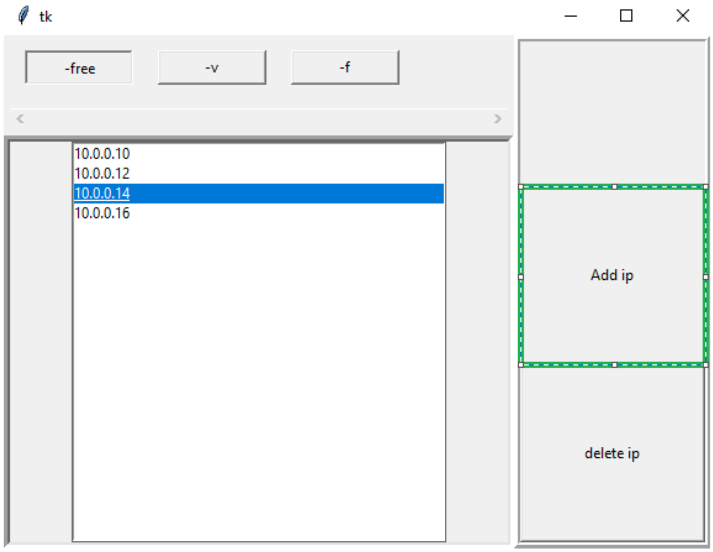
הוספה של כתובת ip מאוסף לאוסף:

בדוגמא אקח כתובת ip מהאוסף "-free" ואוסיף אותה ל "-f".

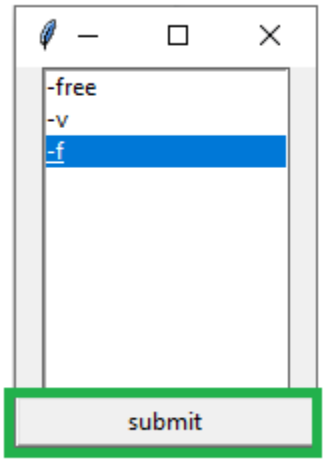
1. יש לבחור באוסף בו יש את הכתובת אותה אנו רוצים להוסיף לאוסף אחר :



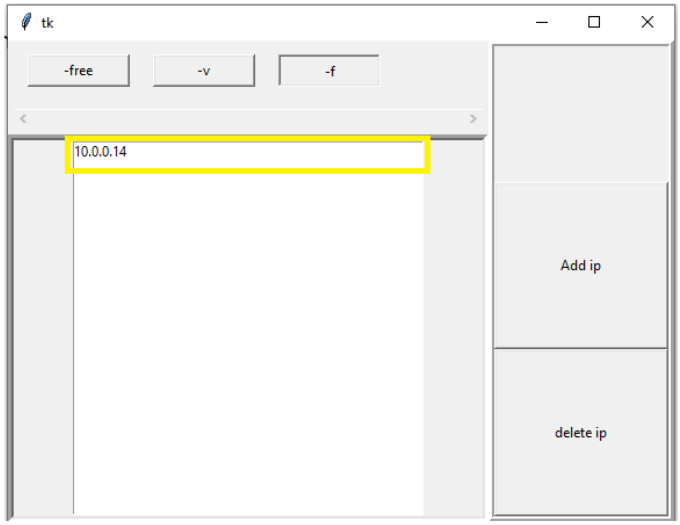
2. עכשיו אנו רואים את המצב לעריכת כתובות. יש לבחור בכתובת אותה אנו רוצים להוסיף לאוסף אחר וללחוץ על כפתור "Add ip":



3. לאחר לחיצה על כפתור "Add ip" תופיע חלונית לבחירת אוסף. יש לבחור את האוסף הרצוי על ידי לחיצה עם העכבר ולאחר מכן ללחוץ על כפתור submit:



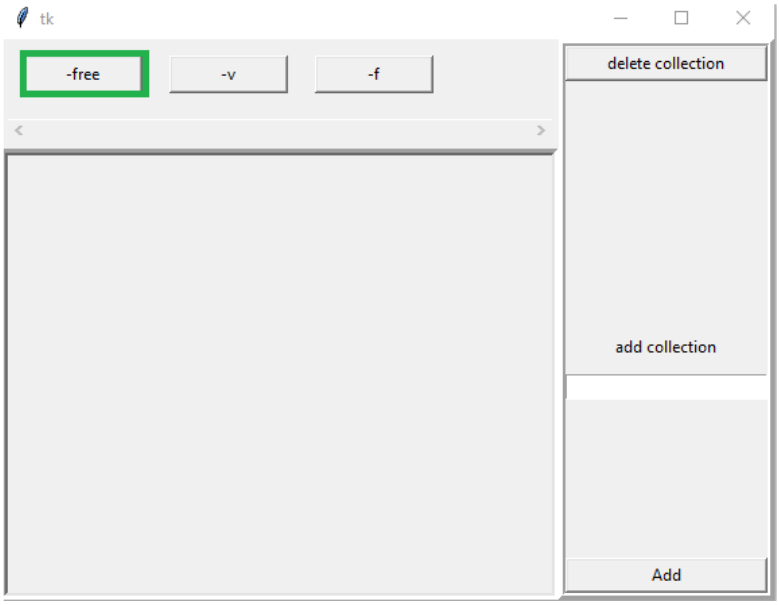
4. לאחר הלחיצה על כפתור "submit", החלונית תעלם ונראה את כתובת הip שבחרנו מופיעה באוסף שבחרנו:



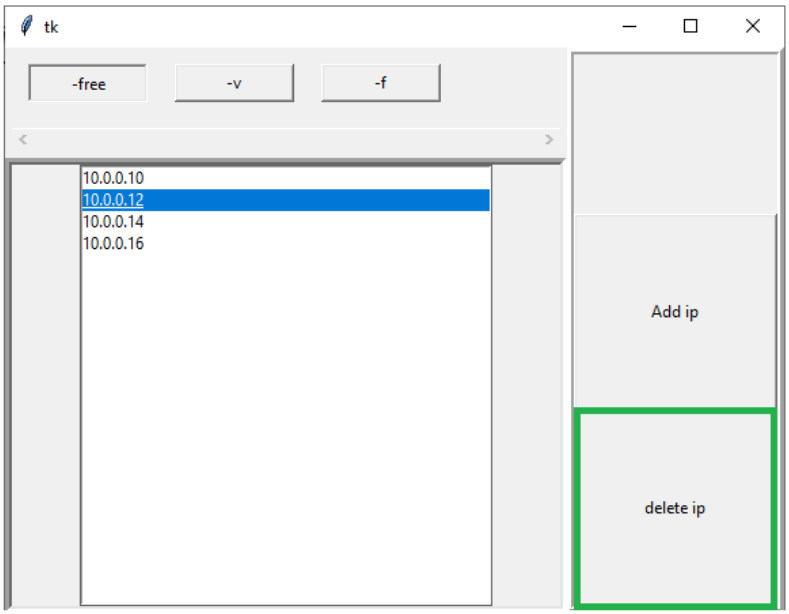
מחיקת כתובת ip:

**** חשוב לדעת שמחיקה של כתובת מהאוסף "-free" תמחק את אותה כתובת מכל האוספים. מאחר ובאוסף זה חייבים להיות כל הכתובות. בשאר האוספים זה ימחוק את הכתובת מאותו אוסף בלבד***

1. לחיצה על האוסף ממנו אנו רוצים למחוק כתובת ip:



2. בחירת כתובת ה ip אותה אנו רוצים למחוק ולחיצה על כפתור "delete ip":



3. לאחר לחיצה על כפתור "delete ip". הכתובת אותה בחרנו תימחק ולא נראה אותה באוסף: