Deijay: Automation Assistant

מגיש: דורון יעקבשטם

ת.ז: 325511954

בית ספר: עירוני ג' ע"ש מוטה גור

שם המנחה: רמי דיין

תאריך ההגשה: 12/6/2022

תוכן עניינים

[מבוא 3](#_Toc105903368)

[הרקע לפרויקט: 3](#_Toc105903369)

[תהליך המחקר: 4](#_Toc105903370)

[סקירת המצב הקיים בשוק: 4](#_Toc105903371)

[חידושים בפרויקט שלי: 5](#_Toc105903372)

[סקירת ספרות: 5](#_Toc105903373)

[פרוטוקול TCP: 5](#_Toc105903374)

[Python sockets: 5](#_Toc105903375)

[Speech Recognition: 5](#_Toc105903376)

[אתגרים מרכזיים: 5](#_Toc105903377)

[בעיות איתם התמודד התלמיד: 6](#_Toc105903378)

[הסיבות לבחירת הנושא ומוטיבציה לעבודה: 6](#_Toc105903379)

[על איזה צורך הפרויקט עונה? 6](#_Toc105903380)

[הצגת פתרונות לבעיה: 6](#_Toc105903381)

[מבנה / ארכיטקטורה של הפרויקט 6](#_Toc105903382)

[הצגת הפתרון המוצע והסיבות לבחירתו: 6](#_Toc105903383)

[Top-down Level Design: 7](#_Toc105903384)

* [מערכת ממשק משתמש: 7](#_Toc105903385)
* [מערכת מילוי בקשות המשתמש: 7](#_Toc105903386)
* [מערכת מעקב ובקרה של התוכנה: 7](#_Toc105903387)

[מחלקות ויחידות הפרויקט: 8](#_Toc105903388)

[צד הלקוח: 8](#_Toc105903389)

[פעולות חיצוניות בצד הלקוח: 12](#_Toc105903390)

[תיאור אלגוריתם הלקוח: 13](#_Toc105903391)

[יחידות הלקוח, והסברים: 13](#_Toc105903392)

[צד השרת: 13](#_Toc105903393)

[ LogRecordSocketReceiver 13](#_Toc105903394)

[ LogRecordStreamHandler 14](#_Toc105903395)

[ LoggingHandlerThread 14](#_Toc105903396)

[ ClientHandlerThread: 14](#_Toc105903397)

[פעולות חיצוניות בצד השרת: 15](#_Toc105903398)

[תיאור אלגוריתם השרת: 16](#_Toc105903399)

[מבני נתונים: 17](#_Toc105903400)

[ארכיטקטורת רשת – פרוטוקול תקשורת: 18](#_Toc105903401)

[מטרת השירות: 18](#_Toc105903402)

[אופן תקשורת: 18](#_Toc105903403)

[סוגי הודעות: 18](#_Toc105903404)

[מבנה / אופן התקשורת: 18](#_Toc105903405)

[פרוטוקול תעבורה: 18](#_Toc105903406)

[ממשקים למערכות אחרות: 18](#_Toc105903407)

[תיקיות וקבצים: 19](#_Toc105903408)

[דוגמה לשימוש בקובץ .txt לשמירת נתונים - 19](#_Toc105903409)

[ספריות חיצוניות: 20](#_Toc105903410)

[מדריך למשתמש 20](#_Toc105903411)

[הוראות התקנה - לקוח: 20](#_Toc105903412)

[הוראות התקנה - שרת: 21](#_Toc105903413)

[מידע כללי על קבצים: 21](#_Toc105903414)

[ממשק משתמש מינימלי - אינדיקציות: 21](#_Toc105903415)

[הוראות שימוש: 22](#_Toc105903416)

[מדריך למפתח 23](#_Toc105903417)

[DeijayMain.py: 23](#_Toc105903418)

[משתנים חשובים: 23](#_Toc105903419)

[תוכן: 23](#_Toc105903420)

[voiceOperator.py: 32](#_Toc105903421)

[משתנים חשובים: 32](#_Toc105903422)

[mouseControl.py: 33](#_Toc105903423)

[משתנים: 33](#_Toc105903424)

[dataHandlingServer.py: 34](#_Toc105903425)

[משתנים חשובים: 34](#_Toc105903426)

[רפלקציה 40](#_Toc105903427)

# מבוא

## הרקע לפרויקט:

על מנת להבטיח שאשמור על עניין בפיתוח הפרויקט לאורך זמן רב, החלטתי לבחור ברעיון שיהיה לי בו שימוש אישי ונרחב בחיי היום-יום. בשל כך, פרויקט הסיום הנ"ל אינו עונה על צורך קריטי וספציפי – אלא אמור לעזור לשימוש במחשב בחיי היום-יום להפוך ליותר פשוט, מהיר, וחלק. עם צורת חשיבה זו, החל החיפוש שלי אחר מקורות של אי-הנוחות או הפרעות כלשהן, ולאחר מציאה של מקור שכזה החלה המחשבה על מציאת פתרון. לאחר פרק זמן כלשהו, מצאתי כי אני מבצע פעולות שחוזרות על עצמן בצורה ידנית על המחשב מספר רב של פעמים, פעולות אשר הייתי יכול להגיד לחבר לבצע עבורי כמו פתיחת עמוד או אפליקציה מסוימת והכנסת פרטי משתמש, או פתיחת משחק וידאו וכניסה ללובי, או העברה לפרק הבא של סדרה ללא צורך לקום מן הכורסא... באותו הרגע חשבתי על הרעיון: יישום אוטומציה גמיש - כלומר, לא אוטומציה קשיחה אלא פעולות מגוונות בסדר שונה, ללא צורך בכתיבת קוד של רובוט מאת המשתמש. הרעיון מטבעו היה מוכרח להיות יישום רקע, "בלתי נראה" ככל הניתן לאחר שלבי הכנת סדרי פעולות (הרחבה על שלבי ה"הכנה" תבוא בהמשך הספר), אשר יתרגם פקודות קוליות שנבחרות ע"י המשתמש לסדרת פעולות שיקבעו גם כן על ידי המשתמש, ואשר יבוצעו במהירות וביעילות ככל הניתן על ידי המחשב עם דגש על נוחות וחוסר הפרעה למשתמש בעת השימוש החוזר בפעולות הללו.

## תהליך המחקר:

### סקירת המצב הקיים בשוק:

לאחר העלאת הרעיון – יצירת יישום אוטומציה (בוט גמיש) אשר יאפשר שליטה קולית נוחה במחשב, ביצעתי מספר חיפושים בגוגל אחר מוצרים ואף תכנות "חובבניות" שנוצרו על ידי מתכנתים שחשבו על אותו הרעיון כמוני. מטרתי הייתה לחפש האם קיים כבר מימוש שעונה על כל הצרכים שהעליתי (שכן במצב כזה הייתי משתמש בו וחושב על רעיון חדש לפרויקט), או האם יש ברעיון שלי משהו שעשוי לחדש ולתרום. המחקר בנוסף עזר לי להבין בדיוק איזה יישומים כדאי לי להכניס לפרויקט שלי, ואיזה יישומים כדאי לי להשאיר מחוץ אליו (אין טעם להשקיע זמן בפיתוח יישום שכבר קיים ומצליח בשוק).

בסוף סקירת המצב בשוק, הבנתי כי אין יישום אשר נותן למשתמש הלא-מתכנת את היכולת לשלב אוטומציה של פעולות גמישות במחשב דרך פקודות קוליות. רוב החלופות לאוטומציה בשוק ממוקדות בשימוש בסביבת עבודה כלשהי, ולא בעזרה עם פעולות שבחיי היום-יום. כפי שאציין ברשימה הבאה, תחום האוטומציה עד כה היה שמור למשתמשים אשר יודעים לתכנת רובוטים משלהם. עם זאת, בעת המחקר מצאתי מספר מוצרים מעניינים וחשובים בתחום השליטה הקולית:

* **VoiceComputer for Dragon:** היישום הנ"ל מציע שליטה כמעט מלאה בעכבר המחשב (ובהרחבה, במחשב עצמו) דרך פקודות קוליות פשוטות – מספרים! הוא מבצע זאת על ידי מספור כל האלמנטים האינטראקטיביים על המסך. כל שעל המשתמש לעשות זה להגיד את המספר, והמחשב ילחץ על האלמנט במסך. אמנם המוצר הנ"ל של Dragon מיועד עבור אלו שאינם מסוגלים, מסיבות בריאותיות, להשתמש בעכבר או במקלדת, עם זאת הדבר אינו מונע מלהשתמש במוצר בשביל שליטה מרחוק מדויקת ונוחה במחשב.  
  עם כל הנאמר לעיל, בנוסף לעובדה כי התכנה הזו עולה כסף, היא אינה מספקת את האוטומציה של פעולות אלא שימוש מרחוק במחשב.   
  במהלך ביצוע הפרויקט חשבתי על שימוש ברעיון של שליטה במחשב לפי אלמנטים שעל המסך, (שכן חיפשתי אלגוריתם שימושי בכל דפדפן ובכל אפליקציה חוץ מסוגי .exe) אך לאחר התעמקות בנושא הסתבר שהדבר מתבצע בDragon על ידי בינה מלאכותית בסדר גודל שאין ביכולתי לאמן בעצמי.
* **Windows Speech Recognition:** הכלי הנ"ל מאת Microsoft מגיע בחבילה של מערכת ההפעלה בעלת אותו שם ויוצר. למרות הקלילות שבהתקנה שלו בשל העובדה הזו, התכנה מאוד מוגבלת בטווח היישומים שלה מה שלא מאפשר שימוש בה על מנת ליצור או להשתמש באוטומציה משום סוג - גמיש או קשיח (תזכור: "קשיח" כלומר פעולות שנכתבו מראש לתוך הקוד, כמו תכנות בוטים).
* **פרויקטים של אוטומציה שפורסמו ברשת:** במעבר על מספר פרויקטים שכאלה, הבנתי את המציאות העכשווית של אוטומציה – תכנות רובוטים למטרות ספציפיות, לרוב על ידי מתכנתים או סטודנטים שעובדים בחברות ורוצים להעלות את היעילות של עצמם בעבודה או בלימודים. הפרויקטים שמצאתי שאכן היו קשורים לשיפור נוחות / "איכות חיים" (quality-of-life improvements) היו בעיקר פרויקטים קשיחים גם הם... כלומר התכנה כוללת בתוכה מענה למספר בקשות כמו חיפוש גוגל / פתיחת ערך בוויקיפדיה, או ניגון שיר מסוים, אך אם המשתמש רוצה לבצע אוטומציה לתהליך אישי לסדר פעולות שלו, אין לתכנה איך להיענות לו, שכן אין לה אלגוריתם שמאפשר לה "ללמוד" שירותים חדשים.

כמו כן, יש מספר כלים למתכנתים אשר מאפשרים ליצור אוטומציה (רובוטים) של המחשב כמו Selenium או Twin, אך זו אוטומציה קשיחה בלבד אשר עושה שימוש בכתיבת קוד ולכן אינה רלוונטית ליצירת המוצר שאני כיוונתי אליו.

## 

## חידושים בפרויקט שלי:

כיום כשמבצעים חיפוש על אוטומציה, מקבלים תוצאות חיפוש במתמקדות במתכנת אשר מסוגל ללמוד ממשקים כמו Selenium או Twin. הפרויקט שלי משלב בין שליטה קולית במחשב ליצירת אוטומציה של פעולות ללא צורך בכתיבת שורה אחת של קוד מצד המשתמש, או שימוש בידע של תכנות. Deijay מספק שמירת ויצירת סדרי פעולות ("רובוטים") אינטואיטיבית ופשוטה יחסית, בנוסף לשליטה קולית בעכבר ובמקלדת ושימוש בהם על מנת לתת למשתמש חופש תנועה בתחום הדפדפן ובתחום שולחן העבודה כפי שאראה בהמשך הספר.   
 במילים אחרות ובקצרה, Deijay מעניק שירות שכרגע פתוח רק לחלק מוגבל מאוד מהמשתמשים ברשת לקהל נרחב יותר ללא שום צורך בידע מוקדם, ואף מנסה להרחיב מעט על היכולות של השירות הקיים.

## סקירת ספרות:

כעת, יש צורך לפרט על מספר .כלים בהם עשיתי שימוש במהלך הפרויקט:

### פרוטוקול TCP:

Transmission Control Protocol, בקצרה TCP, הוא פרוטוקול בתקשורת נתונים הפועל בשכבת התעבורה של IP (Internet Protocol), ומגדיר את צורת העברת נתונים בתקשורת בין זוג מחשבים או תוכנות / אפליקציות. כאחד מבין מספר פרוטוקולים, TCP עושה שימוש בפרוטוקול הIP לצורך העברת נתונים. באמצעות הIP, הTCP מוודא את נכונותם של הנתונים, ומבצע אישור קבלת הנתונים או בקשה לשליחה מחדש שלהם אם גילה שהנתונים לא היו תקינים. אמנם פרוטוקול TCP נחשב איטי יחסית לאלטרנטיבות בשל הפעולות הנרחבות שמבצע, עם זאת הוא מתאים ומוצלח מאוד בשליחת כמות קטנה יחסית של נתונים עם דגש על סדר ובטיחות מעל מהירות.

### Python sockets:

דרך הממשק של socket אנו נבצע תקשורת בין השרת ללקוח. הממשק של socket הוא תעתיק של ממשק הsystem calls של יוניקס, כך שיתאים לסביבת העבודה של פייתון – שפה מונחית עצמים. הפונקציה socket() מחזירה אובייקט מסוג socket אשר מחזיק במתודות שמיישמות קריאות מערכת. חשוב לציין כי הפרמטרים של המתודות הם ב"רמה גבוהה" מאשר של קריאות המערכת בשפת C.

### Speech Recognition:

זהו הכלי אשר באמצעותו הפרויקט מתפקד: זיהוי מילים / משפטים. שלא לבלבל עם "זיהוי קולי" / Voice Recognition, שכן זו טכנולוגיה שונה לחלוטין אשר מתמקדת בזיהוי והפרדה של צליל מסוים מהשאר. ב1962 הציגו את מכונת הShoebox אשר הצליחה לזהות סכום מלהיב של 16 מילים. כיום, קיימים ממשקים שונים ברחבי הרשת שמסוגלים לזהות מספר רב הרבה יותר של מילים ומשפטים ובדיוק גבוה הרבה יותר משהיו יכולים לדמיין לפני כל כך הרבה זמן. הטכנולוגיה מוחזקת ע"י טכניקות של בלשנים ואלגוריתם הArtificial Neural Network - "רשת נוירונים": מודל מתמטי אשר מנסה לחקות תהליכים מוחיים על מנת לשמש בלמידת מכונה.

## אתגרים מרכזיים:

### בעיות איתם התמודד התלמיד:

* לדייק בזיהוי רצון המשתמש: הדגשים בפרויקט שלי הם על נוחות וחוסר הפרעה למשתמש. אמנם יחידת הspeech recognition מסייעת לי בתרגום ה"רעש" של המיקרופון למילים, אך האלגוריתם על פיו התכנה תקבע איזה פעולות רוצה המשתמש לבצע, ובאיזה סדר, נקבע על ידי ככותב הקוד. דיוק רצון המשתמש כולל חשיבה על כלל האפשרויות שהוא עשוי לבקש, בכלל סדרי הפעולות. למשל אם הוא מבקש משהו, ומבצע ביטול של הבקשה באותה הנשימה על התכנה לשים את הביטול בעדיפות גבוהה יותר מעל לבקשה הראשונה. האתגר העיצובי והמחשבתי במילוי אחר סדר הפעולות עובר לסדר גודל לא זניח כאשר מגוון הפעולות הופך לנרחב ביותר, וכאשר המילים שמשומשות למספר פעולות נשמעות דומה ואף זהות אחת לשנייה לפעמים.
* מציאת דרך כמה שיותר טובה לגרום לתכנה להצליח לבצע כל פעולה שהמשתמש ירצה שתבצע (מחוץ לקבצי .exe) ללא "למידת מכונה": הקושי באתגר הזה כולל גם את החשיבה על כל הפרטים של הדרך שבה הפתרון יתבטא בקוד, וגם בייעול, ושימוש בדרכים שונות על מנת לקצר תהליכים / לגרום להם להיות נוחים יותר עבור המשתמש.   
  בנוסף, כיצד הופכים תכנה קשיחה לתכנה גמישה שיכולה לבצע פעולות "חדשות" בצורה יעילה וקלה להבנה? בין אם זה על ידי משתמש שרוצה להבין את הדרך הטובה ביותר לשימוש ביישום, או על ידי מפתח שניגש לקוד בפעם הראשונה, האלגוריתם צריך להיות ברור, נוח לעבודה, ופתוח לשיפורים על ידי מפתחים אחרים.

### הסיבות לבחירת הנושא ומוטיבציה לעבודה:

בפעם הראשונה ששמעתי על כך שנבצע פרויקט, ריגש אותי לחשוב שאוכל לעבוד על משהו חדש, לאתגר את עצמי בחשיבה על בעיה שלא מצאתי עליה פתרונות בפורומים או באתרי מידע, ולענות על צורך קיים ואף להשתמש בתוצר בעתיד לאחר שהוא הושלם.

### על איזה צורך הפרויקט עונה?

הפרויקט פותר בעיה של בזבזנות וחוסר יעילות בפעולות חזרתיות, ובו זמנית עונה על צורך של שליטה במחשב ממרחק מסוים (כמובן, המרחק הנ"ל תלוי ברגישות המיקרופון של המשתמש)

## הצגת פתרונות לבעיה:

על מנת לדייק בזיהוי רצון המשתמש, היו רק 2 פתרונות שנשמעו יעילים תחילה – שימוש באלגוריתמים של לימוד מכונה, או שימוש במילות מפתח. החלטתי לבסוף לעשות שימוש במילות מפתח. ארחיב על בחירתי בפתרון זה מעל "לימוד מכונה" בפרק הבא.

על מנת לגרום לקוד "קשיח" להתנהג בצורה שונה, יש לכתוב אותו בצורה כזו שמאפשרת לסדר אותו מחדש באופן פשוט ונוזלי. אתאר איך הדבר מתבצע בפרק הבא: "מבנה/ארכיטקטורה של הפרויקט".

# מבנה / ארכיטקטורה של הפרויקט

## הצגת הפתרון המוצע והסיבות לבחירתו:

על מנת לדייק בזיהוי רצון המשתמש, היו רק 2 פתרונות שנשמעו יעילים תחילה – שימוש באלגוריתמים של לימוד מכונה, או שימוש במילות מפתח. לימוד מכונה היה עדיף לדעתי, אך אימון של בינה מלאכותית מכל סוג דורש כמות מידע מסוימת. בנוסף, אחד החסרונות של למידת מכונה הוא הניחושים שמבצעת המכונה. אמנם כאשר מספקים למכונה כמות מספקת של מידע בתהליך האימון, בעיית סיכויי הכשלון בהערכה שלה יורדים משמעותית... לאחר חשיבה ממושכת וחיפוש באינטרנט, לא הגעתי לפתרון לבעיית זיהוי כוונה של אדם לפי טקסט בלבד בצורה שתשמש אותי, ולכן החלטתי על שימוש במילות מפתח על מנת לסווג את רצונות המשתמש. יש לשיטה הזו חסרונות כמובן: המשתמש לא ידע את מילות המפתח או לחלופין הוא יצטרך לזכור אותן – מה שפוגע משמעותית בנוחות השימוש. בנוסף מילות מפתח עלולות להיות משומשות למספר פעולות יחדיו על ידי המשתמש. בשל החסרונות שנובעים מן הבחירה הזו אתגרים עיצוביים נוספים עלו בתהליך הקידוד, אך אלו נפתרו כמעט לחלוטין לטעמי בשלב הסופי של הפרויקט. החיסרון המשמעותי היחיד לדעתי שנותר, הוא שעל מנת להוסיף מילות מפתח חדשות (אם מפתח מנסה להרחיב את התכנה) יש להשקיע כמות מסוימת של מחשבה בנוגע לעדיפות ששמים עליהן ולדאוג שהצירופים שלהן לא מתלכדים עם אלה של פעולות אחרות.

התפקוד הבסיסי ביותר של המוצר הסופי הוא שליטה בפעולות העכבר / המקלדת באמצעות פקודות קוליות "קשיחות" (חלקן מקבלות קלט, אך הפעולה שמתבצעת נותרת זהה מטבעה). באמצעות אבני הבניין הללו, הפרויקט מסוגל לבצע את התפקוד העיקרי שלו: אוטומציה גמישה. למשתמש יש את היכולת לבנות סדר פעולות ולקרוא לו בשם כלשהו. כאשר ירצה המשתמש לחזור על סדר הפעולות הנ"ל, הוא יגיד את השם שנבחר בצירוף עם מילת המפתח בצורה הבאה: "protocol [chosen name]". משום שהפעולות ה"קשיחות" כוללות כמעט את כל הפעולות של העכבר ומספר רב מתוך הפעולות של המקלדת (ישנם מקלדות / עכברים מיוחדים עם פעולות מעבר לפעולות הבסיסיות, ובמקלדת ישנם קיצורי דרך שהחלטתי שלא רלוונטיים לרוב המשתמשים)

## Top-down Level Design:

הפרויקט משלב מספר מערכות שמתפקדות במקביל: מערכת ממשק המשתמש, מערכת מילוי בקשות המשתמש, ומערכת מעקב ובקרה של התוכנה.

* מערכת ממשק משתמש: מערכת הממשק מהווה את התיווך בין המשתמש לחלק בפרויקט אשר מבצע את מטרות התוכנה.   
  המערכת כוללת את לולאת הmain בצד הלקוח, ואחראית על עדכון הגרפיקה ועיבוד הקלט הקולי של המשתמש לטקסט ע"י API (נעשה בתוך יחידת voiceOperator), הפרדת הטקסט המעובד שהתקבל לבקשות יחידות והכנסתן למבנה נתונים queue, לשימוש על ידי מערכת מילוי בקשות המשתמש.
* מערכת מילוי בקשות המשתמש: מערכת מילוי הבקשות מהווה את עיקר הפרויקט ומשתרעת על מספר יחידות בצד הלקוח וגם בשרת עצמו, אשר בעל גישה לקבצי ה.txt עליהם שמור מידע החיוני לפעולות האוטומציה.   
  המערכת כוללת מספר threads בצד השרת ובצד הלקוח.   
  המערכת הנ"ל אחראית על מילוי בקשות המשתמש במהירות ולפי סדר, באופן קריא וקל להבנה. בקשות המשתמש יכולות לכלול פעולות עכבר / מקלדת, פעולות של ביצוע סט אוטומציה שמור (פרוטוקול) או שמירת סט פרוטוקול חדש. על מנת לבצע כל פעולת אוטומציה, המערכת שואבת את המידע הרלוונטי ממאגר הנתונים (.txt) או לחלופין מזריקה את המידע לתוכו. משום שרק השרת בעל גישה לקבצי ה.txt, על מנת לבצע פעולות פרוטוקול הלקוח מבצע תקשורת TCP עם השרת בעת הצורך.
* מערכת מעקב ובקרה של התוכנה: מערכת המעקב והבקרה קיימת לטובת המפתח בעיקר לצורך תיקון באגים, אך היא גם שימושית למשתמש שרוצה להבין כיצד הפרויקט עובד, ואף לעקוב אחר פעולותיו.   
  המערכת כוללת קריאות לוגים ושמירת קבצים שלהם על הלקוח וכן על השרת כגיבוי למפתח, ברמת DEBUG ומעלה, מפורטות ורבות לאורך כל התוכנה אשר מתארות את סדר הפעולות של התוכנה והאופן בו הפרויקט מתפקד. יש קריאות לרמות INFO ERROR CRITICAL, פחותות במספרן לעומת קריאות הDEBUG, אשר מספקות מידע של ביצוע בקשות או כשלון בביצוע, וסיבת הכשלון. יש צורך במערכת שכזו בפרויקט שכזה, שכן בשל ההישענות על מילות מפתח, יש ערך רב למפתח במידע על הדרך בה הפרויקט מסווג ומבצע בקשות (כמו למשל: מציאת מילים מאוד דומות למילות מפתח שאינן תואמות, כמו למשל quik ו-click). בתור המפתח של הפרויקט, אני יכול להגיד בביטחון שכתיבת הפרויקט הייתה קרובה לבלתי-אפשרית ללא המעקב הסיזיפי אחר פעולות הקוד, שכן קשה מאוד לדעת היכן התבצעה הטעות, או האם מקור הטעות מגיע מAPI או מהקוד שלי. מעבר לשימוש האדיב למפתח, שימוש בלוגים של INFO יכול לעזור למשתמש לעקוב אחר דרך השימוש שלו בתוכנה, אם המידע מעניין אותו.   
  אני מחשיב את המעקב והבקרה כמערכת משל עצמה שכן כל לקוח מנהל עם השרת תקשורת TCP של מידע הלוגים שלו, ועבור כל לקוח השרת צריך לספק thread נוסף ותמידי (עד שהלקוח מתנתק) על מנת להתמודד עם המעקב שלו. המערכת מספקת שימוש נרחב למפתח, שימוש כלשהו למשתמש, וצורכת משאבים (לא-זניחים יחסית) מן המחשב.

## מחלקות ויחידות הפרויקט:

### צד הלקוח:

בצד הלקוח ישנם 4 מחלקות:

* **ProtocolController**: מקבל פירוט מלא בטבלה, שכן כל מטרתו ליצור אובייקט על מנת לחסוך מספר שורות קוד ולהפוך אותו לקריא ליותר בו-זמנית.  
  תכונות:
  + protocol\_recording: רשימה ריקה. תוך בקשת הקלטה, הרשימה מתחילה להתמלא בבקשות אשר מבקש המשתמש. לאחר שמירת הפרוטוקול על ידי המשתמש, הרשימה הזו מתרוקנת.
  + protocol\_recording\_switches: מילון בעל זוג מפתחות מסוג מחרוזת, בעלי ערך בוליאני צמוד. התכונה הזו שולטת בזרימת פעולות התוכנית לפי סטטוס הבקשות שקשורות לשמירת סטים חדשים של אוטומציה.
* **RecordingDot**: שימוש המחלקה הזו סובב סביב השימוש של האובייקט שלה, אשר נבנה בפעם היחידה בתחילת התוכנית ואשר מהווה את המראה הגרפי של הפרויקט – הנקודה החצי-שקופה שמופיעה מדי 10 שניות בצד ימין של המסך. המחלקה יורשת מtkinter.Tk (ספריה חיצונית tkinter), והבנאי שלה יוצר אובייקט מאותו הסוג. המחלקה הנ"ל מהווה תקשורת חסרת-הפרעות יחסית ומודיעה למשתמש מתי התוכנה מקשיבה לו.  
  תכונות:
  + diameter: קוטר המעגל, int.
  + color: צבע המעגל, מחרוזת.
  + Visible: מצב הנראות שלו, בוליאני.
* **Speaker**: מחלקה שמטרתה היא תקשורת נוספת עם המשתמש, הפעם בצורת דיבור text-to-speech (שימוש בספריה חיצונית pyttsx3). יש שימוש במחלקה כאשר יש צורך בלהודיע למשתמש הודעה מסוימת, למשל אם אירעה תקלה שהייתה קשורה לקלט שלו (למשל: פקודה קולית שלא זוהתה) אשר ביכולתו לתקן (למשל: לדבר בצורה ברורה יותר).  
  תכונות:
  + speaker: אובייקט של ספריית pyttsx3 (text-to-speech)
  + sex: קובע את קול הדובר (זכר/נקבה). מחרוזת.
* **RequestHandlerThread**: המחלקה הגדולה ביותר בפרויקט, המרכזת של רוב התפקודים של התוכנית והמתקשרת עם השרת, תוך ירושה ושימוש בthreading.Thread.   
  מטרת המחלקה – לסווג את בקשות המשתמש לשירותים קיימים בתוכנה ומתן מענה מתאים, זה באופן כמה שיותר נקי ונוח לקריאה מצד המפתח.   
  תכונות:
  + noError: ערך בוליאני. הופך לFalse במקרה שאירעה תקלה, מה שמשנה את זרימת התוכנית.
  + job: ערכה ההתחלתי None, אך משתנה לשל reference לאחת ממתודות המחלקה בשימוש. חוזרת לNone לאחר כל שימוש.
  + current\_request: ערך מחרוזת היוצא מן התור request\_queue.
  + request\_queue: תור הבקשות המועבר כreference לאובייקט של המחלקה ע"י פונקציית main
  + sock: אובייקט socket השומר על החיבור לשרת
  + timer: שומר זמני הרצה, רלוונטי למפתח.
  + program\_switch: הערך הבוליאני האחראי על הלולאה הראשית, מועבר בreference לאובייקט של המחלקה.
  + pcl\_tracker: האובייקט האחראי לניהול סטטוס בקשות הפרוטוקולים/שמירת אוטומציות חדשות. מועבר גם הוא בreference לאובייקט של המחלקה.
  + keyword\_sorter: מילון המתאים בין reference של מתודות המחלקה למילות מפתח. אלגוריתם רץ על המילון הזה על מנת למצוא את השירות המתאים לבקשת הלקוח.

הערה לגבי טבלאות המתודות והפונקציות:

לכל פונקציה שנכנסת בתוך מחלקה, ההנחה היא כי חלק מהקלט שלה הוא תכונות המחלקה בצורה בלתי-נפרדת, ולכן לא יוזכרו תכונות מחלקה בקלט של הטבלה.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם מחלקה: | שם פעולה: | מטרת/תיאור פעולה: | קלט: | פלט: |
| ProtocolController | \_\_init\_\_ | יצירת אובייקט המחזיק בזוג תכונות, בעזרתן ניתן לבצע שמירת פרוטוקול חדש. |  | אובייקט המחזיק ברשימה ריקה, ובמילון בעל 2 מפתחות המתאימים לערכים בוליאנים: "האם מקליטים פרוטוקול כעת?", "האם הבקשה היא לבצע שמירת פרוטוקול?" |
| RecordingDot | \_\_init\_\_ | יצירת אובייקט מסוג tkinter.Tk אשר מראה אינדיקציות למשתמש | יש שימוש בגודל מסך המתקבל משימוש בספריה חיצונית user32 | אובייקט הגרפיקה העיקרית לפרויקט |
| toggle\_visibility | לאפשר לגרפיקה "להיעלם" מהמסך ולהיראות שוב בקריאה חוזרת לפעולה | self.visible תכונה של האובייקט |  |
| Speaker | speak | הודעת מידע ספציפי למשתמש בעת תקלה או בעת הצלחה של פעולות מסוימות | טקסט כלשהו | text-to-speech, פלט קולי של קריאת הטקסט שבקלט |
| RequestHandlerThread | run | הפעולה הראשית של הthread |  |  |
| receive\_from\_server | מקבלת מידע מהשרת | מידע מהשרת (רשת) | מחזירה את המידע |
| handle\_server\_response | שימוש במידע שהתקבל מן השרת על מנת למלא את בקשת המשתמש | סוג בקשה, סוג מידע. | פלט קולי של מידע שהתקבל מן השרת.  מחזירה אחד מן העוקבים:  queue.Queue שמכיל את המידע מן השרת, או מידע מן השרת ללא עריכה. |
| send\_then\_handle | שליחת בקשת עבודה לשרת וקבלת השירות ממנו | מפתח בקשה, בקשה,  מידע מהשרת. | שליחה לשרת של: מפתח בקשה, אורך בקשה, ולבסוף בקשה.  מחזיר:  queue.Queue |
| start\_rec\_protocol | התחלת הקלטה של פרוטוקול, כלומר מעקב אחר הפעולות שהמשתמש מבצע באמצעות התוכנית |  | פלט קולי |
| save\_rec\_protocol | סיום הקלטה של פרוטוקול, שלב מקדים לשמירת הפרוטוקול. |  |  |
| execute\_protocol | ביצוע פרוטוקול | הפרוטוקול בצורת תור queue.Queue | הכנסת הבקשות שמרכיבות את הפרוטוקול בתור הבקשות המרכזי |
| move\_mouse\_pointer | תיווך בין יחידת mouseControl ליחידת הלקוח המרכזית, מעבר לפעולה mouse\_movement | מידע מן המשתמש |  |
| save\_location\_entry | הכנסת הפרמטרים לsend\_then\_handle עבור בקשת "שמירת מיקום עכבר" | מיקום העכבר, מידע מן המשתמש |  |
| fast\_travel | תנועת העכבר לקואורדינטות ספציפיות על המסך | מידע מן המשתמש |  |
| click\_mouse | שימוש בפעולת הלחיצה של העכבר | מידע מן המשתמש |  |
| scroll\_mouse | שימוש בפעולת הגלילה של העכבר | מידע מן המשתמש |  |
| keyboard\_type | שימוש במקלדת למטרת כתיבה | מידע מן המשתמש (הטקסט שייכתב) |  |
| press\_enter | לחיצה על מקש מקלדת |  |  |
| press\_left\_arrow | לחיצה על מקש מקלדת |  |  |
| press\_right\_arrow | לחיצה על מקש מקלדת |  |  |
| press\_up\_arrow | לחיצה על מקש מקלדת |  |  |
| press\_down\_arrow | לחיצה על מקש מקלדת |  |  |
| press\_backspace | לחיצה על מקש מקלדת |  |  |
| press\_tab | לחיצה על מקש מקלדת |  |  |
| back\_history\_browser | בדפדפן, חזור לעמוד האחרון |  |  |
| forward\_history\_browser | בדפדפן, עבור לעמוד הבא |  |  |
| open\_new\_tab | בדפדפן, פתח לשונית חדשה |  |  |
| open\_new\_window | בדפדפן, פתח חלון חדש |  |  |
| next\_tab\_browser | בדפדפן, עבור ללשונית הבאה |  |  |
| close\_window | סגור את החלון הפתוח כעת (alt+f4) |  |  |
| close\_tab | בדפדפן, סגור את הלשונית הפתוחה כעת |  |  |
| add\_bookmark | בדפדפן, הוסף את העמוד הנוכחי למועדפים |  |  |
| find\_in\_browser | בדפדפן, פתח את סרגל חיפוש המילים |  |  |
| open\_history\_browser | בדפדפן, בפתח היסטוריה |  |  |
| refresh\_browser | בדפדפן, אתחל את העמוד מחדש |  |  |
| search\_bar\_browser | בדפדפן, בוחר בסרגל החיפוש. |  |  |
| play\_pause\_video | התחל/עצור תהליך של סרטון. |  |  |
| junk | נועדה שיקראו לה כאשר המשתמש התחרט על הבקשה האחרונה (שטרם התבצעה) |  |  |
| shutdown\_program | סגור את תוכנית הפרויקט |  | שינוי ערך בוליאני program\_switch |

### פעולות חיצוניות בצד הלקוח:

|  |  |  |  |
| --- | --- | --- | --- |
| שם פעולה: | מטרת פעולה: | קלט: | פלט: |
| main | הפעולה הראשית |  |  |
| monkey\_patched\_circle  |  (מתבצעת קביעה בתוכנית)  =  Canvas.create\_circle | לעשות override לפונקציית יצירת אליפסה, כך שיהיה ניתן ליצור באמצעותה עיגול בפשטות | x, y, r – מיקום מרכז המעגל וגודל הרדיוס. | self.create\_oval(x-r, y-r, x+r, y+r, \*\*kwargs)  הפונקציה מחזירה פונקציה נוספת (ללא קריאה!), ואיתה עושים override. |
| speak | text-to-speech על מנת לספק אינדיקציות למשתמש במצבים מסוימים | טקסט | text-to-speech פלט קולי |
| setup\_logging | מגדיר את הקונפיגורציה של הlogger | fmt  הפורמט המכתיב את דרך תצוגת המידע |  |
| main\_setup | יצירת חיבור socket עם השרת, והגדרת המשתנים סביבם התוכנה מתפקדת |  | מחזירה את הבאים:  socket, ProtocolController, RecordingDot, Queue, program\_switch (=True) |
| from\_mic | אחראית יחד עם listen\_mic על קליטה ועיבוד של מידע הזורם דרך המיקרופון (speech-to-text) באופן מסונכרן עם הגרפיקה. | גרפיקה איתה קיים סנכרון, אודיו מהמיקרופון | מחזירה טקסט |
| listen\_mic | פונקציה שנועדה להיות משומשת בתור Thread בתוך from\_mic.  מבצעת את קליט המידע הזורם דרך המיקרופון תוך סנכרון עם נראות הגרפיקה | אובייקטים של המיקרופון, של מתרגם הspeech-to-text, ושל הגרפיקה. בנוסף תור המכיל את המידע הנכנס מן המיקרופון |  |
| mouse\_location\_save\_data | הפונקציה מבצעת חלק אחד מבקשת המשתמש לsave mouse location. (החלק מתבטא בפלט>>>) | מידע מן המשתמש (שם מיקום), ומיקום העכבר. | מחזירה סט של 2 נתונים המופרדים ע"י תו הפרדה @: שם מיקום, וקואורדינטות על המסך. |
| mouse\_location\_get\_entry | שאיבת שם / מפתח המיקום אליו מבקש המשתמש לזוז מתוך טקסט הבקשה | מידע מן המשתמש | מחזירה את שם / מפתח המיקום. |

### תיאור אלגוריתם הלקוח:

### יחידות הלקוח, והסברים:

* DeijayMain.py: היחידה העיקרית של הפרויקט, מכילה את כל התפקודים המרכזיים ומבצעת את כל אלגוריתם הלקוח לעיל.
* voiceOperator.py: אחראית על תפקודי הspeech-to-text וה-text-to-speech. בנוסף היחידה הזו מנהלת את סנכרון הגרפיקה עם תפקודים אלו. למעשה היא מתווכת בין היחידה העיקרית למשתמש.  
  משמעות סנכרון הגרפיקה: הגרפיקה שבחרתי היא נקודה חצי-שקופה שנעלמת מדי 10 שניות ומופיעה מחדש בצד ימין העליון של המסך. הסנכרון מתבצע כך שכאשר הנקודה נראית התוכנה אוספת מידע מן המיקרופון, וכאשר הנקודה אינה נראית התוכנה אינה פנויה להקשבה למיקרופון. כלומר הגרפיקה מהווה אינדיקציה למשתמש – מתי התוכנה מקשיבה, ומתי היא לא.
* mouseControl.py: יחידה שפועלת כמעין "מחסן" למספר פונקציות אשר נקראות ע"י מערכת מילוי הבקשות ביחידה המרכזית.

## צד השרת:

בצד השרת ישנם 4 מחלקות:

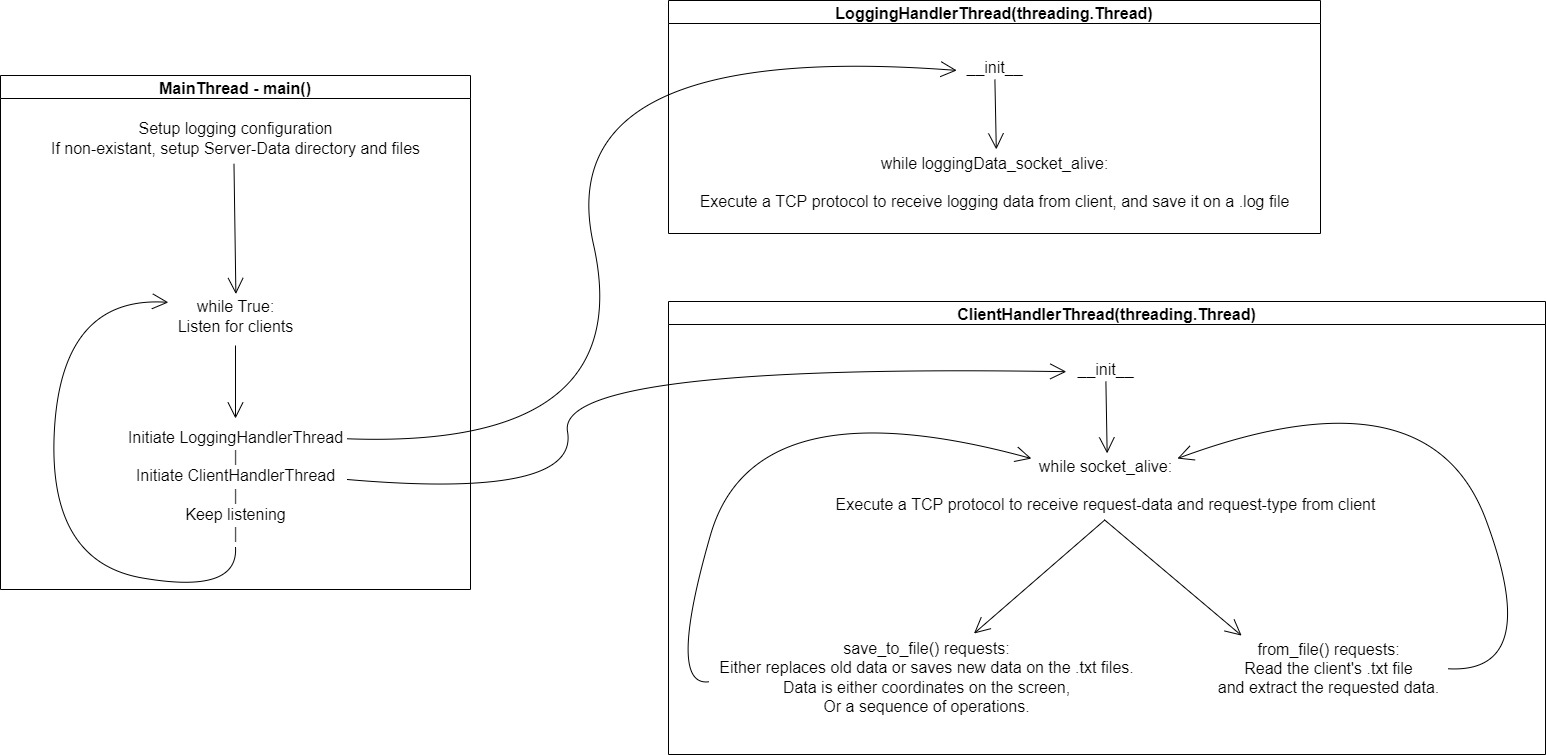
* LogRecordSocketReceiver: המחלקה יורשת מsocketserver.ThreadingTCPServer (שימוש בספריה חיצונית socketserver). מטרת המחלקה ליצור ולשמור על חיבור בפרוטוקול TCP בין הלקוח לשרת, אשר מזרים מידע מסוג log באופן כמעט-תמידי.  
  בבניית האובייקט, המחלקה עושה שימוש במחלקה נוספת אשר ארחיב עליה בפסקה הבאה.  
  תכונות:
  + abort: ערך int המייצג ערך בוליאני (0 או 1). האם היה ניתוק socket?
  + Timeout: ערך int המייצג ערך בוליאני (0 או 1). האם היה timeout?
  + logname: הערך בdefault הוא None, אך ניתן להגדיר ערך לתכונה בהגדרת האובייקט.
* LogRecordStreamHandler: המחלקה יורשת מsocketserver.StreamRequestHandler (שימוש בספריה חיצונית socketserver), והיא חסרת בנאי. כל logger צריך קונפיגורציה כלשהי לדרך בה הוא ינהג עם המידע שיוצר. קונפיגורציה זו נעשית על ידי handler. במידה ולא ניתן לlogger קונפיגורציה עדיין, המחלקה הזו תיצור לוגים עם הקונפיגורציה שהגיעה עם המידע דרך הsocket. עם זאת, כפי שאסביר בפסקה הבאה, אכן יצרתי קונפיגורציה לlogger של השרת.  
  תכונות: רק תורשתיות.
* LoggingHandlerThread: המחלקה יורשת מthreading.Thread (שימוש בספריה חיצונית threading). מטרת המחלקה ליצור אובייקט Thread הבנוי ספציפית עבור שירות logging. אכן כשקוראים לאובייקט עם פונקציית start בתחילת קשר עם לקוח חדש, הוא מבצע קונפיגורציה לפי כתובת הIP של המשתמש (הIP מהווה מעין "שם משתמש" בפרויקט) וקורא לאובייקט LogRecordSocketReceiver על מנת להתמודד עם זרם המידע שמגיע מן הלקוח.  
  תכונות:
  + client\_address: שומר את כתובת הIP של הלקוח.
* ClientHandlerThread: המחלקה השנייה בגודלה בפרויקט, זה משום שהיא אחראית על כל פעולות השרת: שמירה ומשיכה של מידע מקובץ לפי IP של המשתמש, ותקשורת מסוג TCP עם הלקוח.   
  חשוב לציין כי מעבר לשליחת אורך ההודעה לפני שליחת ההודעה, בתקשורת הזו יש צורך בשליחת קוד בעל 4-תווים לפני שליחת אורך המידע. הקוד מודיע לשרת על סוג הבקשה שתשלח בקרוב: svlc, gtlc, svpc, gtpc: save/get location, save/get protocol.  
  תכונות:
  + sock: אובייקט socket האחראי על החיבור ללקוח.
  + client\_address: שומר את כתובת הIP של הלקוח.
  + current\_job\_data: לפני קבלת ערך מן הלקוח (רשת), הערך נקבע כNone.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| שם מחלקה: | שם פעולה: | מטרת פעולה: | קלט: | פלט: |
| LogRecordStreamReceiver | serve\_until\_stopped | לשמור על חיבור הTCP עם הLogger של הלקוח.  משמע: בדיקה חוזרת ונשנית של סטטוס הsocket ושימוש בhandler כל עוד החיבור עדיין קיים. |  |  |
| LogRecordStreamHandler | handle | הhandler של LogRecordStreamReceiver.  מטפל במספר בקשות: כל אחת מצופה להיות באורך 4-בייטים (אורך), ומיד לאחר מכן מצופה LogRecord בפורמט pickle. | מידע מן הלקוח (רשת) לפי שתואר ב"מטרת הפעולה" |  |
| unPickle | פריקת ערך הpickle למידע שמיש (ההכנסה לpickle נעשית על מנת להעביר אותו בsocket) | מידע מסוג pickle | מחזירה את המידע לאחר פריקתו |
| handleLogRecord | אם לא הוחלט על logger, ואם מצוין במידע של הlog שם של logger, משתמשים בו על מנת לעבוד עם המידע. | log record שמגיע מהלקוח (רשת) |  |
| LoggingHandlerThread | run | הפעלת שירות הlogging תוך שימוש במחלקות ובפעולות שלעיל. |  |  |
| ClientHandlerThread | run | ניהול ומתן מענה לבקשות שמגיעות מן המשתמש | מידע מן הלקוח (משתמש) | שולחת מידע חזרה אל הלקוח |
| receive\_from\_client | פרוטוקול TCP לתקשורת בקשה-תשובה. |  | מחזירה את מידע הבקשה מהלקוח |
| save\_to\_file | שומרת סטים של מידע בתוך קבצי .txt באופן אסוף עם תו הפרדה ייחודי לכל קובץ טקסט: @, ] | סוג מידע (פרוטוקול או מיקום) |  |
| from\_file | שואבת סטים של מידע מתוך קבצי .txt לפי תו ההפרדה וסוג האיסוף הייחודי שלהם.  בוחרת את הסט ע"י המידע מהלקוח (שם המיקום/הפרוטוקול) | סוג מידע (פרוטוקול או מיקום) | מחזירה את הערך השמור בחלק השני של סט המידע השמור הרצוי |

### פעולות חיצוניות בצד השרת:

|  |  |  |  |
| --- | --- | --- | --- |
| שם פעולה: | מטרת פעולה: | קלט: | פלט: |
| main | הפעולה הראשית |  |  |
| setup\_socket\_logging | מגדיר את הקונפיגורציה של הlogger המתקשר עם מערכת הבקרה של הלקוח. | fmt  הפורמט המכתיב את דרך תצוגת המידע  log\_dir  מיקום התיקייה שתחזיק בקבצי הlog של הלקוח.  client\_address  כתובת הIP של הלקוח. |  |
| setup\_server\_logging | מגדיר את הקונפיגורציה של הlogger הכללי של השרת, אשר גם מתקשר עם מערכת מילוי הבקשות של הלקוח | fmt  הפורמט המכתיב את דרך תצוגת המידע  log\_dir  מיקום התיקייה שתחזיק בקבצי הlog של השרת. |  |

### תיאור אלגוריתם השרת:

משום שלצד השרת רק יחידה אחת, כל ההרחבה הדרושה מוצגת בתרשים הנ"ל:

## מבני נתונים:

בפרויקט נעשה שימוש רב בתור (queue.Queue) במילון (dict), בסט (set) ואף שימוש מינימלי מאוד ברשימה:

* תור: התור הוא מבנה נתונים מהיר ויעיל הפועל לפי FIFO: הראשון שנכנס למבנה, הוא הראשון שיצא. הפעולות שמבצעים עם תור הן הכנסה לתור, הוצאה מן התור, והודעה לתור כאשר "המשימה הושלמה" "task\_done". לעומת רשימה (list), בשימוש בתור יש גישה אך ורק לאיבר הראשון בכל רגע נתון. זו מגבלה שעלולה להפריע במצבים רבים, אך הופכת את התור גם למבנה נתונים בטוח ויעיל ביותר בשימוש נכון; עבודה נכונה עם תור מאפשרת סנכרון ועבודה שיתופית ובטוחה יחסית בmulti-threading אשר עשויה ליצור סדר גם כאשר מבצעים מספר תהליכים (כמעט->) במקביל.   
  השתמשתי בתור בפרויקט בעיקר על מנת ליצור סדר מילוי של בקשות אשר קל להבנה ואף לקריאה. הסיבה העיקרית לבחירה בתור על פני מבני נתונים אחרים היא הבטיחות והסדר שהיא יוצרת בסביבת multi-threading, בנוסף למהירות ולקריאות שלה.
* מילון: מילון הוא מבנה נתונים גמיש ומגוון בשימושיו, ויוצר סדר למפתח אשר משתמש בו. המילון הוא אוסף של סטים של key:value, כאשר key וvalue יכולים להיות סוגים שונים ומגוונים של מידע. חיסרון אחד בו הוא המחסור בindex (המספר המכוון על מקום איבר במבנה נתונים, כלומר אין לאיברים מיקום). בנוסף, כאשר קיימים יותר מדי keys המילון מתחיל לקחת כמות מקום גדולה יחסית בזיכרון, אם כי החיסרון הזה לא רלוונטי לפרויקט בעל מידע בסדר גודל כמו שהכנסתי פנימה.  
  על מנת להסביר את משמעות התיאור הראשוני שהענקתי למילון ("...גמיש ומגוון בשימושיו"), אתן את השימושים בפרויקט שלי כמקרים יחידים של שימושים שכאלו:
  + במחלקה RequestHandlerThread, עשיתי שימוש במילון על מנת ליצור קשר בין פונקציה (ללא קריאה!) לרשימה המכילה סטים של מילות מפתח מסוג מחרוזת בצורה הבאה:   
    {open\_new\_tab : [ {‘open’}, {‘tab’} ]}. לאחר שמילאתי את המילון, כל שהיה עליי לעשות זה לעבור על הערכים במילון עד שהתנאי שהצבתי התקבל, ואז לקרוא לפונקציה המתאימה.   
    השתמשתי במילון זה על מנת לקצר / לפשט את התהליך הבלתי-קריא של הif/elif/else שאחרת הייתי צריך לבצע על מנת להתאים בין מילות המפתח לבין הפונקציה אליה יש לקרוא. בנוסף לכך, שימוש זה במילון פורס מילות המפתח של כל שירות בצורה ברורה, ומקל על עבודת המפתח במידה וירצה להוסיף שירותים נוספים או מילות מפתח חדשות.
  + על מנת לבצע קונפיגורציה לlogger, יש לקשר בין שם הlogger לhandler ול-formatter שלו. לאחר שביצעתי את החיבור בעצמי, גיליתי שיש דרך פשוטה יותר לביצוע הדבר - ספריה חיצונית ושמה logging.config (ללא צורך התקנה). ספציפית, הפעולה dictConfig, אשר מאפשרת ליצור קונפיגורציה מלאה וקלה להבנה בתוך מילון. הוספת handlers, loggers, formatters זה עניין פשוט ואינטואיטיבי ביותר כאשר משתמשים במילון.
* סט: סט הוא מבנה נתונים שראה שימוש נרחב בפרויקט שלי יחד עם תור, בין השאר בשל המהירות הגבוהה שלו בכל הקשור למציאת תשובה בוליאנית בתנאי “if x in set” לעומת מבני נתונים אחרים. בנוסף, סטים שימושיים מאוד בכל הקשור לפעולות מתמטיות (למשל intersection בו עשיתי שימוש על מנת למצוא מילות מפתח ולהתאים אותן לשירותים).  
  חשוב לציין כי ההבדל העיקרי בין סט למבני נתונים כמו tuple או רשימה, עם זאת שלא השתמשתי בתכונה הזו בפרויקט הנ"ל, הוא התכונה המעניינת הקובעת כי לא יכולים להיות מספר איברים זהים בסט (דוגמה לצורה בה התכונה מתבטאת, אם כי לא בפרויקט הנ"ל: ניתן למחוק שכפולים ברשימה ע"י העברת/העתקת הערכים לסט והחזרת האיברים לרשימה)
* רשימה: מבנה נתונים הרואה שימוש נרחב בשל השירותים הרבים והנוחים שהוא מספק – index לכל איבר אומר שלבצע סריקת רשימה זו פעולה פשוטה יחסית (בתור למשל, סריקת איברים זה פעולה מסובכת הרבה יותר). רשימה אף מסוגלת לתפקד כתור במידה מוגבלת ע"י שימוש בpop וinsert, אם כי הדבר אינו מומלץ... עם זאת, בפרויקט לא נעשה שימוש נרחב ברשימה, והשימוש שכן התבצע היה בעיקר כתוצאה מפעולת String.split שיוצרת רשימה, או כתוצאה מבחירה לעשות הבדלה בין ערכים לפי הסוג שלהם באמצעות if isinstance.

## ארכיטקטורת רשת – פרוטוקול תקשורת:

מטרת השירות:

השרת מספק ללקוח גישה למאגר הנתונים (קבצי .txt) שעליו ושירותים מוגבלים של כתיבה/קריאה שלהם.

אופן תקשורת:

לקוח – מציג בקשות לשירותים. שרת – מספק שירותים ומענה.

סוגי הודעות:

ישנם 4 סוגים של הודעות לקוח->שרת, אחד לכל שירות שמציע השרת:

* + - svlc – הלקוח מבקש לשמור מידע מסוג שם-מיקום אשר יישלח בהודעה עוקבת, בקובץ ה.txt
    - gtlc – הלקוח מבקש להוציא מקובץ ה.txt מיקום לפי שם שיישלח בהודעה עוקבת.
    - svpc – הלקוח מבקש לשמור מידע מסוג שם-סדר פעולות, אשר יישלח בהודעה עוקבת, בקובץ ה.txt
    - gtpc – הלקוח מבקש להוציא מקובץ ה.txt סדר פעולות לפי שם, אשר יישלח בהודעה עוקבת.

מבנה / אופן התקשורת:

השרת מצפה לקבל סדר הודעות מסוים: סוג בקשה --> אורך מידע + מידע.ראשית, השרת מקבל מחרוזת בעלת 4 תווים, אשר מתארת את סוג הבקשה שתגיע בקרוב מן המשתמש.

השרת שולח בחזרה את הסוג שהתקבל והלקוח מבצע בדיקה על מנת לאשר כי המידע שקיבל לא עבר שינויים. אם לא עבר שינויים, הלקוח שולח זוג הודעות נוספות לשרת: אורך המידע החיוני למילוי הבקשה, והמידע החיוני למילוי הבקשה עצמו.

לאחר מילוי הבקשה במקרה של svlc/svpc, השרת שולח ללקוח אישור הצלחה או הודעת תקלה בהתאם למתרחש.

לאחר העתקת המידע הנחוץ במקרה של gtlc/gtpc, השרת שולח ללקוח את המידע הנחוץ.

פרוטוקול תעבורה:

פרוטוקול התעבורה היחידי שמצא שימוש בפרויקט שלי הוא פרוטוקול הTCP.

## ממשקים למערכות אחרות:

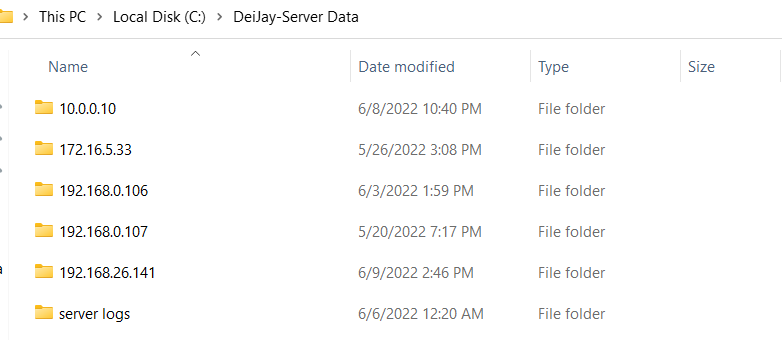
בפרויקט נעשה שימוש מינימלי אך מרכזי וחיוני בAPI של גוגל: speech\_recognizer. הסברתי על הAPI במבוא (תחת כותרת "סקירה ספרותית"), ולכן לא אחזור על ההסבר כאן.

## תיקיות וקבצים:

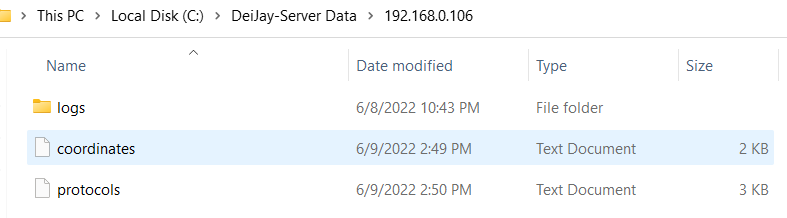
בפרויקט יש צורך לשמור מידע מחוץ לתוכנית בצורה כלשהי אשר תאפשר גישה אל מידע לפי "מפתח" ואפשרות הרחבת מאגר המידע. במהלך הפיתוח החלטתי לבחור בשמירת המידע בקבצי טקסט. (בחירה שאתייחס אליה ברפלקציה...)

בנוסף לקבצי ה.txt שנועדו לשימור מידע לשימוש מאוחר יותר ע"י התוכנית, הפרויקט גם שומר קבצי log של השרת ושל הלקוחות.

עבור כל לקוח (IP) התוכנית בודקת האם קיימת כבר תיקייה מתאימה (לפי IP) ויוצרת תיקייה חדשה אם לא נמצאה כזו. כל תיקיית לקוח נמצאת תחת תיקיית אם אחת הנפתחת בC: כמתואר בתצלום המסך הבא:

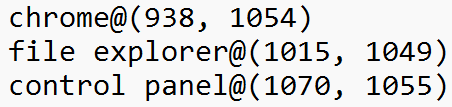


בתצלום הבא, ניתן לראות את הקבצים שבתוך כל תיקיית לקוח (בתיקיית השרת ההבדל הוא שאין קובץ coordinates.txt או protocols.txt, אלא רק קבצי לוג:



## דוגמה לשימוש בקובץ .txt לשמירת נתונים -

התו המפריד בין כל זוג שם-ערך שונה בין שני קבצי הנתונים, אך אופן שמירת הנתונים זהה בין 2 קבצי ה.txt ולכן יש צורך רק בדוגמה אחת. הדוגמה הבאה לקוחה מקובץ coordinates.txt אשר שומר זוגות של שם-מיקום:



## ספריות חיצוניות:

* socket
* socketserver
* logging
* logging.handlers
* logging.config
* queue
* struct
* threading
* time
* select
* tkinter
* ctypes
* pyttsx3
* speech\_recognition
* mouse
* pyautogui
* os – לתיקון תקלה שעלולה לעלות בשל חוסר הרשאות ליצירת תיקייה

# מדריך למשתמש

## הוראות התקנה - לקוח:

הקבצים הנדרשים בצד הלקוח הם קבצי הפייתון: DeijayMain.py, mouseControl.py, voiceOperator.py. הפרויקט מיועד ונבדק על מערכת ההפעלה ווינדוס בלבד.

יש לוודא כי מותקן python 3.9על המחשב, ושהוא תומך בספריות הבאות: logging, logging.handlers, logging.config, queue, socket, struct, threading, time, select, tkinter, ctypes, mouse.

בנוסף לספריות הללו, יש צורך בלבצע התקנה של הAPI החיצוני של גוגל, speech\_recognition, באופן הבא - (המתואר גם בעמוד הרשמי <https://pypi.org/project/SpeechRecognition/>)

* יש לפתוח את הcommand prompt לאחר שפייתון מותקן.
* לאחר מכן יש לרשום: "pip install pyaudio", וללחוץ על מקש האנטר על מנת להתקין את היחידה. אין בפרויקט קריאת import לספריה זו, אך speech\_recognition תלויה בה. בלעדיה, לא יהיה ניתן לבצע תקשורת עם המיקרופון והלקוח לא יתפקד בצורה שאמור.
* כעת האדמה מוכשרת לפקודה הסופית: כותבים "pip install SpeechRecognition" בcommand prompt ולוחצים על מקש האנטר פעם נוספת. לאחר הודעת הצלחה, ובמידה ושאר ההתקנה נעשתה בצורה נכונה, הלקוח אמור להיות מוכן לתפקוד.

על מנת להפעיל את התוכנית יש להריץ את הקובץ DeijayMain.py. שים לב: יש להפעיל את השרת לפני הפעלת הלקוח. בעוד שהלקוח יתפקד בחלקו גם ללא חיבור ללקוח, רוב התפקוד שלו נעלם ללא חיבור זה.

הערה: בעת הגשת ספר הפרויקט לא התבצעה בדיקה של תפקוד שרת-לקוח בין מחשבים שונים.   
אם בהגשה הסופית של הפרויקט נוספת אפשרות כזו, יש גם להכניס כפרמטר הראשון שעולה את כתובת הIP של השרת.

## הוראות התקנה - שרת:

הקובץ הנדרש בצד השרת הוא קובץ הפייתון dataHandlingServer.py בלבד. הפרויקט מיועד ונבדק על מערכת ההפעלה ווינדוס בלבד.

יש לוודא כי מותקן python 3.9על המחשב, ושהוא תומך בספריות הבאות:  
\_pickle, logging, logging.handlers, logging.config, os, socket, socketserver, struct, threading, select .

כדי להפעיל את השרת יש להריץ את הקובץ dataHandlingServer.py. יש להפעיל אותו לפני הפעלת הלקוח.

## מידע כללי על קבצים:

אין צורך בהתקנת דבר נוסף, אך יש לציין למשתמש כי התוכנית עצמה אחראית ליצור תיקייה אחת ללקוח ואחת לשרת על כונן C: של המחשב שמפעיל את הלקוח והמחשב שמפעיל את השרת בהתאמה. בספר הפרויקט הנ"ל יש פרק ושמו "תיקיות קבצים" שמרחיב על שמירת הנתונים הזו שעשויה לסייע בהבנת הנושא.

## ממשק משתמש מינימלי - אינדיקציות:

בתחילת כתיבת הפרויקט, הוחלט כי הוא יהיה כלי אשר שימושו יהווה הפרעה מינימלית למשתמש. אי לכך, אין לפרויקט כמעט רכיב גרפי או כל דבר שהמשתמש עשוי להתעסק בו יתר על המידה.

הפרויקט מסתמך על 2 סוגי אינדיקציות: הנקודה האדומה שבצד המסך, וההודעות הקוליות שמתריעות את המשתמש על מצבים שהוא צריך לדעת מהם (למשל: "שמירת מיקום בוצעה בהצלחה" או "לא נמצא במאגר הנתונים entry התואם את בקשתך").

נקודה אדומה: הנקודה מופיעה כחצי-שקופה, בצד ימין העליון של המסך עד ל10 שניות בכל הופעה. התוכנית מקשיבה ואוספת מידע הזורם דרך המיקרופון. כלומר: כאשר הנקודה האדומה מופיעה, ניתן לדבר ולהשתמש בתוכנית. בהרחבה, כאשר הנקודה לא על המסך התוכנית אינה מקשיבה (הAPI עסוק בדיוור הבקשה למתרגם של Google). משום שהדרך היחידה של המשתמש להשתמש בתוכנית היא באמצעות קלט קולי, זה האינדיקטור היחיד הבלתי-נמנע של הפרויקט עבור המשתמש.

הודעות קוליות: כאשר קורא משהו בתוכנה שדורש את תשומת הלב של המשתמש, התוכנה תבצע פלט קולי של הודעה כלשהי. הפלט מתבצע לאחר שמירת מידע בקבצי הנתונים על מנת שהמשתמש יקבל אישור שלא אירעה תקלה, או לאחר תקלה קריטית בקוד או אי-הצלחה של מילוי בקשה שאירעה מתוך טעות של המשתמש. דוגמאות לסיבות לאי-הצלחות: דיבור לא מובן, בקשה לשירות שלא קיים עדיין, סיפוק מידע חלקי לתוכנית בעת הבקשה – למשל בקשת שמירת מיקום עכבר ללא מתן שם לאותו מיקום.

השימוש בקבצי הlog אינו קיים למען המשתמש בפרויקט זה אלא למען המתכנת/המפתח, ולכן אין יישום ישיר של קבצים אלו בפרויקט.   
עם זאת, ישנם מצבים בהם המשתמש עשוי לרצות לקרוא בקבצים אלו: במידה ומאיזושהי סיבה מתקיימת תקלה קריטית, קבצי הlog השמורים על הלקוח אמורים להכיל את הסיבה לכך. אם הדבר מעניין את המשתמש, התיקייה המחזיקה את כלל קבצי הלוג של הלקוח נמצאת על הכונן שלו, ותמיד מוכנים לקריאה ידנית. ישנם קבצי error, debug וinfo.   
אם מעניין את המשתמש המעקב אחר פעולותיו, יש ביכולתו לעיין בקובץ הinfo אשר מחזיק במידע מינימלי וקריא יחסית לdebug ול-error, אף למי שאינו מתכנת.

## הוראות שימוש בפרויקט:

* התוכנית אומנם מקשיבה לכל רעש כאשר הנקודה האדומה מופיעה, אך **התוכנית תתייחס לרעש רק כאשר היא מזהה את מילת ההקשבה** הקבועה לערך “**DJ**”. כלומר, באיזשהו שלב (לא משנה מתי) של בקשה, יש להגיד את המילה הזו, ניתן להתייחס לזה כמעין "שם" של התוכנית אליו המשתמש פונה.
* **על מנת לחבר פעולות באותו המשפט** למטרות יעילות / נוחות שימוש, ניתן להשתמש במילות חיבור שונות מהשפה היום-יומית (אינטואיטיבית). לדוגמה:

“dj fast travel to from browser YouTube **then** click **and then** move mouse down by 200 **and** double-click”

* **הגדרת פרוטוקולים**: מצריך שימוש במילה **protocol**. פעולת שמירת/ביצוע פרוטוקול מתבצעת ב3 שלבים: התחלת הקלטה, הפסקת הקלטה ומתן שם, ולבסוף קריאה לפרוטוקול בזמן הרצוי. בכל אחד מן השלבים הללו, יש צורך בשימוש במילה **protocol**, והמינוח של הפקודות אינטואיטיבי. דוגמה לדרך אחת לבטא הגדרה של פרוטוקול ושימוש בו: (ישנן יותר מילות מפתח מאלה, אין הגבלה לדרך אחת לבטא כוונת "הגדרת פרוטוקול" או קריאה שלו. זו דוגמה בלבד)

“dj start recording **protocol**” -> “dj fast travel to chrome then click” -> “dj save **protocol** recording as from windows open chrome” -> “dj **protocol** from windows open chrome”

* מעבר לכך, השימוש בתוכנה אמור להיות אינטואיטיבי; אם יש משהו שהמשתמש רוצה שיעשה, הוא צריך להגיד זאת במילים והתוכנית אמורה לזהות את הכוונה שלו לפי מילים מסוימות שיצטרך להשתמש בהן על מנת להביע את כוונתו בכל מקרה (למשל על מנת לפתוח דף חדש בדפדפן, אין דרך להביע את הכוונה הזו מבלי להגיד את המילה open וpage או window).

המשתמש צריך לזכור דבר אחד נוסף: הפרויקט אינו בא לספק את אותם שירותים כמו עוזרים ווירטואליים קיימים, ולכן לא מתפקד בצורה דומה. DeiJay אמור לסייע דרך אוטומציה או שליטה קולית בתהליכים שניתן לבצע באמצעות עכבר ומקלדת, לא להציע סיוע "עוזר ווירטואלי". בקשות כלליות כמו "חפש משהו בגוגל" לא יעבדו. (אלא אם כן המשתמש קבע את הבקשה כפרוטוקול, במקרה זה הבקשה הכביכול כללית הפכה למקרה "צפוי" מבחינת התוכנית, ותתבצע כמתבקש).

# מדריך למפתח

בחלק זה אציג את הקבצים הקיימים בפרויקט שלי, ואת המשתנים הקיימים בהם.  
על מנת לבחון את המחלקות השונות, התכונות שלהן, ואת הפונקציות בקבצים, אנא פנה לפרק "מחלקות ויחידות הפרויקט", שם הסברתי בהרחבה על כל מרכיב במחלקות ובפונקציות של הפרויקט.

הערה: משתנים הכתובים באותיות גדולות בלבד – משתנים שהוגדרו בglobal domain.

## DeijayMain.py:

### משתנים חשובים:

* KEYWORD: מילת ההקשבה של התוכנית.
* LOG\_FORMAT: הפורמט לפיו קבצי הלוג יסתדרו.
* my\_socket: אובייקט מסוג socket.
* pcl\_tracker: אובייקט מסוג ProtocolController.
* red\_dot: אובייקט מסוג RecordingDot, הגרפיקה של הפרויקט
* request\_queue: תור אשר יכיל בקשות משתמש ויועבר כתכונה העיקרית של RequestHandlerThread
* on\_off\_switch: רשימה חד-תאית של ערך בוליאני אשר שולט על הלולאה הראשית.
* sequence\_words: רשימה אשר מכילה מילות חיבור נפוצות למטרות ניתוח טקסט.
* text: ה"רעש" שזורם דרך המיקרופון מתורגם לטקסט ונכנס למשתנה הנ"ל.
* timer\_start: שומר זמני הרצה.
* logging\_config: מילון אשר מגדיר את קונפיגורציית הלוגים.
* keyword\_sorter: משתנה פנימי בRequestHandlerThread. מילון אשר מסייע בהתאמת מילות מפתח לשירות הרצוי (מחליף שימוש בשרשרת if/elif/else)
* SERVER\_IP: בשל מגבלות זמן, מוגדר כIP של הלקוח (localhost)
* PORT: פורט החיבור, מוגדר כ4001

### תוכן:

import logging  
import logging.handlers  
import queue  
import socket  
import struct  
import threading  
import time  
from logging.config import dictConfig  
  
import select  
import tkinter as tk  
import ctypes  
import pyautogui  
import mouse  
  
import mouseControl  
import voiceOperator as VOp  
  
KEYWORD = "dj"  
SERVER\_IP, PORT = socket.gethostbyname(socket.gethostname()), 4001  
  
  
class RecordingDot(tk.Tk):  
  
 def \_\_init\_\_(self, diameter, color, visible=True):  
 tk.Canvas.create\_circle = monkey\_patched\_create\_circle  
 super().\_\_init\_\_()  
 self.diameter = diameter  
 self.color = color  
 self.visible = visible  
  
 self.resizable(False, False)  
 self.overrideredirect(True)  
 user32 = ctypes.windll.user32  
 screensize = user32.GetSystemMetrics(0), user32.GetSystemMetrics(1)  
 self.geometry(f'{self.diameter}x{self.diameter}+{screensize[0] - 40}+{screensize[1] // 10}')  
 self.wm\_attributes("-topmost", True)  
 self.wm\_attributes("-alpha", 0.6)  
 self.wm\_attributes("-transparentcolor", "white")  
  
 canvas = tk.Canvas(self, width=self.diameter, height=self.diameter, borderwidth=0,  
 highlightthickness=0, bg='white')  
 canvas.grid()  
 canvas.create\_circle(self.diameter // 2, self.diameter // 2, self.diameter // 2, fill=self.color,  
 outline="#DDD")  
  
 def toggle\_visibility(self):  
 if self.visible:  
 self.wm\_attributes("-alpha", 0)  
 else:  
 self.wm\_attributes("-alpha", 0.6)  
 self.visible = not self.visible  
  
  
def monkey\_patched\_create\_circle(self, x, y, r, \*\*kwargs):  
 return self.create\_oval(x - r, y - r, x + r, y + r, \*\*kwargs)  
  
  
class ProtocolController:  
 def \_\_init\_\_(self):  
 self.protocol\_recording = []  
 self.protocol\_recording\_switches = {'protocol\_recording': False,  
 'protocol\_save': False}  
  
  
class RequestHandlerThread(threading.Thread):  
  
 def \_\_init\_\_(self, request\_queue, sock, timer, pcl\_tracker, on\_off\_switch):  
 super().\_\_init\_\_(daemon=True)  
 self.noError = True  
 self.job = None  
 self.current\_request = None  
 self.request\_queue = request\_queue  
 self.sock = sock  
 self.timer = timer  
 self.program\_switch = on\_off\_switch  
 self.pcl\_tracker = pcl\_tracker  
  
 def run(self):  
 *"""  
 Matches requests to services in a queue.  
 Follows up on the main function, as a thread.  
 """* while not self.request\_queue.empty():  
 self.job, self.current\_request, self.noError = None, None, True  
  
 self.current\_request = self.request\_queue.get()  
 set\_request = set(self.current\_request.split(" ")) # sets are significantly faster at 'in' statements.  
 logger.debug(f"request: {self.current\_request}")  
 logger.debug(f"Keywords: {set\_request}")  
  
 for key in self.keyword\_sorter:  
 keywords\_match = True  
 if isinstance(self.keyword\_sorter[key], list):  
 for group in self.keyword\_sorter[key]:  
 if not group.intersection(set\_request):  
 keywords\_match = False  
 break  
 elif not self.keyword\_sorter[key].intersection(set\_request):  
 keywords\_match = False  
  
 if keywords\_match:  
 logger.info(f'Request identified in services: {key}')  
 self.job = key  
 break  
  
 if self.job:  
 self.job(self)  
 else:  
 logger.warning(f"Apologies, unknown command: '{self.current\_request}'.")  
 self.noError = False  
  
 if self.noError and self.pcl\_tracker.protocol\_recording\_switches['protocol\_recording']:  
 logger.debug('Recording protocol henceforth...')  
 if self.pcl\_tracker.protocol\_recording\_switches['protocol\_save']:  
 logger.debug('Received request to stop recording. Checking for errors:')  
 try:  
 protocol\_entry = self.current\_request.split(' as ')[1]  
 logger.debug('Successfully extracted protocol entry from sentence.')  
 except IndexError:  
 logger.warning('Stopped listening mid-sentence. Recording preserved')  
 continue  
 protocol\_data = f'{protocol\_entry}{self.pcl\_tracker.protocol\_recording}'  
 logger.debug(f'protocol data organized: {protocol\_data}')  
 self.pcl\_tracker.protocol\_recording\_switches = {item: False for (item, value) in  
 self.pcl\_tracker.protocol\_recording\_switches.items()}  
 logger.debug(f'FALSE??? {self.pcl\_tracker.protocol\_recording\_switches}')  
 self.send\_then\_handle('svpc', protocol\_data)  
 self.pcl\_tracker.protocol\_recording.clear()  
 logger.debug(f'FALSE??? {self.pcl\_tracker.protocol\_recording\_switches}')  
  
 elif self.job != self.start\_rec\_protocol.\_\_func\_\_ \  
 and self.job != self.execute\_protocol.\_\_func\_\_ \  
 and self.job != self.save\_location\_entry.\_\_func\_\_:  
 self.pcl\_tracker.protocol\_recording.append(self.current\_request)  
 logger.debug(f"Appended request into the protocol recording: {self.pcl\_tracker.protocol\_recording}")  
  
 self.request\_queue.task\_done()  
  
 logger.debug(f"total execution time: {time.perf\_counter() - self.timer}")  
 logger.debug("Terminating thread.")  
  
 def receive\_from\_server(self):  
 packed\_length = self.sock.recv(4)  
 data\_len = struct.unpack('i', packed\_length)[0]  
 logger.debug('About to receive data from server:')  
 chunk = self.sock.recv(data\_len)  
 logger.debug('Confirming all data has been received...')  
 while len(chunk) < data\_len:  
 chunk = chunk + self.sock.recv(data\_len - len(chunk))  
  
 return\_msg = chunk.decode()  
 logger.debug(f'Return message received and decoded: {return\_msg}')  
 return return\_msg  
  
 def handle\_server\_response(self, data\_type, job\_type):  
 read\_sockets, write\_sockets, error\_sockets = select.select([self.sock], [], [])  
  
 if read\_sockets[0] == self.sock:  
 return\_msg = self.receive\_from\_server()  
 if job\_type == 'save':  
 speak(return\_msg)  
  
 if 'ERROR: ' not in return\_msg:  
 if data\_type == 'coordinates' and job\_type == 'use':  
 return return\_msg  
 if data\_type == 'protocol' and job\_type == 'use':  
 operations\_string = return\_msg.replace("]", "").replace("'", "")  
 logger.debug(f'Received operations from server: {operations\_string}')  
  
 q = queue.Queue()  
 operations\_lst = operations\_string.split(', ')  
 while operations\_lst:  
 logger.debug(f'Inserting operation into protocol queue: {operations\_lst[0]}')  
 q.put(operations\_lst.pop(0))  
  
 return q  
  
 logger.info(return\_msg)  
 else:  
 logger.error(return\_msg)  
 with self.request\_queue.mutex:  
 logger.debug('Clearing request queue')  
 self.request\_queue.queue.clear()  
 self.noError = False  
 speak(return\_msg)  
  
 def send\_then\_handle(self, job\_key, data):  
 if data is not None:  
 logger.debug(f'Sending job key {job\_key} to server, then awaiting confirmation...')  
 self.sock.send(job\_key.encode('UTF-8'))  
  
 ret\_key = self.sock.recv(4).decode()  
 if ret\_key == job\_key:  
 logger.debug('Confirmed: server received the correct job key. '  
 'Sending data length, followed by the data package')  
  
 self.sock.send(struct.pack('i', len(data)))  
 self.sock.send(data.encode('UTF-8'))  
  
 # Now to handle the data that was returned from the server  
 if job\_key == 'svlc':  
 logger.debug(f'sent {data} to server to save the COORDINATES on it.')  
 self.handle\_server\_response('coordinates', 'save')  
  
 elif job\_key == 'gtlc':  
 logger.debug(f'Sent the entry name {data} to the server to get COORDINATES from it')  
 mouseControl.mouse\_location\_travel(self.handle\_server\_response('coordinates', 'use'), data)  
  
 elif job\_key == 'svpc':  
 logger.debug(f'sent {data} to server to save the PROTOCOL on it.')  
 self.handle\_server\_response('protocol', 'save')  
  
 else: # job key is 'gtpc' (get protocol)  
 logger.debug(f'sent {data} to server to get PROTOCOL OPERATIONS from it.')  
 return self.handle\_server\_response('protocol', 'use')  
 else:  
 logger.critical(f'Request key did not return the same: {ret\_key}')  
 self.noError = False  
 else:  
 logger.error('data to be sent is None, therefore no sending operation was made.')  
 self.noError = False  
  
 def keyboard\_type(self):  
 logger.debug('Identified "keyboard\_type" request')  
 try:  
 sentence = self.current\_request.split("type ", maxsplit=1)[1]  
 except IndexError:  
 logger.critical('The program did not sort the request correctly. URGENTLY find why.')  
 pyautogui.write(self.current\_request)  
 logger.info(f"text written: {self.current\_request}")  
 time.sleep(1)  
  
 def fast\_travel(self):  
 logger.debug('Identified "fast travel" request')  
 coordinates\_entry = mouseControl.mouse\_location\_get\_entry(self.current\_request)  
 self.send\_then\_handle('gtlc', coordinates\_entry)  
  
 def save\_location\_entry(self):  
 logger.debug('Identified "save location entry" request')  
 save\_data = mouseControl.mouse\_location\_save\_data(self.current\_request)  
 self.send\_then\_handle('svlc', save\_data)  
  
 def save\_rec\_protocol(self):  
 logger.debug('Identified "save recording as protocol" request')  
 if self.pcl\_tracker.protocol\_recording\_switches['protocol\_recording']:  
 self.pcl\_tracker.protocol\_recording\_switches['protocol\_save'] = True  
 else:  
 logger.warning("No recording has begun yet, therefore ignoring this request.")  
 self.noError = False  
  
 def start\_rec\_protocol(self):  
 logger.debug('Identified "start recording protocol" request')  
 self.pcl\_tracker.protocol\_recording\_switches['protocol\_recording'] = True  
 speak('Recording')  
  
 def execute\_protocol(self):  
 logger.debug('Identified "execute protocol" request')  
 try:  
 protocol\_entry = self.current\_request.split('protocol ')[1]  
 except IndexError:  
 logger.warning('User omitted protocol entry from request, ignoring request.')  
 self.noError = False  
 return  
 protocol\_queue = self.send\_then\_handle('gtpc', protocol\_entry)  
 if protocol\_queue:  
 # re-organizing request queue:  
 while not self.request\_queue.empty():  
 protocol\_queue.put(self.request\_queue.get())  
 while not protocol\_queue.empty():  
 self.request\_queue.put(protocol\_queue.get())  
  
 def press\_enter(self):  
 logger.debug('Identified "press enter" request')  
 pyautogui.press('enter')  
 logger.info("pressed ENTER on the keyboard.")  
 time.sleep(1)  
  
 def back\_history\_browser(self):  
 logger.debug('Identified "go to last page in browser" request')  
 pyautogui.hotkey('alt', 'left')  
 logger.info("MOVED BACK in browser history.")  
 time.sleep(1)  
  
 def forward\_history\_browser(self):  
 logger.debug('Identified "go to next page in browser" request')  
 pyautogui.hotkey('alt', 'right')  
 logger.info("MOVED FORWARD in browser history.")  
 time.sleep(1)  
  
 def click\_mouse(self):  
 logger.debug('Identified "click the mouse" request')  
 if 'double' in self.current\_request and 'click' in self.current\_request:  
 mouse.double\_click('left')  
 logger.info('double-left clicked')  
 time.sleep(1)  
 elif 'double' in self.current\_request and 'right' in self.current\_request:  
 mouse.double\_click('right')  
 logger.info('double-right clicked')  
 time.sleep(1)  
  
 elif 'right' in self.current\_request:  
 mouse.click('right')  
 logger.info("right clicked")  
 time.sleep(1)  
 else:  
 mouse.click('left')  
 logger.info("left clicked")  
 time.sleep(1)  
  
 def scroll\_mouse(self):  
 logger.debug('Identified "scroll the mouse wheel" request')  
 for i in reversed(range(1, 100)):  
 if f'{i}' in self.current\_request:  
 if 'up' in self.current\_request:  
 mouse.wheel(5)  
  
 elif 'down' in self.current\_request:  
 mouse.wheel(-5)  
  
 def move\_mouse\_pointer(self):  
 logger.debug('Identified "move mouse-pointer" request')  
 num = 0  
 for i in reversed(range(1, 2000)):  
 if f"{i}" in self.current\_request and ("down" in self.current\_request or "right" in self.current\_request):  
 num = i  
 break  
 elif f"{i}" in self.current\_request and ("up" in self.current\_request or "off" in self.current\_request or "left" in self.current\_request):  
 num = -i  
 break  
  
 if "up" in self.current\_request or "off" in self.current\_request or "down" in self.current\_request:  
 mouse.move(0, num, absolute=False, duration=0.1)  
 time.sleep(0.1)  
 logger.info(f"Moved {num} units on the y axis.")  
  
 elif "right" in self.current\_request or "left" in self.current\_request:  
 mouse.move(num, 0, absolute=False, duration=0.1)  
 time.sleep(0.1)  
 logger.info(f"Moved {num} units on the x axis.")  
  
 def press\_left\_arrow(self):  
 logger.debug('Identified "press left arrow" request')  
 pyautogui.press('left')  
 logger.info('pressed LEFT ARROW on the keyboard')  
 time.sleep(0.1)  
  
 def press\_right\_arrow(self):  
 logger.debug('Identified "press right arrow" request')  
 pyautogui.press('right')  
 logger.info('pressed RIGHT ARROW on the keyboard')  
 time.sleep(0.1)  
  
 def press\_up\_arrow(self):  
 logger.debug('Identified "press up arrow" request')  
 pyautogui.press('up')  
 logger.info('pressed UP ARROW on the keyboard')  
 time.sleep(0.1)  
  
 def shutdown\_program(self):  
 logger.debug('Identified "shutdown" request')  
 logger.info("exiting program.\r\n\r\n")  
 self.program\_switch[0] = False  
  
 def press\_down\_arrow(self):  
 logger.debug('Identified "press down arrow" request')  
 pyautogui.press('down')  
 logger.info('pressed DOWN ARROW on the keyboard')  
 time.sleep(0.1)  
  
 def press\_backspace(self):  
 logger.debug('Identified "press backspace" request')  
 pyautogui.press('backspace')  
 logger.info('pressed BACKSPACE on the keyboard. ')  
 time.sleep(0.1)  
  
 def open\_new\_tab(self):  
 logger.debug('Identified "open new tab" request')  
 pyautogui.hotkey('ctrl', 't')  
 logger.info("OPENED NEW TAB in browser.")  
 time.sleep(1)  
  
 def open\_new\_window(self):  
 logger.debug('Identified "open new window" request')  
 pyautogui.hotkey('ctrl', 'n')  
 logger.info("OPENED NEW browser WINDOW.")  
 time.sleep(1)  
  
 def next\_tab\_browser(self):  
 logger.debug('Identified "select next tab in browser" request')  
 pyautogui.hotkey('ctrl', 'tab')  
 logger.info("Cycled to NEXT TAB.")  
 time.sleep(1)  
  
 def press\_tab(self):  
 logger.debug('Identified "press tab" request')  
 pyautogui.press('tab')  
 logger.info('pressed TAB on the keyboard.')  
 time.sleep(0.1)  
  
 def close\_window(self):  
 logger.debug('Identified "close window" request')  
 pyautogui.hotkey('alt', 'f4')  
 logger.info("CLOSED APPLICATION.")  
 time.sleep(1)  
  
 def close\_tab(self):  
 logger.debug('Identified "close tab" request')  
 pyautogui.hotkey('ctrl', 'f4')  
 logger.info("CLOSED TAB in browser.")  
 time.sleep(1)  
  
 def add\_bookmark(self):  
 logger.debug('Identified "add page to bookmarks" request')  
 pyautogui.hotkey('ctrl', 'd')  
 logger.info("BOOKMARKED page in browser.")  
 time.sleep(1)  
  
 def find\_in\_browser(self):  
 logger.debug('Identified "open text finder in browser" request')  
 pyautogui.hotkey('ctrl', 'g')  
 logger.info("opened and selected TEXT FINDER prompt")  
 time.sleep(0.5)  
  
 def open\_history\_browser(self):  
 logger.debug('Identified "open history manager in browser" request')  
 pyautogui.hotkey('ctrl', 'h')  
 logger.info("opened HISTORY tab in browser.")  
 time.sleep(1)  
  
 def refresh\_browser(self):  
 logger.debug('Identified "refresh browser page" request')  
 pyautogui.press('browserrefresh')  
 logger.info("refreshed page in browser.")  
 time.sleep(2)  
  
 def search\_bar\_browser(self):  
 logger.debug('Identified "select search bar in browser" request')  
 pyautogui.hotkey('alt', 'd')  
 pyautogui.press('browsersearch')  
 logger.info("selected search bar.")  
 time.sleep(1)  
  
 def play\_pause\_video(self):  
 logger.debug('Identified "play or pause video" request')  
 pyautogui.press('playpause')  
 logger.info("PLAYED or PAUSED a video.")  
 time.sleep(1)  
  
 def junk(self):  
 logger.debug('Identified "junk" request')  
 self.noError = False  
  
 keyword\_sorter = {  
 keyboard\_type: {'type', 'tyke', 'tight'},  
 junk: {'misunderstanding', 'sorry', 'mistake', 'never', 'mind'},  
 fast\_travel: {'travel', 'trouble'},  
 save\_location\_entry: [{'location', 'coordinates'},  
 {'save', 'safecoin', 'safe', 'safecard', 'safeguard', 'steve'}],  
 save\_rec\_protocol: [{'protocol'}, {'save', 'safe', 'stop'}],  
 start\_rec\_protocol: [{'protocol'}, {'recording', 'record'}],  
 execute\_protocol: {'protocol'},  
 press\_enter: {'enter'},  
 close\_window: [{'close'}, {'window', 'application'}],  
 close\_tab: [{'close', 'clothes'}, {'tab'}],  
 open\_new\_tab: [{'open'}, {'tab'}],  
 open\_new\_window: [{'open'}, {'window'}],  
 add\_bookmark: {'bookmark'},  
 find\_in\_browser: {'find'},  
 open\_history\_browser: {'history'},  
 refresh\_browser: {'refresh'},  
 search\_bar\_browser: [{'select'}, {'search'}],  
 next\_tab\_browser: [{'select', 'switch'}, {'next', 'tab'}],  
 press\_tab: {'tab'},  
 play\_pause\_video: {"play", "pause", "paul's"},  
 back\_history\_browser: [{'back', 'return', 'last'}, {'page'}],  
 forward\_history\_browser: [{'forward', 'next'}, {'page'}],  
 click\_mouse: {'click', 'clicks', 'clicking', 'quik', 'flick', 'slick', 'double-click'},  
 scroll\_mouse: {'scroll'},  
 move\_mouse\_pointer: {'move', 'movie', 'moves'},  
 press\_left\_arrow: {'left'},  
 press\_right\_arrow: {'right'},  
 press\_up\_arrow: {'up', 'off'},  
 shutdown\_program: {'exit', 'bye', 'goodbye', 'shutdown', 'shut'},  
 press\_down\_arrow: {'down'},  
 press\_backspace: {'delete'}  
 }  
  
  
# -------------------------------------------------------------- #  
def speak(sentence):  
 speaker = VOp.Speaker(0, 200)  
 speaker.speak(sentence)  
 del speaker  
  
  
def setup\_logging():  
 fmt = '%(asctime)s | %(threadName)s - %(levelname)s - %(filename)s - %(lineno)s - func=%(funcName)s : "%(message)s"'  
 logging\_config = {  
 'version': 1,  
 'disable\_existing\_loggers': False,  
 'formatters':  
 {  
 'standard':  
 {  
 'format': fmt  
 }  
 },  
 'handlers':  
 {  
 'default':  
 {  
 'class': 'logging.StreamHandler',  
 'formatter': 'standard',  
 'level': 'DEBUG',  
 'stream': 'ext://sys.stdout'  
 },  
 'toSocket':  
 {  
 'class': 'logging.handlers.SocketHandler',  
 'level': 'DEBUG',  
 'host': SERVER\_IP,  
 'port': f'{logging.handlers.DEFAULT\_TCP\_LOGGING\_PORT}'  
 },  
 },  
 'loggers':  
 {  
 'logger':  
 {  
 'handlers': ['default', 'toSocket'],  
 'level': 'DEBUG',  
 'propagate': False  
 }  
 }  
 }  
  
 logging.config.dictConfig(logging\_config)  
  
  
def main\_setup():  
 # logging set-up.  
 setup\_logging()  
 logging.getLogger("comtypes").setLevel(logging.INFO)  
  
 speak("Thank you for waking me up, operator! I am Dayjay.")  
  
 # Socket set-up.  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 s.connect((SERVER\_IP, PORT))  
  
 # Finally, setting up the red\_dot object, the request\_queue and the mutable boolean values inside the return statement.  
 return s, ProtocolController(), RecordingDot(30, "red"), queue.Queue(), [True]  
  
  
def main():  
 my\_socket, pcl\_tracker, red\_dot, request\_queue, on\_off\_switch = main\_setup()  
 sequence\_words = [" and then ", " afterwards ", " finally ", " while ", " meanwhile ", " and ", " than "]  
  
 while on\_off\_switch[0]:  
 print("\n\n\n\n\n ---------------------------------------------------------------")  
 print(f"Number of active threads: {threading.active\_count()}")  
 text = str(VOp.from\_mic(red\_dot).lower())  
  
 if text == "":  
 continue  
  
 if KEYWORD in text: # Listen for the keyword  
 timer\_start = time.perf\_counter()  
  
 text = text.replace(f" {KEYWORD} ", " ").replace(f" {KEYWORD}", "").replace(f"{KEYWORD} ", "")  
 for word in sequence\_words:  
 text = text.replace(word, " then ")  
 if " then " in text:  
 text = text.split(" then ")  
 logger.debug(f"sentence split into requests: {text}")  
  
 if isinstance(text, list): # If the sentence was changed by simplification, it's a sequenced request.  
  
 for nested\_request in text:  
 logger.debug(f"queuing request for request\_handler function: {nested\_request}")  
 request\_queue.put(nested\_request)  
  
 request\_thread = RequestHandlerThread(request\_queue, my\_socket, timer\_start, pcl\_tracker, on\_off\_switch)  
 request\_thread.start()  
 else:  
 logger.debug(f"queuing request for request\_handler function: {text}")  
 request\_queue.put(text)  
  
 request\_thread = RequestHandlerThread(request\_queue, my\_socket, timer\_start, pcl\_tracker, on\_off\_switch)  
 request\_thread.start()  
  
 request\_thread.join()  
 my\_socket.shutdown(socket.SHUT\_RDWR)  
 my\_socket.close()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 logger = logging.getLogger('logger')  
 main()

## voiceOperator.py:

### משתנים חשובים:

* audio: תור המשמש לאכסון הרעש שזורם דרך המיקרופון.
* rec: אובייקט של ספרייה חיצונית מסוג speech\_recognition.Recognizer. אחראי על הAPI לתרגום הרעש לטקסט speech-to-text.
* graphics\_switch: רשימה חד-תאית המחזיקה בערך בוליאני. משמש לסנכרון בין הופעת/היעלמות הנקודה האדומה וקליטת הרעש דרך המיקרופון.
* listening\_graphics: אובייקט המועבר כפרמטר לfrom\_mic. הנקודה האדומה.

תוכן:

import speech\_recognition as sr  
import pyttsx3 as p  
import logging  
import threading  
import time  
import queue  
  
log = logging.getLogger('logger') # get the main logger  
  
  
class Speaker:  
 sex = 0  
 speech\_rate = 0  
 speaker = None  
  
 def \_\_init\_\_(self, sex, speech\_rate):  
 self.sex = sex  
 self.speech\_rate = speech\_rate  
  
 self.speaker = p.init()  
 self.speaker.setProperty("rate", self.speech\_rate)  
 self.speaker.setProperty("voice", self.speaker.getProperty("voices")[self.sex].id)  
  
 def speak(self, text):  
 self.speaker.say(text)  
 self.speaker.runAndWait()  
  
  
def listen\_mic(microphone, speech\_recognizer, audio, graphics\_switch):  
 audio.put(speech\_recognizer.listen(microphone, timeout=None, phrase\_time\_limit=10))  
 graphics\_switch[0] = False  
  
  
def from\_mic(listening\_graphics):  
 rec = sr.Recognizer()  
 graphics\_switch = [True]  
 audio = queue.Queue()  
  
 with sr.Microphone() as source:  
 rec.adjust\_for\_ambient\_noise(source)  
 threading.Thread(target=listen\_mic, args=(source, rec, audio, graphics\_switch)).start()  
  
 listening\_graphics.toggle\_visibility()  
 while graphics\_switch[0]:  
 listening\_graphics.update\_idletasks()  
 listening\_graphics.update()  
 time.sleep(0.01)  
  
 listening\_graphics.toggle\_visibility()  
 listening\_graphics.update\_idletasks()  
 listening\_graphics.update()  
  
 try:  
 re = rec.recognize\_google(audio.get())  
 time.sleep(0.2)  
 return re  
  
 except sr.UnknownValueError:  
 return ""  
 except sr.RequestError:  
 log.critical("Request error given, speech recognition operation failed.")  
 log.critical(  
 "Instructions: Reconnect to the internet. Otherwise, check the Google API key by following the Documentation: https://github.com/Uberi/speech\_recognition/blob/master/reference/library-reference.rst#:~:text=To%20obtain%20your,raise%20this%20limit")  
 return ""  
 except TimeoutError:  
 log.exception(  
 "TimeoutError [WinError 10060] given; speech\_recognition API failed to respond, likely due to bad internet connection.", exc\_info=True)  
 return ""

## mouseControl.py:

### משתנים:

ביחידה זו אין משתנים מעבר לפרמטרים בעלי שם חד-משמעי (coordinates המחזיק במיקום, entry המחזיק בשם, וrequest המחזיק בבקשה). היחידה קיימת על מנת ליצור מעט סדר בקוד.

תוכן:

import mouse  
import time  
import logging  
  
log = logging.getLogger('logger') # get the main logger  
  
  
def mouse\_location\_save\_data(request):  
 current\_mouse\_coordinates = mouse.get\_position()  
 log.debug(f"mouse coordinates: {current\_mouse\_coordinates}")  
 try:  
 entry\_name = request.split(' as ')[1]  
 log.debug(f'New entry name acquired from user: {entry\_name}')  
  
 data\_string = f'{entry\_name}@{current\_mouse\_coordinates}'  
 return data\_string  
  
 except IndexError:  
 log.exception("Syntax problem: the entry that is meant to be saved is the one specified after the word 'as'.")  
 return None  
  
  
def mouse\_location\_get\_entry(request):  
 if " to " in request:  
 sorted\_request = request.split(" to ")  
 elif " the " in request:  
 sorted\_request = request.split(" the ")  
 else:  
 sorted\_request = request.replace("fast travel ", "")  
  
 log.debug(f"sorted request for mouse travel: {sorted\_request}")  
 try:  
 if isinstance(sorted\_request, list):  
 log.debug(f'Fast travel entry extracted from request: {sorted\_request[1]}')  
 return sorted\_request[1]  
 else:  
 log.debug(f'Fast travel entry extracted from request: {sorted\_request}')  
 return sorted\_request  
  
 except IndexError:  
 log.exception("Syntax problem: the entry that is traveled to is the one specified after the word 'to'.",  
 exc\_info=False)  
 return None  
  
  
def mouse\_location\_travel(coordinates, entry):  
 if coordinates:  
 coordinates = coordinates.replace("(", "").replace(")", "").split(", ")  
  
 mouse.move(coordinates[0], coordinates[1], absolute=True, duration=0.1)  
 log.info(f'fast traveled to {entry}')

## dataHandlingServer.py:

### משתנים חשובים:

* main\_logger: אובייקט הlogger.
* server\_socket: אובייקט מסוג socket. מסוגל להחזיק מספר חיבורים בו-זמנית.
* client\_socket: אובייקט מסוג socket, הinstance המחבר בין השרת ללקוח הספציפי.
* address: רשימה דו-תאית, מחזיקה ב2 חלקי כתובת הIP של הלקוח.
* logging\_config: מופיע באותו השם ב2 פונקציות הsetup\_logging כמשתנה מקומי בעל ערך שונה בכל אחת. תמיד מילון אשר מכיל קונפיגורציה כלשהי של הloggers המיועדים.
* LOG\_FORMAT: הפורמט לפיו קבצי הלוג מסתדרים.
* IP: מוגדר כמחרוזת "0.0.0.0", כלומר השרת מקשיב לכל IP באשר הוא שייגש אליו בפורט המתאים.
* PORT: מוגדר כint 4001. הפורט דרכו מתבצע חיבור השרת-לקוח.
* tcp\_log\_server: אובייקט מסוג LogRecordSocketReceiver בעל logname (תכונה) ייחודי ללקוח. אחראי על שמירת logging ללקוח בקבצי הסרבר.

תוכן:

import \_pickle as pickle  
import logging  
import logging.handlers  
import os  
import socket  
import socketserver  
import struct  
import threading  
from logging.config import dictConfig  
  
import select  
  
LOG\_FORMAT = '%(asctime)s | %(threadName)s - %(levelname)s - %(filename)s / %(name)s - %(lineno)s - func=%(funcName)s : "%(message)s"'  
IP, PORT = '0.0.0.0', 4001  
  
  
class LogRecordStreamHandler(socketserver.StreamRequestHandler):  
 *"""  
 Handler for a streaming logging request.  
  
 This basically logs the record using whatever logging policy is  
 configured locally.  
 """* def handle(self):  
 *"""  
 Handle multiple requests - each expected to be a 4-byte length,  
 followed by the LogRecord in pickle format. Logs the record  
 according to whatever policy is configured locally.  
 """* while True:  
 try:  
 chunk = self.connection.recv(4)  
 if len(chunk) < 4:  
 break  
 slen = struct.unpack('>L', chunk)[0]  
 chunk = self.connection.recv(slen)  
 while len(chunk) < slen:  
 chunk = chunk + self.connection.recv(slen - len(chunk))  
 obj = self.unPickle(chunk)  
 record = logging.makeLogRecord(obj)  
 self.handleLogRecord(record)  
 except ConnectionResetError:  
 main\_logger.error('Client crashed or forcibly closed')  
 break  
  
 @staticmethod  
 def unPickle(data):  
 return pickle.loads(data)  
  
 def handleLogRecord(self, record):  
 # if a name is specified, we use the named logger rather than the one  
 # implied by the record.  
 if self.server.logname is not None:  
 name = self.server.logname  
 else:  
 name = record.name  
 logger = logging.getLogger(name)  
 # N.B. EVERY record gets logged. This is because Logger.handle  
 # is normally called AFTER logger-level filtering.  
 logger.handle(record)  
  
  
class LogRecordSocketReceiver(socketserver.ThreadingTCPServer):  
 *"""  
 Simple TCP socket-based logging receiver suitable for testing.  
 """* allow\_reuse\_address = True  
  
 def \_\_init\_\_(self,  
 host=socket.gethostbyname(socket.gethostname()),  
 port=logging.handlers.DEFAULT\_TCP\_LOGGING\_PORT,  
 handler=LogRecordStreamHandler):  
 socketserver.ThreadingTCPServer.\_\_init\_\_(self, (host, port), handler)  
 self.abort = 0  
 self.timeout = 1  
 self.logname = None  
  
 def serve\_until\_stopped(self):  
 abort = 0  
 while not abort:  
 rd, wr, ex = select.select([self.socket.fileno()], [], [], self.timeout)  
 if rd:  
 self.handle\_request()  
 abort = self.abort  
  
  
class ClientHandlerThread(threading.Thread):  
 def \_\_init\_\_(self, sock, client\_address):  
 super().\_\_init\_\_()  
 self.sock = sock  
 self.client\_address = client\_address  
 self.current\_job\_data = None  
  
 def run(self):  
 *"""  
 Constantly check socket status using 'select'.  
 When a request comes in, identify it and  
 Call the appropriate service with the appropriate orders.  
  
 Finally, send the data back (with ERROR label in case of an error)  
 """* while True:  
 main\_logger.debug('Waiting for job key to arrive in socket...')  
 read\_socket, write\_sockets, error\_sockets = select.select([self.sock], [], [])  
 if read\_socket[0] == self.sock:  
 try:  
 job\_key = self.sock.recv(4).decode()  
 except ConnectionResetError:  
 main\_logger.error(f'Client {self.client\_address[0]} crashed or forcibly closed')  
 break  
 if job\_key == "":  
 main\_logger.info(f'Connection was lawfully terminated with {self.client\_address[0]}')  
 break  
 main\_logger.debug(f'Job key arrived in socket: {job\_key}')  
  
 self.sock.send(f'{job\_key}'.encode('UTF-8'))  
 main\_logger.debug('Sent the key back to client for confirmation')  
 self.current\_job\_data = self.receive\_from\_client()  
 main\_logger.debug(f'Received job data: {self.current\_job\_data}. Initiating job\_handler')  
 if job\_key == 'svlc':  
 return\_msg = self.save\_to\_file('coordinates')  
  
 elif job\_key == 'gtlc':  
 return\_msg = self.from\_file('coordinates')  
  
 elif job\_key == 'svpc':  
 return\_msg = self.save\_to\_file('protocols')  
  
 elif job\_key == 'gtpc':  
 return\_msg = self.from\_file('protocols')  
  
 else:  
 main\_logger.error(f'Server does not provide service to job\_key = {job\_key}')  
 return\_msg = f'Server does not provide service to job\_key = {job\_key}'  
  
 main\_logger.debug(return\_msg)  
 main\_logger.debug(f'About to send message to socket: {return\_msg}')  
 self.sock.send(struct.pack('i', len(return\_msg)))  
 self.sock.send(return\_msg.encode('UTF-8'))  
  
 def receive\_from\_client(self):  
 packed\_length = self.sock.recv(4)  
 main\_logger.debug('Received incoming data length, waiting for job data to arrive.')  
 try:  
 data\_len = struct.unpack('i', packed\_length)[0]  
 except struct.error:  
 return None  
 main\_logger.debug('About to receive data from client:')  
 chunk = self.sock.recv(data\_len)  
 while len(chunk) < data\_len:  
 main\_logger.debug('Not all data has been received. Entered receiving loop...')  
 chunk = chunk + self.sock.recv(data\_len - len(chunk))  
  
 job\_data = chunk.decode()  
 main\_logger.debug('Job data received and decoded!')  
 return job\_data  
  
 def save\_to\_file(self, content\_type):  
 if content\_type == 'coordinates':  
 separator = '@'  
 else: # content type is protocols  
 separator = '['  
 entry, content = self.current\_job\_data.split(separator)  
 main\_logger.debug(f'Saving to {content\_type} file: {entry} - {content}')  
  
 if content\_type == 'protocols' and entry == 'junk':  
 main\_logger.info('Received junk keyword, ignoring request...')  
 return 'Received junk keyword, server ignored the data as requested'  
  
 try:  
 with open(fr"C:/DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt",  
 "r+") as f: # Update/create a new entry?  
 for line in f.read().splitlines():  
 split\_line = line.split(separator)  
  
 if entry == split\_line[0]: # If true, update the entry.  
 f.seek(0)  
 current\_file = f.read()  
 past\_content = split\_line[1]  
 main\_logger.debug(f"Past entry content: {past\_content}")  
 replaced\_data = current\_file.replace(past\_content, f"{content}")  
 f.seek(0)  
 f.write(replaced\_data)  
 main\_logger.info(f"Updated {self.client\_address[0]} {content\_type} file.")  
 return f'Successfully updated {content\_type} entry: {entry}'  
  
 with open(fr"C:\DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt", "a+") as f: # Create new entry  
 f.write(f'\r\n{self.current\_job\_data}')  
 main\_logger.info(f"Added {entry} to {self.client\_address[0]} {content\_type} file")  
 return f'Successfully added new entry: {entry}'  
  
 except FileNotFoundError:  
 open(rf"C:/DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt",  
 "x").close() # Create new file, re-call function  
 main\_logger.warning(f"Created new file to save {content\_type} for {self.client\_address[0]}.")  
 return self.save\_to\_file(content\_type)  
  
 except PermissionError:  
 main\_logger.exception(  
 f"Permission error has occurred in creating/accessing the {self.client\_address[0]} {content\_type} file. "  
 f"Seeking permission", exc\_info=True)  
 os.chmod(fr"C:/DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt",  
 755) # Give permission, re-call function  
 main\_logger.warning("Permission 'os.chmod(755)' given.")  
 return self.save\_to\_file(content\_type)  
  
 except IndexError:  
 main\_logger.exception("Sabotage in the text file data could have caused the exception", exc\_info=True)  
 return f'ERROR: IndexError occurred while trying to tamper with the {content\_type} file. Server received an exception log.'  
  
 def from\_file(self, content\_type):  
 if content\_type == 'coordinates':  
 separator = '@'  
 # else -> content\_type is 'protocols':  
 else:  
 separator = '['  
  
 try:  
 with open(fr"C:/DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt", "r") as f:  
 for line in f.read().splitlines():  
 split\_line = line.split(separator)  
 if self.current\_job\_data == split\_line[0]:  
 return\_value = split\_line[1]  
 return return\_value  
  
 main\_logger.error(f"{self.current\_job\_data} entry not found")  
 return f'ERROR: Entry not found in server files: {self.current\_job\_data}'  
  
 except FileNotFoundError:  
 open(f"C:/DeiJay-Server Data/{self.client\_address[0]}/{content\_type}.txt",  
 "x").close() # Create the file, then log the error  
 main\_logger.error(f"no {content\_type} saved yet. ")  
 return f'ERROR: No {content\_type} were saved yet, the file is empty.'  
  
 except IndexError:  
 main\_logger.exception("Sabotage in the text file data could have caused the exception", exc\_info=True)  
 return f'ERROR: IndexError occurred while trying to tamper with the {content\_type} file. Server received an exception log.'  
  
  
class LoggingHandlerThread(threading.Thread):  
 def \_\_init\_\_(self, client\_address):  
 super().\_\_init\_\_()  
 self.client\_address = client\_address  
  
 def run(self):  
 setup\_socket\_logging(rf'C:/DeiJay-Server Data/{self.client\_address[0]}/logs', self.client\_address)  
  
 tcp\_log\_server = LogRecordSocketReceiver()  
 tcp\_log\_server.logname = f'{self.client\_address}'  
 print('About to start TCP logging server...')  
 tcp\_log\_server.serve\_until\_stopped()  
  
  
def setup\_socket\_logging(log\_dir, client\_address, fmt=LOG\_FORMAT):  
 if not os.path.exists(log\_dir):  
 os.makedirs(log\_dir)  
  
 logging\_config = {  
 'version': 1,  
 'disable\_existing\_loggers': False,  
 'formatters':  
 {  
 'standard':  
 {  
 'format': fmt  
 }  
 },  
 'handlers':  
 {  
 'default':  
 {  
 'class': 'logging.StreamHandler',  
 'formatter': 'standard',  
 'level': 'DEBUG',  
 'stream': 'ext://sys.stdout'  
 },  
 'toDebugFile':  
 {  
 'class': 'logging.handlers.RotatingFileHandler',  
 'formatter': 'standard',  
 'level': 'DEBUG',  
 'maxBytes': 1000000,  
 'backupCount': 3,  
 'filename': f'{log\_dir}/debug.log'  
 },  
 'toErrorFile':  
 {  
 'class': 'logging.handlers.RotatingFileHandler',  
 'formatter': 'standard',  
 'level': 'WARNING',  
 'maxBytes': 100000,  
 'backupCount': 7,  
 'filename': f'{log\_dir}/error.log'  
 },  
 },  
 'loggers':  
 {  
 f'{client\_address}':  
 {  
 'handlers': ['default', 'toErrorFile', 'toDebugFile'],  
 'level': 'DEBUG',  
 'propagate': False  
 }  
 }  
 }  
  
 logging.config.dictConfig(logging\_config)  
  
  
def setup\_server\_logging(log\_dir, fmt=LOG\_FORMAT):  
 if not os.path.exists(log\_dir):  
 os.makedirs(log\_dir)  
  
 logging\_config = {  
 'version': 1,  
 'disable\_existing\_loggers': False,  
 'formatters':  
 {  
 'standard':  
 {  
 'format': fmt  
 }  
 },  
 'handlers':  
 {  
 'default':  
 {  
 'class': 'logging.StreamHandler',  
 'formatter': 'standard',  
 'level': 'DEBUG',  
 'stream': 'ext://sys.stderr'  
 },  
 'toDebugFile':  
 {  
 'class': 'logging.handlers.RotatingFileHandler',  
 'formatter': 'standard',  
 'level': 'DEBUG',  
 'maxBytes': 1000000000,  
 'backupCount': 7,  
 'filename': f'{log\_dir}/debug.log'  
 },  
 'toErrorFile':  
 {  
 'class': 'logging.handlers.RotatingFileHandler',  
 'formatter': 'standard',  
 'level': 'WARNING',  
 'maxBytes': 1000000000,  
 'backupCount': 7,  
 'filename': f'{log\_dir}/error.log'  
 },  
 },  
 'loggers':  
 {  
 'serverLogger':  
 {  
 'handlers': ['default', 'toErrorFile', 'toDebugFile'],  
 'level': 'DEBUG',  
 'propagate': False  
 }  
 }  
 }  
  
 logging.config.dictConfig(logging\_config)  
  
  
def main():  
 if not os.path.exists('C:/DeiJay-Server Data'):  
 os.mkdir('C:/DeiJay-Server Data')  
  
 try:  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind((socket.gethostname(), PORT))  
 main\_logger.debug(f"server bound and listening for clients at {PORT}")  
 server\_socket.listen(5)  
  
 while True:  
 client\_socket, address = server\_socket.accept()  
 main\_logger.info(f'Accepted connection from {address[0]}:{address[1]}')  
  
 logging\_handler = LoggingHandlerThread(address)  
 client\_handler = ClientHandlerThread(client\_socket, address)  
 main\_logger.debug('Created threads for client logging and request handling.')  
  
 client\_handler.start()  
 logging\_handler.start()  
  
 except ConnectionResetError:  
 main\_logger.error('Client crashed or forcibly closed')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 setup\_server\_logging('C:/DeiJay-Server Data/server logs')  
 main\_logger = logging.getLogger('serverLogger')  
 main()

## הסבר על keyword\_sorter שבDeijayMain.py:

בתור מפתח, על מנת להוסיף יכולות לתוכנית יש להבין את האלגוריתם שעובר על המילון הזה. האלגוריתם רץ על המילון לפי key. הvalue במילון יכול להיות **רשימה המכילה סטים** או **סט יחיד**. בכל סט יש מילות מפתח. האלגוריתם בודק intersection בין הסט המכיל את המילים שעשה בהם שימוש הלקוח בעת הבקשה, ובין הסט/ים המתאים/ים לkey במילון. במקום להסביר את האלגוריתם צעד-צעד, ניתן לחשוב על זה כך:

בין כל ערך (מילת מפתח) בסט קיים חיבור לוגי "או", ובין כל סט ברשימה קיים חיבור לוגי "וגם". כלומר, על מנת שסט מסוים יוכל לייצג "True", מילה אחת לפחות מתוכו חייבת להימצא בתוך סט הבקשה (הסט שנוצר מתוך בקשת המשתמש). על מנת שהvalue של הkey יוכל לייצג “True”, על כל הסטים ברשימה (אם קיימת) להצליח בבדיקה זו, כלומר מילה אחת לפחות מכל סט שמופיע ברשימה חייבת להופיע בסט הבקשה.  
האלגוריתם יבחר בkey המתאים בצורה שכזו, ויכניס את הערך הנ"ל (reference למתודה כלשהי של האובייקט) למשתנה לו קוראים כפונקציה.

בתור מפתח, על מנת להוסיף מתודה חדשה יש להוסיף מילות מפתח בשילוב שלא מתבצע במקום אחר במילון, ולפי הסדר שמתואר כאן.

# רפלקציה

הפרויקט הזה היווה עבורי חוויה מכוננת. מכל תחומי העניין שלי, תחום העניין היחידי האחר בו השקעתי לאורך חודשים בפרויקט כלשהו, זה למעשה לימודי פסנתר... וגם כאשר אני משווה את האינטנסיביות של בניית וכתיבת הפרויקט עם למידת מנגינה קלאסית מסובכת, הפרויקט הזה יוצר פער עצום על כל דבר שאי פעם השגתי. ברפלקציה הזו ראשית אתאר את החוויה, שנית את מידת שביעות הרצון שלי מהמקום אליו הפרויקט הגיע בעת כתיבת הרפלקציה, שלישית את מה שלמדתי בתהליך ולאחר מכן אכתוב על מספר דברים שהייתי משנה בפרויקט / בתהליך כתיבת הפרויקט אם הייתי יכול להתחיל מההתחלה, דברים הקשורים בעיקר לידע שיש לי כיום ושלא היה לי דאז. לבסוף אכתוב בקצרה על כמה יישומים שלא הצלחתי להכניס לפרויקט עד רגע כתיבת הרפלקציה, והסיכויים שבעת ההגשה הסופית הם כן יגיעו כחלק מן הפרויקט...

במהלך שנותיי במגמת סייבר, למדתי ידע תיאורטי מסוים, בנושאים ספציפיים מאוד. עם זאת, במקום לבחור בפרויקט שקשור ישירות לידע שהיה לי (תקשורת בעיקר), החלטתי משום מה על פרויקט יותר ספונטני בעל אווירה יזמית. כלומר, הבחירה בפרויקט לא נעשתה מטעמי שיקולים מקצועיים (כיצד אני יכול להשתמש בידע שלי בצורה הטובה ביותר), אלא משיקולים של סקרנות ורצון ליצור משהו חדש ושימושי. בנוסף, רציתי להשתמש בהזדמנות הזו לבחון את היצירתיות וכושר פתרון-הבעיות שלי מול גוף בוחן, תוך שאני נתקל בבעיות שאין להם מענה חד-משמעי ברשת. תהליך בניית הפרויקט עצמו היה קשה מאוד מנטלית בהתחלה, שכן הייתי צריך לסמוך שהתמונה של הפרויקט הסופי שציירתי לעצמי בדמיון היא לא סתם דמיונית אלא אפשרית. לאורך זמן הפרויקט החל לרקום צורה מחוץ לדמיון... עם כל פונקציה מוצלחת שנוספה התמונה נראתה ברורה יותר. לקראת סוף הפרויקט (לפני כחודשיים) התחלתי לעשות שימוש בפרויקט בחיי היום-יום שלי כן על מנת למצוא באגים ולחשוב על יישומים נוספים, וכן על מנת להבין אם אכן התוצר משיג את מטרתו. אני יכול להגיד בביטחון שעבור המחשב הנייח שלי, הפרויקט לא רק שימושי אלא הפך לחלק כמעט בלתי-נפרד מהדרך בה אני משתמש במחשב כעת. עבור המחשב הנייד, בשל התלות בחיבור אינטרנטי (API של גוגל...) ובשל הווייפיי הקליטה החלשה אצלנו בבית, מצאתי את יעילות הפרויקט פחותה. עם זאת, הצלחתי בבניית פרויקט שאין לי ספק שאעשה בו שימוש נרחב בעתיד, בעיקר לאחר יצירת מספר יישומים נוספים אשר ישלימו אותו לחלוטין מבחינת נוחות-שימוש (להבהרה: הפרויקט מכיל את כל השימושים שלו כבר עכשיו, ה"יישומים" קשורים לנוחות-שימוש / ממשק משתמש).

איני חושב שזו הגזמה מצידי להעריך כי לפחות 50% מן הזמן שהשקעתי בפרויקט, השקעתי בחיפוש באינטרנט אחר מידע. עם הזמן והניסיון אף למדתי כיצד ואיפה לחפש מידע בצורה מהירה יותר... עם זאת, איני יכול להגיד כי ביצעתי "מחקר ספרותי", שכן המידע שחיפשתי נמצא בפורומים ובאתרים חינוכיים, בשאלות/תשובות של מתכנתים ובסרטוני הסבר... לפעמים המידע היה מזדמן (למדתי על מבני הנתונים בצורה מעמיקה יותר משידעתי מקודם באופן מזדמן) ולפעמים חיפשתי אחר מידע ספציפי – למשל לגבי דרכים לבצע פעולות בצורה יעילה או קריאה יותר באמצעות מבני נתונים (כך הגעתי להבנה שאפשרי להכניס ערך של פונקציה למילון, הבנה שפותחה לרעיון של קיצור שרשרת if/elif/else לאלגוריתם פשוט יחסית שרץ על מילון). אני מרגיש כעת כי המידע שצברתי במהלך הפרויקט נותן לי ראיה שונה על פתירת בעיות משהייתה לי בתחילת הפרויקט.

החרטה העיקרית שלי היא ללא ספק הדרך בה התמודדתי עם יצירת מאגר הנתונים. לפני כשנה, בתחילת הפרויקט, מתוך חוסר ניסיון וחוסר סקרנות בנושא שמירת נתונים (כל הסקרנות שלי הופנתה לבניית הרעיון, לא לביצוע הרעיון באותה התקופה) החלטתי בצורה שרירותית על שמירת נתונים בקבצים. כיום אני יודע כי החלטה על מאגר נתונים לא יכולה להיות שרירותית... אם הייתי עובד עם SQL מלכתחילה, עבודת השרת הייתה פשוטה בהרבה והיה ניתן לחסוך יצירת מספר כה רב של קבצי טקסט (לכל לקוח 2 קבצי טקסט לנתונים, במקום מאגר נתונים **אחד** לכולם שמפריד בין לקוחות וסטים של מידע בצורה יעילה וקריאה יותר).

אם היה לי את הזמן להוסיף אותם, יש לי מספר רעיונות ליישומים שיכולים להפוך את חיי המשתמש לקלים הרבה יותר, עם פוטנציאל כה גבוה עד לכדי הזנחת הצורך של המשתמש להרחיב את מאגר הנתונים ברוב המקרים. הדבר ייעשה על ידי עדכון מאגר הנתונים והוספת נתונים חדשים אליו באופן אוטומטי.  
כעת הדבר מתבצע על ידי המשתמש באופן כמעט-ידני, אך ישנם דרכים להפוך את הדבר לפחות לחצי-אוטומטי לדעתי.   
הדבר יתבצע באופן שונה בדפדפנים ובשולחן העבודה: בדפדפן, ניתן לעשות שימוש בweb scraping על מנת לאתר מיקומים של אלמנטים שונים, ולהגדיר את ה"שם" שצמוד למיקומים הללו לפי תמונה ששמורה במאגר הנתונים מראש (עדכון מאגר הנתונים) או לפי טקסט שמצורף לאלמנט בדפדפן (הוספת נתונים חדשים למאגר). בשולחן העבודה, הדבר עשוי להתבצע באמצעות סריקת צילום מסך לפי 2 אלגוריתמים: אחד יהיה מסוגל לזהות טקסט המצוי על תמונה, והשני יהיה מסוגל לזהות תמונות מסוימות ע"י השוואת תמונות. זוג האלגוריתמים הפועלים במקביל, אם יוזן אליהם מספיק מידע בצורת לוגואים ושמות של אפליקציות/תוכנות ידועות, האלגוריתמים יהיו מסוגלים לזהות חלק נכבד מן האפליקציות שנמצאות על המסך בכל רגע נתון.   
במצב שכזה, המגבלה על הזיהוי האוטומטי תהיה אמורה להיות גודל מאגר הנתונים, דיוק האלגוריתמים בהשוואות שלהם, ורמת הדיוק של/חוסר-הפגמים-ב נתונים הנמצאים במאגר.

לסיכום, אני מאמין כי למדתי והתפתחתי הרבה מפרויקט זה, אם כי הידע התיאורטי שהוצג בו מעט לוקה בחסר לטעמי. אני ללא ספק אמשיך לעבוד עוד על הפרויקט הזה לשימוש אישי, וכן לצבירת עוד ניסיון וידע על בניית מערכות מתקדמות יותר.