ספר פרויקט – שנהב מור

**Static Analysis Platform(SAP)**

**שם המכללה – הרב תחומי עמל ב'**

**שם הסטודנט – שנהב מור**

**ת.ז סטודנט – 322988304**

**שם המנחים – עמית בלומנפלד ואהרון סלינס**

**תאריך הגשה – 5/2021**

תוכן עניינים –

הצעת פרויקט –

**הצעת נושא לפרויקט הנדסאים**

למילוי חלקי של הדרישות לקבלת

תואר הנדסאי במגמת הנדסת תוכנה

1. **שם הפרויקט: Static Analysis Platform(SAP)**

**2. שם הסטודנט: שנהב מור ת.ז: 322988304**

**3. שם המנחה: עמית בלומנפלד, אהרון סלינס תואר: הנדסאי תוכנה**

#### 4. תיאור הפרויקט:

מטרת הפרויקט הינה פיתוח תשתית אחודה המאפשרת למחלקת פיתוח בחייל האוויר לבצע בדיקות על הקוד של מחלקת תוכנה לשיפור האבטחיות ואיכות הקוד.

הפרויקט מאפשר מעבר על קבצי הקוד בצורה יעילה תוך כדי ביצוע אנליזה מעמיקה ואפשרות שינוי פשוטה למתכנת, לשיפור הקוד הסופי.

#### 5. הגדרת הבעיה האלגוריתמית:

כיום בחיל האוויר מפותחות מערכות קריטיות ומבצעיות. בדיקת תקינות הקוד אינה אמינה ומכילה הרבה בעיות. ניתוח הקוד עצמו הינו פשטני ואינו מעמיק, דבר שלא מוצא בעיות לפני ריצת הקוד. בדיקת קוד לוקחת זמן של מפתח מנוסה מהפיתוח עצמו.

הפלטפורמה צריכה להיות דינמית על מנת לאפשר בדיקות נכונות גם כאשר צורת העבודה בצוות משתנה.

#### 6. רקע תיאורטי בתחום הפרוייקט:

תהליך הפיתוח מכיל בדיקה של הקוד מספר רב של פעמים על ידי מתכנתים מנוסים שונים בשל חשיבותו לפעול בצורה נכונה.

תהליך זה אינו מושלם, מכיוון שישנם טעויות אנוש למרות מספר רב של חזרות על מעבר הקוד.

בנוסף תהליך זה לוקח הרבה זמן. ישנם פעמים בהם מתכנת קורא את אותו קוד מספר פעמים בשל שבבדיקה הראשונה אינו מצא את הבעייה ורק לאחר השינויים גילה אותה.

המתכנתים לא מעמיקים יותר מדי בלוגיקה של הקוד, ובעיקר בודקים סטנדרטים בשל כמות הקוד שהם צריכים לקרוא ולהבין כל יום, במקביל הם צריכים לעמוד גם בזמני הפיתוח.

#### 7. תהליכים עיקריים בפרוייקט:

הפרויקט צריך לתמוך במספר תכונות עיקריות כדלקמן:

1. יכולת התממשקות עם קוד אחר, והשגת התוצאה שלו.
2. יכולת לערוך את הקוד הנבדק.
3. שימוש ב REST api.
4. ניהול הרחבות, כלי עזר והגדרות מערכת.
5. זיהוי patterns בקוד c והמבנה שלו.
6. תמיכה בממשק משתמש גרפי.
7. שמירת היסטוריה של ביצועים (logging).
8. תמיכה באנליזה ברמת method, system, mission.
9. תמיכה באנליזת SAST, זיהוי העתקים, עמידה בסטנדרטים, abstract interpretation, model checking.
10. פירסור מידע על הקוד בצורה סימטרית.
11. תמיכה במספר שפות תכנות המשתמשות בפלטפורמה.

#### 8. תיאור הטכנולוגיה

טכנולוגיית ניתוח קוד על בסיס שיטות קיימות.

#### 9.צד שרת

צד השרת יהיה אחראי על לנהל תקשורת מול הרבה מוצרים רשתיים,

לבצע בקשות מהלקוח ולהעביר אליו את המידע הניהולי המבוקש

#### 10. שפת תכנות בצד שרת

שפת C#, REST API

#### 11. צד לקוח

צד הלקוח יבקש מידע על הקוד בנוסף למידע נוסף שהוא צריך ויחזיר לשרת את תוצאות הפעולה שלו בנוסף למידע על הפעילות שלו.

#### 12 . שפת תכנות בצד לקוח

C#, REST API

#### 13. מסד נתונים

מסד נתוני כלים:

\* את הכלים אשר צריך להשתמש בהם

\* את דרך הפעלת הכלי.

\* את תצורת המידע שהכלים צריכים בכדי לפעול.

#### 14. פרוטוקלי תקשורת

REST API

#### 15. לוחות זמנים

|  |  |  |
| --- | --- | --- |
| שלב | טווח תאריכים | |
|  | מ- | עד- |
| ניתוח המערכת | 1/7/2020 | 30/10/2020 |
| עיצוב המערכת | 1/11/2020 | 30/12/2020 |
| פיתוח התוכנה/קידוד | 1/1/2021 | 30/3/2021 |
| ניפוי השגיאות | 1/4/2021 | 15/4/2021 |
| הרצת ניסיון | 16/4/2021 | 30/4/2021 |
| הגשת הפרוייקט | 5/2021 | 5/2021 |

#### 16. חתימת הסטודנט: \_\_\_\_\_\_\_\_\_\_\_

#### 17. חתימת רכז המגמה: \_\_\_\_\_\_\_\_\_\_\_

#### 18. אישור משרד החינוך: \_\_\_\_\_\_\_\_\_\_\_

תקציר –

**רקע –**

בימינו תהליך הפיתוח מכיל בדיקה של הקוד מספר רב של פעמים על ידי מתכנתית מנוסים ושונים בשל חשיבותו לפעול בצורה נכונה.

תהליך זה אינו מושלם, מכיוון שישנם טעויות אנוש רבות בבדיקת קוד למרות מספר רב של חזרות על מעבר הקוד, ניתן להחמיץ טעויות בקלות.

בנוסף תהליך זה לוקח זמן רב. ישנם פעמים בהם מתכנת קורא את אותו קוד מספר רב של פעמים בשל מקרים בהם המתכנת עובר על הקוד ובפעם הראשונה לא מוצא את התקלה ולאחר בדיקת נוספות מוצא את התקלה ומתקן אותה.

המתכנתים לא מעמיקים יותר מידי בלוגיקה של הקוד, ובעיקר בודקים סטנדרטים בשל כמות הקוד שהם צריכים לקרוא ולהבין על בסיס יומיומי, במקביל הם גם צריכים לעמוד גם בזמני הפיתוח שעלולים להיות לחוצים וקשוחים.

**תהליך המחקר –**

בהתחלה חקרתי על כל מיני אפשרויות למקסם את הפתרון לבעיה שלי. בחרתי בשיטה Static Analysis - Static Analysis היא בעצם דרך לתוכנה במחשב לעשות Debug מכל מיני משמעויות וזה בעצם נעשה בלי באמת להריץ את הקוד אלא יש תוכנה הרצה ובודקת את הקוד בעזרתPatterns ועוד. רובו של הפרויקט עוסק בפלטפורמה בעצם שדואגת לחלק של הStatic Analysis היא תחילה בודקת דברים בסיסיים כמו Compiler ברובו Syntax ולאחר מכן מפרסרת את המידע כרצוי ומכינה מאגרי מידע לגבי הקוד. אותם ניתן לשלוח לכלים במקרי הצורך.

לאחר מכן מכיוון וכרגע הפרויקט נוגע אך ורק לקוד בשפת C התחלתי לקרוא ולחקור על דברים בסיסיים בC שצריכים לקרות ושניתן לזהות ולפענך, כל מיני Patterns למיניהם ועוד. לדוגמא לאבחן סוף שורה לאבחן תחילת פעולה וכו..

לאחר מכן התחלתי לעבוד על להכליל את הפלטפורמה כמה שיותר בכדי שאת אותם הכלים שניתן לבנות ולקשר עם הפלטפורמה יהיה אפשר לרשום בכל שפה ולא בשפה אחידה. לדוגמא חלק מההכללה הייתה לעבוד עם שרת Rest API (פרוטוקול HTTP) וכך כלי בכל שפה יכול לבקש דברים מהפלטפורמה ולקבל מידע מפורסר מהפלטפורמה.

ולבסוף באיזה סביבות ובאילו שפות כדאי לי לכתוב חלקים מהפרויקט.

חלק מהמחקר - האם כדאי לי להשתמש לפלטפורמה בC# או בשפה אחרת ? אז החלטתי לעבוד עם C# מכיוון ואני שולט מאוד בשפה כי למדתי אותה בתיכון ובנוסף בשפה היו דברים שהייתי צריך כגון Regex אז החלטתי לעבוד איתה.

האם כדאי לי להשתמש לממשק המשתמש בWPF או בWinform ולבסוף החלטתי להשתמש בWPF מכיוון והוא משמעותית יותר מתקדם.

ולאחר כל המחקר התחלתי לעבוד על תרשימים וכשהכל היה מוכן התחלתי לטפל בקוד.

**סקירת ספרות –**

בדיקות תוכנה, REST/HTTP, WPF MVVM, MD5.

[**בדיקות איכות תוכנה ומה שמסביב**](https://www.a-quality.co.il/p/blog-page.html) **- "דפקט - defect** - פגם ברכיב או מערכת שיכול לגרום לרכיב או למערכת להיכשל בביצוע הפונקציות הנדרשות מהם." המאמר עוסק במושגי יסוד לגבי בדיקות תוכנה QA איכות תוכנה ומה שמסביב. במאמר ניתן למצוא כמעט את כל המושגים שקשורים לבדיקות תוכנה.

[**הסבר על REST**](https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/) **-**

"It is a set of rules that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL."

מאמר זה מסביר מה זה REST וגם קצת על HTTP וגם מה שמסביב כיצד הוא בנוי וכו.

[**הסבר על MVVM**](https://www.codeproject.com/Articles/819294/WPF-MVVM-step-by-step-Basics-to-Advance-Level) **– "** MVVM is an evolution of 3 layer architecture."

הסבר על מה זה MVVM על ההיסטוריה של זה וכיצד ליצור פרויקט בWPF בעזרת שיטת MVVM גם.

הסבר על MD5

[**הסבר על MD5**](https://www.sciencedirect.com/topics/computer-science/message-digest-algorithm-5%23:~:text=Message%20Digest%20Algorithm%205%20(MD5,verify%20the%20integrity%20of%20files.) **-**

"MD5 is most commonly used to verify the integrity of files. However, it is also used in other security protocols"

הסבר על מה זה MD5 כיצד להשתמש בו ומאיפה הרעיון של MD5 הגיע פרוטוקולים ועוד.

**הבעיה איתה התמודדתי –**

כיום בחיל האוויר מפותחות מערכות קריטיות ומבצעיות. בדיקת תקינות הקוד אינה אמינה ומכילה הרבה בעיות. ניתוח הקוד עצמו הינו פשטני ואינו מעמיק, דבר שלא מוצא בעיות לפני ריצת הקוד. בדיקת קוד לוקחת זמן של מפתח מנוסה מהפיתוח עצמו.

ישנם כמה מישורים שהפלטפורמה צריכה לפתור: מישור הקבצים עצמם, מישור השפה, מישור תפניות קוד, ומישור הכלים. לכל מישור יש את הקשיים והבעיות שלו.

הנחת היסוד של הפלטפורמה היא שהקבצים שמתקבלים כקלט לא יהיו באותה מערכת אך בהכרח יהיו באותה רשת. כלומר יש להשיג את אותם קבצים או גישה עליהם. הקבצים עצמם צריכים להיות תקינים ולעבור ממקום למקום ללא שגיאות.

במישור השפה יש לפרסר את השפה בצורה שהכלים יוכלו להשיג מידע בצורה פשוטה וברורה. יש כאן עבודה מסובכת של זיהוי state machine ודיווח על חריגה מהגדרות השפה. דבר זה נהפך ליותר מסובך עקב מצבי קיצון כגון קוד שנמצא בהערות, קוד שלא עבר קימפול אבל ניתן לבדיקה, קוד שמכיל הגדרות לא סטנדרטיות , scopes שונים וקוד של ריצה מקבילית. אם יש כישלון בשלב זה שאר השלבים יכשלו.

כל כלי אשר ירשם לפלטפורמה יבקש את המידע שהוא צריך כדי לרוץ. יכול להיות שהמידע הוא מידע על הפירסור, יכול להיות שהמידע הוא תבנית קוד, יכול להיות שהמידע הינו תוצאה של כלי אחר, ויכול להיות שהכלי יבקש שילוב של הקלטים הללו. הפלטפורמה צריכה לספק את כל הדברים הללו ובנוסף להתייחס לעובדה שהכלים לא בהכרח באותה מערכת. בנוסף לאלגוריתמי חיפוש הפטרנים יש לנו כאן בעיות של halting problem וthe two generals problem. בעית Halting הוכחה כלא פתירה לכן נשתמש באלגוריתמים למיגור הבעיה, אותו דבר נכון עבור בעיית שני הגנרלים.

בנוסף לכל זה הפלטפורמה צריכה להיות נוחה לשימוש.

**יש כמה סיבות עיקריות לבחירת הנושא –**

* בדיקת קוד הינה דבר היכול לקחת זמן רב מאוד (תלוי קוד) .
* בבדיקת קוד המתכנת עלול לא לשים לב לדברים בין השורות ולפספס בעיות בקוד.
* בדיקת קוד על ידי מתכנת במהלך הלחץ לסיים את חובותיו עד לתאריך מסוים יכול להוות לחץ מיותר.

**מוטיבציה לעבודה –**

בעצם כשהתחלתי במכללה בכיתה יג' התחלנו ללמוד C ומאוד התחברתי לשפה. ראיתי שתהליך הבדיקה בשפה הוא ארוך מאוד מכיוון והשפה היא Low level אז ישנם הרבה יותר שורות קוד והרבה יותר אלגוריטמים שנרשמים ידנית ופחות פעולות שנלקחות מהאינטרנט כמו שיש למשל בC# או פייתון. ולכן תהליך הבדיקה היה לוקח לי הרבה זמן והיו פעמים שהייתי מפספס דברים שאחרי זה בימים אחרים שהייתי מסתכל על הקוד שוב לא הייתי קולט את הבעיה ומתקן אותה רק ימים רבים אחרי. וכבר אז היה נראה לי נחמד שיהיה משהו שלבד יבדוק בעצם את הקוד ויקצץ בזמנים.

אחרי שהתקבלתי לתפקיד בפרויקטנטים והבנתי שבחיל האוויר ישנם מערכות קריטיות ומבצעיות וששם בדיקת תקינות הקוד אינה אמינה מאוד ומכילה הרבה בעיות. ניתוח הקוד עצמו הינו פשטני ולא מעמיק, דבר שלא מוצא בעיות לפני ריצת הקוד. בדיקת קוד לוקחת זמן של מפתח מנוסה מהפיתוח עצמו. וחשבתי לעצמי, איזו כמות אדירה ניתן לחסוך אם מתכנת יוכל לעבור על הקוד רק פעם אחת ולא כמות רבה של פעמים. הוא יוכל להרי. את התוכנה ולחסוך לעצמו זמן בכדי לעבוד יותר על הפיתוח.

דבר זה גרם לי לנסות כמה שיותר להכליל דברים אצלי ולעבוד באופן רצוף וארוך על הקוד בכדי לעזור כמה שיותר בפתרון בעיה זו.

**הצגת פתרנות נוספים –**

ישנם המון בעיות שקשורות לבעיה האלגוריטמית אותם פתרתי, בכדי להכליל כמה שיותר את הפלטפורמה כדי שתוכל להריץ כלים מכל שפה ובנוסף גם כלים שלא נמצאים על אותו המחשב ישנם כמה דברים..

* בתחילה הדבר הכי חשוב שהכלים יוכלו להכתב כל שפת תכנות שיודעת לתקשר HTTP מכיוון שהשרת הוא שרת Rest API אז הבקשות הם בקשות HTTP לדוגמא GET,POST וכו.. מעבר הנתונים ברובו הוא בעזרת JSON וכך הכלי מקבל את המידע מסודר בJSON ונותר לו רק להשתמש בו.
* מכיוון וצריך להתחשב באפשרות שכלים ירוצו על מחשבים שונים ולא בהכרח על אותו המחשב אז הפלטפורמה מריצה את כלי בעזרת script שניתן על ידי מתכנת הכלי בזמן הרשמת הכלי למערכת. וכך ניתן להפעיל גם כלי שנמצא על מחשב אחר על שרת לדוגמא. וישנה האפשרות להעביר את המידע בעזרת שרת הRest API לדוגמא את הלוגים ואם הכלי מחזיר איזשהי תוצאה אז היא תועבר על גבי הRest API.
* בבדיקת הקומפילציה הבסיסית שהפלטפורמה עושה צצה בעיה של תוספים לדוגמא Include של קובץ H מסויים שגורם לתוספת של כל מיני Keywords למיניהם משתנים וכו. כי הרי הפלטפורמה לא תכיר את המילים השמורות שהמשתמש ישתמש בהם מכיוון ויכולים להיות מילים שמורות שהוא יצר בעצמו למשל Struct ולכן חוץ מספריות של הGCC אם ישנו קובץ נוסף המשתמש צריך בעזרת ממשק המשתמש להביא כתובות לתיקיות בהן נמצאים הקבצים. וקח הפלטפורמה מוסיפה את כל המילים השמורות וכו מהקבצים הנוספים.
* בעית הHalting אומרת בעצם שכלי יכול לתקוע את כל הפלטפורמה אם הוא תקוע למשל בלולאה אינסופית וכו. וצריך לטפל בה. בשביל לטפל בה אני מבצע בדיקות על הכלי לפני שהוא נכנס למערכת ובודק את זמן הריצה הממוצע שלו לשורת קוד. אם זמן הריצה הוא סביר והוא נוסף לפלטפורמה אז בכל פעם שהוא רץ ישנו טיימר שרץ ובודק אם הוא עובר את זמן הריצה הממוצע שלו לשורת קוד ביותר מידי הוא פשוט יסגר והבדיקות שלו לא יחשבו.. אם יש כלי שסומך על הכלי הזה הוא לא ירוץ גם כן.

**מטרות / יעדים –**

* הרצה מוצלחת של הפלטפורמה על רוב הקודים (גם קודים מלוכלכים / קשים לקריאה / Legacy Code).
* האפשרות להוסיף כלים צריכה להיות פשוטה גם מבחינת קוד וגם מבחינת ההוספה עצמה.
* שימוש בRest Api.
* תמיכה בממשק משתמש גרפי.
* שמירת היסטוריה של ביצועים (Logging).
* יכולת לערוך את הקוד הנבדק.
* פירסור המידע על הקוד בצורה סימטרית.
* כלים יכולים להשתמש בשפה לבחירתם בשביל להשתמש בפלטפורמה.
* זיהוי Patterns בקוד C והמבנה שלו.
* ניהול הרחבות , כלי עזר והגדרות מערכת.

**אתגרים –**

* תכנון והבנת המערכת.
* התייחסות לספריות/קבצים חיצוניים שנמצאים בקוד.
* פרסור המידע כרצוי.
* טיפול בקוד מבולגן ולא קריא.

**מדדי הצלחה למערכת –**

* יש כלי אחד במערכת העובד בצורה מצוינת ללא תקלות וללא Halting Problem.
* ישנו פירסור מידע גדול שהפלטפורמה מבצעת על הקוד שהלקוח נתן, פירסור שמספק את רוב הבקשות.
* הפלטפורמה כמעט ולא טועה במידע שהיא מספקת והמידע הוא סימטרי.
* הפלטפורמה יודעת לזהות שורת קוד ולספק מידע לגבי רוב הקודים (גם legacy code)
* הכלים יוכלו לתקשר עם שרת המידע בלי קשר לשפת התכנות שהם משתמשים.
* הפלטפורמה מספקת קובץ LOG גדול של כל הLOGGING של הכלים ושל הפלטפורמה מסודר ומובן לקריאה.
* תהיה אפשרות לExtension של פעולות בסיסיות כגון להוסיף פעולה Custom לפעולה כמו Malloc או Free.
* הפעלתה של התכנה היא פשוטה קלה ונוחה למשתמש.
* הפלטפורמה או השרת ידעו לספק לכלי אפשרות לבקש תוצאה של כלי אחר.

**רקע תיאורטי –**

בימינו ישנם כמה בדיקות תוכנה –

בדיקות תוכנה הן בעצם תהליך שבו המתכנת מבצע מספר בדיקות בכדי לבדוק את איכות התוכנה ולוודא כי התכנה עומדת בדרישות שהוצבו לה. בדיקות תוכנה מהוות חלק אינטגרלי מתהליכי הנדסת תוכנה ומבטיחה איכות תוכנה.

בדיקות תוכנה ותהליכי [הבטחת איכות תוכנה](https://he.wikipedia.org/wiki/%D7%94%D7%91%D7%98%D7%97%D7%AA_%D7%90%D7%99%D7%9B%D7%95%D7%AA_%D7%AA%D7%95%D7%9B%D7%A0%D7%94) באים לידי ביטוי באופנים שונים במתודולוגיות פיתוח שונות. בחלק ממתודולוגיות פיתוח התוכנה, מקומן של הבדיקות הוא בסוף תהליך הפיתוח (לדוגמא מודל מפל המים).

מודל מפל המים – מבנה המודל בנוי כך שפיתוח התוכנה מתבצע בתהליך שיטתי המורכב משלבים מוגדרים היטב. השלבים מבוצעים אחד אחר השני ובכל שלב יש מיקוד במשימה עיקרית אחת בלבד. זרימה זו נראית כמו זרימתו של מפל מים דרך מספר בריכות. בנוסף דרישה זו שמה דגש על איסוף וניתוח של כל הדרישות כולן קודם לתחילת הפיתוח, וממליצה שתהליך הפיתוח לא יחזור לאחור לאחר ששלב מסוים בו הסתיים.

ישנם 2 סוגי בדיקות עיקריים –

בדיקות קופסה לבנה (White Box) - בדיקות אלו מתבססות על הכרת [קוד המקור](https://he.wikipedia.org/wiki/%D7%A7%D7%95%D7%93_%D7%9E%D7%A7%D7%95%D7%A8) של התוכנה ובניית תוכניות בדיקה המותאמות לנתיבי הזרימה האפשריים של הקוד. בדיקות יחידה עשויות להיות בדיקות מסוג קופסה לבנה. בסוג בדיקות זה, על הבודק להכיר את הלוגיקה של הקוד, וכן, עליו להיות בעל ידע בשפת התכנות בה כתובה התוכנה.

בדיקת קופסה שחורה (Black Box) - בדיקות אלו אינן מכירות את המבנה הפנימי של המערכת ומתבססות על בדיקת הפלט הצפוי לקלט מסוים בהתאם לתכנון מוקדם כלשהו. בדיקות קבלה מתבצעות בשיטה זו בדרך כלל. בסוג בדיקות זה, הבודק חייב לדעת את פירוט דרישות המערכת, וכן, עליו לדעת לאיזה פלט מהתוכנה עליו לצפות עבור קלט מסוים. עם זאת, הבודק אינו חייב להכיר את הלוגיקה של הבעיה או אפילו לדעת את שפת התכנות בה היא כתובה.

בימינו ישנה גם בדיקה שמשלבת את הקופסה הלבנה והשחורה וקוראים לה בדיקת קופסה אפורה (Gray Box) - בדיקות אלו מכירות במבנה הפנימי של המערכת אך משתמשות בידע זה על מנת לבצע בדיקות בסגנון קופסה שחורה. כך לדוגמה שינוי של מאגרי הנתונים לבדיקת פלט מסוים או שימוש ב[הנדסה הפוכה](https://he.wikipedia.org/wiki/%D7%94%D7%A0%D7%93%D7%A1%D7%94_%D7%94%D7%A4%D7%95%D7%9B%D7%94) על מנת לאתר את גבולות הפעולה של המערכת, אלו הן דוגמאות נפוצות לבדיקות בשיטה זו.

בעבר היו בדיקות התוכנה תחום חובבני שבוצע על ידי התוכניתנים עצמם או על ידי עובדים ללא הכשרה מתאימה. כיום הנדסת בדיקות תוכנה היא מקצוע נלמד, בדיקות התוכנה כוללות כתיבת תסריטי בדיקות תוכנה ושימושים מתקדמים **בכלי בדיקה אוטומטיים.**

בדיקות תוכנה ממוכנות –

בדיקות תוכנה ממוכנות (Test Automation) הן בדיקות תוכנה המבוצעות בדרך אוטומטית על ידי שימוש בתוכנת מחשב ייעודית (נפרדת מהתוכנה הנבדקת).

ניתן לממש בדיקות אוטומטיות כתחליף לתהליכי בדיקות ידניים, או בנוסף להן – עבור תרחישים שלאדם יהיה קשה מאוד לבצע אותם (ולעיתים בלתי אפשרי) לבצע ללא מיכון.

ניתן לחלק את בדיקות אלו לשני סוגים –

* בדיקות מבוססות קוד – הממשקים והמודולים השונים של המערכת נבדקים באמצעות קלטים מסוגים שונים תוך וידור התוצאה המוחזרת.
* בדיקות ממשק משתמש – סביבת האוטומציה מייצרת אירועים המדמים אינטראקציה של משתמש עם המערכת כגון שימוש במקלדת ובעכבר, בוחנת את השינויים בממשק המשתמש שנגרמו כתוצאה מכך ומשווה אותם להתנהגות הרצוייה.

כלי בדיקה יכולים לעזור להפוך מטלות לאוטומטיות כגון התקנת מוצר, יצירת נתוני בדיקה, אינטראקציית [ממשק משתמש גרפי](https://he.wikipedia.org/wiki/%D7%9E%D7%9E%D7%A9%D7%A7_%D7%9E%D7%A9%D7%AA%D7%9E%D7%A9_%D7%92%D7%A8%D7%A4%D7%99), זיהוי בעיות וכו', לא בהכרח על ידי הפיכת הבדיקות לאוטומטיות בתצורת רשת קצה-אל-קצה.

הפלטפורמה שלי בעצם היא תוכנה שמתחילה ובודקת קוד של לקוח שהובא לידה בתחילה היא עושה קימפול בסיסי ולאחר מכן היא ממש עושה בדיקות Syntax בדיקות התנהגות תוכנה ובדיקות קוד נוספות לאחר מכן מתחילה לאסוף מידע לגבי הקוד על גבי המילון אותו תשלח לכלים בעתיד במידת הצורך כל זה כמובן זה ללא הרצה של הקוד האמיתי אלה רק התבוננות בו.

Static Analysis -

במדעי המחשב, ניתוח קוד סטטי הוא תהליך אוטומטי לבחינת התנהגות תוכנה ללא הרצתה, וזאת בניגוד לניתוח קוד דינאמי, אשר מחייב את הרצת התוכנה לצורך בחינת התנהגותה. ניתו קוד סטטי מוצא ומעדכן את המנתח אודות שגיאות ודאיות ושגיאות אפשריות ארש עלולות להתרחש בזמן הרצת התוכנה.

לאחר שדיברנו על בדיקות תוכנה שמסדרים לנו את איזור הבדיקה בפלטפורמה שלנו נצטרך להבין כיצד להעביר את המידע שאנחנו צריכים לאגור לכלים כדי שיוכלו לעבוד איתו. צריך לזכור שהכלים לא חייבים להיות על אותו מחשב ויכולים להיות כתובים בכל שפה. ולכן החלטתי להשתמש בשרת Rest API.

שרת Rest API – **Re**presentational **S**tate **T**ransferהוא סגנון תוכנה למימוש שירותי רשת. הרעיון העיקרי הוא הגדרת משאב מערכת שמצבו עובר שינוי כתוצאה מהאינטרקציה בין מספר השירות והצרכן. ברוב המקרים הREST מתבצע במימוש פרוטוקול הHTTP.

פרוטוקול HTTP – פרוקוטול הHTTP הוא פרוטוקול תקושרת שנועד להעברת דפי HTML ואובייקטים שהם מכילים (כגון תמונות קובצי קול וכו..) ברשת האינטרנט. שרתי HTTP הם שרתי התוכן המרכזיים ברשת האינטרנט ודפדפנים הם תוכנות הלקוח הנפוצות ביותר לפרוטוקול HTTP.

התקשורת בHTTP מתחילה ביצירת שיחה בין השרת ללקוח באמצעות פרוטוקול TCP[[1]](#footnote-1) בשכבת התעבורה של פרוטוקול TCP/IP, ונמשכת בסדרה של בקשות ותשובות שנשלחות על ידי הלקוח והשרת בהתאמה בהתחלה הלקוח יוצר חיבור לכתובת הIP ולפורט בו השרת נמצא, בדרך כלל פורט 80 אם זה ברשת. לאחר מכן נשלחת הבקשה,השרת קורא את הבקשה ומחזיר תשובה בהתאם ובדרך כלל סוגר את החיבור על הלקוח לאחר שליחת התשובה.

וכך בדיוק עובד גם שרת הREST ועל גבו אני מעביר את המידע. הכלים מבקשים מה שהם צריכים בפקודות GET[[2]](#footnote-2) ואם הם צריכים להחזיר איזשהי תוצאה או [[3]](#footnote-3)LOG הם יכולים גם להשתמש ב[[4]](#footnote-4)POST. בפקודות GET השרת מחזיר להם ב- Response את המידע שנאגר על הקוד במילון גדול שנשלח על גבי JSON.

וכך הכלים יכולים להשתמש במידע הזה ולתכנן מראש כשבונים את הקוד איזה מידע הם רוצים לקבל מהפלטופורמה ולבקש במקרה הצורך.

**תיאור מצב קיים –**

הפרויקט עוסק בפלטפורמה של Static Analysis בשפת C.

קיים מגוון רחב של חברות שמספקות תוכנות שבמצעות Static Analysis בשפת C. להלן כמה מהם –

* **Cppcheck** – כלי Open Source שבודק מספר תקלות כולל שימוש בSTL[[5]](#footnote-5). תוכנה זו מיועדת גם לC וגם לCPP. כמה Features בתוכנה זו –

1. Bounds Checking for array overruns
2. Classes Checking (CPP)
3. Memory Leaks ועוד

* **Clang** – קומפיילר Open Source שכולל Static Analyzer לשפת C וCPP. כמה Features בתוכנה זו –

1. התוכנה מאוד נגישה ותומכת בGCC.
2. אחת המטרות העיקריות של Clang היא להפחית Memory Footprint ולהפחית את זמן הקומפילציה של התוכנה.

* **Sparse** – כלי Open Source שתוכנת למצוא תקלות בLinux Kernel.
* **Lint** – התוכנה הכי ישנה של Static Analysis לשפת C התוכנה סיפקה מתכנתים במשך זמן גבוה. כמה Features בתוכנה זו –

1. סימון .programming errors
2. Stylistic Errors.
3. Suspicious Constructs.

* **Cppdepend** – מפשטת טיפול בקוד C או CPP בכך שהיא מבססת את האבחונים שלה והדרך בה היא מראה שגיאות וכו. כמה Features בתוכנה זו –

1. Hundred of Clang diagnostics.
2. Support for c++ 14.
3. Declarative code rule over LINQ query.
4. Cppdepend can tell you what has been changed between 2 builds.

על פניו, מכיוון ומדובר במערכת צבאית. יש צורך בכך שהמערכת תהיה מובנה לצבא ובטח שלא תהיה Open Source כמו רוב התוכנות שהצגתי.

**ניתוח חלופות מערכתי –**

בחירת שרת SQL -

המטרה שלי הייתה למצוא את הדרך הטובה ביותר ליצור שרת SQL שבו אשמור את כל הכלים שלי בדרך הכי פשוטה והכי נוחה להעברה. (מכיוון והפרויקט הסופי לא מיועד לרוץ על המחשב שלי אלא על מחשב אחר). ולכן בחנתי את האפשרויות שיש באינטרנט ומצאתי האפשרויות הללו :

* SQL Server
* Mysql
* SQLite3

Sql server – בתוכנה זו ניתן לפתוח את הדטהבייס ואת הטבלאות בפשטות באמצעות ממשק משתמש פשוט מאוד ויש לתוכנה אפשרות לשמור את הדטהבייס שיצרת בקובץ שנקרא .bak בעזרת קובץ זה ניתן להעביר בקלות מאוד דטהבייס ממחשב אחד למחשב אחר.

Mysql – תוכנה זו גם כן ניתן יחסית בפשטות לפתוח דטהבייס, טבלאות חדשות וכו.. בתוכנה זו אין ממשק משתמש לפי מה שקראתי והעבודה היא בעצם בעזרת סקריפטים של קוד, פשוט שורות קוד. ההעברה היא פשוטה יחסית לפי הבנתי.

SQLite3 – בSQLite3 הדטהבייס הוא חלק מהתוכנה והוא רץ עם האפליקציה לעומת Mysql ו- SQL server יצירת הדטהביס והטבלאות גם כן בעזרת סקריפטים של קוד. ההעברה אמורה להיות פשוטה מאוד.

בסופו של דבר בחרתי לבחור בSQL Server מכיוון והיה לי ידע מקדים על התכנה לאחר שבתחילת יד' למדנו קצת על התוכנה וכיצד להשתמש ומבחינת הפרמטרים שהצבתי לעצמי שהתוכנה צריכה לעמוד עליהם היא עומדת בכל הקריטריונים. ולכן בחרתי בה.

בחירת שרת תקשורת בין הכלים לפלטפורמה –

המטרה שלי הייתה למצוא שרת תקשורת שאוכל לטפל בתקשורת בין הכלים לפלטפורמה בצורה הכי הגיונית / יעילה. השרת גם צריך להיות יחסית כללי בכדי שכלי בכל שפה ומכל מחשב יוכלו לתקשר עם השרת והשרת יוכל להחזיר להם תשובה.

האפשרויות שחשבתי עליהן –

* לפתוח SOCKET רגיל לפתיחת תקשורת בין כלי לפלטופרמה.
* שרת REST (HttpListener)
* לפתח שרת REST בעצמי.

בהתחלה פסלתי את הדרך של לפתוח SOCKET לכל כלי מכיוון ודבר זה יכול להוות בעיות בין שפות תכנות שונות ואחד הקריטריונים שלי הוא שלא תהיה בעיה שכלי יהיה מכל שפה. ובגלל שפותחים SOCKET יכולות להיווצר בעיות עם שפות מסוימות. ולכן את רעיון זה פסלתי ונשארתי עם 2 רעיונות. או שאני מפתח בעצמי את השרת REST או שאני משתמש באובייקט ששמו HttpListener בC# שזו דרך פשוטה יחסית לעשות שרת REST בC#. בהתחלה רציתי לממש את הREST בעצמי והבנתי שתהליך זה יכול לקחת יותר מידי זמן והפרויקט שלי הוא בלאו הכי מאוד ארוך אז אין לי יותר מידי זמן להשקיע בשרת REST משל עצמי כי זה לא כל כך רלוונטי בכל מקרה. ולכן בחרתי להשתמש בHttpListener.

**אפיון המערכת שהוגדרה / מוצעת –**

**ניתוח דרישות המערכת –**

**Must have**

* The platform must have an API for other developers to interact and add to it.
* The platform must have at least one form of static analysis tool.
* The platform must represent standard code patterns in files (and extensions).
* The platform must give meta-data about functions (called from file, used by functions, its pattern, exit points, memory handles).
* The platform must consider compile type and compile flags.
* The platform must be symmetric in its analysis.
* The platform must support edit of the original code.
* The platform must support extension to patterns and meta-data.
* The platform must support extension of define/free/malloc/basic states.
* The platform must support saving documentation of the tool actions.
* The platform must support configuration of extensions and tools at any time.
* The platform must have the ability to get result of other tools and feed it to a tool that requests them.
* The platform must have REST API server to support different programming languages for the tools.
* Handle tool halting.

**Should have**

* The platform should support GUI.
* The platform should represent function meta-data in GUI in a way that a developer can understand.
* The platform should have a second static analysis tool.
* The platform should be configurable to which files the tool will use.

**May have**

* The platform may have a specific third static analysis tool we will explain it if all other requirements have been completed.

**מודול המערכת –**

**Platform**

1. Connection To GUI חיבור לממשק המשתמש בכדי לקבל את הפרטים שהלקוח מילא בממשק המשתמש
2. Compile Module חלק הקומפילציה בפלטפורמה, הפלטפורמה בודקת בקובץ שינתן עם עזרה מהתיקיית פרויקט שניתנה לה גם כן וכו.. ובודקת האם יש שגיאות קומפילציה בקובץ שניתן לה. אם כן, היא תפסיק את החיבור עם הלקוח ותשלח את השגיאה. אם לא היא תמשיך הלאה.
3. Parsing Information בעצם החלק בו המערכת מפרסרת את האינפורמציה מהקובץ שהיא קיבלה.
4. Json creating אחראי על להכין את הJSON הגדול ברגע שכל המידע מוכן.
5. שליחת המידע לשרת הREST בכדי שיוכל להתחיל לשלוח מידע לכלים שצריכים.
6. Tool Start אחראי על הפעלת הכלים בזה אחר זה בצורה מתוזמנת לכל אפשרות של משתנה סביבה.
7. Log Creation לאחר שהפלטפורמה מריצה כלי הוא עלול להחזיר קובץ LOG ואת קובץ זה הפלטפורמה משרשרת לקובץ הלוג שהיא יצרה בעבור הקובץ הספציפי שהמשתמש נתן בכדי שבסוף יהיה קובץ LOG אחד מסודר.
8. Sending result שליחת התוצאה לממשק המשתמש.
9. Tool halt care אחראי לוודא שאין כלי שגורם לתקיעת המערכת בגלל שהחלק שלו לרוץ לא

הפסיק.

**GUI**

1. אחראי לקבל את המידע מהלקוח (קובץ, תיקיית הפרויקט וכו..)
2. אחראי להביא את כל הכלים מהדטהבייס ולהציגם.
3. Connection To Platform אחראי להתחבר לפלטפורמה ולשלוח לה את המידע הנתון שקיבל מהלקוח וסידר.
4. אחראי על סדר הכלים. מכיוון וישנם כלים שצריכים תוצאות של כלים אחרים אז ממשק המשתמש מראש מסדר אותם בסדר הנכון כך שלא יהיה מצב בו כלי יופעל לפני שהכלי שהוא מסתמך עליו מופעל.
5. אחראי לקבל גם מידע מיוחד נוסף שהוא מילים מיוחדות לmalloc וFree במקרי הצורך.

**REST Api**

1. אחראי לבקש מהפלטפורמה את המידע שאגרה (JSON)
2. אחראי לטפל בפקודות של כלים שמבקשים ממנו דברים (GET) / שולחים לו דברים (POST) ושלוח Response כמו שצריך.
3. אחראי לדאוג לקבל תוצאה במקרה והכלי מבקש ממנו חיפוש Pattern.
4. דואג לשלוח לפלטפורמה קבצי LOG ולשמור קבצי תוצאה שהוא עלול לקבל מכלים.

**Tools**

1. דואגים כל אחד לנושא בו הוא עוסק (כל כלי מבצע דבר אחר).
2. עלולים לשלוח קבצי תוצאה וקבצי RESULT בחזרה לREST.

**Database**

1. עלול לקבל כלים חדשים במקרה או לקבל עדכונים של כלים במקרה ומישהו מוסיף או מעדכן כלי.

**AddTools GUI**

1. אחראי להביא מהדטהבייס ולהציג את כל הכלים מהדטהבייס.
2. אחראי לשלוח סקריפטים מתאימים לדטהבייס לעדכן / להוסיף / למחוק כלים מהדטהבייס.

**אפיון פונקציונלי –**

**ביצועים עיקריים –**

הביצועיים העיקריים של המערכת הם בעצם כל מה שהפלטפורמה עושה –

בהתחלה הפלטפורמה מקבלת מממשק המשתמש את כל הפרטים לגבי הלקוח ואילו כלים הוא רוצה להריץ ועוד.. לאחר מכן בודקת האם יש בקובץ שקיבלה שגיאות קומפילציה, אם כן היא לא תטפל בקובץ ותשלח ללקוח את בעית הקומפילציה שהיא מצאה. במידה ותהליך הקומפילציה עבר בהצלחה הפלטפורמה תמשיך הלאה לשלב שבה היא אוגרת מידע (חוץ ממידע ספציפי שהיא גם אוגרת במהלך הקומפילציה בשביל לחסוך זמן.) בשלב זה היא תאגור מידע מכל הקובץ שקיבלה לתוך קובץ JSON ענקי שאותה היא שומרת ושולחת לשרת הREST. (כששרת הREST מתחיל לעבוד הוא מבקש ממנה, הוא לא יכול להתחיל לעבוד עד שהסתיים חלק אגירת המידע.) ברגע שנגמר שלב אגירת המידע הפלטפורמה מפעילה את הכלים שמשתמשים בREST בכדי לבקש מידע שהם צריכים באמצעות פרוטוקול HTTP הכל עובדים לפי סדר מסוים שבממשק המשתמש כבר מוחלט (ישנו אלגוריטם שמסדראת הכלים לפי הסדר לפי שליחתם לפלטפורמה בכדי שלא יהיה מצב שיש כלי שמשתמש בכלי אחר והוא מופעל לפני הכלי שהוא צריך.) לאחר שהכל נגמר הכלים שלחו לוגים לפלטפורמה והיא הכינה קובץ לוגים אז הפלטפורמה שולחת את הקובץ החדש אם יש כזה ואם לא אז את הלוגים ובעצם מה שהכלים עשו.

**אילוצים –**

* כלים יכולים להיות מאוחסנים גם על שרתים מחוץ למחשב (כלומר לא חובה שהכלים יהיו על מחשב אחד הם יכולים להיות מרוחקים).
* הכלים יכולים להיות רשומים בכל שפה (ולכן צריך לדאוג לתקשורת שכל שפה תוכל להתחבר בקלות. לכן בחרתי בREST).
* יכולת של כלי להחזיר תוצאה ולכלי אחר לבקש את התוצאה הזו.
* יכולת של כלי להחזיר קבצי LOG לפלטפורמה.
* זיהוי Patterns בקוד C והמבנה שלו.
* יכולת התממשקות עם רוב הקודים בשפת C והשגת התוצאות שלו.
* יכולת לערוך את הקוד הנבדק.

**ארכיטקטורת הפתרון המוצע Top Down Design**

REST Api

Database

Platform

Tools

AddTools GUI

GUI

**ארכיטקטורת רשת**

Rest API

Tools

HTTP GET/POST

Response

GUI

Platform

Send Data

Send Data

A Part Of the Playform

פרוטוקול תקשורת בין הGUI לפלטפורמה.

GUI

Platform

TCP port 11111

שליחת הפרוטוקול של תווי

ASCI

<File Path>,<Folder Path>… הפרוטוקול המלא בהמשך

תחילת התקשורת בין הGUI לפלטפורמה לאחרת לחיצה בGUI על הכפתור Connect.

GUI

Platform

TCP port 11111

שליחת הפרוטוקול של תווי

ASCI

Result string (Success/Error/ETC…)

הודעה לפני סיום התקשורת בין הפלטפורמה לממשק המשתמש.

פרוטוקול תקשורת בין שרת הREST לבין הכלים.

Tools

REST API

TCP port 8081 (REST)

שליחת פרוטוקול

HTTP

HTTP Request

בקשה מהכלי לכיוון הREST (פרוטוקול HTTP) יכול להיות GET POST וכו..

REST API

Tools

HTTP Response

שליחת הResponse של שרת הREST כמובן בפרוטוקול HTTP.

מבנה הJSON יתואר בהמשך ואפשרויות הHTTP Requests יתוארו בהמשך.

**תיאור פרוטוקולי התקשורת –**

ישנם 2 תקשורות בפרויקט.

1. בין הREST לבין הכלים
2. בין הGUI לבין הPlatform

נתחיל בתיאור פרוטוקול התקשורת בין הREST לבין הכלים –

כשהפלטפורמה נפתחת היא פותחת בהתחלה את שרת הREST להאזנה, שם הוא מחכה עד שהיא מסיימת הכל ומתחילה להפעיל כלים שרק הם יכולים לתקשר איתו באמצעות פקודות HTTP. ברגע שהכלים נפתחים אז הם שולחים לו בקשות HTTP לדוגמא GET וPOST והוא עונה עליהם בעזרת פרוטוקול HTTP בResponse. פשוט שימוש רגיל בשרת REST שמסתמך על פרוטוקול HTTP.

בין הGUI לבין הPlatform –

הGUI הוא קובץ נפרד מהפלטפורמה ובשביל להתחיל לבדוק על קובץ חדש פותחים אותו. כשפותחים אותו הוא והפלטפורמה מבצעים חיבור באמצעות SOCKET בC# פשוט. הלקוח מקליד אינפורמציה ולוחץ Connect (בהנחה שהפלטפורמה מופעלת). ברגע שלוחצים Connect הGUI מתחיל להכין את כל הפרוטוקול של ההודעה שהוא ישלח לPlatform. ההודעה שהוא שולח לפלטפורמה נראית כך –

<File Path>,<Folder Project Path>,<GCCPath>,<anotherExtensions>,

<FolderDestination>,tools={<Tool1>,<Tool2>},memory={<memory1>,<memory2>},

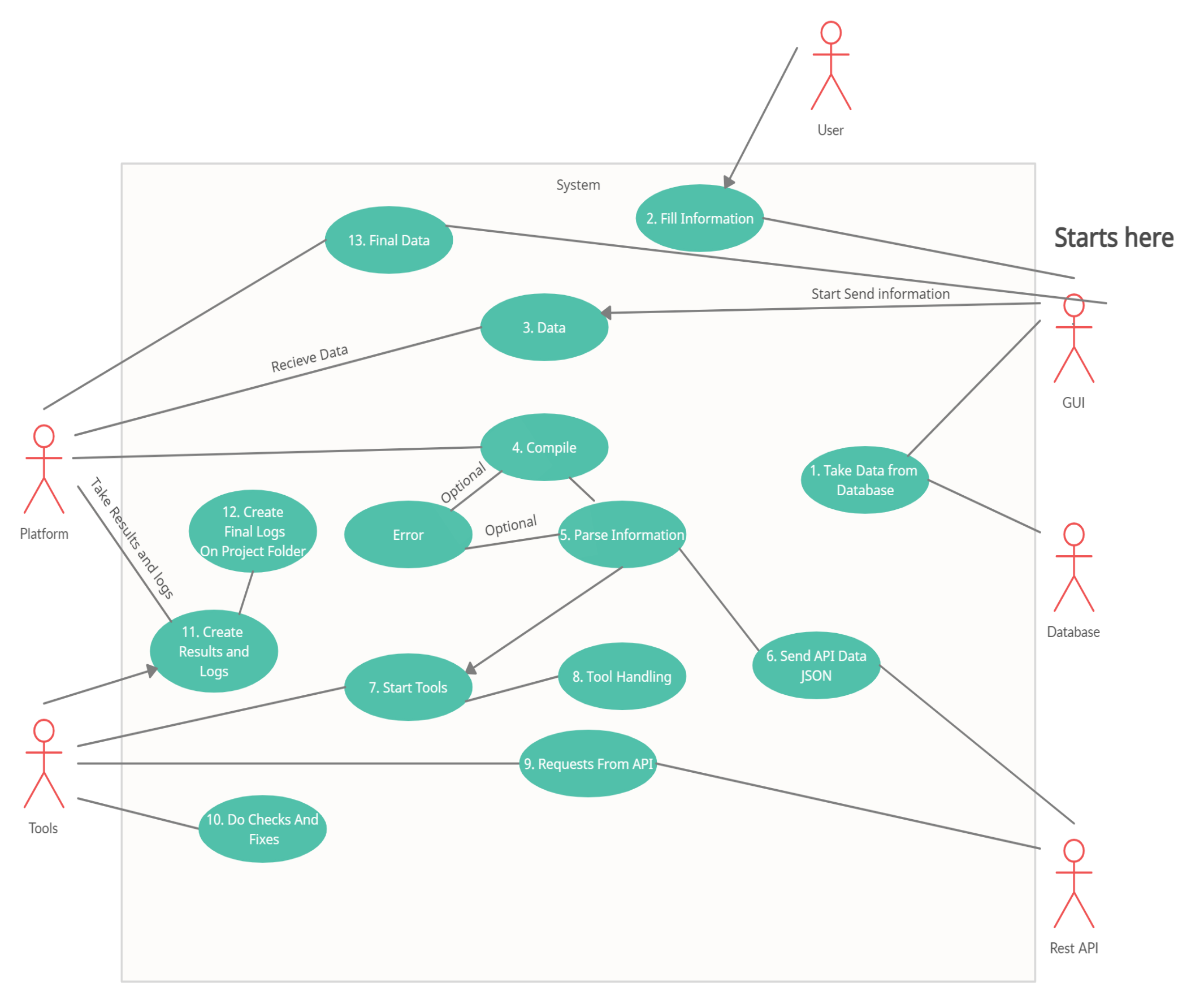
free={<Free1>,<Free2>},environmentVariablePath={<Path>}

חוץ מה3 הראשונים כולם אופציונלים. יכול להיות גם רק כלי אחד ויכולים להיות גם 10 כנל על כל השאר לדוגמא memory שבעצם אומר סימונים מיוחדים לmalloc לדוגמא safe\_malloc או בעצם דרך אחרת שהשתמשת בפעולה זו לא חייבת להיות בהודעה שנשלחת לפלטפורמה. רק אם הלקוח בחר להוסיף memory אז היא תופיע.

בפרויקט ישנם כמה שרתים וכמה לקוחות. מכיוון ובפלטפורמה ישנו שרת REST, GUI, פלטפורמה וכלים אז אסביר את ההיררכיה.

בתוך הפלטפורמה ישנו שרת Rest Api שהוא בעצם השרת שדואג לכל בקשות הכלים. זאת אומרת במצב זה השרת REST הוא השרת והלקוח הם הכלים.

הפלטפורמה עצמה היא גם שרת כשמדברים על החיבור בינה לבין הGUI. בעצם הGUI הוא הלקוח של הפלטפורמה שהיא השרת.

**תיאור הUC בפרויקט –**

**תיאור הUC –**

1. ברגע שמפעילים את הGUI ישר הוא ניגש אל הDatabase ומביא את כל הכלים ומציג אותם בGUI.
2. לאחר מכן המשתמש ממלא את האינפורמציה המתאימה. (קוד, אילו כלים הוא רוצה להשתמש, תיקיית פרויקט וכו..
3. לאחר מכן הGUI מסדר את המידע בפרוטוקול המתאים שמוסבר בספר הפרויקט לפני חלק זה, ולאחר שהמידע מסודר הוא שולח אותו אל הפלטפורמה המחוברת איתו בתקשורת של SOCKET.
4. הפלטפורמה מתחילה את הריצה שלה על הקוד בקימפול בסיסי על הקוד בו נבדקים דברים בסיסיים שבודקים קומפיילרים משם יש מקרה בו יש למשתמש ארור קומפילציה בקוד ולכן צריך להפסיק את הריצה על הקוד מכיוון והוא תקול.
5. לאחר מכן אם הקומפילציה עברה ללא שגיאות קומפילציה אז פירסור המידע מתחיל, בפרסור המידע הפלטפורמה משתמש בדברים שאגרה תוך כדי בדיקת הקומפילציה ובנוסף בדברים שהיא מפרסרת תוך כדי הפירסור עצמו. הפירסור נמצא בתוך קובץ JSON גדול אותו הREST יקבל בהמשך.
6. שליחת הJSON כולו לשרת הREST ומשם הREST מוכן לקבל בקשות מכלים.
7. הפלטפורמה מפעילה את הכלים בסדר המתאים כך שאין כלי שסומך על תוצאה של כלי אחר והוא ירוץ לפניו, זאת אומרת יש סדר בריצה והם רצים אחד אחרי השני לפי הסדר שהGUI בדק וקבע בהתחלה.
8. במהלך שהכלים רצים ישנו את הTool Handling שבודק האם הכלים נתקעים. אם הם נתקעים אז כל הפלטפורמה תעצור ותבקש לקרוא לאחראי מערכת שיבדוק מה קרה, תגיד איזה כלי נתקע ותמליץ אולי למחוק כרגע את כל מי שקשור אליו עד שיתוקן.
9. הכלי יכול להתחיל וסביר להניח שיתחיל לבקש בקשות מהREST.
10. הכלי יעשה את הבדיקות שהוא אמור לבדוק ואם הוא צריך לתקן קוד אז זה הזמן בו הוא מתקן את הקוד וכו.
11. הכלים מכינים ושולחים את התוצאות שלהם שהם יצרו לREST בפקודת POST והREST שומר את זה אצלו למקרה שמישהו יבקש את התוצאה של אחד הכלים. כל פעם שבודקים משתני סביבה אחרים אז התוצאות שהREST שמר מתאפסות והוא אוסף אותם מחדש. קבצי הLOG גם כן נשלחים בפקודת POST אל הREST ואותם הפלטפורמה לוקחת.
12. לאחר שנפלטפורמה לקחה את קבצי הLOG הוא יוצרת קובץ TXT בתיקיית פרויקט שהלקוח סיפק לה ושמה שם את קובץ הLOG הגדול שהיא יצרה בעצם שילוב של כל קבצי הLOG ועוד דברים נוספים שהפלטפורמה מוסיפה.
13. שליחת הקבצים הסופיים לGUI ומשם ללקוח.

פונקציות עיקריות בפרויקט –

השמת זמן ממוצע לשורת קוד לכל כלי חדש –

פונקציה לבדיקת קומפילציה –

פונקציה למיפוי המידע על כל הקוד ובניית הגייסון –

פונקצייה לסדר הכלים כך שהכלים הראשונים יהיו הכלים שלא צריכים כלים אחרים והסדר הוא כך שלא יהיה מצב בו יש כלי שצריך כלי אחר והכלי האחר עוד לא היה –

פונקציית סידור הסקוופים -

**תיאור התוכנה -**

**שפות תכנות –**

**C#** היא שפת תכנות עילית מרובת-פרדיגמות[[6]](#footnote-6), מונחית עצמים בעיקרה, המשלבת רעיונות כמו טיפוסיות חזקה, אימפרטיביות, הצהרתיות, פונקציונליות, פרוצדורליות וגנריות. השפה פותחה על ידי "מיקרוסופט" בשנת 2000 כחלק מפרויקט דוט נט ותוקננה בשנים 2005 ו-2006 על ידי ארגון התקינה [[7]](#footnote-7)Ecma .

**SQL** **-** SQL קיצור של Structured Query Language היא שפת מחשב הצהרתית לטיפול ועיבוד מידע בבסיסי נתונים יחסיים, שפותחה על ידי, והתבססה במקור על אלגברה רלציונית השפה מאפשרת שליפת נתונים ועדכונם ויצירת טבלה ושינויה. או בקיצור הDatabase שלי.

**סביבת עבודה –**

**SQL Server –** שרת הSQL שלי הוא בנוי על Microsoft SQL Server שזו היא מערכת לניהול בסיס נתונים במודל היחסי של חברת מיקרוסופט. שפת הפיתוח שבאמצעותה מועברות הפקודות למערכת על ידי כתיבת פקודות או שימוש בממשק גרפי ששולח פקודות השפה הזו היא SQL.

**SQL Server Management Studio –** זוהי סביבת עבודה של מייקרוסופט גם כן, בה ניתן להשתמש בSQL Server שהסברתי וזה בעצם ממשק משתמש לכתיבת שאילתות יצירת טבלאות ועוד.. בעצם דרך קלה ליצור טבלאות ודטהבייסים בSQL.

**Visual Studio –** סביבת פיתוח מבית חברת מיקרוסופט, המאפשרת למתכנתים לפתח תוכניות מחשב ואתרי אינטרנט אינטראקטיביים. היא מיועדת לפיתוח יישומים עבור סביבות היא מיועדת לפיתוח יישומים עבור סביבות [[8]](#footnote-8)Win32 ואפליקציות של Windows 10.

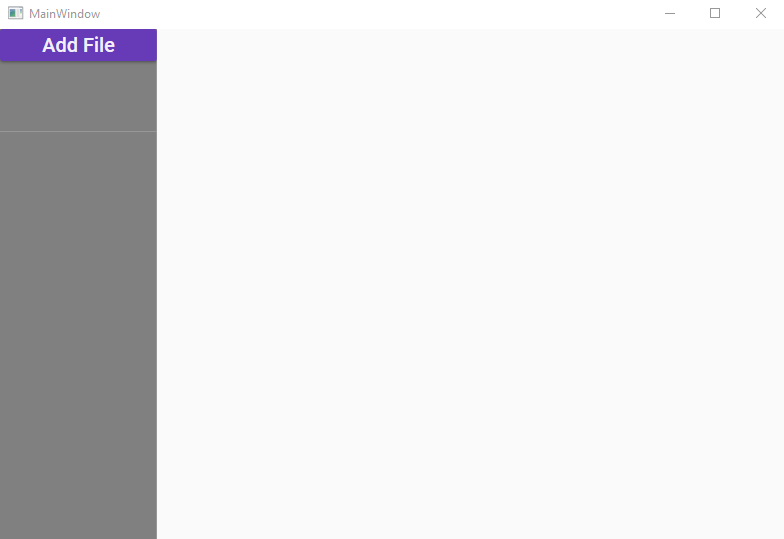
שפות תכנות שמגיעות כברירת מחדל בויזואל סטודיו –

* TypeScript
* C,C++
* C#

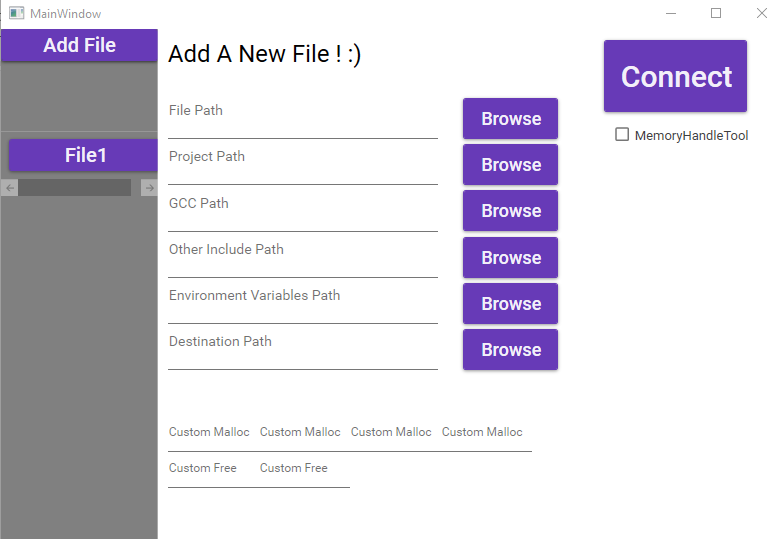
**.Net -**  היא שכבת תוכנה של מיקרוסופט שבאמצעותה מקודדות ומורצות תוכנות אחרות. שכבה זו מספקת לתוכנות שרצות מעליה סט כלים ושירותים כגון: ניהול זיכרון, שליטה בהתקני קלט/פלט, הצפנת הודעות וכולי, כך שהמתכנתים אינם צריכים לכתוב ספריות עזר המספקות שירותים אלה, אלא רק את הלוגיקה העיקרית של התוכנה.

**WPF - Windows Presentation Foundation** בקיצור WPF הוא מודל וכלי פיתוח תוכנה מבית מיקרוסופט למערכת ההפעלה חלונות, עם דגש על עיצוב גרפי מתקדם של ממשק המשתמש. WPF החל דרכו תחת שם הקוד Avalon, פורסם במגזין MSDN[[9]](#footnote-9)  בסוף 2004, הושק ב-2006 ויועד לשילוב במערכת ההפעלה שנקראה בשם הקוד Longhorn ונודעה לאחר מכן בשמה ויסטה. המודל הושק כחלק מפלטפורמת ‎.NET 3 החל מחבילת הפיתוח ויז'ואל סטודיו.

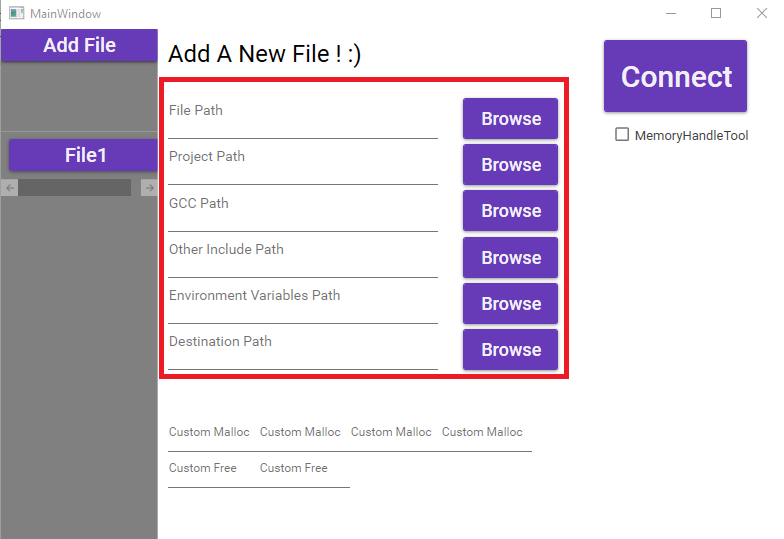
**תיאור מסכים –**



זהו הדף הראשי לממשק המשתמש של הפעלת התוכנה לבדיקת קודים בסי (Static Analysis Platform) שעליו מוסיפים קבצים לבדיקה.



זה הדף שמגיעים אליו לאחר שלוחצים על Add File פה צריך להקיש את הנתונים שמבקשים ולאחר מכן ללחוץ על Connect.



בריבוע שמסומן באדום צריך לרשום –

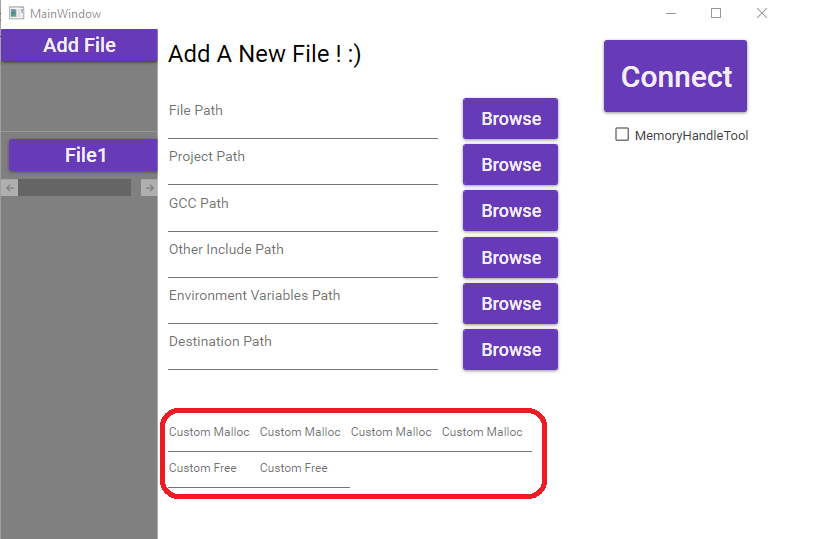
1. File Path - הכתובת של הקובץ שאותו רוצים לבדוק בשפת C כמובן.
2. Project Path – הכתובת של התיקייה של הפרוייקט.
3. GCC Path – כתובת הCompiler GCC.
4. Other Include Path – בעת הצורך אם יש ספרייה או תיקייה נוספת שקשורה לפרויקט ניתן להוסיף אותה בכתובת זו.
5. Environment Variables Path – משתני המערכת של הקוד. אם אין ניתן ליצור קובץ טקסט ריק והתוכנה לבד תבין שאין משתני מערכת. אם יש נתן לכתוב אותם ככה :

#define >evar1<

#define <evar2>

…

1. Destination Path – אם יש כלי שמחזיר קובץ אז הוא יחזיר את הקובץ לאותה תיקייה שהוספתם שם.



בריבוע שמסומן באדום ניתן לרשום –

1. Custom Mallocs – אם משתמשים באיזושהי תיקייה מיוחדת של מאלוקים אז ניתן לרשום פה את הCustom Malloc.
2. Custom Frees – אם משתמשים בתיקייה מיוחדת לFree אז ניתן להוסיף את השם של הפעולות שהם Custom Frees.

1. TCP – פרוטוקול בתקשורת נתונים הפועל בשכבות התעבורה של מודל הOSI ומבטיח העברה אמינה של נתונים בין שתי תחנות ברשת מחשבים. [↑](#footnote-ref-1)
2. GET REQUEST – מיועדת לקבל אובייקט שנמצא על השרת, בכתובת שניתנת בתחילה ההודעה. [↑](#footnote-ref-2)
3. LOG – הסבר קצר על מה שקרה (לדוגמא מה שונה בקוד על ידי הכלי). [↑](#footnote-ref-3)
4. POST – בקשות המכילות גוף הודעה. בקשות POST משמשות בדרך כלל לשליחה של מחרוזות לשרת. למשל מחרוזות ארוכות וכו.. [↑](#footnote-ref-4)
5. STL – Standard Template Library תיקייה לCPP שהשפיעה המון על הספריה הבסיסית של הCPP. [↑](#footnote-ref-5)
6. פרדיגמות - פרדיגמת תכנות היא סט מוסכמות לכתיבת תוכנה, עם שפת תכנות מסוימת. קיימות מספר פרדיגמות תכנות וכל שפת תכנות יכולה לשמש לכתיבת תוכנה בפרדיגמה אחת לפחות. [↑](#footnote-ref-6)
7. Ecma היא ארגון תקנים למערכות מידע ותקשורת. היא רכשה את שמה הנוכחי בשנת 1994, כאשר התאחדות יצרני המחשבים האירופי (ECMA) שינתה את שמה כדי לשקף את טווח ההגעה והפעילות העולמית של הארגון. כתוצאה מכך, השם כבר לא נחשב לראשי תיבות אלא כבר לשם אמיתי. [↑](#footnote-ref-7)
8. Win32 – Win32 זה השם של הWindows-API בגרסת 32 ביט.

   API זה אוסף של פקודות עבור תוכניתנים כדי שהם יוכלו להפעיל פקודות על ה- windows . לדוגמה, כשאתה מריץ משחק הוא נפתח על מסך מלא, זו פקודה של המערכת הפעלה, כשאתה מתקין תכנה והתכנה מעתיקה לך קבצים למחשב - ההעתקה היא גם פקודה של מערכת ההפעלה. [↑](#footnote-ref-8)
9. - **MSDN** או בשמה המלא  **Microsoft Developer Network**, היא רשת מבית מיקרוסופט שנועדה לאפשר למפתחי תוכנה, בודקי תוכנה ומפתחי חומרה לקבל מידע אודות תשתיות התוכנה ש-Microsoft מספקת תיעוד API תחביר שפה ועוד.. [↑](#footnote-ref-9)