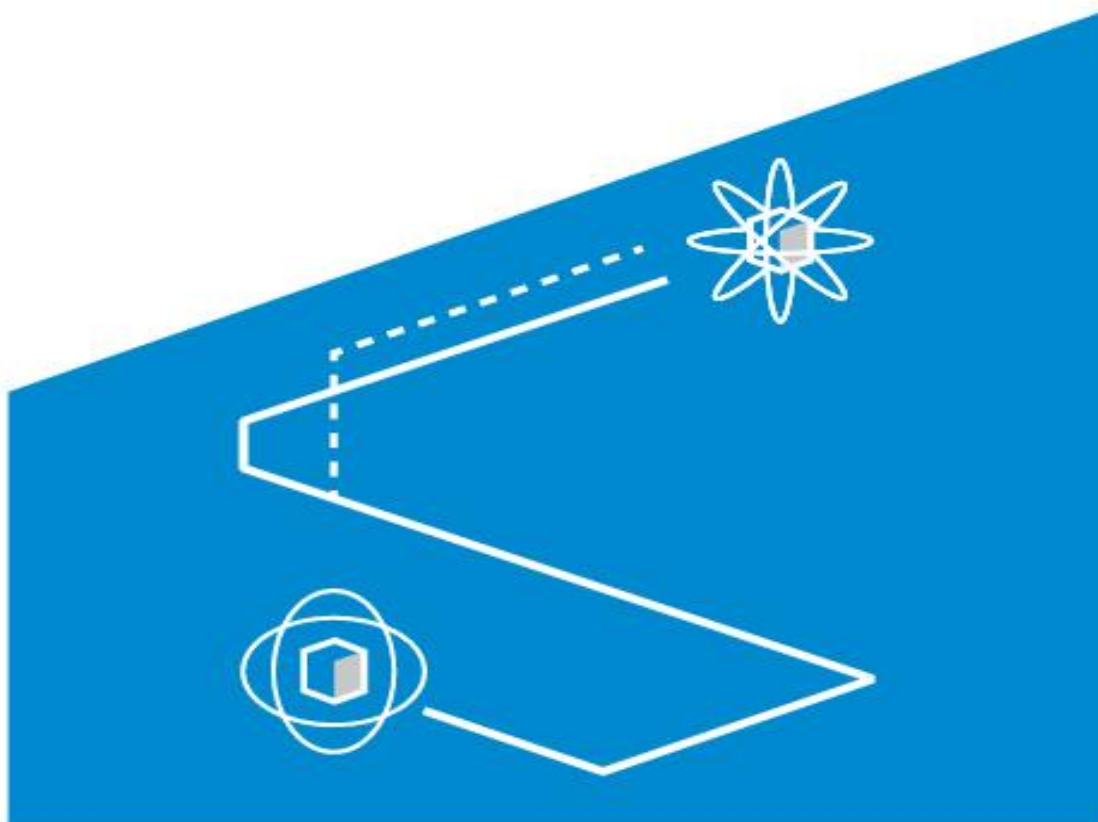


המכללה  
הטכנולוגית  
באר שבע

בשיתוף דגם WORLD  
קדימה מדע



# פרויקט גמר



**מדור חשבונות סטודנטים ופרויקטים**  
רחוב בזל 71, ת.ד. 45 באר-שבע 8410001  
טל: 08-6462502 | פקס: 08-6462501 | [gabrielak@tcb.ac.il](mailto:gabrielak@tcb.ac.il)

1954-ב | [www.tcb.ac.il](http://www.tcb.ac.il)

המכללה הטכנולוגית באר שבע קדימה מדע (ע"ר)

נושא : מערכת לניהול מוסך.

מוגש ע"י :

קובי ביטון ואיתי אזולאי

קורס :

תוכנה

מנחה :

מיכאל חרסונסקי



חתימת מנחה :

---

ראש מגמה :

איגור ברגמן



חתימת ראש מגמה :

---

באר שבע 2023.

תאריך: \_\_\_\_\_

לכבוד  
יחידת הפרויקטים  
מה"ט

**\*\* יש לקרוא את הנספח להצעת הפרויקט בקובץ הצעת פרויקט PDF**

**הצעה לפרויקט גמר**

**א. פרטי הסטודנטים**

שם הסטודנט	ת.ז. 9 ספרות	כתובת	טלפון נייד	תאריך סיום הלימודים
קובי ביטון	316505387	האנדרטה 12 באר שבע	0524892070	2023
איתי אזולאי	205612955	אלי סיני 2/50 נתיבות	0527557909	2023

שם המכללה המכללה הטכנולוגית באר שבע. סמל המכללה :

72204 מסלול ההכשרה : הנדסאים.

מגמת לימוד : תוכנה

מקום ביצוע הפרויקט : המכללה הטכנולוגית באר שבע

**ב. פרטי המנחה האישי**

שם המנחה *	כתובת	טלפון נייד	תואר	מקום עבודה/תפקיד
מר חרסונסקי מיכאל	אביתר הכהן 7, באר שבע	052-4738238	תואר שני	המכללה הטכנולוגית באר שבע

\* עבור מנחה אישי חדש יש לצרף קורות חיים, ניסיון מקצועי ותעודות השכלה לאישור מה"ט.

חתימת הסטודנט

חתימת הגורם המקצועי מטעם מה"ט

חתימת המנחה האישי

1. שם הפרויקט : ממשק ניהול מוסך "Pimp My Ride"

2. רקע:

1.2 תיאור ורקע כללי:

המערכת משמשת לניהול המוסך: פתיחת כרטיסי עבודה, ניהול עובדים וניהול מלאי, לו"ז לקוחות ואפשרות לקביעת תור.

2.2 מטרות המערכת:

- אפשרות לקביעת תור מצד לקוח
- קבלת הצעת מחיר לגבי טיפול/ תיקון תקלה
- פתיחת כרטיס עבודה
- ניהול מלאי
- ניהול עובדים (שעות, שכר וכו)
- הזמנת חלפים/מוצרים מספקים

3. סקירת מצב קיים בשוק, אילו בעיות קיימות:

כיום המצב במוסכים הוא כזה שלא מאפשר ללקוח לקבוע תור מבלי ליצור קשר טלפונית או להגיע פיזית למוסך, דבר שמכביד ומאריך את תהליך קבלת השירות מהמוסך. הצעת מחיר דיגיטלית במקום טפסי נייר.

4. מה הפרויקט אמור לחדש או לשפר:

הפרויקט אמור להקל על השתמש לקבל שירות דיגיטלי מהמוסך, החל מקביעת תור דיגיטלי דרך משך זמן טיפול ועד להצעת מחיר דיגיטלית, וכך הלקוח יכול לקבל החלטות מבלי לבזבז זמן מיותר במענה טלפוני או בהגעה למקום.

5. דרישות מערכת ופונקציונאליות:

1.5 - דרישות מערכת:

המערכת תפעל ע"י תוכנת מחשב שמותאמת למערכת הפעלה WINDOWS שתמומש ע"י visual forms C#, בנוסף יהיה אפליקציית צד לקוח שתמומש ע"י REACT NATIVE. צד השרת ימומש ע"י C#, הנתונים יישמרו בבסיס נתונים מקומי (SQL SERVER).

2.5 - דרישות פונקציונליות:  
רשימת דרישות המשתמש מהמערכת, מהן הפעולות בהן נדרשת המערכת לתמוך.  
1.2.5 - דרישות משתמש (מכונאי) :  
1.1.2.5 - דיווח שעות עבודה

2.2.5 - דרישות משתמש (יועץ שירות) :  
1.2.2.5 - התחברות למערכת  
2.2.2.5 - עדכון פרטיים אישיים וסיסמא  
3.2.2.5 - דיווח שעות עבודה  
4.2.2.5 - פתיחה וסגירה של הצעות מחיר וכרטיסי עבודה.

3.2.5 – דרישות משתמש (מנהל) :  
1.3.2.5 – כל דרישות משתמש יועץ שירות ובנוסף:  
2.3.2.5 – עדכון פרטיים אישיים לעובדים.  
3.3.2.5 – הזמנת מלאים מספקים וניהול מלאי.

4.2.5 – דרישות מנהל מערכת  
1.4.2.5 – הוספה/הסרה/עדכון של משתמשים  
2.4.2.5 עריכת בסיס נתונים (יצרנים \ דגמים \ מלאי \ ספקים \ משתמשים)

6. בעיות צפויות במהלך הפיתוח ופתרונות)תפעוליות, טכנולוגיות, עומס ועוד):

1.6 – תיאור הבעיות הללו כפועל יוצא של דרישות המשתמש מהתוכנה:  
בעיה 1: כפילויות של הצעות מחיר לאותו הרכב.

1.6 – פתרון אפשרי לבעיה מס 1:

שמירת הצעות מחיר לפי מספר ייחודי של ההצעה ולא לפי מספר רכב.

7. פתרון טכנולוגי נבחר:

1.7 טופולוגית הפתרון- כלומר: פרישת המערכת, היכן יתבצע יישום המערכת (deployment), מרכיבי הפרישה. הנ"ל ברמת מערכת (לדוג' פרויקט פיתוח אתר אינטרנט: המערכת מורכבת משרת, ממשק משתמש בצד הלקוח, DB's, טווח תקשורת-אינטרנט, המערכת תיושם ברשת האינטרנט, יש להציג את דיאגרמת המערכת וכו')

2.7 טכנולוגיות בשימוש. איזה ומדוע בכמה מילים) #C - פיתוח באמצעות #C מאפשר תכנות ויזואלי לתוכנה שהולכת לרוץ על מערכת ההפעלה של ווינדוס.

– Sql server בסיס נתונים שיתחבר ל#C.

3.7 שפות הפיתוח: איזה שפות ומדוע בכמה מילים?)

#C 7.3.1

4.7 תיאור הארכיטקטורה הנבחרת- הסבר בכמה מילים מדוע

הארכיטקטורה הנבחרת היא חלוקה ל 3 שכבות – 3 Tier

Architecture

הפרויקט מבוסס על database ולכן מומלץ להשתמש בגישת מודל השכבות.

לגישה יש יתרון של תחזוקה קלה. בשל החלוקה לשכבות ניתן לבצע שינויים בשכבה כלשהי בקלות ומבלי להשפיע על השכבות האחרות.

5.7 חלוקה לתכניות ומודולי ם

שכבת התצוגה: שכבת ממשק המשתמש

שכבת הלוגיקה העסקית: שכבת עיבוד המידע ושליחתו לשכבת התצוגה שכבת הנתונים:

שכבה של מסד נתונים האחראית על שמירה עדכון והוספה של מידע.

6.7 סביבת השרת (מקומי, וירטואלי, ענן, שירות

אירוח) שרת מקומי ותחנות עבודה המחוברות באותה הרשת

7.7 ממשק המשתמש/לקוח – GUI

מוצג על גבי תוכנה של ווינדוס.

8.7 ממשקים למערכות אחרות/ API: הדפסת הצעות מחיר + כרטיסי עבודה +

תלושי שכר + דוחות + חשבונית.

9.7 שימוש בחבילות תוכנה:

#C

0.8 שימוש במבני נתונים וארגון קבצים

1.8 פירוט מבני הנתונים

**Employee**: Id, FullName, ContactInformation, JobDefinition

**EmployeeEntry**: EmployeeId, EntryTime, ExitTime

**WorkTicket**: CarId, Complaint, WorkTicketOfferId, EmployeeId

**WorkTicketOfferId**: WorkTicketOfferId, CarId, PartsLayout, WorkLayout

**PartsCollection**: PartId, PartName, CostToConsumer, CostToGarage, Quantity,  
SafeQuantity

**ClientsCollection**: CarId, CarOwnerName, CarOwnerContacts, CarHistoryTable

**CarHistoryTable**: CarId, WorkTicketHistory, Warranty

**SuppliersCollection**: SupplierId, SupplierName, SupplierContact

**SuppliersOrders**: OrderId, OrderInformation, OrderDate, SupplierId

**ScheduleCollection**: CarId, Date, WorkTicketOfferId

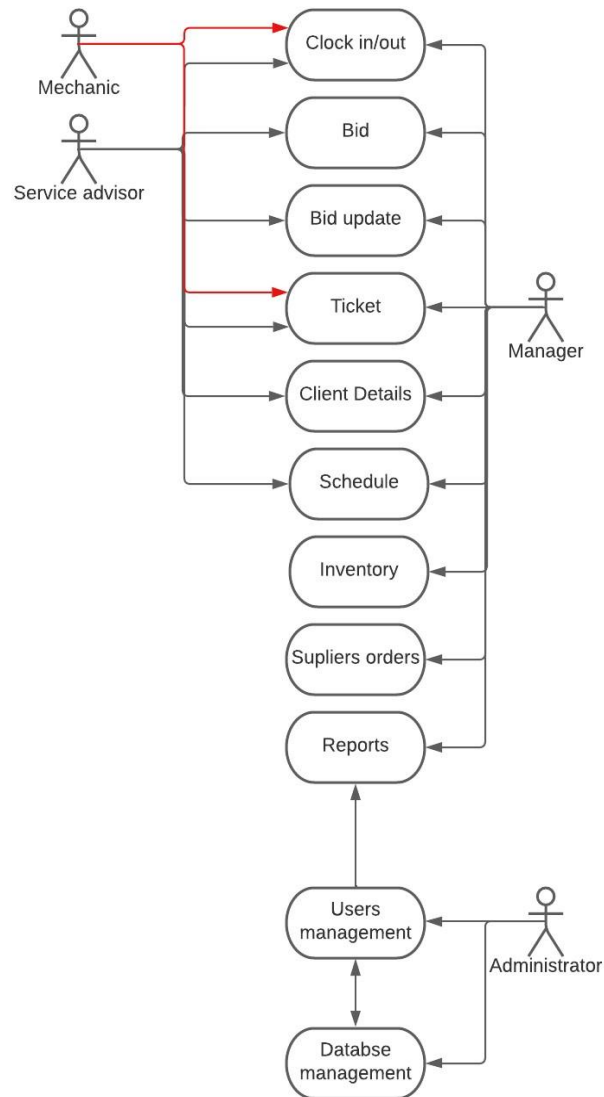
2.8 נא פרט את שיטת האיחסון (מאגר, קבצים ובאיזה טכנולוגיה)

בסיס נתונים SQL Server

3.8 נא ציין מגננוני התאוששות מנפילה/קריסה/תמיכה בטראנזקציות.

המערכת מיועדת לשימוש בארגונים קטנים ומספר הפניות במקביל למסד הנתונים

והשרת אינו צפוי להיות גדול כלל.



המרכיב

10. תיאור האלגוריתמי - חישובי

10.1 איזה בעיה בא לפתור, איך יפתור?

אלגוריתם ניהול מלאי יסייע במניעת כפל הזמנות בנוסף יתריע על מלאי שעומד להיגמר.

אלגוריתם ניהול כרטיסי עבודה שתפקידו למנוע כפילויות של כרטיסי עבודה.

10.2 איסוף מידע וניתוחים סטטיסטיים (אנליטיות)

חישוב שעות עבודה לעובד.



חלקים שנצרכו ועלויות ממוצעות לתקופת זמן.	
ניתוח הזמנות מספקים כולל דוחות של עלויות פר תקופת זמן.	
דוחות הכנסות והוצאות של המוסך.	
תיאור / התייחסות לנושאי אבטחת מידע	11
פרטי משתמשים רגישים וסיסמאות יוצפנו באמצעות ספריית security.Cryptography.	
המערכת מותקנת על שרת מקומי וללא גישה לרשת.	
משאבים הנדרשים לפרויקט:	12
12.1 מספר שעות המוקדש לפרויקט, חלוקת עבודה בין חברי הצוות:	
סה"כ 350 שעות עבודה.	
חלוקי שווה בין חברי הצוות.	
12.2 ציוד נדרש:	
מחשבים לכתיבת קוד.	
12.3 תוכנות נדרשות:	
Visual studio code	
SQLServer	
12.4 ידע חדש שנדרש ללמוד לצורך ביצוע הפרוייקט:	
התממשקות להדפסה מתוך התוכנה.	
12.5 ספרות ומקורות מידע:	
אינטרנט.	
מרצה.	
13 תוכנית עבודה לשלבים למימוש הפרוייקט:	
יזום הרעיון.	
ניתוח מערכת.	

ניתוח מבנה נתונים.

כתיבת קוד מערכת.

עיצוב.

בדיקות תוכנה.

14 תכנון הבדיקות שיבוצעו:

מספר	מספר הדרישה במס מך האפיון	מקרה הבדיקה	ידנית / אוטומטית	חשיבות
1	5.2.1.1	דיווח שעות עבודה	ידנית	
2	5.2.2	עדכון פרטים אישיים	ידנית	
3	5.2.2.4	פתיחה וסגירה של הצעות מחיר וכ רטיסי עבודה (וידוא שדות, וידוא פונ קציונליות תקינה, בדיקת כפל, בדיקת חישוב עלויות תיקון	ידנית	
4	5.2.3.3	הזמנת מלאים / התראה על מלאי שעומד להיגמר	ידנית	
5	5.2.4	מנהל מערכת- ניהול בסיס נתונים וניהול משתמשים	ידנית	

15 ( version control ) בקרת גרסאות



חתימת המנחה האישי



חתימת הסטודנט

ג. הערות ראש המגמה במכללה

-----

מאושר \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

ד. אישור ראש המגמה



שם: \_\_\_\_\_ ברגמן איגור חתימה : \_\_\_\_\_ תאריך: 23/10/23

-----

ה. הערות הגורם המקצועי מטעם מה"ט

-----

-----

ו. אישור הגורם המקצועי מטעם מה"ט



2023/01/30

תאריך:

חתימה:

שם:

סימוכין : כלל 2

### הצהרת סטודנט

סטודנט 1 – שם : איתי אזולאי תעודת זהות : 205612955

סטודנט 2 – שם : קובי ביטון תעודת זהות : 316505387

שם המכלל בה לומדים הסטודנטים: המכללה הטכנולוגית באר – שבע קדימה מדע (ע"ר)

אנו החתומים מטה מצהירים בזאת כי פחיקט הגמר וספר הפחיקט המצ"ב נעשו על ידינו בלבד.  
פחיקט הגמר נעשה על סמך הנושאים שלמדנו במכללה ובאופן עצמאי.  
פחיקט הגמר וספר הפחיקט נעשו על בסיס הנחיית של המנחה האישי.  
מקוחת המידע בהם השתמשנו לביצוע פחיקט הגמר מצוינים ברשימת המקוחת בספר הפחיקט.  
אנו מודעים לאחריות שהננו מקבלים על עצמנו על ידי חתימתנו על הצהרה זו שכל הנאמר בה את  
ורק אמת.

סטודנט 1 – חתימה:  תאריך: 23/10/2023

סטודנט 2 – חתימה:  תאריך: 23/10/2023

### אישור המנחה האישי

הריני מאשר שהפחיקט בוצע בהנחייתי, קראתי את ספר הפחיקט ומצאתי כי הוא מוכן לצורך  
הגשת הסטודנטים להגנה על פחיקט הגמר.

שם המנחה: מיכאל חרסונסקי חתימה:  תאריך: 11/07/2023

**אישור ראש המגמה**

הריני מאשר שספר הפרויקט מוכן לצורך הגשת הסטודנטים להגנה על פרויקט הגמר.

שם ראש המגמה: איגור ברגמן חתימה:  תאריך: 11/07/2023

## תקציר הפרויקט Pimp my ride

הפרויקט הוא מערכת ניהול מוסך רכבים הפועלת בסביבת Windows ומשתמשת בשפת תכנות #C עם ממשק משתמש באמצעות WinForms. המערכת נוצרה כדי לספק פתרון נוח ויעיל לניהול ותחזוקת רכבים במוסך.

התוכנה מטרתה לספק יעול בניהול ובתחזוקת הרכבים, כולל ניהול לקוחות, רכבים, פעולות תחזוקה, ותיקונים.

הפרויקט כולל בתוכו:

### 1. ניהול לקוחות:

- הוספה, מחיקה ועדכון של פרטי לקוחות.
- צפייה ברשימת לקוחות ופרטיהם.

### 2. ניהול רכבים:

- הוספה ועדכון של פרטי רכבים, כולל דגם, שנת ייצור, ומספר רישוי.
- תצוגה וניהול של רשימת רכבים במוסך.

### 3. תחזוקה ותיקונים:

- רישום פעולות תחזוקה ותיקונים לכל רכב.
- היסטורית טיפולים עבור רכבים שטופלו במוסך.

### 4. ממשק משתמש ידידותי:

- עיצוב והפעלה נוחים של ממשק משתמש באמצעות WinForms.
- קלות בשימוש, כולל תפריטים וכפתורים לפעולות נפוצות.

### 5. אחסון נתונים:

- שימוש בממסד נתונים SQL Server שרץ על שרת מקומי.

### טכנולוגיות שוטפות:

הפרויקט משתמש בשפת תכנות #C עם סביבת פיתוח Visual Studio ומשתמש ב WinForms כטכנולוגיה לבניית ממשק המשתמש.

### השגחה על קוד:

הקוד נכתב בסגנון נקי וארגוני, כולל תיעוד רלוונטי להבטחת קריאות ותחזוקה קלה בעתיד.

### לסיכום:

המערכת מספקת פתרון מתקדם ויעיל לניהול מוסך רכבים, מסייעת בשמירה על רשומות נכונות ומספקת ממשק משתמש ידידותי המאפשר ניהול יעיל ונוח של פרטים ופעולות שונות.

## תוכן עניינים:

1. מבוא.
2. עקרונות התכנון.
  - 2.1 בניית Controllers ו – handlers לכל ישות (טבלה) בבסיס הנתונים.
  - 2.2 שימושים בספריות.
3. תיאור העבודה
  - 3.1 ממשק המשתמש.
  - 3.2 צד השרת.
  - 3.3 בסיס הנתונים.
4. מסכי המשתמש
  - Login 4.1
  - Dashboard 4.2
  - Tickets 4.3
    - 4.3.1 יצירת כרטיס עבודה חדש.
    - 4.3.2 כל הכרטיסים הפתוחים.
      - 4.3.2.1 כרטיס עבודה.
        - 4.3.2.1.1 הוספת חלקים לכרטיס עבודה.
        - 4.3.2.1.2 הוספת עבודות לכרטיס עבודה.
        - 4.3.2.1.3 סגירת כרטיס עבודה(תשלום).
          - 4.3.2.1.3.1 תשלום במזומן.
          - 4.3.2.1.3.2 תשלום בכרטיס אשראי.
      - 4.3.3 יצירת הצעת מחיר.
      - 4.3.4 הצעות מחיר פתוחות.
    - Clients 4.4
      - 4.4.1 חיפוש לקוח.
      - 4.4.2 טבלת כל הלקוחות.
      - 4.4.3 יצירת לקוח חדש.
    - Storage 4.5
      - 4.5.1 טבלת כל הספקים.
      - 4.5.2 יצירת ספק חדש.
      - 4.5.3 יצירת הזמנה מספק.
      - 4.5.4 כל החלקים במחסן.
      - 4.5.5 כל ההזמנות שבוצעו.
    - Admin 4.6
      - 4.6.1 יצירת משתמשים חדשים.
      - 4.6.2 עמוד תחזוקה הכולל בתוכו הכנסת מידע למערכת.
      - 4.6.3 הכנסת דגמי רכבים חדשים למערכת.
  5. קוד המערכת
    - 5.1 ממשק המשתמש.
    - 5.2 שרת העיבוד.
    - 5.3 בסיס הנתונים.



## **1. מבוא :**

בעידן המודרני, המערכות הממוחשבות תובנה נוספת ומעמיקה בכמעט כל תחום של חיינו. עם עליית הטכנולוגיה, הצורך בכלים דיגיטליים לניהול יעיל וממוקד של מגוון תחומים עולה. אחד מתחומים אלו הוא תחום ניהול הרכבים והמוסכים, שבו מתעסקים בשולי הטכנולוגיה וההנדסה יחד.

בהקפדה על האספקטים השונים של ניהול ותחזוקת הרכב, החלטנו לפתח מערכת ניהול מוסך רכבים באמצעות תוכנה. המערכת שפותחה בשפת תכנות #C באמצעות סביבת הפיתוח Visual Studio ובעזרת טכנולוגיית WinForms, מיועדת להקל ולשדרג את תהליכי הניהול והתחזוקה של רכבים במוסך.

## **מטרה :**

המטרה העיקרית של המערכת היא ליצור כלי יעיל וידידותי למשתמש המסייע בניהול ובמעקב אחר פעולות התחזוקה והתיקונים של רכבים במוסך. באמצעות ממשק משתמש אינטואיטיבי ותכונות יעילות, המערכת מאפשרת למנהלי המוסך ולצוות התחזוקה לשמור על מעקב מדויק ומעודכן של רכביהם ולשרת את לקוחותיהם בצורה מקצועית ויעילה.

## **תכנון ומבנה :**

התוכנה פותחה בהתבסס על ארכיטקטורת קוד יציבה וקלה לתחזוקה, ותומכת בניהול רשומות מקומי באמצעות בסיס נתונים. בניית הממשק נעשתה באמצעות WinForms, מה שמבטיח יעילות ונוחות בשימוש.

בעזרת הפרויקט, אנחנו מקווים לתרום למתקדמים בתחום ניהול הרכב והמוסך ולהפוך את תחום זה ליותר יעיל וזמין, עם מערכות מתקדמות וידידותיות למשתמש.

## 2. עקרונות התכנון:

בחרנו להשתמש בארכיטקטורת REST API אשר משתמשת במודל שפשוט, יעיל ונפוץ עבור שירותים רשתיים. הנה סיכום קצר על טעמי לבחירה זו:

### יעילות ניהולית:

- REST (Representational State Transfer) משתמשת במודל יעיל וקל שהקלט שלו קריא וברור. זה מפשט את התקשורת בין השרת והלקוח, וגורם לפרויקט להיות נגיש וקל לתחזוקה.

### סטנדרטיזציה:

- REST הוא סטנדרט נפוץ כאשר הוא חלק מהאינטרנט ומערכת ה-WWW. זה מאפשר איכות קוד גבוהה, נגישות ושיבוץ נתונים בצורה יחידה ואחידה בפרויקט.

### תקשורת גמישה:

- REST מספק גמישות בתקשורת, וזהו יתרון חשוב בפרויקטים שבהם ישנם מספר פלטפורמות ולקוחות שונים.

### אוטונומיה בכתיבה:

- באמצעות REST, כל רשות בקצה הלקוח והשרת יכולה לפעול באופן עצמאי ולא תלויה יותר מדי בשינויים בצד השני. זה מקנה ניתוח והתקנות באופן נפרד.

### תמיכה בפורמטים פתוחים:

- REST מתמקדת בשימור קריאות ונגישות של המידע, בדרך כלל תחת פורמטים פתוחים כמו JSON או XML. זה מאפשר ללקוחות ולשרתים שונים להבין ולתקשר בצורה יעילה.

בחירת REST API מציינת את המחויבות לבנות מערכת יעילה, קלה לתחזוקה, ומותאמת לכמה שווקים ופלטפורמות ככל שזה אפשרי. היא מספקת פתרון טכנולוגי פשוט ויעיל שמתמקד בהעברת מידע בצורה ברורה וקריאה, עומדת בסטנדרטים נפוצים, ומאפשרת תקשורת נגישה ויעילה בין רכיבי המערכת.

## 2.1 Controllers and Handlers

### Controllers (בקרים):

- המחלקות האחראיות על לקבל ולתת קלט ופלט למערכת.
- כל בקר מתמקד בתפקיד מסוים, כגון ניהול לקוחות, רכבים, או פעולות תחזוקה.
- מטרתם העיקרית היא להפנות בקשות הגעה מהלקוחות לפונקציות הנכונות לטפל בהן.

### HANDLERS (טפלים):

- מחלקות אשר מיועדות לטפל בבקשות קצה, כל טפל יורש ממחלקה אבסטרקטית אשר מיועדת לטפל במקרה ספציפי.
- נוצרו כך שכל הטפלים ייכתבו לפי נהלים מוגדרים מראש, מה שמבטיח יחידות קוד, קריאות, ותחזוקה יעילה.

המבנה שלנו נבנה בצורה אשר בה הוא נוקט בגישה פונקציונלית, קצרה וברורה, המקנה את היכולת לפתח בקלות פונקציות חדשות ולתחזק את הקוד בצורה יעילה. המודל שנבחר מסייע לנו לקיים כתיבת קוד בצורה אחידה ומאורגנת היטב בפרויקט, וכמובן מוודא שכל הקוד עומד בנהלים ובסטנדרטים שהוגדרו מראש.

## 2.2 שימוש בספריות חיצוניות :

בפרויקט שלנו, נעשה שימוש יעיל ומועיל בספריות חיצוניות לצורך ייבוא והשתמשות בפונקציות ויכולות נוספות מקוד מוכן. הנה סיכום שמסביר את השימוש בספריות חיצוניות בפרויקט:

### הגדרת מערכת :

- הפרויקט משתמש בהגדרות חיצוניות לפרק או להגדיר משאבים מסוימים. דוגמת כך היא השימוש בספריית *Newtonsoft.Json* לטיפול בנתוני JSON.

### פעולות RESTful :

- ספריות חיצוניות משמשות לקבל ולשלוח בקשות HTTP, כמו אותו יכולת הכתיבה והקריאה מובנית ב-C# לא מתקנת את הפרויקט.

- לדוגמה, שימוש בספריית *HttpClient* לביצוע קריאות ל-API חיצוני.

### ניהול נתונים :

- עבודה עם מסדי נתונים או פורמטים מסוימים יכולה להינתן יחסית בקלות באמצעות ספריות חיצוניות. לדוגמה, שימוש ב-Entity Framework לשימוש נוח ויעיל עם מסדי נתונים.

### תיעוד וקוד יעיל :

- השימוש בספריות מקל על תיעוד הקוד, כי זה יכול להקל על ההבנה של מקורות חיצוניים וכך להפחית את סיכוני השגיאה.

לסיכום, שימוש יעיל בספריות חיצוניות בפרויקט מאפשר פתרונות מוכנים לשימוש, ביצועים יעילים ותמיכה בתחומים ספציפיים כמו ניהול נתונים ותקשורת עם מערכות חיצוניות.

### 3. תיאור העבודה:

בפרויקט זה, בחרנו להשתמש בארכיטקטורת "לקוח-שרת", באמצעות תקשורת מבוססת API.

#### צד הלקוח:

צד הלקוח, הידוע גם כ Front-End הוא ממשק המשתמש או האפליקציה שבה המשתמשים מתקשרים, בחרנו להשתמש בארכיטקטורה זו על מנת שבמקרה בעתיד נוכל להרחיב את השימוש בפרויקט וליצור עוד ממשקי משתמש, כגון אפליקציית ווב, אפליקציית מובייל וכו'.

#### צד השרת:

השרת אחראי לעבוד הבקשות מהלקוח, לביצוע הלוגיקה הנדרשת ולניהול התקשורת עם מסד הנתונים, כשהשרת מקבל בקשת HTTP מהלקוח הוא מעבד את הבקשה, נכנס להתקשרות עם מסד הנתונים (במידת הצורך) ומחזיר מענה HTTP ללקוח.

#### מסד הנתונים:

מסד הנתונים אחראי לאחסון ולניהול המידע של התוכנה, בחרנו להשתמש בSql Server, אך בזכות הבחירה בארכיטקטורה זו, ניתן להרחיב ולהשתמש במספר מסדי נתונים כפי שנצטרך.

לסיכום, ארכיטקטורת לקוח-שרת מספקת הפרדה של תפקידים ומאפשרת לכל אחד מהצדדים להתפתח באופן עצמאי ללא תלות אחד בשני, בנוסף מסייעת לסקל את השרת ומסד הנתונים בכדי להקל בעומסים האפשריים בזמן שימוש בתוכנה זו.

### **3.1 ממשק המשתמש:**

WinForms היא טכנולוגיה מבוססת תוכנה שמספקת חוויות משתמש מקומיות. טכנולוגיה זו מאפשרת אמינות גבוהה וביצועים מהירים בסביבת Windows.

WinForms מספקת יכולות עיצוב מתקדמות ותפעול בסביבת Windows.

### **3.2 צד השרת:**

צד השרת מהווה את שכבת התקשורת והעיבוד בין ממשק המשתמש לבסיס הנתונים.

השרת מקבל את בקשות ממשק המשתמש, מבצע פעולות עיבוד שונות כגון בדיקת תקינות הנתונים, מתקשר מול בסיס הנתונים ומחזיר את התגובה המתאימה בחזרה אל ממשק המשתמש. השרת מכיל בעיקר מודלים של טבלאות שונות, בקרים וטפלים לכל אחת מהבקשות ומהמודלים.

השרת נכתב בשפת #C ומיישם את תשתית asp.net.

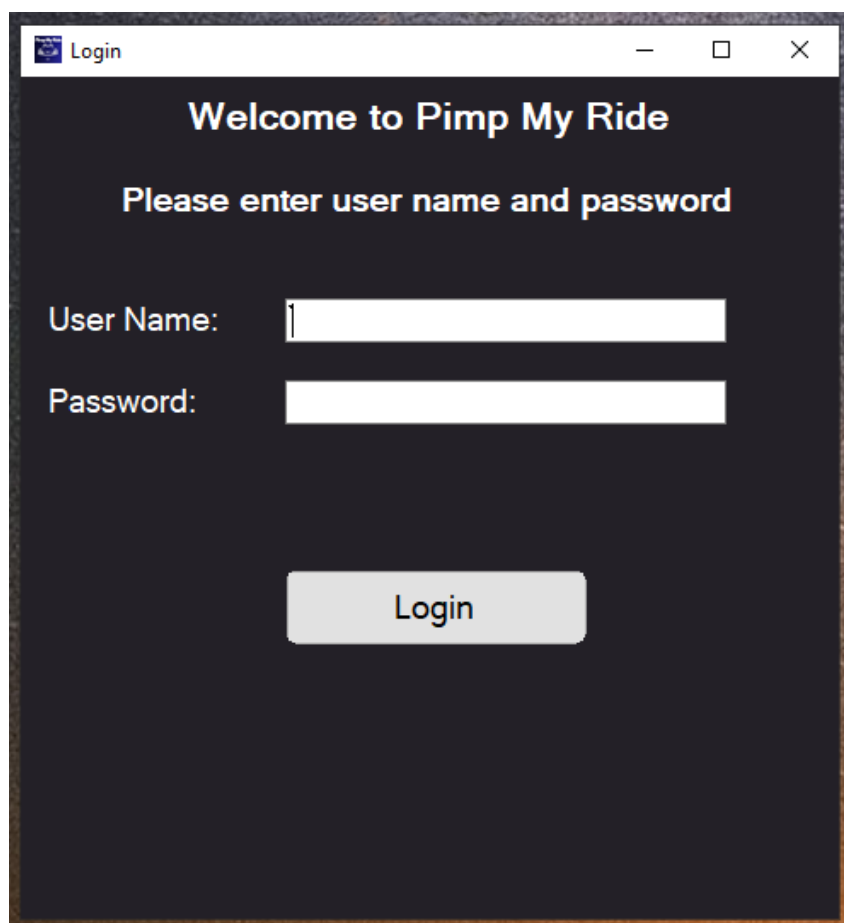
### **3.3 בסיס הנתונים:**

בבסיס הנתונים מאוחסן כל המידע הקשור לניהול מוסך, החל מניהול לקוחות ומכוניות, דרך מלאי מחסן ועד להיסטורית טיפולים. בסיס הנתונים מאפשר שליפת נתונים דרך פקודות שמקבל מצד השרת.

בסיס הנתונים בו השתמשנו הוא SQLSERVER.

#### 4. מסכי המשתמש:

#### 4.1 Login – מסך התחברות למערכת:



The screenshot shows a Windows-style application window titled "Login". The window has a dark blue background with white text. At the top, it says "Welcome to Pimp My Ride". Below that, it says "Please enter user name and password". There are two input fields: "User Name:" and "Password:". Below the input fields is a "Login" button. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Login

Welcome to Pimp My Ride

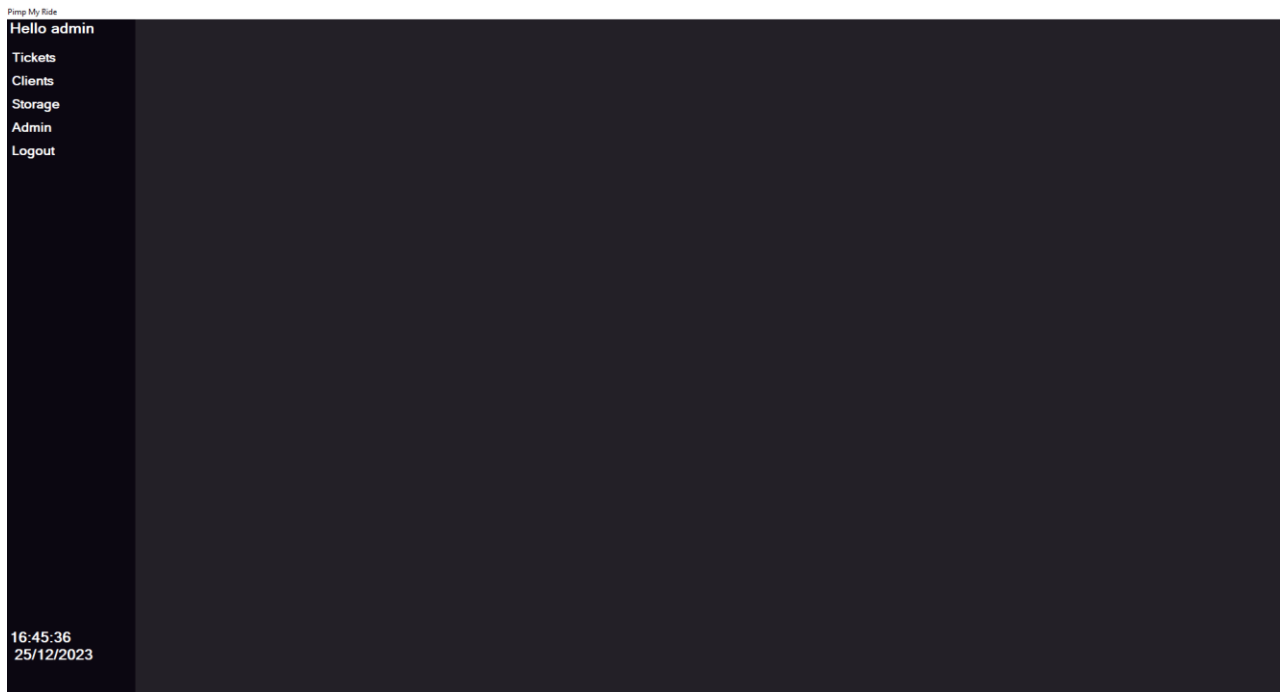
Please enter user name and password

User Name:

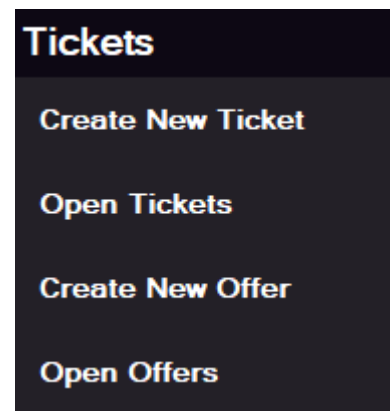
Password:

Login

## : Dashboard 4.2



## : Tickets 4.3





:Create new ticket 4.3.1

Car Number: \*  

Search car

Car details

Full Name:  
Phone number:  
Email:

Manufacture:  
Model:  
Year:  
Engine:  
Vin Number:  
KM:

Cause of arrival

Create ticket

:Open tickets 4.3.2

Car Number :  

Search

	Ticket Id	Car Id	Client Id	Problems	Date	Price
>	2051	6893164	205612955	oil leak	12/5/2023 7:44 PM	600

Select

### :Selected ticket 4.3.2.1

Full Name:	itay azoulay	Manufacture:	Mazda
Phone number:	0527557909	Model:	3
Email:	itay.az21@gmail.com	Year:	2008
		Engine:	1600
		Vin Number:	asdasd
		KM:	30000

Cause of arrival  
oil leak

Part Id	Part Name	Price	Quantity	Discount	Unit Price
ab12	mazda 3 a...	600.00	2.00	0.00	300.00

Labor Id	Description	Price	Time	Discount
----------	-------------	-------	------	----------

Total part price : 600  
Discount : 0  
Total price : 600

Total labor price : 0  
Discount : 0  
Total price : 0

Add PartUpdate PartDelete PartAdd LaborUpdate LaborDelete Labor

Total price : 600

Export to pdfDelete TicketClose Ticket

### :Add parts to ticket 4.3.2.1.1

Part Name :

Part Id	Part Name	Price	Quantity
ab12	mazda 3 axle	300.00	4.00
Ae12	Mazda Brake Rotors	200.00	44.00
Ae123	Motul oil 5w30	40.00	91.00
ter8457	spark plugs for mitsubishi	80.00	36.00

Amount :

Add Part

**:Add labors to ticket 4.3.2.1.2**

Labor Name :

	Id	Description	Price	Time	Discount
▶	2	15K Maintenance	300.00	1.00	0.00
	3	30K Maintenance	300.00	1.50	0.00
	1004	dissassembly of gear	400.00	3.00	0.00

**:Close ticket 4.3.2.1.3**

Cash

Amount :

### :Pay with cash 4.3.2.1.3.1

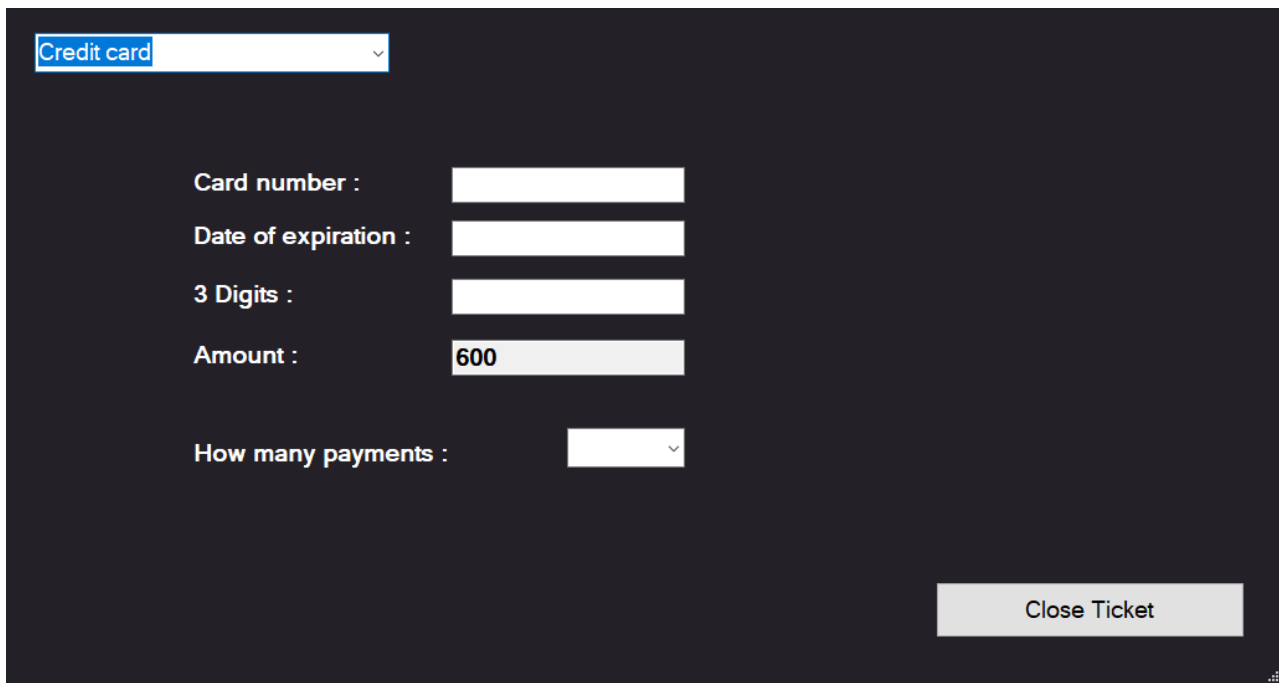


Cash

Amount : 600

Close Ticket

### :Pay with credit card 4.3.2.1.3.2



Credit card

Card number :

Date of expiration :

3 Digits :

Amount : 600

How many payments :

Close Ticket

### :Create new offer 4.3.3

Car Number: \*

30473301

Search car

Car details

Full Name:

alona

Phone number:

0507164753

Email:

alonazarankin@gmail

Manufacture:

Mazda

Model:

3

Year:

2018

Engine:

2000

Vin Number:

asd

Offer details:

Create Offer

### :Open offers 4.3.4

Car Number :

Search

Select

### : Clients 4.4

Clients

Search

All Clients

Create New Client

### :Search client 4.4.1

Client ID: *	<input type="text" value="205612955"/>	Car Number: *	<input type="text" value="6893164"/>
Full Name:	<input type="text" value="itay azoulay"/>	Manufacture:	<input type="text" value="Mazda"/>
Phone:	<input type="text" value="0527557909"/>	Model:	<input type="text" value="3"/>
Email:	<input type="text" value="itay.az21@gmail.com"/>	Year:	<input type="text" value="2008"/>
Address:	<input type="text" value="netivot"/>	Engine:	<input type="text" value="1600"/>
		Vin Number:	<input type="text" value="asdasd"/>
<div><input type="button" value="Search client"/> <input type="button" value="Update"/> <input type="button" value="Add Car"/> <input type="button" value="Car history"/></div>			

### :All clients 4.4.2

	Id	Client Name	Client Phone	Email	Client address
▶	205612955	itay azoulay	0527557909	itay.az21@gmail.com	netivot
	316505387	kobi biton	0524892070	kobiton6991@gmail.com	beer sheva
	323655803	alona	0507164753	alonazarankin@gmail.com	

### : Create new client 4.4.3

Create new client			
Client ID: *	<input type="text"/>	Car Number: *	<input type="text"/>
Full Name: *	<input type="text"/>	Manufacture: *	<input type="text"/>
Phone: *	<input type="text"/>	Model: *	<input type="text"/>
Email:	<input type="text"/>	Year: *	<input type="text"/>
Address:	<input type="text"/>	Engine: *	<input type="text"/>
		KM: *	<input type="text"/>
		Vin Number: *	<input type="text"/>
<input type="button" value="Create client"/>			

**:Storage 4.5**

Storage

All Suppliers

Create New Supplier

New Order

All parts

All Orders

**:All suppliers 4.5.1**

	Supplier Id	Supplier Name	Supplier Address	Supplier Phone	Supplier Email
▶	123456	itay azoulay	netivot	0527557909	itay.az21@gmail.com
	205612955	KB Rims	eilat	0524892070	kobiton6991@gmail.com

Supplier Id :

Supplier Name :

Supplier Adress :

Supplier Phone :

Supplier Email :

Update Supplier

Delete Supplier

**: Create new supplier 4.5.2**

**Create new supplier**

**Supplier ID: \***

**Supplier Full Name: \***

**Supplier Address :**

**Supplier Phone: \***

**Supplier Email:**

**Create Supplier**



### : Create new order 4.5.3

New Order

Choose Supplier

itay azoulay

Part Name : \*

General Brake Pads

Part ID : \*

456456

Part Price : \*

220

Quantity : \*

100

	partId	partName	price	quantity
▶	789456	Brake fluids	100	200
*				

More Parts

Sum Order : 20000

Place Order

### All parts 4.5.4

	Id	Part Name	Price	Quantity
▶	ab12	mazda 3 axle	300.00	4.00
	Ae12	Mazda Brake Rotors	200.00	44.00
	Ae123	Motul oil 5w30	40.00	91.00
	ter8457	spark plugs for mitsubishi	80.00	36.00

Part ID :

Part Name:

Price:

Quantity:

Create Part

Update Part

Delete Part

Warning

Notice: mazda 3 axle quantity is less than 10.

OK

## :All orders 4.5.5

	Order Id	Supplier Name	Date	Price
▸	27	KB Rims	11/24/2023 3:46:12 PM	1000
	28	itay azoulay	11/24/2023 5:08:01 PM	5000
	29	alona brakes	11/25/2023 2:14:32 PM	3000

	partId	partName	price	quantity
▸	323333	brakes	100.00	10.00

Show Details

## :Admin 4.6

<b>Admin</b>
<b>Users</b>
<b>Maintenance</b>
<b>Cars DB</b>

## :Users 4.6.1

	Id	User Name	Password	Email	Job Title
▸	1	admin	admin	admin@admin.com	Manager

User Name:

Password:

Email:

Job Title:

Create User

Update User

Delete User

## :Maintenance 4.6.2

Labors

	Id	Description	Price	Time	discount
▶	1	15K Service	300.00	1.00	0.00
	2	30K Service	450.00	1.50	0.00
	3	45K Service	600.00	2.00	0.00
	4	60K Service	600.00	2.00	0.00
	5	75K Service	300.00	1.00	0.00
	6	90K Service	900.00	3.00	0.00
	7	105K Service	450.00	1.50	0.00
	8	Brake pads Replacement	300.00	1.00	0.00
	9	Brake rotors and pads Replacement	450.00	1.50	0.00
	10	Axle Replacement	300.00	1.00	0.00

Description :

Price :

Time :

Create Labor

Update Labor

Delete Labor

## :Cars DB 4.6.3

Manufacture name:

Model name:

Add model

Delete model

## 5. קוד המערכת

### 5.1 ממשק המשתמש:

#### Login form

```
// an http request method that for login
2 references
public async void LoginRequest(string username, string password)
{
    if(username == String.Empty || password == String.Empty)
    {
        MessageBox.Show("All fields are required", "Error");
        return;
    }

    LoginRequest loginRequest = new LoginRequest(username, password);

    try
    {
        HttpResponseMessage response = await Program.client.PostAsJsonAsync(
            "login/", loginRequest);

        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsStringAsync();

            dashboardForm = new Dashboard(result, UserNameetxt.Text);
            dashboardForm.Show();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

## Create new ticket :

```
// an http request method that creates a new ticket
1 reference
public async void createNewTicket()
{
    try
    {
        CreateNewTicketRequest createNewTicketRequest = new CreateNewTicketRequest(carNumberTxt.Text, clientManufactureTxt.Text, clientModelTxt.Text, clientEngineTxt.Text,
        ,int.Parse(clientYearTxt.Text),int.Parse(clientKmTxt.Text), clientVinNumberTxt.Text, clientId, clientFullNameTxt.Text,
        clientPhoneNumberTxt.Text, clientEmailTxt.Text, causeOfArrivalTxt.Text);
        HttpResponseMessage response = await Program.client.PostAsJsonAsync("Tickets/", createNewTicketRequest);
        if(response.IsSuccessStatusCode)
        {
            MessageBox.Show("Ticket Created!");
        }
        else
        {
            await ErrorHandler.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

## Open tickets:

```
// an http request method that gets all open tickets
2 references
public async void GetAllTickets()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("Tickets/");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<GetAllTicketsResponse>>(responseResult);

            AllTicketsDataGridview.DataSource = jsonResult;

            AllTicketsDataGridview.Columns["ticketId"].HeaderText = "Ticket Id";
            AllTicketsDataGridview.Columns["carId"].HeaderText = "Car Id";
            AllTicketsDataGridview.Columns["clientId"].HeaderText = "Client Id";
            AllTicketsDataGridview.Columns["problems"].HeaderText = "Problems";
            AllTicketsDataGridview.Columns["dateTime"].HeaderText = "Date";
            AllTicketsDataGridview.Columns["price"].HeaderText = "Price";
            try
            {
                ticketId = AllTicketsDataGridview.Rows[0].Cells[0].Value.ToString();
            }
            catch (Exception e)
            {
                MessageBox.Show("No open tickets");
            }
        }
        else
        {
            await ErrorHandler.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

## Close ticket:

```
// an http request method that closes the ticket
1 reference
private async void CloseTicketById(string ticketId)
{
    try
    {
        HttpResponseMessage response = await Program.client.DeleteAsync("Tickets/closeTicket/" + ticketId);

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Ticket closed!");
            this.Close();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButton.OK, MessageBoxIcon.Error);
    }
}
```

## All parts :

```
// an http request method that for getting all parts
4 references
public async void GetAllParts()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("StorageHandler/getParts/");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<GetAllPartsRequest>>(responseResult);
            allPartsDataGrid.DataSource = jsonResult;

            allPartsDataGrid.Columns["partId"].HeaderText = "Id";
            allPartsDataGrid.Columns["partName"].HeaderText = "Part Name";
            allPartsDataGrid.Columns["price"].HeaderText = "Price";
            allPartsDataGrid.Columns["quantity"].HeaderText = "Quantity";

            foreach(GetAllPartsRequest part in jsonResult)
            {
                if(part.quantity < 10)
                {
                    MessageBox.Show($"Notice: {part.partName} quantity is less than 10.", "Warning", MessageBoxButton.OK, MessageBoxIcon.Information);
                }
            }
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButton.OK, MessageBoxIcon.Error);
    }
}
```

```

// an http request method that for creating a new part
1 reference
private async void CreateNewPart()
{
    CreateNewPartRequest createNewPartRequest = new CreateNewPartRequest();
    createNewPartRequest.partId = partIdTxt.Text;
    createNewPartRequest.price = decimal.Parse(partPriceTxt.Text);
    createNewPartRequest.partName = partNameTxt.Text;
    createNewPartRequest.quantity = decimal.Parse(quantityTxt.Text);

    try
    {
        HttpResponseMessage response = await Program.client.PostAsJsonAsync("StorageHandler/createNewPart", createNewPartRequest);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Added successfully", "Success", MessageBoxButtons.OK);
            GetAllParts();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for updating a part
1 reference
private async void UpdatePartById()
{
    CreateNewPartRequest createNewPartRequest = new CreateNewPartRequest();
    createNewPartRequest.partId = partIdTxt.Text;
    createNewPartRequest.price = decimal.Parse(partPriceTxt.Text);
    createNewPartRequest.partName = partNameTxt.Text;
    createNewPartRequest.quantity = decimal.Parse(quantityTxt.Text);

    try
    {
        HttpResponseMessage response = await Program.client.PutAsJsonAsync("StorageHandler/updatePart/" + partIdTxt.Text, createNewPartRequest);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Part updated successfully!", "Success", MessageBoxButtons.OK);
            GetAllParts();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```
// an http request method that for deleting a part
1 reference
private async void DeletePartById()
{
    try
    {
        HttpResponseMessage response = await Program.client.DeleteAsync("StorageHandler/deletePart/" + partIdTxt.Text);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Part deleted successfully!", "Success", MessageBoxButtons.OK);
            GetAllParts();
        }
        else
        {
            await ErrorHandler.HandleErrorResponse(response);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

**All supplier:**



```

// an http request method that for getting al suppliers
3 references
private async void GetAllSuppliers()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("StorageHandler/getAllSuppliers/");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<GetAllSuppliersRequest>>(responseResult);
            allSuppliersDataGrid.DataSource = jsonResult;

            allSuppliersDataGrid.Columns["supplierId"].HeaderText = "Supplier Id";
            allSuppliersDataGrid.Columns["supplierName"].HeaderText = "Supplier Name";
            allSuppliersDataGrid.Columns["supplierAddress"].HeaderText = "Supplier Address";
            allSuppliersDataGrid.Columns["supplierPhone"].HeaderText = "Supplier Phone";
            allSuppliersDataGrid.Columns["supplierEmail"].HeaderText = "Supplier Email";
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for updating a supplier
1 reference
private async void UpdateSupplierById()
{
    GetAllSuppliersRequest getAllSuppliersRequest = new GetAllSuppliersRequest();
    getAllSuppliersRequest.supplierId = supplierIdTxt.Text;
    getAllSuppliersRequest.supplierName = supplierNameTxt.Text;
    getAllSuppliersRequest.supplierAddress = supplierAddressTxt.Text;
    getAllSuppliersRequest.supplierPhone = supplierPhoneTxt.Text;
    getAllSuppliersRequest.supplierEmail = supplierEmailTxt.Text;
    getAllSuppliersRequest.orders = new List<Order>();

    try
    {
        HttpResponseMessage response = await Program.client.PutAsJsonAsync("StorageHandler/updateSupplier/" + supplierIdTxt.Text, getAllSuppliersRequest);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Supplier updated successfully!", "Success", MessageBoxButtons.OK);
            GetAllSuppliers();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```
// an http request method that for deleting a supplier
1 reference
private async void DeleteSupplierById()
{
    try
    {
        HttpResponseMessage response = await Program.client.DeleteAsync("StorageHandler/deleteSupplierById/" + supplierIdTxt.Text);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Supplier deleted successfully!", "Success", MessageBoxButtons.OK);
            GetAllSuppliers();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
```

```
// an http request method that for creating a new supplier
1 reference
private async void CreateNewSupplierRequest(string supplierId, string supplierName, string supplierAddress, string supplierPhone, string supplierEmail)
{
    CreateSupplierRequest createSupplierRequest = new CreateSupplierRequest();

    if (supplierAddress == String.Empty)
    {
        createSupplierRequest.supplierId = supplierId;
        createSupplierRequest.supplierPhone = supplierPhone;
        createSupplierRequest.supplierName = supplierName;
        if (supplierEmail != String.Empty)
        {
            createSupplierRequest.supplierEmail = supplierEmail;
        }
    }
    else
    {
        createSupplierRequest.supplierId = supplierId;
        createSupplierRequest.supplierPhone = supplierPhone;
        createSupplierRequest.supplierName = supplierName;
        createSupplierRequest.supplierAddress = supplierAddress;
        if (supplierEmail != String.Empty)
        {
            createSupplierRequest.supplierEmail = supplierEmail;
        }
    }
    try
    {
        HttpResponseMessage response = await Program.client.PostAsJsonAsync("StorageHandler/newSupplier/", createSupplierRequest);
        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsStringAsync();
            MessageBox.Show("Supplier " + supplierName + " Added successfully", "success");

            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show("Supplier " + supplierName + " Already exists", "conflict");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
```

Create new order:

```

// an http request method for creating a new order
1 reference
private async void createNewOrder()
{
    CreateNewOrderRequest createNewOrderRequest = new CreateNewOrderRequest
    {
        parts = parts.ToList(),
        supplierName = suppliersName[chooseSupplierComboBox.SelectedIndex],
        price = sumPrice
    };

    try
    {
        HttpResponseMessage response = await Program.client.PostAsJsonAsync("StorageHandler/newOrder", createNewOrderRequest);

        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsStringAsync();
            MessageBox.Show("Order Added successfully", "success");

            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show("Order Already exists", "conflict");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

## All clients :

```
// an http request method that for getting all clients
1 reference
public async void GetAllClients()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("client/");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<GetAllClientRequest>>(responseResult);

            AllClientsGridView.DataSource = jsonResult;

            AllClientsGridView.Columns["clientId"].HeaderText = "Id";
            AllClientsGridView.Columns["name"].HeaderText = "Client Name";
            AllClientsGridView.Columns["phone"].HeaderText = "Client Phone";
            AllClientsGridView.Columns["email"].HeaderText = "Email";
            AllClientsGridView.Columns["address"].HeaderText = "Client address";
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```

// an http request method that for searching c client by id
1 reference
public async void searchClientById(string id)
{
    HttpResponseMessage response = await Program.client.GetAsync("client/getClientById/" + id);

    try
    {
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            jsonResult = JsonConvert.DeserializeObject<Client>(responseResult);
            changeVisibilityToTrue();

            searchClientFullNameTxt.Text = jsonResult.name;
            searchClientEmailTxt.Text = jsonResult.email;
            searchClientAddressTxt.Text = jsonResult.address;
            searchClientPhoneTxt.Text = jsonResult.phone;
            carNumberComboBox.Text = jsonResult.cars[0].carId.ToString();
            searchCarManuTxt.Text = jsonResult.cars[0].carManufacture.ToString();
            searchCarModelTxt.Text = jsonResult.cars[0].carModel.ToString();
            searchCarYearTxt.Text = jsonResult.cars[0].carYear.ToString();
            searchCarEngineTxt.Text = jsonResult.cars[0].carEngine.ToString();
            searchVinNumberTxt.Text = jsonResult.cars[0].vinNumber.ToString();
            foreach (Car c in jsonResult.cars)
            {
                carNumberComboBox.Items.Add(c.carId.ToString());
            }
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for searching a client by car number
1 reference
public async void searchClientByCarId(string id)
{
    HttpResponseMessage response = await Program.client.GetAsync("client/getClientByCarId/" + id);
    try
    {
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<Client>(responseResult);
            changeVisibilityToTrue();

            searchClientIdTxt.Text = jsonResult.id;
            searchClientFullNameTxt.Text = jsonResult.name;
            searchClientEmailTxt.Text = jsonResult.email;
            searchClientAddressTxt.Text = jsonResult.address;
            searchClientPhoneTxt.Text = jsonResult.phone;
            carNumberComboBox.Text = jsonResult.cars[0].carId.ToString();
            searchCarManuTxt.Text = jsonResult.cars[0].carManufacture.ToString();
            searchCarModelTxt.Text = jsonResult.cars[0].carModel.ToString();
            searchCarYearTxt.Text = jsonResult.cars[0].carYear.ToString();
            searchCarEngineTxt.Text = jsonResult.cars[0].carEngine.ToString();
            searchVinNumberTxt.Text = jsonResult.cars[0].vinNumber.ToString();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for updating a client information
1 reference
private async void UpdateClient(Client client, Car car)
{
    CreateClientRequest createClientRequest = new CreateClientRequest
    {
        id = client.clientId,
        name = client.name,
        phone = client.phone,
        email = client.email,
        address = client.address,
        carId = car.carId,
        carManufacture = car.carManufacture,
        carModel = car.carModel,
        carEngine = car.carEngine,
        carYear = car.carYear,
        carKilometer = car.carKilometer,
        vinNumber = car.vinNumber
    };

    HttpResponseMessage response = await Program.client.PutAsJsonAsync(
        "client/" + client.clientId, createClientRequest);

    try
    {
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Client " + client.name + " Updated successfully", "success");
            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show($"*Client {client.name} or car {car.carManufacture} {car.carModel} already exist");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for adding another car to a specific client
1 reference
private async void addCarToClient()
{
    AddCarToClientRequest addCarToClientRequest = new AddCarToClientRequest
    {
        carId = carNumberComboBox.Text,
        clientId = searchClientIdTxt.Text,
        carManufacture = searchCarManuTxt.Text,
        carModel = searchCarModelTxt.Text,
        carEngine = searchCarEngineTxt.Text,
        carYear = int.Parse(searchCarYearTxt.Text),
        vinNumber = searchVinNumberTxt.Text,
        dateTime = DateTime.Now,
    };

    HttpResponseMessage response = await Program.client.PutAsJsonAsync("client/client/addCar/" + addCarToClientRequest.clientId, addCarToClientRequest);

    try
    {
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show($"Car {addCarToClientRequest.carManufacture} {addCarToClientRequest.carModel} has been added successfully to client {searchClientFullNameTxt.Text}", "success");
            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show($"Client {searchClientFullNameTxt.Text} or car {addCarToClientRequest.carManufacture} {addCarToClientRequest.carModel} already exist");
        }
        else
        {
            await ErrorHandler.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for getting the car history
1 reference
private async void GetCarHistoryByCarId(string carId)
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("client/getHistory/" + carId);

        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<Models.Ticket>>(responseResult);

            carHistoryDataGridView.DataSource = jsonResult;
            ticketId = carHistoryDataGridView.Rows[0].Cells[0].Value.ToString();

            carHistoryDataGridView.Columns["ticketId"].HeaderText = "Ticket Id";
            carHistoryDataGridView.Columns["carId"].HeaderText = "Car Number";
            carHistoryDataGridView.Columns["clientId"].HeaderText = "Client Id";
            carHistoryDataGridView.Columns["dateTime"].HeaderText = "Date";
            carHistoryDataGridView.Columns["price"].HeaderText = "Total price";
            carHistoryDataGridView.Columns["state"].HeaderText = "State";
        }

        else
        {
            await ErrorHandler.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

All users :



```

1 reference
public async void GetAllUsers()
{
    HttpResponseMessage response = await Program.client.GetAsync("user/");
    try
    {
        if (response.IsSuccessStatusCode)
        {
            string result = await response.Content.ReadAsStringAsync();
            List<GetAllUsersRequest> list = JsonConvert.DeserializeObject<List<GetAllUsersRequest>>(result);

            AllUsersGridView.DataSource = list;

            AllUsersGridView.Columns["Id"].HeaderText = "Id";
            AllUsersGridView.Columns["UserName"].HeaderText = "User Name";
            AllUsersGridView.Columns["Password"].HeaderText = "Password";
            AllUsersGridView.Columns["Email"].HeaderText = "Email";
            AllUsersGridView.Columns["JobTitle"].HeaderText = "Job Title";
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for creating new user
1 reference
private async void createUserRequest(string username, string password, string email, string jobTitle)
{
    CreateUserRequest createUserRequest = new CreateUserRequest
    {
        UserName = username,
        Password = password,
        Email = email,
        JobTitle = jobTitle == "Manager" ? JobTitleClass.JobTitle.Manager : jobTitle == "ServiceAdvisor" ? JobTitleClass.JobTitle.ServiceAdvisor : JobTitleClass.JobTitle.Warehouse
    };
    HttpResponseMessage response = await Program.client.PostAsJsonAsync(
        "user/", createUserRequest);

    try
    {
        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsStringAsync();
            MessageBox.Show("user " + username + " Added successfully", "success");

            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show("user " + username + " Already exists", "conflict");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for updating an existing user
1 reference
public async void UpdateUser(User user)
{
    HttpResponseMessage response = await Program.client.PutAsJsonAsync("user/" + user.Id, user);
    try
    {
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("User " + user.UserName + " updated succesfully");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    finally
    {
        this.Refresh();
    }
}
}

```

```

// an http request method that for deleting a specific user
1 reference
public async void DeleteUser(User user)
{
    HttpResponseMessage response = await Program.client.DeleteAsync("user/" + user.Id);
    try
    {
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("User " + user.UserName + " deleted succesfully");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}

```

**Maintenance :**

```
// an http request method that for getting all the labors
4 references
private async void GetAllLabors()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("Tickets/getLabors");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<List<Labor>>(responseResult);
            allLaborsDataGridView.DataSource = jsonResult;

            allLaborsDataGridView.Columns["Id"].HeaderText = "Id";
            allLaborsDataGridView.Columns["description"].HeaderText = "Description";
            allLaborsDataGridView.Columns["price"].HeaderText = "Price";
            allLaborsDataGridView.Columns["time"].HeaderText = "Time";
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```

// an http request method that for creating a new labor
1 reference
private async void CreateNewLabor( string description, decimal price, decimal time)
{
    Labor labor = new Labor(description, price, time);

    try
    {
        HttpResponseMessage response = await Program.client.PostAsJsonAsync("Tickets/createLabor", labor);

        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Added successfully", "Success", MessageBoxButtons.OK);
            GetAllLabors();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for updating an existing labor
1 reference
private async void UpdateLaborById(int Id, string description, decimal price, decimal time)
{
    Labor labor = new Labor(Id, description, price, time);

    try
    {
        HttpResponseMessage response = await Program.client.PutAsJsonAsync("Tickets/updateLaborById/" + Id, labor);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Labor updated successfully!", "Success", MessageBoxButtons.OK);
            GetAllLabors();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

// an http request method that for deleting an existing labor
1 reference
private async void DeleteLaborById(int Id)
{
    try
    {
        HttpResponseMessage response = await Program.client.DeleteAsync("Tickets/deleteLaborById/" + Id);
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Labor deleted successfully!", "Success", MessageBoxButtons.OK);
            GetAllLabors();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```



```

// an http request method that for getting the monthly statistics
1 reference
private async void GetIncome()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("statistics");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<StatisticsResponse>(responseResult);

            statisticsChart.Series["Income"].Points.AddXY("Today", jsonResult.todaysIncome);
            statisticsChart.Series["Income"].Points.AddXY("This Week", jsonResult.thisWeeksIncome);
            statisticsChart.Series["Income"].Points.AddXY("This Month", jsonResult.thisMonthIncome);
            statisticsChart.Series["Income"].Points.AddXY("This Year", jsonResult.thisYearIncome);
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

catch (Exception ex)
{
    MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

```

// an http request method that for getting the yearly statistics
1 reference
private async void GetYearlyIncome()
{
    try
    {
        HttpResponseMessage response = await Program.client.GetAsync("statistics/yearly");
        if (response.IsSuccessStatusCode)
        {
            var responseResult = await response.Content.ReadAsStringAsync();
            var jsonResult = JsonConvert.DeserializeObject<YearlyStatisticsResponse>(responseResult);

            yearlyIncomeChart.Series.Clear();
            Series series = new Series("Income");

            for (int month = 1; month <= 12; month++)
            {
                double income = GetIncomeForMonth(jsonResult, month);
                series.Points.AddXY(month, income);
            }

            yearlyIncomeChart.Series.Add(series);

            for (int i = 1; i <= 12; i++)
            {
                yearlyIncomeChart.Series["Income"].Points[i - 1].AxisLabel = CultureInfo.CurrentCulture.DateTimeFormat.GetMonthName(i);
            }

            yearlyIncomeChart.ChartAreas[0].AxisX.Minimum = 1;
            yearlyIncomeChart.ChartAreas[0].AxisX.Maximum = 12;
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```
// a method for getting statistics per month
1 reference
private double GetIncomeForMonth(YearlyStatisticsResponse statistics, int month)
{
    switch (month)
    {
        case 1: return statistics.januaryIncome;
        case 2: return statistics.februaryIncome;
        case 3: return statistics.marchIncome;
        case 4: return statistics.aprilIncome;
        case 5: return statistics.mayIncome;
        case 6: return statistics.juneIncome;
        case 7: return statistics.julyIncome;
        case 8: return statistics.augustIncome;
        case 9: return statistics.septemberIncome;
        case 10: return statistics.octoberIncome;
        case 11: return statistics.novemberIncome;
        case 12: return statistics.decemberIncome;
        default: return 0;
    }
}
```

## Adding manufactures and models:

```
// an http request method that for adding the manufacture and model
1 reference
private async void CreateManufactureWithModel(string manufactureName, string modelName)
{
    CreateManufactureWithModelRequest createManufactureWithModelRequest = new CreateManufactureWithModelRequest(manufactureName, modelName);
    HttpResponseMessage response = await Program.client.PostAsJsonAsync(
        "StorageHandler/", createManufactureWithModelRequest);

    try
    {
        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsStringAsync();
            MessageBox.Show("Manufacture " + createManufactureWithModelRequest.manufacturerName + " model " + createManufactureWithModelRequest.modelName + " added successfully", "success");
            this.Close();
        }
        else if (response.StatusCode.Equals(HttpStatusCode.Conflict))
        {
            MessageBox.Show("Manufacture " + createManufactureWithModelRequest.manufacturerName + " model " + createManufactureWithModelRequest.modelName + " Already exists", "conflict");
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```
1 reference
private async void DeleteManufactureAndModel(string manufactureName, string modelName)
{
    HttpResponseMessage response = await Program.client.DeleteAsync("StorageHandler/removeModel/" + manufactureName + "/" + modelName);
    try
    {
        if (response.IsSuccessStatusCode)
        {
            MessageBox.Show("Deleted successfully", "success");
            this.Close();
        }
        else
        {
            await ErrorHandling.HandleErrorResponse(response);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

## 5.2 קוד של שרת העיבוד:

### Garage context:

```
// creating a customized database context that inherits from EF Core Database context
57 references
public class GarageContext : DbContext
{
    // setting up tables as properties for the context
    15 references
    public DbSet<Client> Clients { get; set; }
    9 references
    public DbSet<Suppliers> Suppliers { get; set; }
    14 references
    public DbSet<Part> Part { get; set; }
    9 references
    public DbSet<Car> Car { get; set; }
    8 references
    public DbSet<User> User { get; set; }
    6 references
    public DbSet<Manufacture> Manufacture { get; set; }
    8 references
    public DbSet<Model> Model { get; set; }
    27 references
    public DbSet<Ticket> Ticket { get; set; }
    9 references
    public DbSet<Labor> Labor { get; set; }
    5 references
    public DbSet<Order> Order { get; set; }
    0 references
    public DbSet<OrderPart> OrderPart { get; set; }
    0 references
    public DbSet<TicketPart> TicketPart { get; set; }
    0 references
    public DbSet<TicketLabor> TicketLabor { get; set; }
```



```

// here we are configuring the entities that were defined previously with primary keys and relationships
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Part>();

    modelBuilder.Entity<Ticket>();

    // an entity that has a one to many relationship
    modelBuilder.Entity<Client>(model =>
    {
        model.HasMany(e => e.cars)
            .WithOne(e => e.Client)
            .HasForeignKey(e => e.clientId);
    });
}

// configures the seeding of data for the Labor entity, this enables initial data for it
modelBuilder.Entity<Labor>(model =>
{
    model.HasData(
        new { Id = 1, description = "15K Service", price = 300m, time = 1.0m, discount = 0.0m },
        new { Id = 2, description = "30K Service", price = 450m, time = 1.5m, discount = 0.0m },
        new { Id = 3, description = "45K Service", price = 600m, time = 2.0m, discount = 0.0m },
        new { Id = 4, description = "60K Service", price = 600m, time = 2.0m, discount = 0.0m },
        new { Id = 5, description = "75K Service", price = 300m, time = 1.0m, discount = 0.0m },
        new { Id = 6, description = "90K Service", price = 900m, time = 3.0m, discount = 0.0m },
        new { Id = 7, description = "105K Service", price = 450m, time = 1.5m, discount = 0.0m },
        new { Id = 8, description = "Brake pads Replacement", price = 300m, time = 1.0m, discount = 0.0m },
        new { Id = 9, description = "Brake rotors and pads Replacement", price = 450m, time = 1.5m, discount = 0.0m },
        new { Id = 10, description = "Axle Replacement", price = 300m, time = 1.0m, discount = 0.0m }
    );
});

```

Model example :

```

13 references
public class Ticket
{
    11 references
    [Key] public int ticketId { get; set; }
    11 references
    [Required] public string carId { get; set; }
    4 references
    [Required] public string clientFullName { get; set; }
    6 references
    [Required] public string clientId { get; set; }
    4 references
    [Required] public string clientPhone { get; set; }
    4 references
    [Required] public string clientEmail { get; set; }
    5 references
    [Required] public string problems { get; set; }
    20 references
    [Required] public List<TicketPart> parts { get; set; } = new();
    16 references
    [Required] public List<TicketLabor> labors { get; set; } = new();
    7 references
    public DateTime dateTime { get; set; }
    19 references
    public double totalPartsPrice { get; set; }
    19 references
    public double totalPartsDiscount { get; set; }

    20 references
    public double totalLaborPrice { get; set; }
    19 references
    public double totalLaborDiscount { get; set; }
    13 references
    public double price { get; set; }
    [Required]
    [EnumDataType(typeof(JobTitle))]
    [JsonConverter(typeof(Newtonsoft.Json.Converters.StringEnumConverter))]
    15 references
    public TicketType state { get; set; }
}

```

Server :

```

99+ references
public class Server
{
    // creating a static property of the database context
    public static GarageContext context;

    // a function to start the server
    1 reference
    public void Start(string[] args)
    {
        // creatin a new instant of the database context
        context = new GarageContext();

        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddControllers();
        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseHttpsRedirection();

        app.MapControllers();

        // function for backup every day at midnight
        OthersHandler.ExecuteAtMidnight();

        app.Run();
    }
}

```

## Base controller :

```

// creating a abstract class for the context
6 references
public abstract class GarageController : ControllerBase
{
    // creating a handler prop for the abstract class so that every class that inherits will need to use a handler
    protected Handler handler;
}

```

## Base handler :

```

// an interface for handling requests
8 references
public interface Handler
{
}

```

## Base request:

```
10 references  
public interface Request  
{  
}
```

## Controller example:

```
// base route controller for login that inherits from GarageController  
[Route("/login")]  
[ApiController]  
1 reference  
public class LoginController : GarageController  
{  
    // assigning the handler to the login handler within the constructor  
    0 references  
    public LoginController()  
    {  
        this.handler = new LoginHandler();  
    }  
  
    // post controller for signin in a user  
  
    [HttpPost]  
    0 references  
    public ActionResult Login([FromBody]LoginRequest request)  
    {  
        return ((LoginHandler)handler).HandleCreate(request);  
    }  
}
```

## Handler example :

```
// login handler that handles all the login http requests regarding login  
2 references  
public class LoginHandler : CreateHandler  
{  
    // a function that recives a login request as a parameter  
    // it checks if the user information provided matches the one on the database  
    // if everything checks out it returns status 200,  
    // otherwise it returns a customized failure response  
    2 references  
    public ActionResult HandleCreate(Request request)  
    {  
        LoginRequest loginRequest = (LoginRequest)request;  
  
        var user = Server.Server.context.User.SingleOrDefault(u => u.UserName == loginRequest.UserName && u.Password == loginRequest.Password);  
  
        if (user == null)  
        {  
            return ErrorHandler.onFailure("User not found", "Not found");  
        }  
  
        return new OkObjectResult(user.JobTitle);  
    }  
}
```

## Request example:

```
3 references
public class LoginRequest : Request
{
    1 reference
    public string UserName { get; set; }
    1 reference
    public string Password { get; set; }
}
```

## Response example:

```
3 references
public class ClientResponse
{
    1 reference
    public ClientResponse(string id, string name, string phone, string email, string address, List<CarResponse> cars) {
        this.name=name;
        this.id = id;
        this.phone=phone;
        this.email=email;
        this.address=address;
        this.cars=cars;
    }
    1 reference
    public string id { get; set; } = null!;
    1 reference
    public string name { get; set; } = null!;
    1 reference
    public string phone { get; set; } = null!;
    1 reference
    public string? email { get; set; }
    1 reference
    public string? address { get; set; }
    1 reference
    public List<CarResponse> cars { get; set; }
}
```

## Error handling :

```
// a class that was build for error handling within the handlers for the http requests
70 references
public class ErrorHandler
{
    // a static function that recives 2 necessary parameters and other optional parameters for error handling
    // string message - what message will be in the frontend
    // string error code - message header that will be shown in the frontend
    // int status code that is initialized to 404 (not found), the reason behind this is that most errors occurred based on not found errors
    // string details - initialized to null, for extension purposes if decided moving forward,
    // list of validation error if give
    // the function builds a json object with all the necessary information for giving an error for the frontend to use
    70 references
    public static ActionResult onFailure(string message, string errorCode, int statusCode = StatusCodes.Status404NotFound, string details = null, List<ValidationError> validationErrors = null)
    {
        var errorResponse = new ErrorResponse
        {
            Success = false,
            Message = message,
            StatusCode = statusCode,
            ErrorCode = errorCode,
            Details = details,
            ValidationErrors = validationErrors
        };

        JsonResult jsonResult = new JsonResult(errorResponse);
        jsonResult.StatusCode = statusCode;

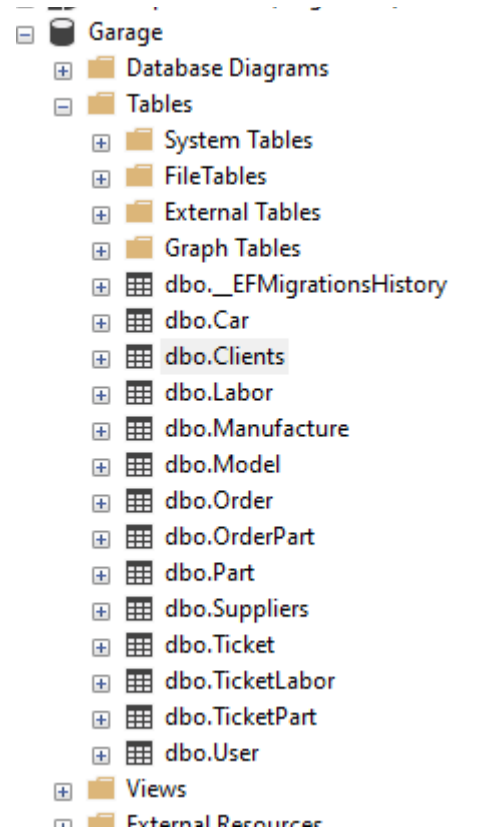
        return jsonResult;
    }
}
```

## Error response :

```
public class ErrorResponse
{
    public bool Success { get; set; } = false;
    public string Message { get; set; }
    public int StatusCode { get; set; }
    1 reference
    public string ErrorCode { get; set; }
    1 reference
    public string Details { get; set; }
    1 reference
    public List<ValidationError> ValidationErrors { get; set; }
}

2 references
public class ValidationError
{
    0 references
    public string Field { get; set; }
    0 references
    public string Message { get; set; }
}
```

### 5.3 בסיס הנתונים:



#### Client table example

DESKTOP-T1R5O2U.Garage - dbo.Clients						
	clientId	name	phone	email	address	dateTime
	205612955	itay azoulay	0527557909	itay.az21@gmai...	netivot	2023-12-05 19:3...
	316505387	kobi biton	0524892070	kobiton6991@g...	beer sheva	2023-11-29 17:0...
	323655803	alona	0507164753	alonazarankin...		2023-11-29 15:5...