בית הספר הרב תחומי
רשת תיכונים ומכללות מקבוצת עמל

שם: שלי בן ציון

ת.ז.: 327889606

שם המנחה: גולן מור

חלופה: הגנת סייבר ומערכות הפעלה

יוני 2023

תוכן עניינים

תקציר ורציונל הפרויקט

בכל ארגון שבו מחשבים קיים מנהל מערכת (System Administrator). תפקידו להיות אחראי על התחזוקה והתפקוד השוטף של מערכת המחשבים בארגון. לעקוב אחרי כל כך הרבה פרמטרים בכל כך הרבה מחשבים יכולה להיות עבודה קשה, ועל כן המוצר שלי, "NETVIGILANT", בא לעזור למנהל המערכת לעקוב אחרי הפעילות של כל המחשבים ברשת.

למנהל המערכת תהיה האפשרות לראות איזה מחשב במערכת דלוק ואיזה לא. עבור המחשבים הכבויים תהיה לו האפשרות להדליק אותם דרך המחשב שלו. עבור המחשבים הדלוקים תהיה לו האפשרות לצפות במידע חשוב על הביצועים, הרשת והחומרה של אותם מחשבים. המוצר יזין לתוך מסד נתונים את חלק מהמידע וינתח אותו. בנוסף הוא יתריע למנהל המערכת במקרה וחלק מהמידע חורג מתווך מוגדר מראש. שירות נוסף שהמוצר ייתן הדגרה של תהליכים שאסור שירוצו על המחשבים במערכת, והתרעה על ריצה שלהם במקרה והם רצים.
על כן קהל היעד של המוצר הוא מנהלי מערכת למניהם.

מדוע בחרתי בפרויקט הזה?

בחרתי לעשות את הפרויקט הזה כפרויקט הגמר שלי כחלק ממגמת הנדסת תוכנה מכמה סיבות.

- רציתי לעסוק במשהו שישלב הרבה אלמנטים שונים כך שאוכל ללמוד ולהתנסות בהרבה דברים בדרך. בשביל הפרויקט שלי השתמשתי למשל בתקשורת, התממשקתי עם מערכת ההפעלה, רכשתי ידע על החומרה של המחשב ועוד. מאוד שמחתי להזדמנות ללמוד תוך כדי עשיה, "ללכלך את הידיים" ולבנות משהו שמיש.
- רציתי ליצור כלי שיוכל לשמש ארגונים ולהקל על ניהול מערכת המחשבים שלהם.

מבוא ורקע כללי

כל חברה בימינו עובדת עם מחשבים רבים. בלי אותם מחשבים, החברה לא הייתה יכולה להתקיים כי בעזרתם עושים הכל – מבצעים פעולות, מתקשרים עם שאר העולם, שומרים מידע ועוד. ולכן לכל חברה יש בעל תפקיד "מנהל מערכת". יש למנהל המערכת עבודה מרובה. עליו:

- לוודא שכל המחשבים עובדים בצורה תקינה.
- מידי פעם לעדכן גרסאות של כל המחשבים והתוכנות בהם.
- לבצע הגדרות והתקנות של ציוד.
- לעקוב אחרי ביצועי המערכת.
- להבטיח שכל המחשבים עובדים קשורה.

ועוד הרבה.

אם נדמיין חברה גדולה, עם מלא מחשבים ובניין ענק עם כמה קומות, אנו מבינים שהעבודה שלו לא קלה. יש לו הרבה אחריות והוא חייב לעקוב אחרי תפקוד כל המחשבים ברשת תמיד. תפקוד לקוי מצידו יכול להוביל להשלכות שישפיעו על החברה כולה.

לכן הפרויקט שלי בא לעזור למנהל הרשת להצליח לתפקד בצורה יותר יעילה ויותר פשוטה בשבילו. NetVigilant – עוזר למנהל המערכת תמיד להיות דרוך. בעזרתו הוא יכול לפקח אחרי התפקוד של המחשבים השונים עליהם הוא אחראי ברשת. יתר על כן, הוא יכול גם לקבל התרעות במקרה ומשהו חריג. כך הרבה יותר קל לו לעקוב אחרי פעילות הרשת ולדעת כשקיימות בעיות.

מטרת הפרויקט

מה המוצר המוגמר אמור לבצע:

המוצר אמור לעזור למנהל מערכת מחשבים לנהל את המערכת בצורה יעילה ונוחה. אצלו במחשב יהיה השרת, איתו הוא יוכל לראות מי כל המחשבים ברשת והאם הם דלוקים.
עבור כל המחשבים הדלוקים הוא יוכל לראות מידע שימושי כגון הביצועים של כל מחשב, מידע על – התרעות במקרה והנתונים – טמפרטורה של הcpu, שימוש cpu, או זיכרון עוברים רף מסוים אשר הוא מגדיר. יתר על כן, הוא יוכל להגדיר "forbidden processes", תהליכים שאסור שירוצו על המחשבים ברשת, ויקבל התרעה ואפשרות לראות את רשימת התהליכים הרצים על אותו המחשב ומידע עליהם, במקרה ואותם תהליכים רצים.
עבור כל הלקוחות התוכנה תרוץ ברקע.

דרישות מרכזיות:

דרישות פונקציונליות:

- המוצר יאפשר למנהל המערכת לראות את כל המחשבים ברשת הפנימית.
- עבור כל מחשב דולק, מנהל המערכת יוכל לראות את המידע הבא:
  - כתובת ip וכתובת mac.
  - מידע על החומרה של המחשב.
  - מידע על הביצועים של המחשב.
  - מידע על התהליכים הרצים על המחשב.
  - מידע על הusers במחשב.
  - מידע על הnetwork interface.
  - מידע על הפורטים במחשב.
  - Network information.
  - מידע על הdrives במחשב.
- המוצר יתריע למנהל המערכת בעזרת חלון pop up והודעה למייל כאשר אחד מהנתונים – טמפרטורה של הcpu, שימוש הcpu או זיכרון עברו רף מסוים.
- המוצר יאפשר למנהל המערכת להגדיר תהליכים שאסור שירוצו על המחשבים ברשת ויתריע לו בעזרת חלון pop up והודעה למייל במקרה ואחד מהתהליכים המוגדרים כאסורים רצים.
- עבור כל מחשב כבוי, מנהל המערכת יהיה מסוגל להדליק אותו מרחוק.

דרישות לא פונקציונליות:

- ממשק אינטואיטיבי וקל לעבודה עבור מנהל המערכת.
- יהיה מספר לא מוגבל של לקוחות ברשת.
- המידע שיתקבל על המחשבים יהיה מדויק ומהימן.
- השרת והלקוח יפעלו בצורה אוטומטית כשהמחשב נדלק.

- עבור הלקוחות, המוצר יעבוד לגמרי בקרע.

אילוצים:

- המוצר יעבוד רק על windows 10 או 11.
- עבור כל מחשב ברשת יהיה צריך להגדיר את קוד הלקוח אינדיבידואלית.

תרחישים במערכת:

1. שימוש ראשוני – שרת:

- מנהל המערכת יצטרך לעדכן את כתובת המייל שלו בקובץ בשם admin_info.txt.

2. שימוש ראשוני – לקוח:

- על מנהל המערכת להגדיר בקבצי client_info.txt את כתובת הip של המחשב עליו רץ השרת, את רף הנתונים cpu, memory וtempraturе, ויצטרך להגדיר שם תהליכים שאסור שירוצו על המחשב.

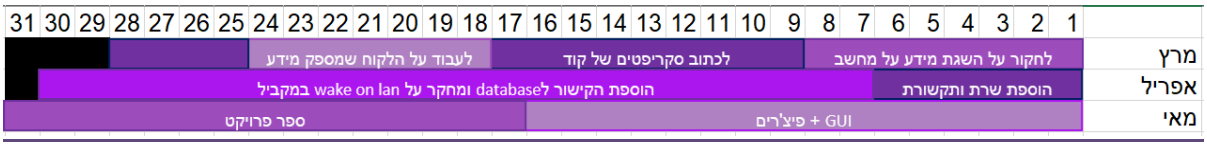
3. שימוש שוטף – שרת:

- מנהל המערכת יראה בGUI את כל המחשבים ברשת.
- הוא יהיה מסוגל לצפות במידע עבור כל מחשב –
  - מידע על החומרה
  - מידע על הרשת
  - מידע על הדיסקים
  - מידע על המשתמשים
  - מידע על הפורטים הפתוחים
- יהיה מסוגל להדליק מרחוק מחשבים כבויים על ידי לחיצה ימנית עליהם.
- יהיה מסוגל להגדיר תהליכים שאסור שירוצו על המחשבים ברשת.
- יקבל התרעת pop up והתראה במייל במקרה ואחד המחשבים עבר את הרף שמנהל המערכת הגדיר בקובץ של הcpu או הזיכרון או הטמפרטורה. בנוסף יוכל לקבל קובץ שמראה בתוכו את כל התהליכים הרצים על המחשב ומידע עליהם.

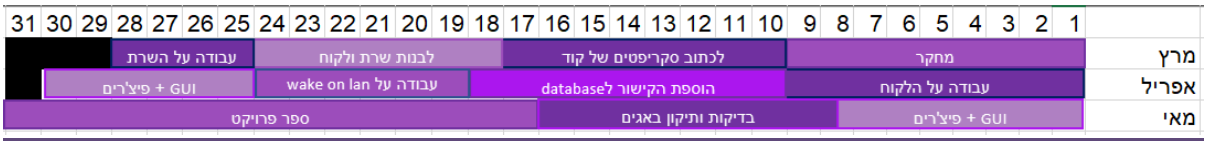
4. שימוש שוטף – לקוח:

- עבור הלקוח הכל יקרה לגמרי ברקע.

לו"ז משוער



לו"ז בפועל



שפת התכנות וסביבת העבודה

Python -

שפת התכנות העיקרית שהשתמשתי בה כדי לכתוב את הפרויקט היא python 3.11, תוך שימוש בסביבת הרצה של Visual Studio Code.
Python היא שפת תנות עילית המאפשרת תכנות מונחה עצמים, משפות התכנות הפופולאריות ביותר שקיימות. מאוד נוח לעבוד איתה בעיקר בגלל התמיכה הרחבה שלה במספר רב של ספריות ומודלים שונים. בחרתי בשפה הזו בגלל הנוחות שבכתיבה וקריאה שלה, ובגלל התמיכה שלה בספריות ומודלים שעזרו לי לבנות את הפרויקט. איתה כתבתי את כל הפרויקט – את התקשורת בין השרת והלקוח, את השגת המידע על כל לקוח ברשת, ואת ממשק המשתמש.

SQL –

בשביל לנהל את מסד הנתונים שלי השתמשתי בספריה של python בשם SQLite. הספרייה הזו מאפשרת להריץ פקודות בSQL בתוך קוד בpython כדי לנהל מסד נתונים. כאמור, SQL היא שפה בה אפשר להשתמש כדי לעבד מידע במסדי נתונים, המאפשרת שליפה והוספה של רשומות ונתונים לטבלאות, יצירה של טבלאות ושינוין. השתמשתי בה כדי לעדכן את מסד הנתונים שלי בעת כניסה של לקוח חדש/עדכון סטטוס החיבור של לקוח ישן, ובשביל לשמור את המידע שכל לקוח שולח לשרת במסד הנתונים.

ניסוח וניתוח הבעיה האלגוריתמית

בעיה ראשונה:

הדרך להשיג מידע על המחשבים – כחלק מהפרויקט שלי הייתי צריכה למצוא דרך להשיג מידע שונה אודות הביצועים, החומרה, הרשת ועוד, עבור כל מחשב במערכת.

בעיה שניה:

לאפשר כמה clients במקביל – בגלל שהשרת שלי צריך להיות מחובר לכל הלקוחות ברשת במקביל, הייתי צריכה למצוא דרך לאפשר זאת.

בעיה שלישית:

עדכון הdatabase – עבדתי עם sqlite3 כדי לעדכן את הdatabase. הייתי צריכה לעדכן אותו מהמון threads שונים, בהתחשב בזה שלכל client במערכת אני יוצרת thread לעדכונים שוטפים של בדיקות תקינות. sqlite3 לא יכול לעבוד עם אותו חיבור לdatabase מthread שונה מהthread בו הוא נוצר, ולכן היה עלי למצוא פתרון לעדכון הdatabase מthreads שונים.

בעיה רביעית:

ריצה אוטומטית – עבור השרת והלקוחות, המוצר שלי אמור להתחיל לרוץ אוטומטית ברגע שהמחשב נדלק. כך הלקוחות מתחברים ישר כשהם יכולים לשרת ומנהל המערכת יכול לראות את כל המחשבים הדלוקים ברשת. הייתי צריכה למצוא דרך לאפשר זאת.

תיאור אלגוריתמים קיימים

פתרונות קיימים לבעיה ראשונה:

Python היא שפה שתומכת בספריות רבות, לכן היו הרבה פתרונות לבעיה זו.

1. Psutil (python system and process utilities)

זוהי ספריה בpython שיכולה לספק מידע על תהליכים שרצים על המחשב ועל הביצועים שלו, למשל cpu, memory, disks, network, sensors ועוד. הספרייה הזו מאוד שימושית לבדיקת הביצועים של המחשב וניהול הריצה של תהליכים במחשב. הספרייה אכן תומכת גם בwindows אך חלק מהפונקציות שלה לא תומכות בו.

הספרייה לא מגיעה אוטומטית עם python ולכן יש להתקין אותה באמצעות הרצת הפקודה הבאה בcmd:

```
pip install psutil
```

הספרייה יכולה למשל לספק את שימוש הcpu בעזרת הפונקציה psutil.cpu_times() אך לאחר שניסיתי אותה כמה פעמים ובכמה דרכים שונות, שמתי לב שהמידע שהיא מחזירה הוא לא קונסיסטנטי, לא תואם למציאות והרבה פעמים היא מחזירה פשוט 0.0.

חוץ מזה, היה לי חשוב להשיג מידע על החיישנים של המחשב בעיקר כדי לגלות את טמפרטורת המחשב – מידע שחשוב שמנהל מערכת ייחשף אליו כדי לוודא שטווח הטמפרטורות של המחשב

תקין. לעומת זאת, כל מה שקשור בחיישני טמפרטורה וחיישני מאוורר למשל בספריה psutil, לא תומך בwindows, אלא בlinux.

2. WMI (Windows Management Instrumentation)

זהו מודל שעוטף את WMI ומאפשר לנו להשתמש ביכולות שלו. WMI הוא יישום של Microsoft למודל שמאפשר קבלה של כמעט כל פיסת מידע מכל מחשב אם הוא רץ עם ההרשאות המתאמות. כך אנחנו מקבלים דרך לגשת ולנהל מידע ומשאבי מערכת שונים. בעזרתו אנחנו יכולים להתממשק בpython למערכת ההפעלה (בין היתר) ולקבל מידע שונה עליה, למשל, מידע על המערכת (חומרה, גרסת מערכת ההפעלה ועוד), מידע על המשתמשים ואבטחה (שמות הusers של המחשב, סיסמאות ועוד), מידע על תהליכים שרצים על המחשב, ועוד המון מידע שימושי כשמדובר על פיקוח על רשת.

גם את הספרייה הזו יש להתקין באמצעות הרצת הפקודה הבאה בcmd:

```
Pip install wmi
```

3. Open hardware monitor

זוהי תוכנה open source שדרכה אפשר לראות מידע שונה על תפקוד המחשב, למשל, חישני טמפרטורה, מידע על הcpu והgpu, חומרה, וולטים ועוד. אפשר לראות דרכה את טמפרטורת הcpu בעזרת חיישני הטמפרטורה שנמצאים במעבדי intel וADM, שתי החברות הנפוצות בעולם למעבדים. התוכנה חינמית ואכן תומכת בwindows.

כאשר מורידים אותה מהאתר, מקבלים תיקייה דחוסה שבה נמצא קובץ הexe אך גם נמצא קובץ dll. אפשר להשתמש בdll כדי להשיג את המידע הנחוץ לנו בpython.

כדי להשתמש בdll בתוך הקוד, אפשר להשתמש בספריה wmi (Windows Management Instrumentation) של python. הספרייה מספקת ממשק של מערכת ההפעלה שדרכו ניתן להשיג מידע על רכיבים שונים. במקום להתממשק אליה נוכל להתממשק אל התוכנה כך:

```
w = wmi.WMI(namespace="root\OpenHardwareMonitor")
```

וכך לגשת למידע הנדרש.

כשניסיתי את הדרך הזו ראיתי שהמידע המסופק אכן תואם למציאות וקונסיסטנטי. הדבר היחיד שלא הסתדר הוא שכאשר התוכנה הייתה סגורה, חלק מהערכים המוחזרים היו None תמיד. לכן יש צורך לפתוח את התוכנה ברקע וכך להשתמש במידע שבה.

התוכנה כן יחסית ישנה, העדכון האחרון שלה היה ב2020, אבל היא עונה על חלק מהצרכים בצורה טובה.

פתרונות קיימים לבעיה שניה:

אני יוצרת שרת tcp בפרויקט תוך שימוש בספרייה "socket" של python. מכיוון ואני צריכה שכמה מחשבים יתחברו במקביל לשרת שלי, הייתי צריכה לבחור בדרך אחת מהדרכים הבאות כדי לאפשר זאת:

1. Asyncio

ספרייה לכתיבת קוד במקביל בעזרת פקודות של await/async. הספרייה משמשת כבסיס למספר מסגרות אסינכרוניות של Python המספקות רשתות ושרתי אינטרנט בעלי ביצועים גבוהים, ספריות חיבור למסד נתונים, ועוד. בעזרתה אנחנו נמנעים משימוש בthreads מרובים אך היא מורכבת יותר ליישום ולניפוי באגים בהשוואה לפתרונות אחרים הקיימים.

2. Select

מודל שמאפשר גישה לפונקציות select() ו()pull שזמינות ברוב מערכות ההפעלה. כך אנו מנהלים רשימה של לקוחות פעילים ומפקחים על הפעולות שלהם בעזרת הפונקציות האלו. כשלקוח למשל שולח מידע או כשמתקבל חיבור חדש, הsocket המתאים הופך להיות מוכן, ואפשר לטפל בו בהתאם. כך אנחנו נמנעים מיצירת threads מרובים והגישה הזו יותר חסכנית במשאבים, אך יותר קשה לניהול ותחזוקה.

3. Threading

בעזרת מודל זה אנחנו יכולים להקנות לכל לקוח thread משלו. כאשר מתקבל חיבור חדש, נוצר thread חדש אשר מטפל באותו הלקוח. הגישה הזו בהחלט נוחה ומאפשרת חיבור של כמה לקוחות במקביל, אבל כאשר מדובר על מספר רב של לקוחות היא יכולה להיות "בזבזנית" במשאבי המערכת.

פתרונות קיימים לבעיה שלישית:

כדי להשתמש בספרייה sqlite3 לצורך ניהול database, עלינו ליצור connection לdatabase, וממנו ליצור cursor שמאפשר לנו להריץ פקודות בSQL כדי לשלוף נתונים ולהכניס אותם לטבלאות השונות בdatabase. את הconnection והcursor לא ניתן להעביר בין threads וחייבים להשתמש בהם רק בthread בו הם נוצרו, sqlite לא תומך בmultithreading. מכיוון ואני משתמשת בmultithreading ויוצרת thread חדש עבור כל לקוח שמתחבר לשרת, הייתי צריכה למצוא דרך לנהל זאת תוך עמידה במגבלות של הספרייה ושמירה על הערכים של clean code.

פתרונות:

1. הפתרון הראשון שחשבתי עליו הוא ליצור connection וcursor חדש עבור כל מתודה בשרת שצריכה לגשת לdatabase. כך אני לא עוברת על המגבלות של הספרייה ובכל מקום שצריכה להיות הידברות עם הdatabase נוצר חיבור משלו. אמנם השיטה הזו מובילה לקוד מסורבל ולא נוח לעבודה איתו.

2. הפתרון השני עליו חשבתי הוא יצירה של thread שכל האחריות שלו היא עדכון של הdatabase. יצרתי רשימה גלובלית שאליה כל thread יכול היה להוסיף את מה שהיה צריך. כמובן שבשביל

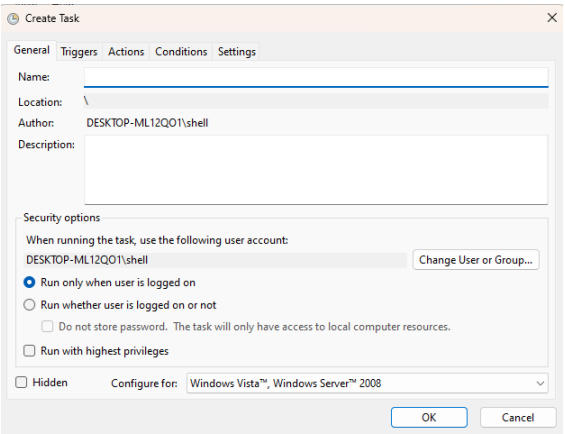
זה אני השתמשתי בlock, דרך להבטיח שברגע שthread מנסה לגשת לרשימה ולשנות אותה אף thread אחר לא יכול יהיה להפריע לו באמצע, באמצעות כך שאני עושה למנעול acquire לפני הכתיבה וrelease אחריה. הthread שאחראי על עדכון הdatabase קורא מהרשימה בצורה של FIFO (first in first out) כדי שהמשימות יתבצעו בסדר שבה נכנסו לרשימה, ולפי המידע ששלף הוא מעדכן את הdatabase. כך יש לי רק חיבור אחד לdatabase שקיים באופן קבוע.

3. הפתרון השלישי עליו חשבתי הוא ליצור מחלקה שכל תפקידה היא ניהול הdatabase. היא תאתחל אותו ואת הטבלאות בו במידה ולא אותחלו כבר, והיא תהיה אחראית על הוספה ועדכון המידע בו. כך אפשר להכיל בתוך השרת instance של המחלקה, ולהשתמש בה מכל thread בעזרת lock.

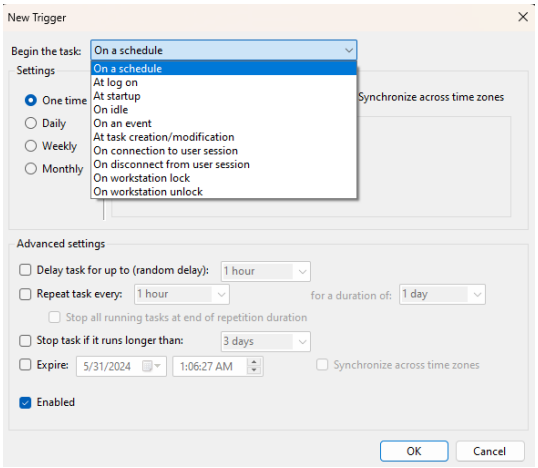
<u>פתרונות קיימים לבעיה רביעית:</u>

1. Task scheduler

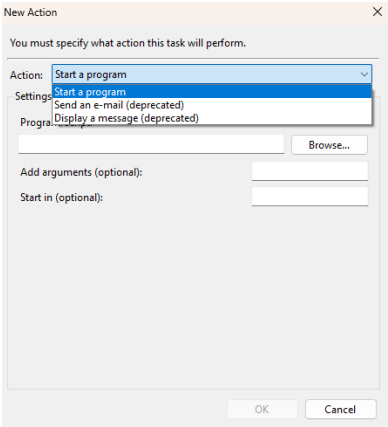
זהו כלי של windows שמאפשר להגדיר tasks – משימות שעל המחשב לבצע ומתי עליו לבצע אותן. הוא יכול למשל להפעיל אפליקציה, לשלוח הודעת מייל או להציג הודעה על המחשב.
כאשר פותחים אותו על המחשב (נמצא אוטומטית במחשבים בעלי מערכת הפעלה windows) אפשר ללחוץ על הכפתור Create task ובכך ליצור משימה חדשה.
לאחר מכן נפתח חלון בו אפשר להגדיר את שם המשימה ולשים לה תיאור. בנוסף אפשר לתת לה פריווילגיות של admin ולהגדיר את המשימה כמוסתרת (hidden).



לאחר מכן ניתן לעבור לחלק של Triggers ולהגיד מתי המשימה צריכה לקרות:



עכשיו נגדיר מה המשימה בactions:



אפשר גם להוסיף תנאים (בחלון conditions) ולשנות את ההגדרות (בחלון settings).
כך ניתן להגדיר משימה שתתחיל את הקוד התוכנית כאשר המחשב נדלק. חשוב גם לתת למשימה את הפריווילגיות הגבוהות ביותר, כי הלקוחות והשרת שבניתי דורשים פריווילגיות גבוהות כדי לגשת למידע על המחשב.

2. Registry

זהו מאגר המרכזי של ההגדרות המשמשות את מערכת ההפעלה. הוא מכיל בתוכו הגדרות ממאפיינים של מערכת ההפעלה ונתונים על חומרה ועד מסך הרקע של שולחן העבודה. כדי שנוכל לגרום לתוכנה שלנו לרוץ נצטרך ליצור שני קבצים:
קובץ batch – קובץ שמכיל שורות טקסט של פקודות למערכת ההפעלה כדי להפעיל תוכניות שונות. השורות הללו לעיתים זהות לחלוטין לפקודות שנותנים למערכת ההפעלה דרך הshell.
קבצי batch נמצאים בשימוש נרחב למטרות שונות, כולל ניהול מערכת, התקנת תוכנה, פעולות גיבוי ושחזור ומשימות חוזרות. הם מספקים דרך פשוטה ויעילה לבצע אוטומציה של משימות במערכות Windows ללא צורך בידע מתקדם בתכנות.
בתוך קובץ הbatch צריך לכתוב פקודה דומה לזו שבshell כדי להריץ את התוכנית:

```
python ***path***
```

קובץ vbs – קובץ סקריפט שנכתב בשפת התכנות VBScript, שפותחה על ידי מיקרוסופט. קבצי VBS מספקים דרך גמישה ועוצמתית להפוך משימות לאוטומטיות במערכות Windows, המאפשרות למשתמשים ליצור סקריפטים מותאמים אישית לביצוע מגוון רחב של פעולות ביעילות.
קובץ הvbs הוא הקובץ שמפעיל את קובץ הbatch וגורם לו לרוץ ברקע בצורה אוטומטית.
לאחר שיצרנו את הקבצים וכתבנו בהם את הנדרש, צריך להגיע לpath הבא בregistry:
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
ובו ליצור string value חדש. בvalue data שלו נשים את הpath המלא לקובץ הvbs שיצרנו.
לאחר מכן התוכנית שלנו תרוץ באופן אוטומטי כאשר המחשב נדלק.

הפתרון הנבחר

פתרון בעיה ראשונה:

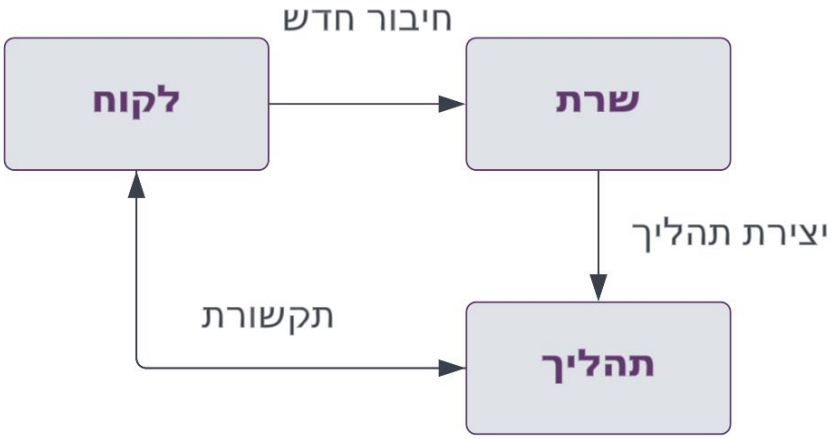
בשביל לקבל מידע אודות המחשב בclients בחרתי להשתמש בכל הפתרונות.

יצרתי מודל בשם snmp_server.py ובתוכו כתבתי פעולות שנותנות לי מידע שונה אודות המחשב. קראתי לו כך מכיוון וSNMP הוא פרוטוקול לניהול רשת (Simple Network Management Protocol) שמשמש בעיקר רשתות TCP, הוא מספק אמצעי לניתור ובקרה של התקני רשת, איסוף סטטיסטיקות, ביצועים ואבטחה ברשת.

בתוכו יש לי כמה פונקציות וכל אחת משתמשת בחלק מפתרון אחר. בגלל ששמתי לב שכל אחד מהפתרונות מספק לי מידע לא מושלם, החלטתי לשלב את כולם לפתרון אחד ולהשתמש בכל הספריות שציינתי כדי לאסוף מידע שרלוונטי לי.

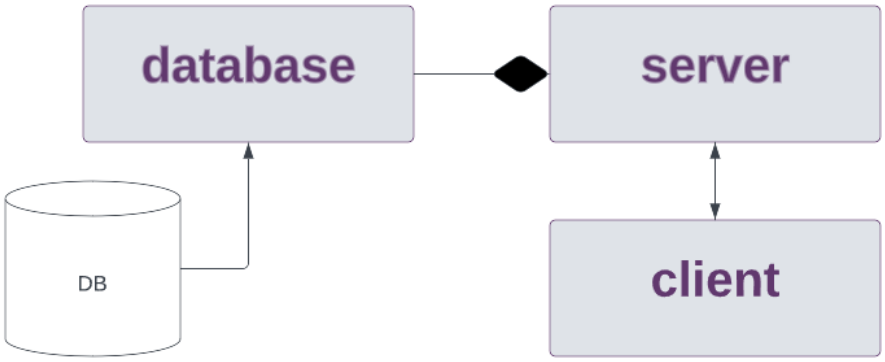
פתרון בעיה שניה:

הפתרון שבחרתי להשתמש בו הוא שימוש בthreading. בחרתי בפתרון זה הן בגלל הנוחות והפשטות שלו ביחד לשאר הפתרונות שמצאתי, והן בגלל ההכרות וההתעסקות הקודמת שלי עם הספרייה במסגרות שונות. אמנם השיטה הזו היא יותר "בזבזנית", אך היא יותר נוחה לתחזוקה ולעבודה איתה.



פתרון בעיה שלישית:

בשביל לעדכן ולהוסיף נתונים לdatabase בחרתי בפתרון השלישי. ניסיתי לבנות את הפרויקט שלי לפי עקרונות של clean code ולכן בחרתי בפתרון זה. Clean code מתאר את הדרך שלנו לכתוב קוד נקי שקל לעבוד איתו. ישנם הרבה עקרונות כדי לעמוד בסטנדרטים של clean code, ואחד מהם הוא לנסות שלכל אלמנט בקוד תהיה מטרה אחת. למשל, כל פונקציה עושה משהו אחד בלבד. לכן החלטתי לחלק את העבודה של השרת למחלקה שמטפלת בdatabase ומחלקה שמטפלת בתקשורת, לכל אחד תפקיד מוגדר וברור.

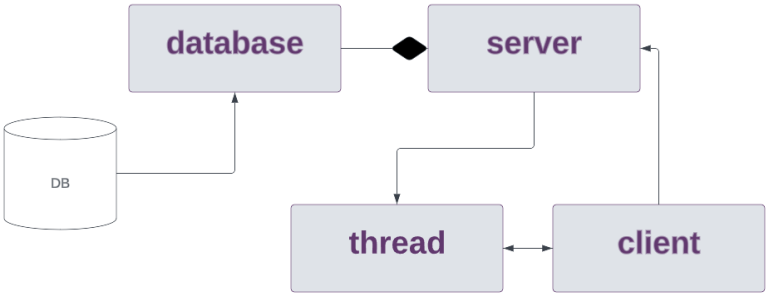


פתרון בעיה רביעית :

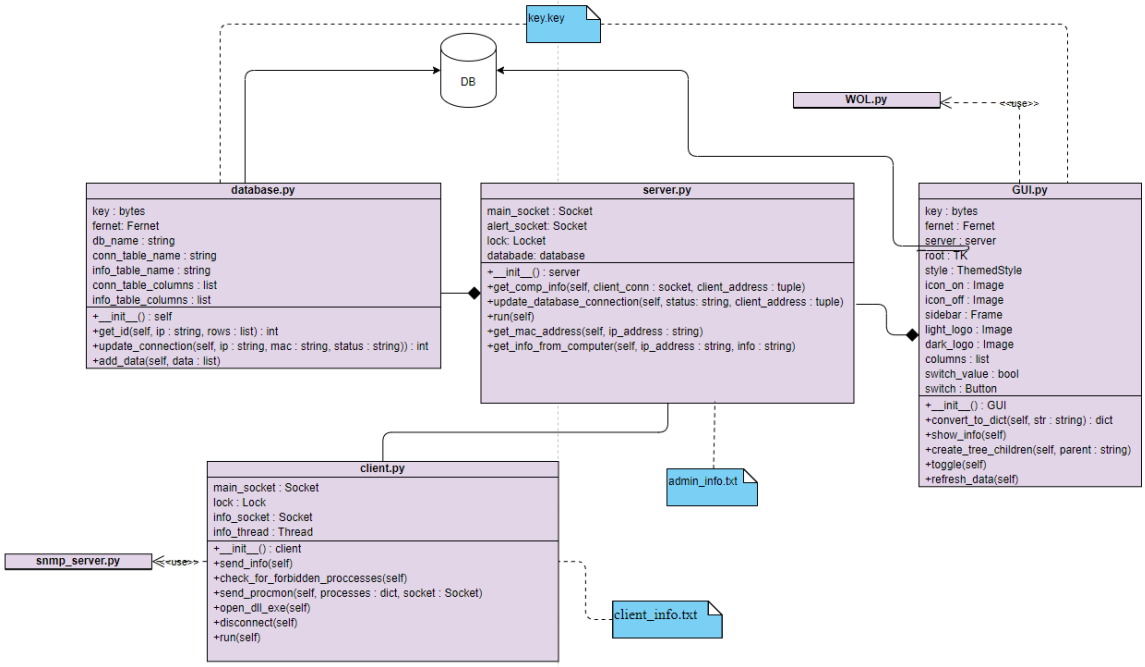
לפתרון הבעיה הרביעית בחרתי להשתמש בדרך השנייה. Task scheduler מאפשר גמישות רבה יותר ואפשרויות לתזמון משימות, כולל הפעלה שלהם עם הרשאות. שימוש בtask scheduler מבטיח פעילות מהימנה ומאפשר לשנות קונפיגורציות במקרה הצורך.

יתר על כך, עבודה עם הtask scheduler מבטיחה שלא יבוצעו שינויים לא רצויים בregistry. כל שינוי קטן בregistry עלול לפגוע רבות ביציבות המערכת ולכן כדאי להיות מאוד זהירים בעבודה איתה.

מבנה המערכת :

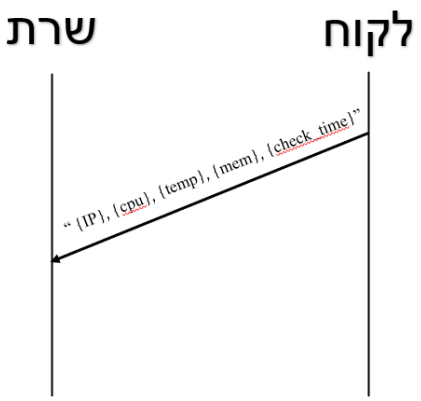


מבנה Classes :



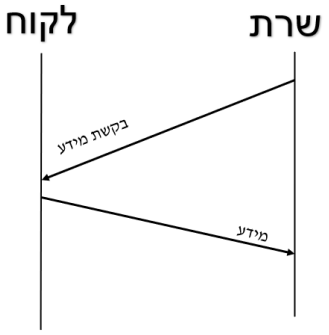
פרוטוקול תקשורת :

א. עדכון שוטף של cpu, memory וtemprature דרך thread מיוחד של עדכונים. כל 30 שניות כל לקוח שולח לשרת את העדכון, והשרת מביא את המידע למחלקה database.py שמוסיפה אותו לdatabase.



ב. כל 30 שניות מתבעצת גם בדיקה על כל הprocesses הרצים על המחשב האם אחד מהם נמצא ברשימה של processes שאסור שירוצו על המחשב. במקרה והקוד זיהה process שמוגדר כאסור, זאת התקשורת :

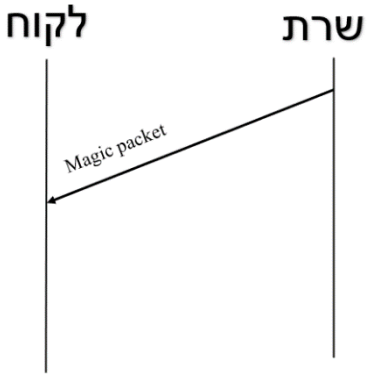
בקשות מידע שונות מהלקוח:



– WOL

Wake on lan - הפעלת מחשב מרחוק

טכנולוגיה שמאפשרת להפעיל (״להעיר״) מחשב כבוי מרחוק על ידי שליחת ״magic packet״ (פקטה מיוחדת). זה יעבוד רק אם המחשב תומך בפרוטוקול בהגדרות הBIOS שלו. עקרון הפעולה הוא שגם כאשר המחשב כבוי ישאר חשמל שיגיע לכרטיס הרשת, וברגע שהוא יזהה שהוא קיבל ״פקטת קסם״, הוא יעיר את המחשב.



יש לציין שללקוח איך דרך לאיים על האתר/ המערכת. המידע הרגיש בdatabase מוצפן באמצעות הצפנה סימטרית (כתובות הip והmac) ולא עובר בין השרת ללקוחות מידע רגיש. הפרויקט כן יודע להתמודד עם קריסה של אחד הצדדים בתקשורת בעזרת שימוש בtry וexcept. חוץ מזה אין דרך לאיים על המערכת כי היא לא מתעסקת במידע רגיש.

פיתוח הפתרון בשכלול הקוד עם שפת התכנות

קבלת מידע על המחשב

בניתי מודל שיאפשר לי להגיד מידע שונה אודות המחשב. בשביל לקבל את המידע השתמשתי בספריות שונות – psutil ו wmi, בנוסף השתמשתי ב open hardware monitor.

פונקציה שמחזירה את הזכרון:

```
def get_virtual_mem():
    # return virtual mem precentage
    return psutil.virtual_memory().percent
```

פונקציה שמחזירה מידע על הdrives:

```
def get_drives_info():
    drives_info = []

    # list of all drives names
    drives = [disk.device for disk in psutil.disk_partitions()]

    # return info for each drive in dict
    for drive in drives:
        disk_info = psutil.disk_usage(drive)
        drives_info.append({
            'name' : drive,
            'total' : str(bytes_to_GB(disk_info.total)) + 'GB',
            'used' : str(bytes_to_GB(disk_info.used)) + 'GB',
            'free ' : str(bytes_to_GB(disk_info.free)) + 'GB',
            'percent' : disk_info.percent,
        })

    return drives_info
```

פונקציה שמחזירה מידע על הרשת:

```
'''
bytes_sent: number of bytes sent
bytes_recv: number of bytes received
packets_sent: number of packets sent
packets_recv: number of packets received
errin: total number of errors while receiving
errout: total number of errors while sending
dropin: total number of incoming packets which were dropped
dropout: total number of outgoing packets which were dropped
'''
def get_network_info():
    info = psutil.net_io_counters()
    return {
        'bytes_sent' : info.bytes_sent,
        'bytes_recv' : info.bytes_recv,
        'packets_sent' : info.packets_sent,
        'packets_recv' : info.packets_recv,
```

```
    'errors_sending' : info.errout,
    'errors_reciving' : info.errin,
    'packets_dropped_sending' : info.dropout,
    'packets_dropped_reciving' : info.dropin,
  }
```

פונקציה שמחזירה מידע על החיבורים הפתוחים שעל פרוטי המחשב:

```
'''
family: the address family, either AF_INET, AF_INET6 or AF_UNIX.
type: the address type, either SOCK_STREAM, SOCK_DGRAM or SOCK_SEQPACKET.
laddr: the local address as a (ip, port) named tuple.
raddr: the remote address as a (ip, port) named tuple.
status: represents the status of a TCP connection.
pid: the PID of the process which opened the socket.
'''
def get_connections_info():
  info = psutil.net_connections()
  connections_info = []
  for connection in info:
    connections_info.append(
      {
        'family' : connection.family,
        'type' : connection.type,
        'local address' : connection.laddr,
        'remote address' : connection.raddr,
        'status' : connection.status,
        'pid' : connection.pid
      }
    )
  return connections_info
```

פונקציה שמחזירה מידע על הסוללה. אם רץ על לפטופ, נקבל את אחוז הסוללה, הערכה של כמות השניות עד שתגמר הסוללה, והאם החשמל מחובר. במקרה שרץ על מחשב נייח, יחזיר None.

```
'''
psutil.sensors_battery() returns:
percent: battery power left as a percentage.
secsleft: approximation of the seconds left before the battery runs. If power is connected, returns -
psutil.POWER_TIME_UNLIMITED. If can't be determined, returns -
psutil.POWER_TIME_UNKNOWN.
power_plugged: True if the AC power cable is connected, False if not or None if it can't be determined.
'''
def get_battery_info():
  return psutil.sensors_battery()
```

פונקציה שמחזירה את שמות הusers הקיימים על המחשב:

```
def get_users_info():
  users_info = psutil.users()
  users_names = []
# for each user add its name to the list
```

```
    for user in users_info:
        users_names.append(user[0])
    return users_names
```

פונקציה פנימית שמחזירה מידע מתוך open hardware monitor עם שימוש בספריה wmi :

```
def __get_info(s_type, s_name):
    # connect to openHardwareMonitor
    w = wmi.WMI(namespace="root\OpenHardwareMonitor")
    # get the computer sensors
    sensors = w.Sensor()
    for sensor in sensors:
        if sensor.SensorType==s_type:
            if sensor.Name == s_name:
                # return the value according to the type and name given
                return sensor.Value
```

פונקציה שמשתמשת בפונקציה הפנימית ומחזירה את טמפרטורת הcpu :

```
def get_cpu_temp():
    # returns the CPU package temp
    return __get_info('Temperature', 'CPU Package')
```

פונקציה שמשתמשת בפונקציה הפנימית ומחזירה את הcpu :

```
def get_cpu():
    # returns the total CPU load
    return __get_info('Load', 'CPU Total')
```

פונקציה שמחזירה את טמפרטורת הGPU :

```
def get_gpu_temp():
    # returns the GPU core temp
    return __get_info('Temperature', 'GPU Core')
```

פונקציה שמחזירה את הGPU :

```
def get_gpu():
    # returns the GPU core load
    return __get_info('Load', 'GPU Core')
```

שימוש בספריה wmi כדי לקבל מידע על המעבד :

```
def get_processor():
    pc = wmi.WMI()
    # returns the processor name
    return pc.Win32_Processor()[0].Name.strip()
```

שימוש בספריה wmi כדי לקבל מידע על הגרסה של מערכת ההפעלה :

```
def get_os():
    pc = wmi.WMI()
    os_info = pc.Win32_OperatingSystem()
    # returns the name of the operating system
    return os_info[0].Name
```

קיבוץ והחזרה של נתונים שונים שנוגעים לחומרה כמילון :

```
def get_hardware_info():
    d = { }
    # processor type
    d['Processor'] = get_processor()
    # os type and version
    d['OS'] = get_os()
    # cpu usage
    d['CPU'] = get_cpu()
    # cpu package temprature
    d['CPU temp'] = get_cpu_temp()
    # gpu usage
    d['GPU'] = get_gpu()
    # gpu core temprature
    d['GPU temp'] = get_gpu_temp()
    # virtual memory
    d['Memory'] = get_virtual_mem()
    # battery information, None for desktop computer
    d['Battery'] = get_battery_info()
    return d
```

פעולה שמחזירה רשימה של מילונים שמכילים בתוכם מידע על כל התהליכים הרצים על המחשב. הפעולה משתמשת ב()process.oneshot , פעולה שאיתה לא מתבצעות קריאות נפרדות עבור כל תכונה של התהליך – מהיר ויעיל יותר. המידע אותו אוספים על כל thread הוא –

- ID של התהליך (במקרה ו0 ממשיכים הלאה, תהליכים עם ID 0 הם בדרך כלל תהליכים שאחראיים על תזמונים ועל swaps, לא תורמים לנו.
- שם התהליך
- הזמן בו הוא נוצר (במקרה ולא ניתן לקבל משתמשים בboot time)
- מספר הליבות שבהן הוא יכול לעבוד (במקרה ולא ניתן לגשת, 0)
- שימוש הcpu שלו
- הstatus שלו
- העדיפות שלו, niceness (במקרה ולא ניתן לגשת, 0)
- שימוש הזיכרון שלו (במקרה ולא ניתן לגשת, 0)
- מספר הthreads שלו
- המשתמש שגרם לו לרוץ(במקרה ולא ניתן לגשת, N/A כלומר מערכת ההפעלה גרמה לתהליך הזה לרוץ)

```
def get_processes_info():
  # the list to contain all process dictionaries
  processes = []
  for process in psutil.process_iter():
    # get all process info in one shot (more efficient, without making separate calls for each attribute)
    with process.oneshot():
```

```
        # get the process id
        pid = process.pid

        if pid == 0:
            # Swapper or sched process, useless to see
            continue

        # get the name of the file executed
        name = process.name()

        # get the time the process was spawned
        try:
            create_time = datetime.fromtimestamp(process.create_time())
        except OSError:
            # system processes, using boot time instead
            create_time = datetime.fromtimestamp(psutil.boot_time())

        try:
            # get the number of CPU cores that can execute this process
            cores = len(process.cpu_affinity())
        except psutil.AccessDenied:
            cores = 0

        # get the CPU usage percentage
        cpu_usage = process.cpu_percent()

        # get the status of the process (running, idle, etc.)
        status = process.status()

        try:
            # get the process "niceness" (priority)
            nice = int(process.nice())
        except psutil.AccessDenied:
            nice = 0

        try:
            # get the memory usage in mbytes
            memory_usage = process.memory_full_info().uss / 1000000
        except psutil.AccessDenied:
            memory_usage = 0

        #number of threads the process has
        n_threads = process.num_threads()

        # get the username of user spawned the process
        try:
            username = process.username()
        except psutil.AccessDenied:
    # os created this process
```

```
            username = "N/A"
        processes.append({
        'pid': pid, 'name': name, 'create_time': create_time,
        'cores': cores, 'cpu_usage': cpu_usage, 'status': status, 'nice': nice,
        'memory_usage': memory_usage, 'n_threads': n_threads, 'username': username,
        })
    return processes
```

איפשור של חיבור של כמה לקוחות במקביל

לגבי הבעיה השניה, השתמשתי בthreading ועבור כל לקוח חדש שהתחבר לשרת פתחתי thread דרכו הוא שלח עדכונים של הcpu, טמפרטורה וזיכרון בשוטף. בתוך השרת:

```
def run(self):
    while True:
        # accept a new client - main socket
        conn, addr = self.main_socket.accept()
        # accept a new client - alert socket
        alert_conn, alert_addr = self.alert_socket.accept()

        # put the socket in a global dict to ask for information
        self.lock.acquire()
        global COMPUTERS
        COMPUTERS[addr[0]] = conn
        self.lock.release()

        # thread that supplies client info
        info_thread = threading.Thread(
            target=self.get_comp_info,
            args=(
                alert_conn,
                alert_addr,
            ),
        )

        # start the thread to get information about the client's cpu' temp and memory
        info_thread.start()
```

בלקוח:

```
def __init__(self):
    # create main socket
    self.main_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # client tries to connect the server
    connection_established = False
    while not connection_established:
        # incase the server is not up yet
        try:
            self.main_socket.connect((IP, MAIN_PORT))
            connection_established = True
        except Exception as e:
            # try again
            pass


    # connect the socket responsible for sending the info for the database
    self.info_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.info_socket.connect((IP, ALERT_PORT))
```

```
# run the thread in the background to send the info every 30 seconds
self.info_thread = threading.Thread(target=self.send_info)
self.info_thread.start()
```

כך אני יוצרת עבור כל לקוח שמתחבר לשרת שני sockets שונים על שני פורטים שונים. אחד מהם הוא לתקשורת הרגילה בניהם – כשאר השרת מבקש מידע מהלקוח הוא מספר לו אותו. השני הוא בשביל לשלוח עדכונים שוטפים לשרת לגבי הביצועים של המחשב, והשמה שלהם מצד השרת בdatabase. עבור כל לקוח אני שומרת את הsocket המרכזי שלו במילון כאשר המפתחות הן כתובות הip. כך אני יכולה לשלוף מהמילון את הsocket במקרה ואני רוצה לבקש מידע ממחשב מסוים שמחובר למחשב של הadmin. כך גם אני מקבלת עדכונים שוטפים בdatabase לגבי הביצועים של כל מחשב.

חיבור הקוד עם הdatabase

המחלקה database מתממשקת עם מסד הנתונים ודרכה ניתן לעדכן ולהוסיף נתונים לטבלאות במסד.
המחלקה server, מכילה אובייקט של המחלקה הזו, וכך יכולה לערוך את מסד הנתונים בעת הצורך.
בנוסף, הצפנתי את העמודות בטבלת החיבורים שמכילות את כתובות הip והmac, כדי שאנשים שיש להם גישה למסד הנתונים לא ישתמשו בהם לרעה. ההצפנה שהשתמשתי בה היא הצפנה סימטרית – יצרתי מפתח שאותו שמרתי בקובץ key.key, ואת ההצפנה והפיענוח אני במצעת באמצאות Fernet, פעולה בpython שמאפשרת שימוש בהצפנה סימטרית.
פעולת הבנאי במחלקה database, שמאתחלת את גם מסד הנתונים (למקרה ולא אותחל כבר לפני):

```
def __init__(self):
    # key
    file = open('key.key', 'rb') # rb = read bytes
    self.key  = file.read()
    file.close()
    self.fernet = Fernet(self.key)

    # create connection and cursor for the db
    self.db_name = "ipconnections.db"
    db_conn = sqlite3.connect(self.db_name)
    db_cursor = db_conn.cursor()

    # tables names
    self.conn_table_name = "Connections"
    self.info_table_name = "CompInfo"

    # columns for the table that describes the different computers on the net
    self.conn_table_columns = [
        "id",
        "ip_address",
        "mac_address",
        "connection_status",
    ]

    # columns for the table that saves the computer's updates on their performance
    self.info_table_columns = [
        "id",
        "cpu",
        "temperature",
        "memory",
        "check_time"
    ]

    # create connections table
    db_cursor.execute(
        f"""CREATE TABLE IF NOT EXISTS {self.conn_table_name} (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            ip_address TEXT NOT NULL,
            mac_address TEXT NOT NULL,
```

```
            connection_status TEXT NOT NULL)"""
    )

    # create computers info table
    db_cursor.execute(
      f"""CREATE TABLE IF NOT EXISTS {self.info_table_name} (
            id INTEGER NOT NULL,
            cpu FLOAT NOT NULL,
            temperature FLOAT NOT NULL,
            memory FLOAT NOT NULL,
            check_time TIMESTAMP not null,
            FOREIGN KEY (id) REFERENCES ip_addresses (id))"""
    )

    # commit the changes and close the connection
    db_conn.commit()
    db_conn.close()
```

שתי הטבלאות שלי – Connections, CompInfo, מקושרות אחת לשניה בעזרת הID, שהוא המפתח הראשי בטבלה Connections. היחד הוא יחס של אחד לרבים כי בעוד בטבלת החיבורים יופיע כל ID פעם אחד בלבד, בטבלה של המידע הוא יופיע כמה פעמים מכיוון וכל מחשב מבצע כמה בדיקות. לכן כדי להתנהל עם טבלת המידע, בניתי פעולה שממירה בין ip נתון לid שלו בטבלת החיבורים:

```
def get_id(self, ip, rows):
    for row in rows:
        decrypted_ip = self.fernet.decrypt(row[1]).decode()
        if decrypted_ip == ip:
            return row[0]
```

פעולה שמעדכנת את טבלת החיבורים במקרה של:

- חיבור של לקוח חדש
- חיבור של לקוח ישן
- ניתוק של לקוח

```
def update_connection(self, ip, mac, status):
        # create new connection and cursor
    db_conn = sqlite3.connect(self.db_name)
    db_cursor = db_conn.cursor()


    encrypted_ip = self.fernet.encrypt(ip.encode())
    encrypted_mac = self.fernet.encrypt(mac.encode())

    # check if the ip is already in the table
    db_cursor.execute(
            f"SELECT {self.conn_table_columns[0]},{self.conn_table_columns[1]} FROM
{self.conn_table_name}",
        )
```

```
    rows = db_cursor.fetchall()

    wanted_id = self.get_id(ip, rows)

    # check if new ip or not
    if wanted_id is not None:
        print('here', status)
        # if exist update connection status to the one given
        db_cursor.execute(
            f"UPDATE {self.conn_table_name} SET {self.conn_table_columns[3]} = ? WHERE
{self.conn_table_columns[0]}=?",
            (status, wanted_id),
        )
    else:
        # if not exist add the new ip to the table
        db_cursor.execute(
            f"INSERT INTO {self.conn_table_name} ({', '.join(self.conn_table_columns[1:])}) VALUES (?, ?,
?)",
            (encrypted_ip, encrypted_mac, status),
        )
    db_conn.commit()
    db_conn.close()
```

פונקציה שמוסיפה מידע לטבלת המידע מהעדכונים השותפים של הלקוחות:

```
def add_data(self, data):
# create new connection and cursor
    db_conn = sqlite3.connect(self.db_name)
    db_cursor = db_conn.cursor()

    # find the id of the ip
    db_cursor.execute(
        f"SELECT {self.conn_table_columns[0]}, {self.conn_table_columns[1]} FROM
{self.conn_table_name}",
    )
    rows = db_cursor.fetchall()

    wanted_id = self.get_id(data[0], rows)

    # add the colomn with all of the info
    values = [wanted_id] + data[1:]
    db_cursor.execute(
        f"INSERT INTO {self.info_table_name} ({', '.join(self.info_table_columns)}) VALUES (?, ?, ?, ?,
?)",
        (values),
    )
    db_conn.commit()
    db_conn.close()
```

בגלל שהפעולות נקראת מthreads שונים במחלקת השרת, אני חייבת ליצור connection וcursor חדש עבור כל שימוש בהן. בנוסף, כאשר הפעולות נקראות בשרת, אני משתמשת בlock כדי לוודא שכמה threads לא מנסים לשנות את הנתונים בdatabase במקביל, דבר שיכול לגרום להתנגשויות ולעיוות של המידע.

WOL (wake on lan)

זוהי היכולת שלנו להעיר מחשב מרחוק. הקוד ששולח את "חבילת הקסם" משתמש בספרייה בpython בשם wakeonlan ובפירוט בפעולה send_magic_packet :

```
from wakeonlan import send_magic_packet

def wake_device(mac, ip):
    send_magic_packet(mac,ip_address=ip)
    # Magic Packet Sent
```

החבילה נשלחת למחשב ומתקבלת בכרטיס ברשת שלו. הוא מזהה שזו "חבילת הקסם" וכך מדליק את המחשב. כמובן שלפני כן יש לאפשר בBIOS שכרטיס הרשת יקבל קצת חשמל בזמן שהמחשב כבוי, כדי שיוכל לשים לב אם נשלחת חבילת קסם.

תיאור המודולים של מערכת התוכנה

בשביל להריץ את התוכנה ואת האלמנטים השונים בה, המערכת משתמשת בכמה מודלים.
להלן פירוט של מודלי המערכת:

- server.py

מחלקת השרת אשר ימצא על המחשב של מנהל המערכת (system administrator).
מתקשר עם לקוחות המערכת.

- client.py

מחלקה המייצגת את שאר המחשבים במערכת - הלקוחות אשר מתקשרים עם השרת.

- database.py

מחלקה אשר אחראית על אחראית על עדכון הנתונים בdatabase.

- Snmp_server.py

מודל אשר מאפשר לכל לקוח ברשת לקבל מידע שונה עליו, כגון הperformance שלו, מידע על הusers השונים על המחשב, מידע על הprocesses השונים הרצים על המחשב, ועוד.

- WOL.py

מודל אשר מאפשר שימוש בWOL – wake on lan, הדלקה מרחוק של מחשבים.

- GUI.py

מחלקה היוצרת את ממשק המשתמש של השרת.

מודלים/ספריות מרכזיות מיובאות:

Psutil, wmi –

מקנות מידע על היבטים שונים במחשב.

socket –

מאפשר ליצור שרת ולקוח בפרוטוקול tcp.

Threading –

מאפשר יצירה ושימוש בתהליכונים – לבצע כמה קטעי קוד במקביל.

sqlite3 –

מאפשר חיבור לdatabase והרצת פקודות בSQL לשם עריכתו.

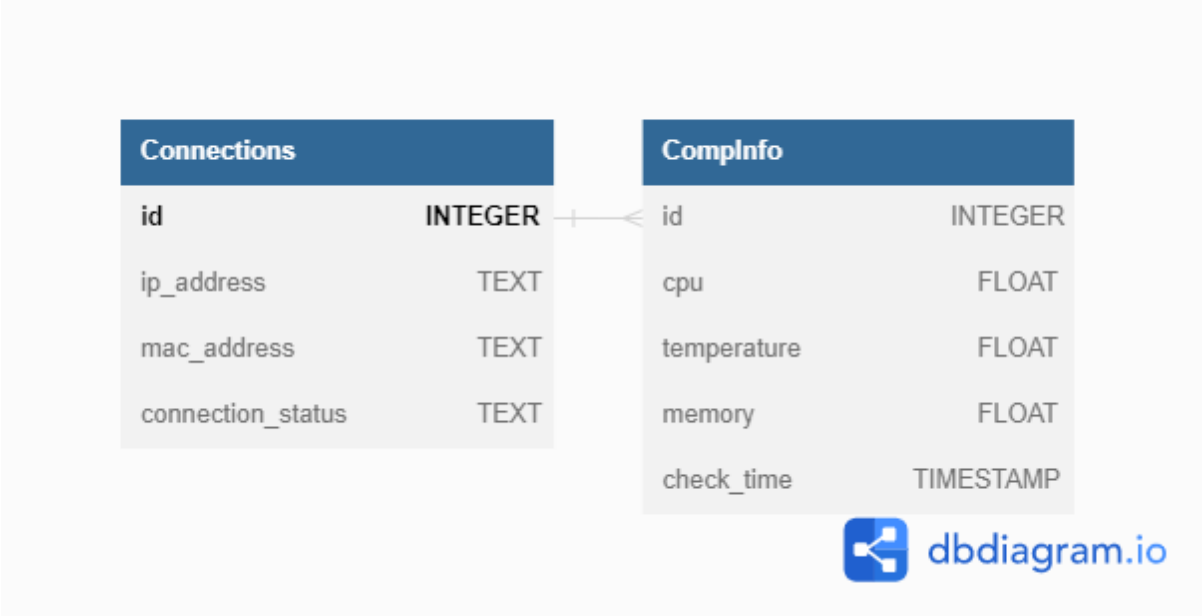
cryptography.fernet –
מאפשר הצפנה של מידע רגיש אשר מאוחסן בdatabase.

Tkinter –
מאפשר יצירה של ממשק משתמש.

תיעוד הקוד

פירוט מבניי הנתונים:

שם המבנה	תפקיד המבנה
Ipconnections.db	מכיל מידע שונה אודות הלקוחות ברשת. מסד הנתונים הזה מכיל שתי טבלאות. אחת מהן – Connections, מכילה id (מפתח ראשי) אינטג'ר, את כתובות הipה של הלקוחות, מחרוזת, כתובות הmac שלהם, מחרוזת, ואת סטטוס החיבור שלהם ("on"/"off"). השניה - CompInfo,מכילה id שמקשר אותה לטבלה הראשונה, אינטג'ר, מכילה מידע אודות הביצועים של המחשב – cpu, temperature וmemory (כולם floats). בנוסף היא מכילה TIMESTAMP של הזמן בו המדידה התבצעה.



פירוט הפעולות ותפקידן:

server.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def __init__ (self)	מקבלת את מחלקת השרת	אובייקט של המחלקה.	בנאי, מאתחל את השרת.
def get_comp_info(self, client_con, client_address)	מקבלת את מחלקת השרת, את socket עדכון המידע ואת כתובת הלקוח	-	מקבלת מידע שוטף על ביצועי הלקוח ומעדכנת את מסד הנתונים.

def get_mac_address(self, ip_address)	מקבלת את מחלקת השרת וכתובת ip	כתובת mac	מוצאת את כתובת המac של מחשב ברשת הפנימית לפי כתובת הip שלו.
def update_database_connection(self, client_address, status)	מקבלת את מחלקת השרת, כתובת של לקוח וסטטוס החיבור שלו לשרת	-	משתמשת באובייקט של database כדי לעדכן את מסד הנתונים.
def run(self)	מקבלת את מחלקת השרת	-	מטפלת בחיבורים של לקוחות חדשים.
def get_info_from_compter(self, ip, info)	מקבלת את מחלקת השרת, כתובת ip ומידע שיש לקבל מהלקוח	מידע מהלקוח	מבקשת מידע מסוים מלקוח שמחובר כרגע לשרת.

database.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def __init__(self)	מקבלת את המחלקה	אובייקט של המחלקה.	בנאי, מאתחל את מסד הנתונים ואת האובייקט.
def get_id(self, ip, rows)	מקבלת את המחלקה, כתובת ip ואת השורות מטבלת החיבורים	את הID של אותו הIP	מוצאת את הID של IP מסוים מטבלת החיבורים.
def update_connection(self, ip, mac, status)	מקבלת את המחלקה, כתובת ip וmac וסטטוס חיבור לשרת.	-	מעדכנת את סטטוס חיבור הלקוח/ מוסיפה לקוח חדש לטבלת החיבורים.
def add_data(self, data)	מקבלת את המחלקה ואת המידע שיש לעדכן בטבלת המידע השוטף.	-	מקבלת מידע ומעדכנת את טבלת המידע.

client.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד

def __init__(self)	מקבלת את מחלקת הלקוח.	אובייקט של המחלקה.	בנאי, מאתחל את הלקוח.
def send_info(self)	מקבלת את מחלקת הלקוח.	-	שולחת לשרת כל 30 שניות עדכון לגבי ביצועי המחשב.
def check_for_forbidden_proccesses (self)	מקבלת את מחלקת הלקוח.	-	במקרה ורצים תהליכים שהוגדרו כאסורים על המחשב, שולחת לשרת עדכון שרץ תהליך אסור ושולחת את כל התהליכים שרצים ומידע עליהם.
def send_procmon(self, processes, socket)	מקבלת את מחלקת הלקוח, רשימת התהליכים הרצים על המחשב וחיבור socket.	-	שולחת את כל התהליכים הרצים על המחשב ומידע עליהם בכמה חבילות לשרת.
def open_dll_exe(self)	מקבלת את מחלקת הלקוח.	-	מריצה ברקע את הקובץ openhardwaremonitor.exe כדי לאפשר שימוש בdll.
def disconnect(self)	מקבלת את מחלקת הלקוח.	-	מנתקת את ההידברות עם השרת.
def run(self)	מקבלת את מחלקת הלקוח.	-	מתקשרת עם השרת בעזרת הsocket הראשי.

WOL.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def wake_device(mac, ip)	כתובת mac וכתובת ip	-	מעירה מחשב כבוי ברשת

GUI.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def __init__(self)	מקבלת את המחלקה	מחזירה אובייקט של המחלקה	בנאי, מאתחלת את אובייקט המחלקה ואת הGUI
def convert_to_dict(self, str)	מקבלת את המחלקה ומחרוזת	מילון	ממירה את המחרוזת הנתונה למילון

def show_info(self)	מקבלת את המחלקה	-	מציגה את המידע המבוקש בGUI
def toggle(self)	מקבלת את המחלקה	-	אחראי על החלפת הmodes של החלון (light ו dark mode mode)
def refresh_data(self)	מקבלת את המחלקה	-	מעדכן את המידע המוצג בtreeview
def create_tree_children (self, parent)	מקבלת את המחלקה שורת ההורה בtreevies	-	מוסיפה להורה בtreeview ילדים

snmp_server.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def get_virtual_mem()	-	זיכרון של המחשב	מחזירה את הזיכרון של המחשב.
def get_drives_info()	-	מידע על הדיסקים במחשב	מחזירה מידע על הדיסקים במחשב.
def bytes_to_GB(bytes)	בייטים	ג'יגה בייטים	ממירה בייטים לג'יגה בייטים.
def get_network_info()	-	מילון שמכיל מידע על הרשת.	מחזירה מילון שמכיל מידע על הרשת – מידע שנשלח ומידע שהתקבל במחשב.
def get_connections_info()	-	מילון עם מידע על החיבורים בפורטים השונים במחשב.	מחזירה מילון שמכיל מידע על החיבורים של המחשב – מי יזם, מה סוג החיבור ומה הסטטוס.
def get_battery_info()	-	מידע על הסוללה של המחשב.	מחזירה מידע על הסוללה – אחוזים, זמן שתחזיק והאם החשמל מחובר.
def get_users_info()	-	רשימה של שמות הusers על המחשב.	מחזירה רשימה של שמות הusers על המחשב.

def __get_info(s_type, s_name)	סוג ושם של חיישן	מידע על חיישן	משתמשת ב open hardware monitor כדי להשיג מידע על חיישנים במחשב.
def get_cpu_temp()	-	טמפרטורת ה cpu package	משתמשת בפעולה הפנימית כדי להחזיר את טמפרטורת הcpu
def get_gpu_temp()	-	טמפרטורת ה gpu core	משתמשת בפעולה הפנימית כדי להחזיר את טמפרטורת הgpu
def get_cpu ()	-	שימוש הcpu במחשב	משתמשת בפעולה הפנימית כדי להחזיר את שימוש הcpu
def get_processes_info ()	-	רשימת מילונים של כל התהליכים הרצים על המחשב ומידע עליהם	מנתחת מידע על התהליכים הרצים על המחשב ומחזירה אותו.
def get_os()	-	מידע על מערכת ההפעלה	מחזירה מידע על מערכת ההפעלה – שמה והגרסה שלה.
def get_processor()	-	מידע על המעבד	מחזירה את סוג ושם המעבד של המחשב.
def get_hardware_info()	-	מילון שמכיל מידע על החומרה של המחשב.	מחזירה מילון שמכיל מידע על המעבד, מערכת ההפעלה וביצועי המחשב.

השוואת העבודה עם פתרונות ויישומים קיימים

בגלל שהמוצר שלי עושה שני דברים עיקריים – פיקוח על רשת והשגת מידע ממחשב אודות החומרה והביצועים שלו, אחלק את החלק הזה לשניים.

מוצרים להצגת מידע על המחשב –

•Speccy - פותח על ידי החברה Piriform, ניט תכנת שירות חינמית שפועלת על windows. היא מציגה למשתמש מידע על הרכיבים ועל התוכנות שנמצאים במחשב. התוכנה מציגה תכונות כגון פרטים על המעבד, על הדיסקים במחשב, ונתונים על הזיכרון. ההבדל בין המערכת שלי לSpeccy הוא שהמערכת שלי מציגה את כל הפרטים הללו, אך גם פרטים שלא קשורים בחומרה, למשל מידע על הרשת.

•Task Manager - מציג מידע שימושי על המחשב. הוא מספק מידע על ביצועי המחשב ועל תוכנות שונות, למשל התהליכים הרצים על המחשב, מידע על הזיכרון, ומידע על הCPU. המערכת שלי תלויה בספריות חיצוניות אך בגלל שtask manager הוא חלק מwindows יש לו גישה הרבה יותר פשוטה לכל הנתונים. למרות זאת, המערכת שלי מספקת גם שירותים כמו הגדרת תהליכים אסורים, בניגוד ל task manager.

מוצרים לפיקוח על רשת –

• Naigos – תוכנת מחשב חינמית וopen source שממנטרת על מערות, רשתות ותשתיות. המערכת נותנת גישה לפיקוח על המערכות העיקריות במחשב וברשת. היא מספקת שירותים של התרעה במקרה שדבר משתבש וגם מתריעה כשהבעיה נפתרה. הבדל עיקרי בין Naigos למערכת שלי הוא ברמת הפירוט של הרכיבים במחשב. בעוד Naigos נותנת דגש יותר רב על תרעה בעת בעיה, המוצר שלי מספק גם שימוש יותר שוטף והצגת מידע יותר מפורט על כל מחשב ברשת.

•Zabbix – תוכנה open source לניהול רשתות, שרתים מכונות וירטואליות וענן. אוסף מידע על הרשת, מערכות ההפעלה של המשתמשים, ועל שרתים. ההבדל העיקרי בין המערכת שלי לבין Zabbix הוא התמיכה הרחבה של Zabbix בשרתים, מכונות וירטואליות וענן, בעוד הפרויקט שלי תומך ברשת מחשבים.

הערכת הפתרון לעומת התכנון והמלצות לשיפורו

את החלקים המרכזיים של הפרויקט שלי הצלחתי להוציא לפועל. הצלחתי ליצור שרת שיחוברו אליו לקוחות ברשת, ובעזרתו הצלחתי לפקח על רשת מחשבים ועל כל מחשב בנפרד לראות את הביצועים ונתונים שונים עליו. ביחס לזמן שהיה לי אני מרוצה מהמערכת שבניתי ומהידע והניסיון שרכשתי, אבל זה לא אומר שאין לה מקום לשיפור. הייתי רוצה להמשיך לעבוד עליה גם אחרי ההגשה ולהוסיף לה עוד פיצ׳רים, כגון עדכון אפליקציות ותוכנות על המחשבים ברשת מהמחשב של מנהל המערכת (בעזרת winget), פיצ׳ר שיעזור לניהול מסדי הנתונים וכאשר הם התמלאו יספק שירות של אחסון הטבלה בקובץ, ועוד. אהבתי את הרעיון שלי לפרויקט כי אפשר לבנות עליו עוד ועוד פיצ׳רים וכך להפוך אותו לרחב יותר, מה שכמובן תלוי במגבלת הזמן, המשאב הכי יקר בפרויקט זה.

תיאור של הממשק למשתמש – הוראות הפעלה

* הוראות ההפעלה נכתבו בלשון זכר בשל טעמי נוחות בלבד.

שלום מנהל מערכת נכבד,

באמצעות התוכנה הזו תוכל לעשות את העבודה שלך בצורה יעילה יותר וקלה יותר.
תרצה להתקין את תוכנת השרת על המחשב שלך, כך שתרוץ תמיד כשהמחשב דלוק. בנוסף, תרצה לשים את כתובת המייל שלך בקובץ admin_info.txt. את תוכנת הלקוחות, תרצה להתקין על כל מחשבי המערכת, כך שיפעלו עליהם לגמרי בקרע. תרצה לשים את כתובת הip שעליו רץ השרת בקובץ המצורף לתוכנת הלקוח - client_info.txt. יתר על כן, תוכל להכניס שם רף של cpu, memory וטמפרטורת cpu שהיית רוצה שיהיו עבור על מחשב, בנוסף לרשימה של תהליכים שאסור שירוצו על המחשב מבחינתך.

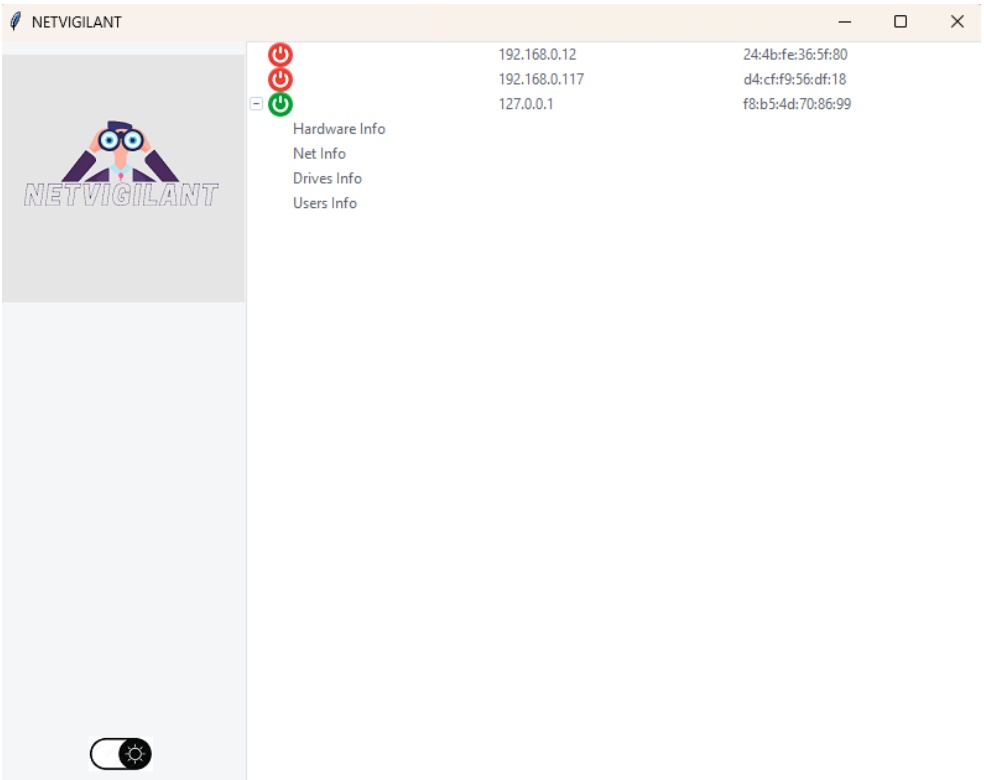
כאשר השרת מותקן ומופעל, יפתח חלון בו תוכל לראות את כל המחשבים עליהם תוכנת הלקוח רצה. תוכל לראות את המידע בצורת עץ, ולראות אילו מחשבים דלוקים ואילו מחשבים כבויים כרגע. עבור כל המחשבים הדלוקים, תוכל לראות סוגים שונים של פיסות מידע עליהם, כמו למשל כתובות הip והmac שלהם, מידע שקשור בביצועי המחשב, מידע שקשור ברשת, במשתמשים ועוד. אם תלחץ על סימן ה+ ליד שם המחשב, השורה תתרחב ותוכל לראות קטגוריות מידע שונות. ברגע שתלחץ על אחת מהן, יפתח חלון בו תוכל לראות את המידע של אותה הקטגוריה.

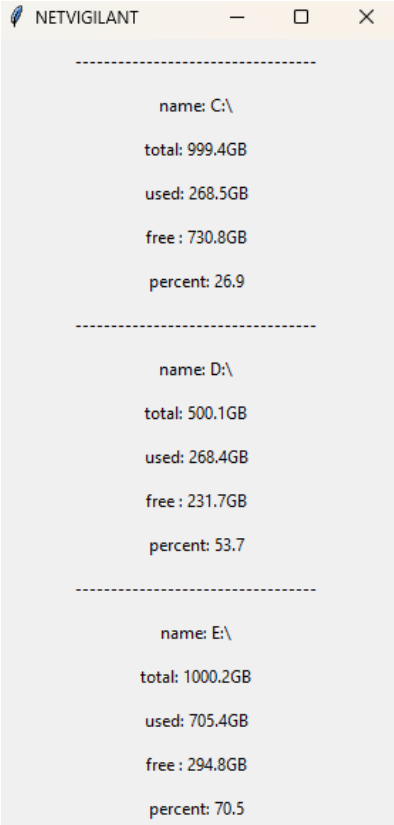
יתרה מכך, במקרה ואחד המחשבים עבר את הרף שהגדרת עבור הcpu, memory, או טמפרטורת הcpu, תקבל מייל והודעת pop up שיודיעו לך על המחשב שזה קרה אצלו. בדומה לכך, אם תהליך שהגדרת כאסור ירוץ על אחד המחשבים, תקבל הודעה למייל והודעת pop up עם אפשרות לקבל קובץ שבו כל התהליכים שרצו על המחשב באותו הזמן, בנוסף למידע עליהם.

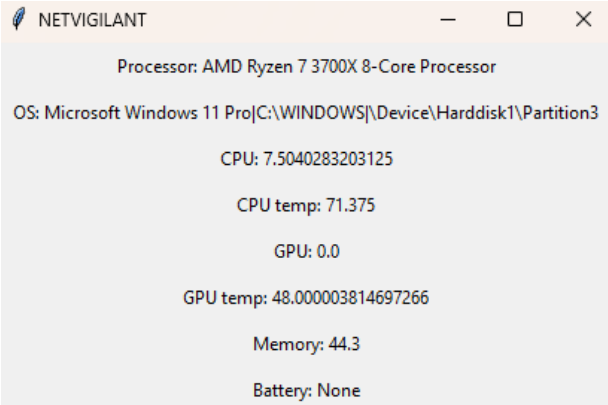
עבור המחשבים הכבויים ברשת, תוכל להקליק עליהם לחיצה ימינית ולבחור באפשרות של wake on lan – הדלקה מרחוק של מחשב. בשביל להשתמש בה, תצטרך לדאוג בהגדרות הBIOS של המחשבים שהם אכן תומכים בwake on lan.

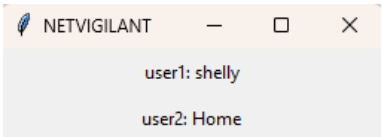
בהצלחה רבה! ושימוש נעים.

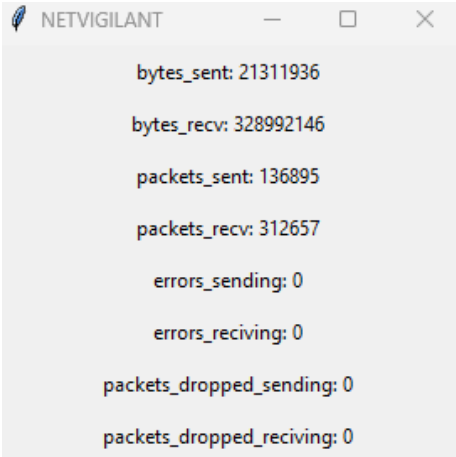
ממשק השרת


NETVIGILANT
192.168.0.12
192.168.0.117
127.0.0.1
24:4b:fe:36:5f:80
d4:cf:f9:56:df:18
f8:b5:4d:70:86:99
Hardware Info
Net Info
Drives Info
Users Info
NETVIGILANT

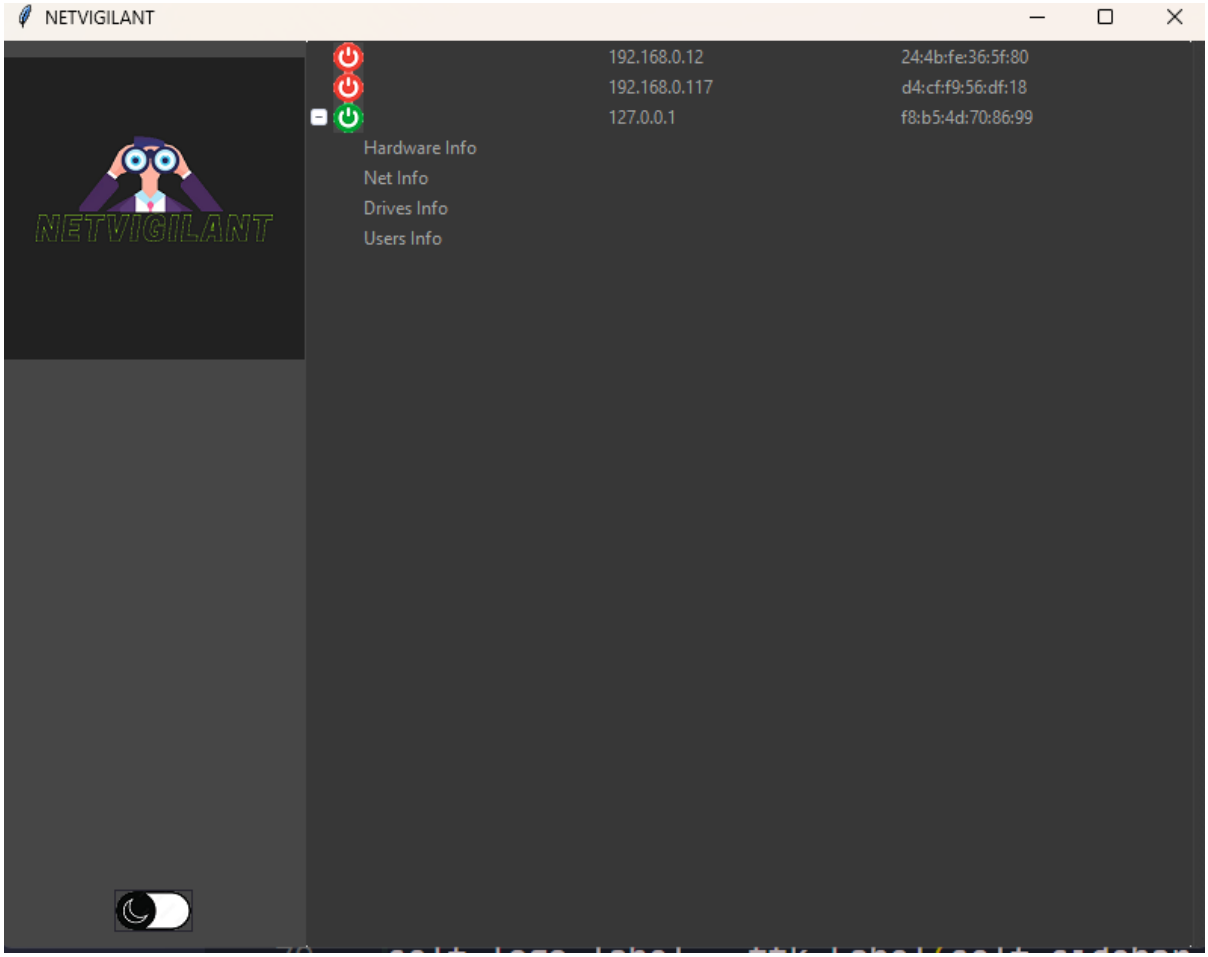

NETVIGILANT
---
name: C:\
total: 999.4GB
used: 268.5GB
free : 730.8GB
percent: 26.9
---
name: D:\
total: 500.1GB
used: 268.4GB
free : 231.7GB
percent: 53.7
---
name: E:\
total: 1000.2GB
used: 705.4GB
free : 294.8GB
percent: 70.5


NETVIGILANT
Processor: AMD Ryzen 7 3700X 8-Core Processor
OS: Microsoft Windows 11 Pro|C:\WINDOWS|\Device\Harddisk1\Partition3
CPU: 7.5040283203125
CPU temp: 71.375
GPU: 0.0
GPU temp: 48.000003814697266
Memory: 44.3
Battery: None


NETVIGILANT
user1: shelly
user2: Home


NETVIGILANT
bytes_sent: 21311936
bytes_recv: 328992146
packets_sent: 136895
packets_recv: 312657
errors_sending: 0
errors_reciving: 0
packets_dropped_sending: 0
packets_dropped_reciving: 0

בנוסף לתמיכה בdark mode על ידי לחיצה על הכתפור למטה :



מבט אישי על העבודה ותהליך הפיתוח

העבודה על הפרויקט הייתה מאוד מאתגרת בשבילי. בהתחלה היה לי קשה לחשוב על נושא לפרויקט שבאמת יעניין אותי – לא רציתי לעשות משהו סתמי רק כדי לקבל ציון, רציתי לבנות מערכת שאני אהיה גאה בה, ואלמד ואתנסה תוך העבודה עליה. לכן לקח לי הרבה זמן לגבש רעיון כמו שצריך. לאחר שבאמת התחלתי לעבוד על הפרויקט, בהדרגה נחשפתי עוד ועוד לנושאים חדשים עבורי ומעניינים, והעמקתי את הידע שלי בנושאים שהכרתי וגם בנושאים שלא.

התחלתי את תהליך הכתיבה במחקר על הדרכים השונות שיש לי לקבל מידע אודות ביצועי המחשב. ככל שגיליתי על עוד מידע שאני יכולה להשיג, עלו לי עוד ועוד רעיונות לפיצ'רים שאני יכולה להוסיף לפרויקט. ככל שהזמן עבר, הרעיון שלי לפרויקט נהיה מגובש יותר ויותר והתחלתי להבין איך המערכת שלי תראה. זה נתן לי הרבה מוטיבציה להתנסות ו"ללכלך את הידיים", וכשראיתי שהכול מתחיל להתחבר ביחד – השרת והלקוח, התקשורת עם הdatabase, המידע שאני משיגה על מחשבים שונים, נהניתי מההרגשה שאני בונה משהו אמיתי בסדר גודל הרבה יתר משמעותי מכל מה שעשיתי עד עכשיו. למרות שסדר הגודל של הפרויקט היו אחת החששות שלי בתחילת הדרך, עכשיו אני ממש שמחה שהייתה לי ההזדמנות לעשות משהו כזה.

הייתי צריכה להתגבר על הבעיות שהצגתי בפרק תיאור הבעיה האלגוריתמית, ולשם כך חלק מזמן העבודה שלי הוקדש אך ורק למחקר. כך זכיתי לצבור ידע רב תוך עשיית הפרויקט, ידע שאני מתרגשת להשתמש בו בחיים האמיתיים, בצבא, בעבודה, ובפרויקטים אישיים שאני אעסוק בהם.

הבעיה הכי משמעותית שהתמודדתי איתה במהלך עשיית הפרויקט היא הזמן. בהתחשב בזה שלצערי הייתי כ3 שבועות בחו"ל בחודש אפריל, הייתי צריכה לתכנן את הזמן שלי טוב כדי להספיק הכל. למרות זאת הבאתי איתי מחשב וגם שם ניסיתי לעבוד כמה שיותר. אני כמובן מרגישה שבלי הלחץ של הזמן הייתי מספיקה הרבה יותר (כמו כל דבר בחיים) אך אני מרוצה ממה שהספקתי. רוב מה שתכננתי שאספיק יצא לפועל, ותאם לציפיות שלי מתחילת העבודה על הפרויקט.

בנוסף לכך, אני מלאת מוטיבציה להמשיך ולשפר את המוצר שלי גם אחרי הצגת הפרויקט כדי להמשיך לצבור ידע ולהוסיף פיצ'רים שונים שחשבתי עליהם אבל עקב מגבלת הזמן לא יצאו לפועל.

לסיכום, תוך העבודה על הפרויקט הרגשתי שאני מרחיבה את אופקיי ושהידע שלי רק עולה ומתחזק. נהניתי להתנסות וללמוד, נהניתי להעשיר את הידע שלי ולהיות סקרנית, ונהניתי משהשמחה של למצוא פתרון לבעיה אחרי שהייתי מתוסכלת ותקועה עליה במשך ימים. אני מרגישה שהפרויקט שירת את המטרה שלי, גם מבחינה מקצועית וגם מהבחינה שלמדתי הרבה בתהליך על עבודה עצמאית, על תוכנה, על מערכת ההפעלה, ועוד.

ביבליוגרפיה

1. Stack Overflow - Where Developers Learn, Share, & Build Careers. (n.d.).
Stack Overflow. https://stackoverflow.com/

2. GeeksforGeeks | A computer science portal for geeks. (n.d.).
GeeksforGeeks. https://www.geeksforgeeks.org/

3. Wikipedia. (n.d.).
Wikipedia. https://en.wikipedia.org/wiki/Main_Page

4. psutil documentation: psutil 5.9.5 documentation. (n.d.).
psutil documentation. https://psutil.readthedocs.io/en/latest/

5. Python Automation project : Run Python scripts Automatically in backgroud on windows startup. (2020, August 26).
YouTube. https://www.youtube.com/watch?v=XWV9tatoPQI&ab_channel=CodeBear

6. Hardware Information Tool in Python. (2022, July 08).
YouTube. https://www.youtube.com/watch?v=_9ThkldEg0c&t=6s&ab_channel=NeuralNine

7. Welcome to Python.org. (n.d.).
Python.org. https://www.python.org/doc/

8. G. (2023, April 24). Build software better, together.
GitHub. https://github.com/topics/forums

קוד התוכנית

server.py

```
import uuid
from joblib import load
import socket
import threading
import sqlite3
import os
from threading import Lock
import subprocess
from database import database


MAIN_PORT = 65432
ALERT_PORT = 65431

# dict to save the ip and socket of computers
COMPUTERS = { }

class server:

  def __init__(self):

    # start main socket
    self.main_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.main_socket.bind(("0.0.0.0", MAIN_PORT))
    self.main_socket.listen(5)
    print("> MAIN SERVER ON")

    # start the socket for the info and the alerts
    self.alert_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.alert_socket.bind(("0.0.0.0", ALERT_PORT))
    self.alert_socket.listen(5)
    print("> INFO SERVER ON")

    # create lock instance
    self.lock = Lock()
```

```
        # create DB handler instance
        self.database = database()



    def get_comp_info(self, client_conn, client_address):
        self.update_database_connection(client_address, 'on')
        try:
            while True:

                try:
                    data = client_conn.recv(1024).decode()

                    if data == 'bye':
                        break

                    data = data.split(",")

                    # append the data to the global list so it will be added to the database
                    if len(data) == 5:
                        self.lock.acquire()
                        self.database.add_data(data)
                        self.lock.release()

                    # forbidden process running
                    if len(data) == 2:
                        forbidden_process = data[1]

                        # Receive the number of packets to expect
                        num_packets_data = client_conn.recv(4)
                        num_packets = int.from_bytes(num_packets_data, byteorder="big")
                        print(num_packets)

                        # Receive the serialized data packets and reassemble them
                        received_data = b""
                        for _ in range(num_packets):
                            packet = client_conn.recv(4096)
                            print(packet)
                            received_data += packet
                        print(received_data.decode())
```

```
                print(forbidden_process)

            except Exception as e:
                break
        except:
            pass
        finally:
            # finally close the socket
            client_conn.close()
            self.update_database_connection(client_address, 'off')
            self.lock.acquire()
            global COMPUTERS
            COMPUTERS.pop(client_address[0])
            self.lock.release()
            print("DONE WITH THIS THREAD")


    def get_mac_address(self, ip_address):
        try:
            arp_command = ["arp", "-a"]
            output = subprocess.check_output(arp_command).decode()
            mac_address = output.split()
            mac_address = mac_address[mac_address.index(ip_address) + 1]
        except:
            mac_address = uuid.getnode()
            mac_address = ":".join(
                [
                    "{:02x}".format((mac_address >> elements) & 0xFF)
                    for elements in range(0, 8 * 6, 8)
                ][::-1]
            )
        return mac_address


    def update_database_connection(self, client_address, status):
        self.lock.acquire()
        print('here')
        self.database.update_connection(
            *[
                client_address[0],
```

```
            self.get_mac_address(client_address[0]),
            status,
        ]
    )
    self.lock.release()


def run(self):
    while True:
        # accept a new client - main socket
        conn, addr = self.main_socket.accept()
        # accept a new client - alert socket
        alert_conn, alert_addr = self.alert_socket.accept()

        # put the socket in a global dict to ask for information
        self.lock.acquire()
        global COMPUTERS
        COMPUTERS[addr[0]] = conn
        self.lock.release()

        # thread that supplies client info
        info_thread = threading.Thread(
            target=self.get_comp_info,
            args=(
                alert_conn,
                alert_addr,
            ),
        )

        # start the thread to get information about the client's cpu' temp and memory
        info_thread.start()

def get_info_from_computer(self, ip, info):
    global COMPUTERS
    try:
        self.lock.acquire()
        socket = COMPUTERS[ip]
        self.lock.release()

        socket.send(info.encode())
```

```
        return socket.recv(1064).decode()


    except:
        print('no such ip')
```

database.py

```
import sqlite3
from cryptography.fernet import Fernet

class database:

    def __init__(self):
        # key
        file = open('key.key', 'rb') # rb = read bytes
        self.key  = file.read()
        file.close()
        self.fernet = Fernet(self.key)

        # create connection and cursor for the db
        self.db_name = "ipconnections.db"
        db_conn = sqlite3.connect(self.db_name)
        db_cursor = db_conn.cursor()

        # tables names
        self.conn_table_name = "Connections"
        self.info_table_name = "CompInfo"

        # columns for the table that describes the different computers on the net
        self.conn_table_columns = [
            "id",
            "ip_address",
            "mac_address",
            "connection_status",
        ]

        # columns for the table that saves the computer's updates on their performance
        self.info_table_columns = [
            "id",
            "cpu",
            "temperature",
            "memory",
            "check_time"
        ]
```

```
        # create connections table
        db_cursor.execute(
            f"""CREATE TABLE IF NOT EXISTS {self.conn_table_name} (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                ip_address TEXT NOT NULL,
                mac_address TEXT NOT NULL,
                connection_status TEXT NOT NULL)"""
        )

        # create computers info table
        db_cursor.execute(
            f"""CREATE TABLE IF NOT EXISTS {self.info_table_name} (
                id INTEGER NOT NULL,
                cpu FLOAT NOT NULL,
                temperature FLOAT NOT NULL,
                memory FLOAT NOT NULL,
                check_time TIMESTAMP not null,
                FOREIGN KEY (id) REFERENCES ip_addresses (id))"""
        )

        # commit the changes and close the connection
        db_conn.commit()
        db_conn.close()

    def get_id(self, ip, rows):
        for row in rows:
            decrypted_ip = self.fernet.decrypt(row[1]).decode()
            if decrypted_ip == ip:
                return row[0]


    def update_connection(self, ip, mac, status):
        # create new connection and cursor
        db_conn = sqlite3.connect(self.db_name)
        db_cursor = db_conn.cursor()



        encrypted_ip = self.fernet.encrypt(ip.encode())
        encrypted_mac = self.fernet.encrypt(mac.encode())
```

```
        # check if the ip is already in the table
        db_cursor.execute(
                f"SELECT {self.conn_table_columns[0]},{self.conn_table_columns[1]} FROM
{self.conn_table_name}",
            )
        rows = db_cursor.fetchall()

        wanted_id = self.get_id(ip, rows)

        # check if new ip or not
        if wanted_id is not None:
            print('here', status)
            # if exist update connection status to the one given
            db_cursor.execute(
                f"UPDATE {self.conn_table_name} SET {self.conn_table_columns[3]} = ? WHERE
{self.conn_table_columns[0]}=?",
                (status, wanted_id),
            )
        else:
            # if not exist add the new ip to the table
            db_cursor.execute(
                f"INSERT INTO {self.conn_table_name} ({', '.join(self.conn_table_columns[1:])})
VALUES (?, ?, ?)",
                (encrypted_ip, encrypted_mac, status),
            )
        db_conn.commit()
        db_conn.close()

    def add_data(self, data):
        # create new connection and cursor
        db_conn = sqlite3.connect(self.db_name)
        db_cursor = db_conn.cursor()

        # find the id of the ip
        db_cursor.execute(
            f"SELECT {self.conn_table_columns[0]}, {self.conn_table_columns[1]} FROM
{self.conn_table_name}",
        )
        rows = db_cursor.fetchall()
```

```
    wanted_id = self.get_id(data[0], rows)

    # add the colomn with all of the info
    values = [wanted_id] + data[1:]
    db_cursor.execute(
        f"INSERT INTO {self.info_table_name} ({', '.join(self.info_table_columns)})
VALUES (?, ?, ?, ?, ?)",
        (values),
    )
    db_conn.commit()
    db_conn.close()
```

client.py

```
import socket
import sys
import psutil
import time
import snmp_server
import threading
import subprocess
from pyuac import main_requires_admin
import pandas as pd
import os
from joblib import dump
from threading import Lock


IP = "127.0.0.1" # add file to store ip
MAIN_PORT = 65432
ALERT_PORT = 65431

CHECK_SECONDS = 30
THREAD_ALIVE = True
DISCONNECTING = False

FORBIDDEN_PROCESSES_NAMES = ["Notepad.exe"]


class client:
    def __init__(self):
        # create main socket
        self.main_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # client tries to connect the server
        connection_established = False
        while not connection_established:
            # incase the server is not up yet
            try:
                self.main_socket.connect((IP, MAIN_PORT))
                connection_established = True
            except Exception as e:
                # try again
```

```
            pass


        # connect the socket responsinble for the info for the database
        self.info_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.info_socket.connect((IP, ALERT_PORT))

        # run a thread in the background to send the info
        self.info_thread = threading.Thread(target=self.send_info)
        global THREAD_ALIVE
        THREAD_ALIVE = True
        self.info_thread.start()

        # create lock
        self.lock = Lock()
        # open the exe file so i can use the dll
        self.open_dll_exe()

        self.run()


    def send_info(self):
        init_time = time.time()
        while THREAD_ALIVE:
            self.lock.acquire()
            if time.time() - init_time > CHECK_SECONDS:
                check_time = str(
                    time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
                ).strip()
                cpu = str(snmp_server.get_cpu()).strip()  # TODO: fix cpu
                mem = str(snmp_server.get_virtual_mem()).strip()
                temp = str(snmp_server.get_cpu_temp())
                msg = str(", ".join([IP, cpu, temp, mem, check_time]))
                self.info_socket.send(msg.encode())

                self.check_for_forbidden_proccesses()

                init_time = time.time()
            self.lock.release()
```

```
def check_for_forbidden_proccesses(self):
    processes = snmp_server.get_processes_info()
    for forbidden_process in FORBIDDEN_PROCESSES_NAMES:
        for process in processes:
            if process["name"] == forbidden_process:
                msg = f"FORBDDEN PROCCESS RUNNING, {forbidden_process[0]}"
                self.info_socket.send(msg.encode())
                self.send_procmon(processes, self.info_socket)

def send_procmon(self, processes, socket):
    data = str(processes)
    # Send the serialized data over the socket in packets
    packet_size = 4096
    total_size = len(data.encode())
    num_packets = total_size // packet_size + 1

    # Send the number of packets to expect
    socket.send(num_packets.to_bytes(4, byteorder="big"))

    # Send the serialized data in packets
    for i in range(num_packets):
        start = i * packet_size
        end = min(start + packet_size, total_size)
        packet = data[start: end]
        socket.send(packet.encode())

@main_requires_admin
def open_dll_exe(self):
    # Specify the path to the EXE file
    exe_path = r"sources\OpenHardwareMonitor\OpenHardwareMonitor.exe"
    exe_path = "\\".join([os.getcwd(), exe_path])

    # runs the exe as a daemon process
    subprocess.Popen([exe_path])

def disconnect(self):
    # sends the server "bye" to notify disconnection
    self.lock.acquire()
    self.info_socket.send("bye".encode())
    self.lock.release()
```

```
        # notifies the other thread it has to finish
        global THREAD_ALIVE
        THREAD_ALIVE = False

        # close the main socket of the client
        self.main_socket.close()

    def run(self):
        while not DISCONNECTING:
            try:
                info = self.main_socket.recv(1064).decode()
                print(info)
                if info == 'hardware':
                    data = snmp_server.get_hardware_info()
                elif info == 'users':
                    data =snmp_server.get_users_info()
                elif info == 'net':
                    data = snmp_server.get_network_info()
                elif info == 'connections':
                    data = snmp_server.get_connections_info()
                else:
                    data = snmp_server.get_drives_info()
                print(data)
                data = str(data).encode()
                self.main_socket.send(data)
            except Exception as e:
                print(e)

c = client()
time.sleep(30)
c.disconnect()
```

GUI.py

```
import threading
import tkinter as tk
from tkinter import ttk
from ttkthemes import ThemedStyle
import sqlite3
from server import server
from cryptography.fernet import Fernet


class GUI:
    def refresh_data(self):
        db_conn = sqlite3.connect('ipconnections.db')
        db_cursor = db_conn.cursor()

        db_cursor.execute("SELECT * FROM Connections")
        self.data = db_cursor.fetchall()

        # Clear the existing treeview items
        self.tree.delete(*self.tree.get_children())

        # Insert updated items into the treeview
        for computer in self.data:
            ip = self.fernet.decrypt(computer[1]).decode()
            mac = self.fernet.decrypt(computer[2]).decode()
            if computer[3] == "on":
                parent = self.tree.insert(
                    "",
                    "end",
                    text="",
                    values=(ip, mac),
                    open=True,
                    tags="on",
                )
                self.tree.item(parent, image=self.on_icon, tags="on")
                self.create_tree_children(parent)
            else:
                parent = self.tree.insert(
                    "",
                    "end",
```

```
            text="",
            values=(ip, mac),
            open=False,
            tags="off",
        )
        self.tree.item(parent, image=self.off_icon, tags="off")

    # Schedule the next refresh
    self.root.after(1000, self.refresh_data)

def __init__(self):
    # key
    file = open('key.key', 'rb') # rb = read bytes
    self.key  = file.read()
    file.close()
    self.fernet = Fernet(self.key)

    # contain an instance of the server in the GUI
    self.server = server()
    server_thread = threading.Thread(target= self.server.run)
    server_thread.start()

    self.root = tk.Tk()
    self.root.title("NETVIGILANT")
    self.root.geometry("800x600")
    self.style = ThemedStyle(ttk.Style())
    self.style.set_theme("arc")

    # on and off icons
    self.on_icon = tk.PhotoImage(file="sources/Images/green_dot.png")
    self.off_icon = tk.PhotoImage(file="sources/Images/red_dot.png")

    # Create the sidebar
    self.sidebar = ttk.Frame(self.root, width=200)
    self.sidebar.pack(side="left", fill="y")

    # Logo
    self.light_logo = tk.PhotoImage(file="sources/Images/llogo.png")
    self.dark_logo = tk.PhotoImage(file="sources/Images/dlogo.png")
    self.logo_label = ttk.Label(self.sidebar, image=self.light_logo)
```

```
        self.logo_label.pack(pady=10)

        # Define the columns
        self.columns = [ "ip", "mac"]

        # Create a treeview widget
        self.tree = ttk.Treeview(self.root, selectmode= 'browse')
        self.tree.pack(side="left", fill="both", expand=True)

        self.tree['columns'] = self.columns
        self.tree['show'] = 'headings'
        self.tree['show'] = 'tree'

        # Configure the treeview
        self.tree.heading("#0", text="#")
        self.tree.heading("ip", text="IP")
        self.tree.heading("mac", text="MAC")
        self.tree.column("#0", width=150, minwidth=10)
        self.tree.column("ip", width=150, minwidth=150)
        self.tree.column("mac", width=150, minwidth=150)

        # Set the icons for the TreeView
        self.tree.tag_configure("on", image=self.on_icon)
        self.tree.tag_configure("off", image=self.off_icon)

        self.refresh_data()
        self.root.after(1000, self.refresh_data)

        # Add a scrollbar to the treeview
        scrollbar = ttk.Scrollbar(self.root, orient="vertical", command=self.tree.yview)
        scrollbar.pack(side="right", fill="y")
        self.tree.configure(yscrollcommand=scrollbar.set)

        # Bind function to TreeviewSelect event
        self.tree.bind("<<TreeviewSelect>>", self.show_info)

        # Dark and light mode switch
        self.switch_value = True

        # Load light and dark mode images
```

```
        self.light = tk.PhotoImage(file="sources/Images/lightMode.png")
        self.dark = tk.PhotoImage(file="sources/Images/darkMode.png")

        # Create a button to toggle between light and dark themes
        self.switch = tk.Button(
            self.sidebar,
            image=self.light,
            bd=0,
            bg="white",
            activebackground="white",
            command=self.toggle,
        )
        self.switch.pack(padx=50, pady=10, side="bottom")

        self.root.mainloop()

    def toggle(self):
        if self.switch_value:
            self.style.set_theme("equilux")
            self.switch.config(
                image=self.dark, bg="#26242f", activebackground="#26242f"
            )
            self.logo_label.config(image=self.dark_logo)
            self.switch_value = False
        else:
            self.style.set_theme("arc")
            self.switch.config(image=self.light, bg="white", activebackground="white")
            self.logo_label.config(image=self.light_logo)
            self.switch_value = True

    def create_tree_children(self, parent):
        # Insert child items under each parent row
        self.tree.insert(parent, "end", text="Hardware Info")
        self.tree.insert(parent, "end", text="Net Info")
        self.tree.insert(parent, "end", text="Drives Info")
        self.tree.insert(parent, "end", text="Users Info")


    def show_info(self):
        try:
```

```
        # Get the selected item
        selected_item = self.tree.selection()[0]

        # Check if the selected item has a parent
        if self.tree.parent(selected_item):

            # Get the parent item's IP address
            parent_item = self.tree.parent(selected_item)
            parent_ip = self.tree.item(parent_item)["values"][0]

            # Get the child item's text
            child_text = self.tree.item(selected_item, "text")

            # Get the data from the server using the parent IP address
            if child_text == 'Hardware Info':
                print('here')
                info = 'hardware'
            elif child_text == 'Net Info':
                info = 'net'
            elif child_text == 'Drives Info':
                info = 'drives'
            elif child_text == 'Users Info':
                info = 'users'
            #elif child_text == 'Connections Info':
            #    info = 'connections'

            try:
                data = self.convert_to_dict(self.server.get_info_from_computer(parent_ip, info))

                # Create a new window and display the data
                new_window = tk.Toplevel(self.root)
                for key, value in data.items():
                    label = tk.Label(new_window, text=f"{key}: {value}")
                    label.pack(padx=10, pady=5)

            except Exception as e:
                print(e)

    except:
        pass
```

```
    def convert_to_dict(self, str):
        str = str.split(',')
        dict = { }
        for i in str:
            i = i.strip('}{][')
            i = i.split(':', 1)
            print(i)
            dict[i[0].strip('"\'')] = i[1].strip('"\'')
        return dict

if __name__ == "__main__":
    server_gui = GUI()
```

snmp_server.py

```
from datetime import datetime
import platform
import time
import clr
import pandas as pd
import psutil
import threading
import sched
import wmi
import cpuinfo
import pythoncom


def get_virtual_mem():
    # return virtual mem precentage
    return psutil.virtual_memory().percent

def get_drives_info():
    drives_info = []

    # list of all drives names
    drives = [disk.device for disk in psutil.disk_partitions()]

    # return info for each drive in dict
    for drive in drives:
        disk_info = psutil.disk_usage(drive)
        drives_info.append({
            'name' : drive,
            'total' : str(bytes_to_GB(disk_info.total)) + 'GB',
            'used' : str(bytes_to_GB(disk_info.used)) + 'GB',
            'free ' : str(bytes_to_GB(disk_info.free)) + 'GB',
            'percent' : disk_info.percent,
        })

    return drives_info

def bytes_to_GB(bytes):
    return round(bytes/1000000000, 1)
```

```
'''
bytes_sent: number of bytes sent
bytes_recv: number of bytes received
packets_sent: number of packets sent
packets_recv: number of packets received
errin: total number of errors while receiving
errout: total number of errors while sending
dropin: total number of incoming packets which were dropped
dropout: total number of outgoing packets which were dropped (always 0 on macOS and
BSD)
'''
def get_network_info():
    info = psutil.net_io_counters()
    return {
        'bytes_sent' : info.bytes_sent,
        'bytes_recv' : info.bytes_recv,
        'packets_sent' : info.packets_sent,
        'packets_recv' : info.packets_recv,
        'errors_sending' : info.errout,
        'errors_reciving' : info.errin,
        'packets_dropped_sending' : info.dropout,
        'packets_dropped_reciving' : info.dropin,
    }

'''
fd: the socket file descriptor. If the connection refers to the current process this may be passed
to socket.fromfd to obtain a usable socket object. On Windows and SunOS this is always set
to -1.
family: the address family, either AF_INET, AF_INET6 or AF_UNIX.
type: the address type, either SOCK_STREAM, SOCK_DGRAM or SOCK_SEQPACKET.
laddr: the local address as a (ip, port) named tuple or a path in case of AF_UNIX sockets. For
UNIX sockets see notes below.
raddr: the remote address as a (ip, port) named tuple or an absolute path in case of UNIX
sockets. When the remote endpoint is not connected you'll get an empty tuple (AF_INET*) or
"" (AF_UNIX). For UNIX sockets see notes below.
status: represents the status of a TCP connection. The return value is one of the
psutil.CONN_* constants (a string). For UDP and UNIX sockets this is always going to be
psutil.CONN_NONE.
```

```
pid: the PID of the process which opened the socket, if retrievable, else None. On some
platforms (e.g. Linux) the availability of this field changes depending on process privileges
(root is needed).
'''
def get_connections_info():
    info = psutil.net_connections()
    connections_info = []
    for connection in info:
        connections_info.append(
            {
                'family' : connection.family,
                'type' : connection.type,
                'local address' : connection.laddr,
                'remote address' : connection.raddr,
                'status' : connection.status,
                'pid' : connection.pid
            }
        )
    return connections_info


'''
Return the addresses associated to each NIC (network interface card) installed on the system
as a dictionary whose keys are the NIC names and value is a list of named tuples for each
address assigned to the NIC. Each named tuple includes 5 fields:

family: the address family, either AF_INET or AF_INET6 or psutil.AF_LINK, which refers
to a MAC address.
address: the primary NIC address (always set).
netmask: the netmask address (may be None).
broadcast: the broadcast address (may be None).
ptp: stands for "point to point"; it's the destination address on a point to point interface
(typically a VPN). broadcast and ptp are mutually exclusive. May be None.
'''
def get_network_interface_info():
    info =  psutil.net_if_addrs()
    return info

'''
psutil.sensors_battery() returns:
```

```
percent: battery power left as a percentage.
secsleft: a rough approximation of how many seconds are left before the battery runs out of
power. If the AC power cable is connected this is set to psutil.POWER_TIME_UNLIMITED.
If it can't be determined it is set to psutil.POWER_TIME_UNKNOWN.
power_plugged: True if the AC power cable is connected, False if not or None if it can't be
determined.
'''
def get_battery_info():
    return psutil.sensors_battery()

'''
name: the name of the user.
terminal: the tty or pseudo-tty associated with the user, if any, else None.
host: the host name associated with the entry, if any.
started: the creation time as a floating point number expressed in seconds since the epoch.
pid: the PID of the login process (like sshd, tmux, gdm-session-worker, …). On Windows and
OpenBSD this is always set to None.
'''
def get_users_info():
    users_info = psutil.users()
    users_names = []
    for user in users_info:
        # for each user add its name to the list
        users_names.append(user[0])
    return users_names

print(get_users_info())


def __get_info(s_type, s_name):
    # connect to openHardwareMonitor
    w = wmi.WMI(namespace="root\OpenHardwareMonitor")
    # get the computer sensors
    sensors = w.Sensor()
    for sensor in sensors:
        if sensor.SensorType==s_type:
            if sensor.Name == s_name:
                # return the value according to the type and name given
                return sensor.Value
```

```
def get_cpu_temp():
    # returns the CPU package temp
    return __get_info('Temperature', 'CPU Package')

def get_gpu_temp():
    # returns the GPU core temp
    return __get_info('Temperature', 'GPU Core')

def get_cpu():
    # returns the total CPU load
    return __get_info('Load', 'CPU Total')

def get_gpu():
    # returns the GPU core load
    return __get_info('Load', 'GPU Core')

def get_processes_info():
    # the list to contain all process dictionaries
    processes = []
    for process in psutil.process_iter():
        # get all process info in one shot (more efficient, without making separate calls for each
attribute)
        with process.oneshot():
            # get the process id
            pid = process.pid

            if pid == 0:
                # Swapper or sched process, useless to see
                continue

            # get the name of the file executed
            name = process.name()

            # get the time the process was spawned
            try:
                create_time = datetime.fromtimestamp(process.create_time())
            except OSError:
                # system processes, using boot time instead
                create_time = datetime.fromtimestamp(psutil.boot_time())
```

```
        try:
            # get the number of CPU cores that can execute this process
            cores = len(process.cpu_affinity())
        except psutil.AccessDenied:
            cores = 0

        # get the CPU usage percentage
        cpu_usage = process.cpu_percent()

        # get the status of the process (running, idle, etc.)
        status = process.status()

        try:
            # get the process "niceness" (priority)
            nice = int(process.nice())
        except psutil.AccessDenied:
            nice = 0

        try:
            # get the memory usage in mbytes
            memory_usage = process.memory_full_info().uss / 1000000
        except psutil.AccessDenied:
            memory_usage = 0

        #number of threads the process has
        n_threads = process.num_threads()

        # get the username of user spawned the process
        try:
            username = process.username()
        except psutil.AccessDenied:
            # os created this process
            username = "N/A"
        processes.append({
        'pid': pid, 'name': name, 'create_time': create_time,
        'cores': cores, 'cpu_usage': cpu_usage, 'status': status, 'nice': nice,
        'memory_usage': memory_usage, 'n_threads': n_threads, 'username': username,
        })
    return processes
```

```
def get_os():
    pc = wmi.WMI()
    os_info = pc.Win32_OperatingSystem()
    # returns the name of the operating system
    return os_info[0].Name

def get_processor():
    pc = wmi.WMI()
    # returns the processor name
    return pc.Win32_Processor()[0].Name.strip()

def get_hardware_info():
        d = { }
        # processor type
        d['Processor'] = get_processor()
        # os type and version
        d['OS'] = get_os()
        # cpu usage
        d['CPU'] = get_cpu()
        # cpu package temprature
        d['CPU temp'] = get_cpu_temp()
        # gpu usage
        d['GPU'] = get_gpu()
        # gpu core temprature
        d['GPU temp'] = get_gpu_temp()
        # virtual memory
        d['Memory'] = get_virtual_mem()
        # battery information, None for desktop computer
        d['Battery'] = get_battery_info()
        return d
```

WOL.py

```
from wakeonlan import send_magic_packet

def wake_device(mac, ip):
    send_magic_packet(mac,ip_address=ip)
    # Magic Packet Sent
```