שם התלמידה: תמר אינדורסקי

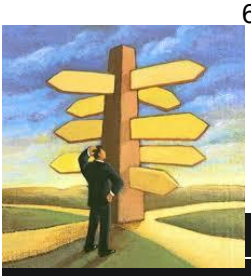
מ.ז. 325192086

סמינר: דרכי חנה

שם המנחה: המורה שרה שנהב

תאריך הגשה: יז' סיון תשפ"ב (16.06.22)

תוכן העניינים



תודות

ראשית ברצונו להודות לבורא עולם, הוא הנותן לי את הכח והיכולת להגיע לידי סיום פרויקט בהיקף גדול שכזה ורמה גבוהה, ממנו הרעיונות והפתרונות וכמה שנודה זה אפילו לא מתחיל...

- הערכתי הרבה נתונה להורי היקרים שהתענינו בכל שלב בפרויקט, והשתדלו לעשות הכל כדי שאגיש פרויקט מושלם ברוגע ובנחת.
- תודה לרכזת, המורה עדינה שטיינברג שעושה הכל כדי שנגיש פרויקטים הכי מושלמים שאפשר. וכן על העזרה והאכפתיות הרבה בעת ביצוע הפרויקט.
- למנחה- גב' שרה שנהב ,שתמכה התענינה ועודדה לאורך כל הדרך.
- לגב' פסי סלוצק על העזרה הרבה בכתיבת הפרויקט ,בניסוח האלגוריתם בהתחברות לgoogle maps ולספריות נוספות.
- לגב' המניק על הסיוע המעשי בכתיבת הקוד ובאיתור בעיות למיניהן.

הצעת פרויקט

סמל מוסד: 711903

שם מכללה: סמינר דרכי חנה

שם הסטודנט: תמר אינדורסקי

ת"ז הסטודנט: 325192086

שם הפרויקט: Find your way

תיאור הפרויקט: אפליקציה לחישוב הדרך המהירה והקצרה ביותר לחלק את הסחורה מהמפעל לחנויות. באמצעות משאית אחת או יותר.

הספק יכניס את הפרטים הנדרשים לצורך החישוב.

האתר יחשב עבור כל משאית את המסלול שעליה לעבור בין החנויות בהתאם לקיבולת המשאית, לכמות הסחורה המוזמנת , למיקום החנות ולשעות שבהן החנויות מוכנות לקבל סחורה. לאחר ביצוע החישוב תינתן האפשרות לנהגי המשאיות להיכנס לאפליקציה ולראות את המסלול המיועד עבורם.

הגדרת הבעיה האלגוריתמית:

חישוב המסלול הקצר ביותר להובלת הסחורה לחנויות בהתאם לאילוצים הקיימים.

רקע תאורטי בתחום הפרויקט:

כיום חלק נכבד מהסכום שאנו משלמים על המוצרים נובע מהעלויות היקרות של ההובלה. תכנון ההובלה אינו יעיל וגורם לשימוש מיותר במשאיות, בכח אדם ובדלק.

האתר יחשב את המסלול האופטימלי ויחסוך בעלויות ההובלה.

תהליכים עיקריים בפרויקט:

- מציאת המרחקים בין החנויות והמפעל.
- מציאת זמני ההגעה המשוערים לכל דרך.
- חישוב המסלול.
- הצגת המסלול הסופי במפה לכל נהג.

תיאור הטכנולוגיה: שרת, לקוח.

שפת תכנות בצד השרת: #C

שפת תכנות בצד הלקוח: REACT

מסד נתונים: SQL

לוחות זמנים:

1. חקר מצב קים- אוקטובר
2. הגדרת הדרישות- אוקטובר
3. אפיון המערכת- נובמבר
4. אפיון בסיס הנתונים- דצמבר

5. עיצוב המערכת- דצמבר
6. בנית התכנה- ינואר-פברואר
7. בדיקות- מרץ
8. הכנת תיק פרויקט- אפריל
9. הטמעת המערכת- מאי
10. הגשת פרויקט סופי- יוני

חתימת הסטודנט:

חתימת רכז המגמה:

אישור משרד החינוך:

מבוא

אחד היישומים החשובים ביותר לאופטימיזציה הוא ניתוב כלי רכב. חישוב וניתוב נכון של כלי הרכב, מוריד משמעותית את זמני הנסיעה ובכך חוסך עלויות רבות. כתוצאה מכך, עומסי התנועה יורדים באופן משמעותי, וזיהום האוויר אף הוא יורד באופן ניכר . ממחקרים שונים מתברר, שתכנון נכון של מסלול עבור חברות הובלה, מוריד כ- 30% מהעלויות, וזאת מבלי להשקיע כמעט, רק למצוא תוכנה יעילה שתבצע זאת!

ארגונים שונים נותנים שירות להזמנות עם צי רכבים .לדוגמה, חנות רהיטים גדולה עשויה להשתמש במספר משאיות כדי להעביר רהיטים לבתים . חברת מיחזור שומנים מתמחה עשויה לנתב משאיות ממתקן כדי לאסוף גריז משומש ממסעדות .מחלקת בריאות עשויה לקבוע ביקורי בדיקה יומיים לכל אחד מפקחי הבריאות שלו.
הבעיה המשותפת לדוגמאות המפורטות לעיל היא בעיית ניתוב הרכב .(VRP) כל ארגון צריך לקבוע אילו הזמנות (בתים, מסעדות או אתרי פיקוח) צריכות לקבל שירות בכל מסלול (משאית או פקח) ובאיזה רצף יש לבקר את ההזמנות .המטרה העיקרית היא לתת שירות הטוב ביותר להזמנות ולמזער את עלות התפעול הכוללת של צי הרכבים .כך, ה VRP-מוצא את המסלולים הטובים ביותר עבור צי רכבים לשירות הזמנות רבות.

בפרויקט זה הדגש העיקרי מונח על ספקים שמשתמשים במספר משאיות כדי לחלק את הסחורה בין החנויות.

האפליקציה נועדה לסייע לספקים, המתעסקים עם חנויות רבות.

לשם הדוגמא, הספק של מאפית אנגל, מחויב לספק לחנויות המזמינות לחם מידי יום.

יש ברשותו משאיות של אנגל הממונות על חלוקת הסחורה.

האפליקציה תחשב עבור כל משאית מסלול אופטימלי של החנויות שבהן היא תחלק את הלחם,

כך שכל חנות תקבל את כמות הלחם הדרושה עבורה.

האפליקציה תתייחס גם לאילוצים הבאים:

מצד המשאית – כמות הלחם שהמשאית מכילה לא תעלה על כמות ההזמנות מהחנויות.

מצד החנויות- כל חנות תקבל את הלחם בטווח מסוים של השעות שהיא דורשת. לדוגמא החנות נפתחת ב 6:00 בבוקר, בעל החנות רוצה שהלחם יגיע עד השעה 08:00. האפליקציה תדאג לכך שהמשאית תגיע לחנות בשעות 06:00-08:00.

תקציר

הפרויקט מבצע את הפעולות הבאות:

הגדרת נתונים:

שלב ראשון הוא תהליך הגדרת הנתונים. החברה מכניסה את נתוני כלי הרכב שברשותה ואת נתוני ההזמנות, החניות ומיקומן לאתר. המערכת תתבסס על נתונים אלו ותגדיר אילוצי זמנים, קיבולת, ומרחקים בין נקודות.

שיבוץ אוטומטי

שלב שני הוא ביצוע השיבוץ. השיבוץ הוא הפעולה העיקרית בפרויקט, הושקע בו מאמץ רב, על מנת שיהיה מדויק ויעיל. פעולה זו מסדרת מסלול לכל מספר הזמנות ומתאימה לו את כלי הרכב, בצורה אופטימלית ע"י התחשבות בכל האילוצים .

הצגת המסלולים לספק

שלב שלישי הוא הצגת המסלולים. הספק יוכל לצפות במפת המסלולים המתארת את השיבוץ הסופי.

וכן כל נהג יוכל להיכנס לאפליקציה ולראות במפה את המסלול שהוכן עבורו.

הממשק

הפרויקט פותח בסביבת עבודה. NET בשיתוף טכנולוגית REACT, API & WEB , המאפשרים בנית אתר של שרת- לקוח בצורה מובנית וברורה האתר מספק ממשק משתמש נוח תוך שימוש בטכנולוגיתREACT .

הקוד

פונקציית צד השרת נכתבו בשפת C# תוך חלוקה נכונה ומסודרת של קוד.

האלגוריתם

תפקיד האלגוריתם המרכזי הוא שיבוץ הזמנות לרכבים וחישוב המסלול האופטימלי.

הספר

המעיין בספר זה ילמד על אופן פעילות האתר, המחלקות, הפונקציות ומבנה מסד הנתונים. וכן הסיבות שהביאו לצורת הפתרון, הפתרון בעצמו ומדריך למשתמש.

היקף הפרויקט

1000שעות לפחות, והרבה הרבה מעבר.

סקירת ספרות

תוך כדי העבודה על הפרויקט, נעזרתי באתרים הבאים לצורך קוד/ עיצוב/ פתרון בעיות:

- Google maps
- W3School
- Npm
- StackOverFlow
- Mui
- react-bootstrap
- Developers .google optimizationrouting vrp
- Altexsoft
- Frontiersin
- Hindawi

אתגרים מרכזיים

הבעיה המרכזית

הבעיה העיקרית בפרויקט היא חישוב המסלול עבור כל משאית, תוך התחשבות באילוצים הקיימים.

נימוקים בבחירת הנושא

מבחינה מקצועית ,רציתי להתנסות במציאת אלגוריתם יעיל המשלב שיבוץ עם מציאת המסלול קצר ביותר. וכן אתגר אותי ללמוד להתעסק עם הממשק של google maps לצורך חישוב המרחקים והצגת המסלול במפה.

צרכים עליהם הפרויקט עונה

הפרויקט מתאים לכל מפעל/חברה המספקת באופן סדיר סחורה לחנויות, ,שימוש באפליקציה זו תעניק לספק ראש נקי מבעיות, בטחון בחיסכון בעלויות, באיכות השירות, ובניצול מקסימלי של זמן ודלק.

מצב קיים

קיימת אפליקציה אשר מבצעת את החישוב הנ"ל אך ממשק המשתמש אינו ברור יש צורך למלא אלפי נתונים וקשה מאוד להשתמש בה.

ניתוח חלופות מערכתי

- בניית רשת זרימה בשיטה של פורד פלקרסון :

עבור כל חנות ניצור צומת

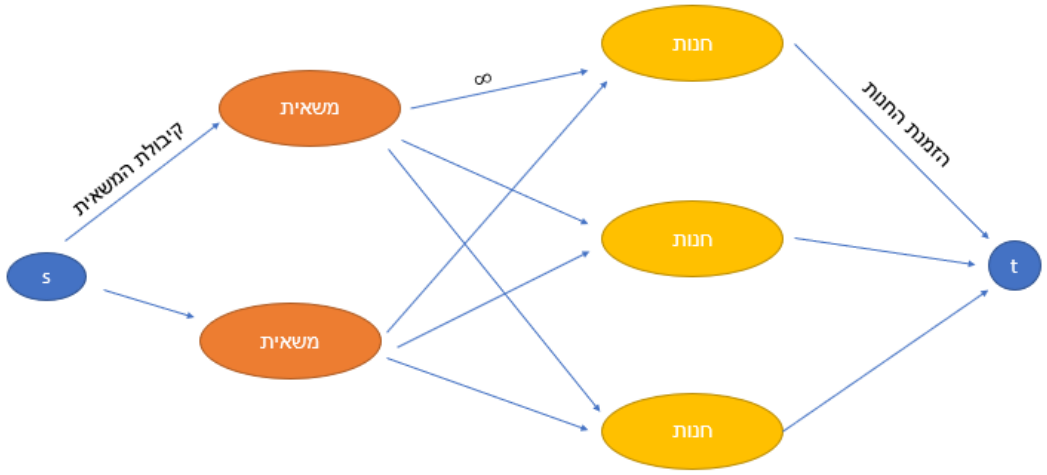
עבור כל משאית ניצור צומת

מצומת המקור S נוציא קשתות לכל משאית ומשקלן יהיה כמספר יחידות הקיבולת של המשאית

מכל צומת המייצג משאית תצא קשת לכל צומת המייצג חנות ומשקלה יהיה ∞.

מכל צומת המייצג חנות תצא קשת לצומת T ,צומת היעד, ומשקלה יהיה כספר יחדות הסחורה שהחנות הזמינה.

הגרף יראה כך:



לאחר בניית הגרף ניתן להריץ עליו את האלגוריתם של פורד פלקרסון למציאת זרימה מקסימלית.

אם הזרימה המקסימלית שווה לסך הדרישות של החנויות הפתרון עונה על האילוץ הראשון (דרישות החנויות לא יעלו על סף קיבולת המשאית).

נמצא את כל הזרימות השוות לזרימה המקסימלית ונבחר מבניהם את זה שעומד גם בדרישות הזמן ואורך המסלול קצר ככל הניתן.

סיבוכיות: O((m*n)!)

הדרך הזו אמנם תיתן פתרון אופטימלי אך בגלל הסיבוכיות הגבוהה, הפתרון , בסופו של דבר, שונה. כפי שיפורט בהמשך.

תיאור החלופה הנבחרת והנימוקים לבחירתה

שימוש בספריית OR TOOLS זו ספריה של Google שמציעה פתרון לבעיות מסוג זה, בהן יש אלפי פתרונות ואם נעבור על כולם נגיע לסיבוכיות בלתי אפשרית.

הספרייה מציעה פתרון מלא לבעיה תוך התחשבות בכל האילוצים ובזמן ריצה סביר.

מטרות ויעדים

בנייה של מערכת נוחה וקלה לשימוש המספקת מענה לבעיה בפשטות וביעילות.

מטרות נוספות במערכת -

- הפחתת עומסי התנועה ושמירה וצמצום זיהום האוויר הנובע מכך .
- חיסכון בדלק, רכבים וזמן.
- בניית אתר יפה ונוח למשתמש.

יעדי המערכת:

- תכנון ושיבוץ המסלולים.
- יצירת ממשק משתמש המאפשר קליטת נתונים בצורה קלה ונוחה.
- הצגת המידע בצורה ברורה ונוחה.

תיחום המערכת

המערכת מטפלת ב:

- הזנת נתוני כלי הרכב, הקיבולת שלהם, פרטי החנויות ומיקומן בממשק נוח ויעיל.
- חישוב המסלולים האופטימליים ושיבוץ ההזמנות לרכבים בהתאם לאילוצים שנקבעו.
- הצגת תוצאות חישוב המסלול לספק וכן לכל נהג לחוד.

פיתוח עתידי:

- במקרה בו לא ימצא שיבוץ העונה על האילוצים הקיימים, המערכת תציג מספר שיבוצים העונים על מירב האילוצים המתאפשר.
- המערכת תתמוך בעריכת המסלול ע"י המשתמש.

אפיון המערכת

סביבת פיתוח

חומרה: מעבד i7 GB8 RAM

עמדת פיתוח: מחשב Intel

מערכת ההפעלה: Windows10

שפות תוכנה: #C תוך שימוש בטכנולוגית net core.

כלי תוכנה לפיתוח המערכת: Microsoft Visual Studio 2019

מסד נתונים: SQL Server

עמדת משתמש מינימאלית:

- חומרה: RAM 4GB i5
- מערכת הפעלה: Windows 7 ומעלה.
- חיבור לרשת: נדרש
- תוכנות: Internet Explorer 7 or higher or chrome

תיאור הארכיטקטורה

הפרויקט מחולק ל:

- מסד נתונים-SQL
- צד השרת
- צד הלקוח

בצד השרת הארכיטקטורה של הפתרון – מודל שלושת השכבות.

האפליקציה מחולקת למספר שכבות:

- **DAL** - כאן אנו פונים לבסיס הנתונים ,יש לנו את המודל שמכיל את הטבלאות של בסיס הנתונים.

  בתוכו מוכלת שכבת DTO:(Data Transfer Object)

  לעיתים יש פערים בין שכבות ה Data -לבין מה שהלקוח רואה בפועל על המסך, ולכן אנו עושים שימוש ב- DTO שזו מחלקה המייחצנת משתנים ואין לה פונקציות. המחלקות האלה הן אלה שתחזורנה לצד לקוח, ולא האובייקטים של הDB
- **BLL-** בשכבה זו כתובה הלוגיקה העסקית
- **API-** שכבה זו היא בעצם ה-controllers **.**

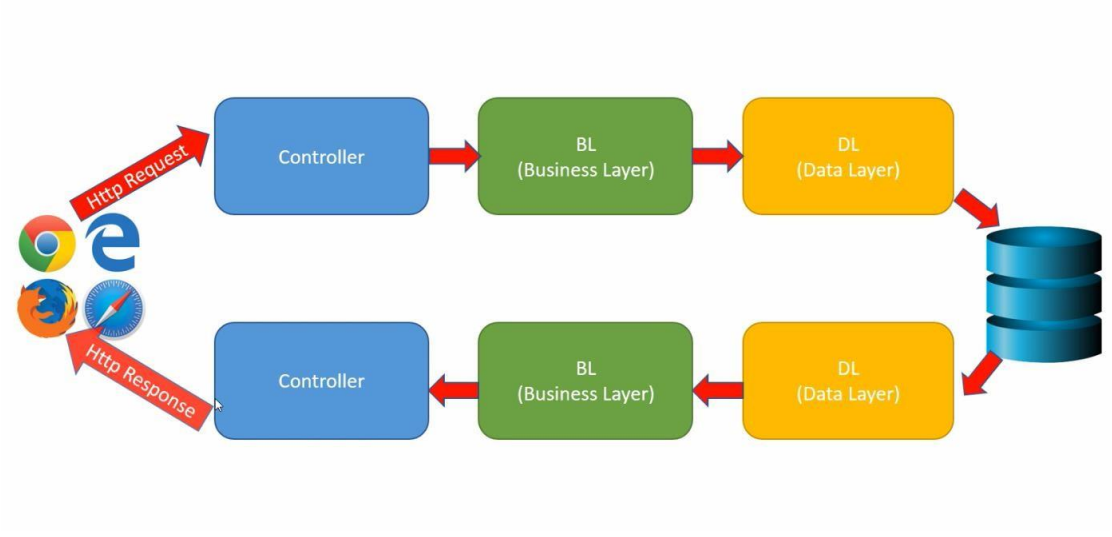
  השכבה הזו אחראית על ההתקשרות בין ה-frontend ה REACT לבין ה-server ה-#C

  בשכבה זו יש את קריאות ה http:

  Get, Delete, Post, Put

בשיטה זו כל שכבה מתקשרת רק עם השכבה הסמוכה לה!

תרשים זרימה:



תיאור הרכיבים בפתרון

טכנולוגיית התקשורת שרת – לקוח: Web API

צד שרת: .net core

שפת תכנות בצד שרת: C#

צד לקוח: React

שפת תכנות בצד לקוח: html+javascript

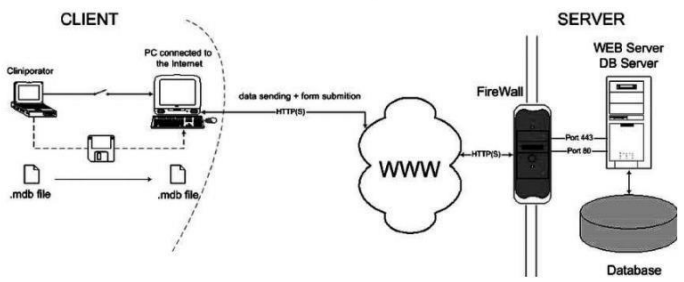
מסד נתונים: SQL Server

תיאור פרוטוקול התקשורת

המערכת מורכבת משרת IIS המריץ את האתר בסביבת ה- Server

מסד נתונים- database של Sql Server

ממשק משתמש בצד הלקוח: דפדפן אינטרנט כלשהו, internet explorer, firefox, chrome.



אפיון המערכת

ארכיטקטורת המערכת

Web – API

ארכיטקטורת שרת לקוח,

בארכיטקטורה זו קיימות 3 שכבות שדרכן עוברים הנתונים בצורה מסודרת וקבועה מראש.

תיכון מפורט

צד שרת: **C#**

צד לקוח: React

מסד נתונים: SQL Server

חלופות לתיכון המערכת

Angular / python / java/C++ וכו"

תיאור התוכנה

סביבת עבודה

Visual Studio 2019 , Visual Studio Code.

שפות תכנות

.WebAPI / C# / React: html, css

מבנה המערכת

בסיס הנתונים

מהלך ניתוח המערכת עבר חשיבה רבה כיצד לבנות את מסד הנתונים בצורה הנכונה והמדויקת ביותר.

מאגר הנתונים הרלוונטיים למערכת מכיל אוסף גדול של נתונים. עם זאת, הפרויקט שלנו המתמקד דווקא בחישוב אינו צריך את כל הנתונים אלא את רובם.

בכל זאת הושקעו שעות רבות באפיון בסיס הנתונים כך שיכיל רק את הנחוץ בלי כפילות נתונים, ובאופן שיאפשר פיתוח ושדרוג של המערכת בלי זעזועים לחלקים הקיימים.

הנתונים נשמרים בצורה טבלאית בבסיס נתונים SQL-Server.

באתחול המערכת נאגרים במסד הנתונים מיקומי והזמנות החנויות וכן נתוני הרכבים והקיבולת שלהם.

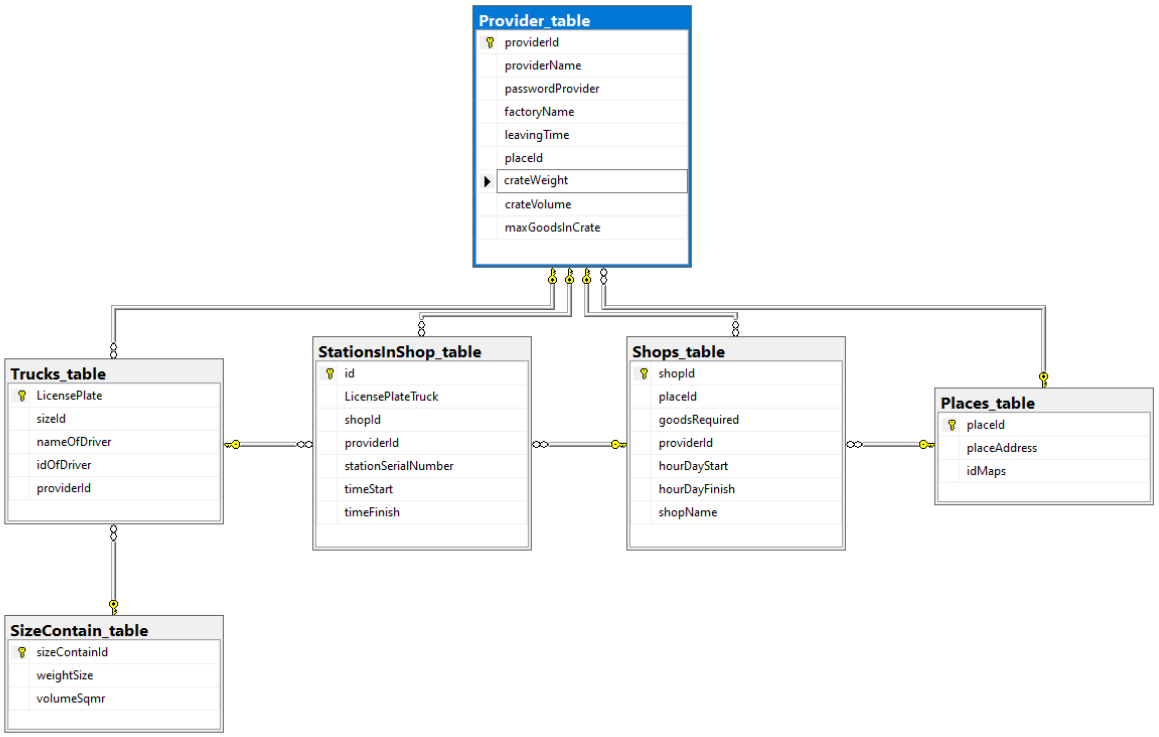
לאחר מכן תוצאות החישוב נשמרות במסד הנתונים. בכל עדכון של אחד מנתוני המערכת, מתבצע חישוב מחודש ועדכון התוצאות במסד הנתונים.

תרשים מסד נתונים

מסד הנתונים מכיל טבלאות עם קשרי גומלין ביניהם ע"מ לאפשר שמירה על חוקיות הנתונים ותאימות תבין החלקים השונים.

כמו כן מכיל המסד שדות מפתח שמונעים כפילות נתונים אסורה ומזרזים את שליפת הנתונים.

בתרשים הבא ניתן לראות את הטבלאות במסד:



טבלאות

הנתונים נשמרים בתוך טבלאות, השדות נשמרים בצורה טבלאית כשלכל שדה יש שם שנשמר במערכת SqlServer, ולפי השמות של השדות נשלפים הנתונים.

טבלת חנויות- shops

הטבלה מכילה את נתוני החנויות הבסיסיים.

מפתחות	שם השדה	סוג השדה	תיאור	שדה חובה
PK	shopId	int	קוד החנות	✓
FK לטבלת מקומות	placeId	int	קוד מקום	✓
	goodsRequired	int	כמות הסחורה שהחנות הזמינה	✓

FK לטבלת ספקים	providerId	Char(9)	ת.ז. הספק אליו שייכת החנות	✓
	hourDayStart	time	מאיזה שעה החנות רוצה לקבל סחורה	✓
	hourDayFinish	time	עד איזה שעה החנות רוצה לקבל סחורה	✓
	shopName	string(50)	שם החנות	

טבלת מקומות-places

מפתחות	שם השדה	סוג השדה	תאור	שדה חובה
PK	placeId	int	קוד	✓
	placeAddress	String(50)	כתובת	✓
	idMaps	String(150)	מחרוזת המייצגת את המקום בגוגל מפות	✓

טבלת ספקים-providers

מפתחות	שם השדה	סוג השדה	תאור	שדה חובה
PK	providerId	Char(9)	ת.ז. ספק	✓
	providerName	String(20)	שם הספק	✓
	passwordProvider	String(15)	סיסמא	✓
	factoryName	String(30)	שם החברה/המפעל	✓
	leavingTime	time	השעה שבה יוצאות המשאיות מהמפעל .	✓
FK לטבלת מקומות	placeId	int	קוד מקום	✓
	crateWeight	float	משקל ארגז סחורה (בק"ג)	✓
	crateVolume	int	נפח ארגז סחורה (במ"ר)	✓
	maxGoodsInCrate	int	כמות יחידות הסחורה הנכנסות בארגז אחד.	✓

הערה: כדי לחשב את כמות הסחורה הנכנסת במשאית מסוימת, הספק מתבקש להזין משקל ונפח ארגז סחורה, ואת כמות יחידות הסחורה הנכנסות בארגז אחד.

כך ניתן לבדוק את כמות יחידות הסחורה שמכילה כל משאית.

טבלת הגבלות קיבולת -sizeContain

מפתחות	שם השדה	סוג השדה	תאור	שדה חובה
PK	sizeContainId	int	קוד קיבולת	✓
	weightSize	float	הגבלת משקל (בק"ג)	✓
	volumeSqmr	int	הגבלת נפח (במ"ר)	✓

טבלת משאיות- trucks

מפתחות	שם השדה	סוג השדה	תאור	שדה חובה
PK	LicensePlate	String(8)	מספר לוחית רישוי משאית	✓
FK **לטבלת הגבלות קיבולת**	sizeId	int	קוד הגבלת קיבולת	✓
	nameOfDriver	String(20)	שם הנהג	✓
	idOfDriver	Char(9)	ת.ז. הנהג	✓
FK **לטבלת ספקים**	providerId	Char(9)	ת.ז. של הספק אליו שייכת המשאית	✓

טבלת stationInShop -תחנות

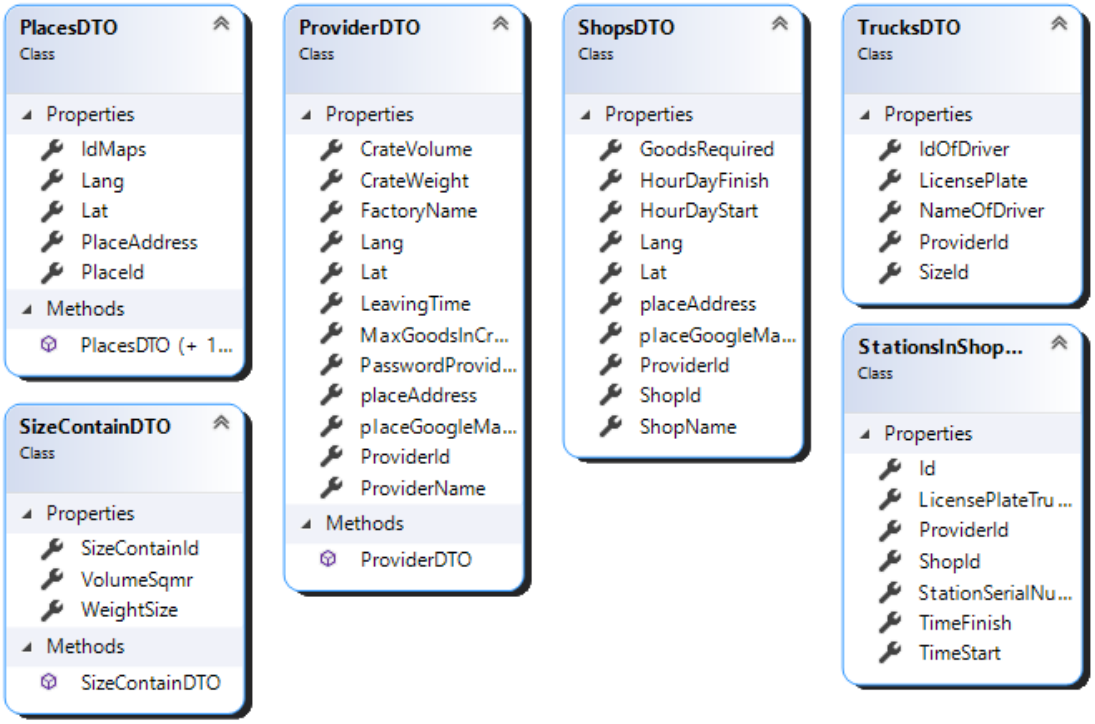
טבלה זו מייצגת את תוצאות השיבוץ, אלו התחנות במסלול לכל משאית.

מפתחות	שם השדה	סוג השדה	תאור	שדה חובה
PK	id	int	קוד תחנה	✓
FK **לטבלת משאיות**	LicensePlateTruck	String(8)	מספר לוחית רישוי משאית	✓
FK **לטבלת חנויות**	shopId	int	קוד חנות	✓
FK **לטבלת ספקים**	providerId	Char(9)	ת.ז. ספק	✓
	stationSerialNumber	int	מספר סידורי של התחנה במסלול	✓
	timeStart	time	זמן היציאה לדרך	
	timeFinish	time	זמן הסיום	

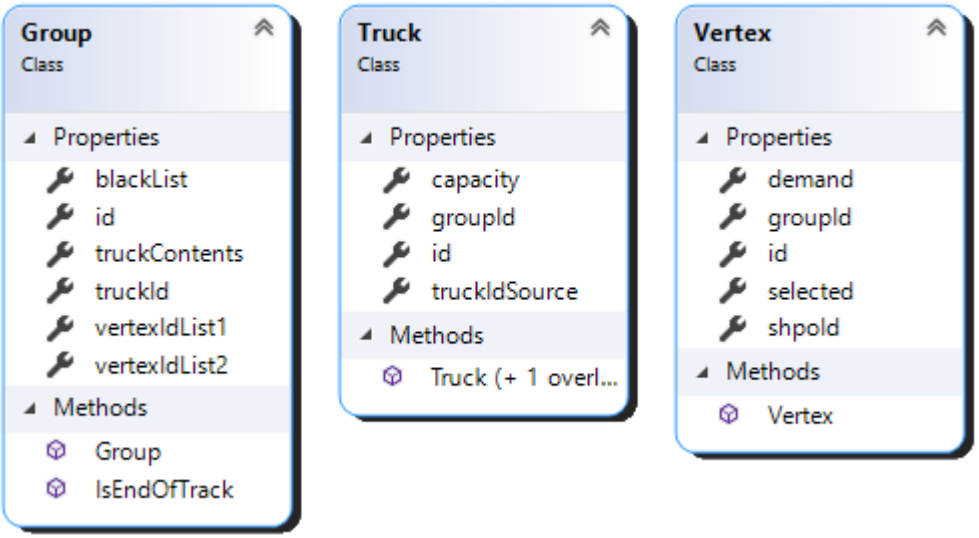
תרשים מחלקות הפרויקט

מחלקות ה – **D**ata**T**ransfer**O**bject

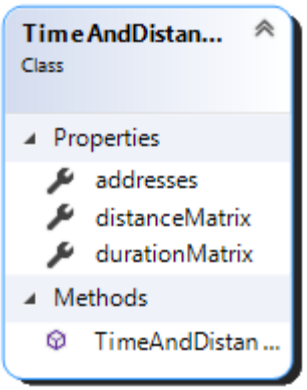
DTO– זו מחלקה המייחצנת משתנים ואין לה פונקציות. המחלקות האלה הן שתחזורנה לצד לקוח, ולא האובייקטים של ה -DB . לעיתים יש פערים בין שכבות ה Data -לבין מה שהלקוח רואה בפועל על המסך , ולכן אנו מחזירות ללקוח אובייקט DTO ומבצעות המרה בין האובייקטים של הDB לאובייקטים של DTO.



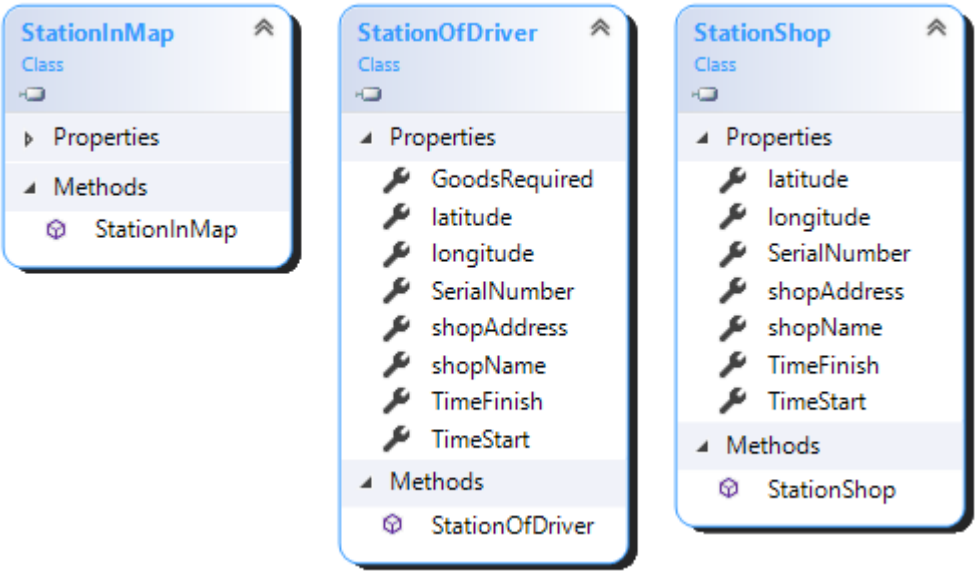
מחלקות הקוד



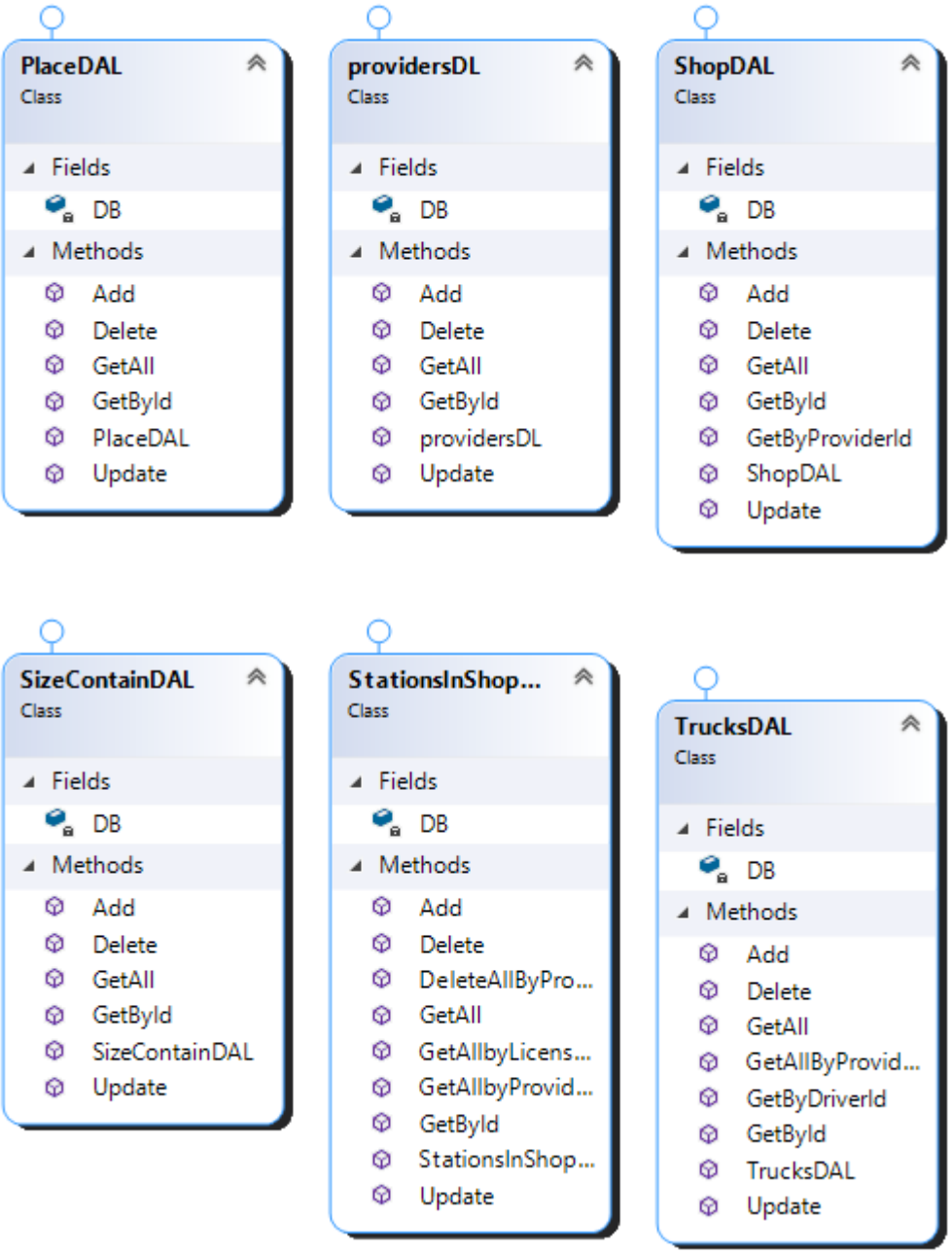
מחלקת ה-google maps



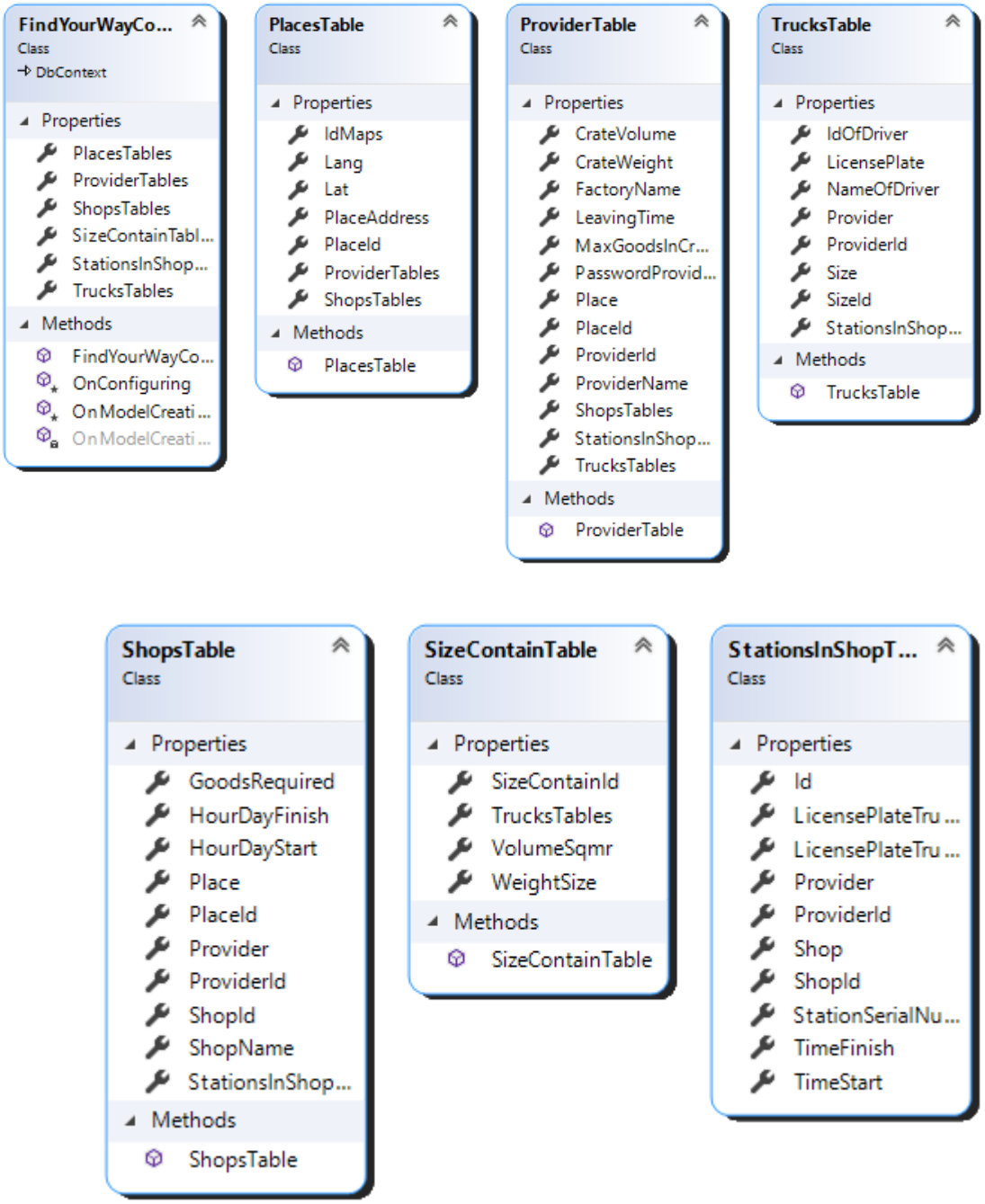
מחלקות הצגת המפה



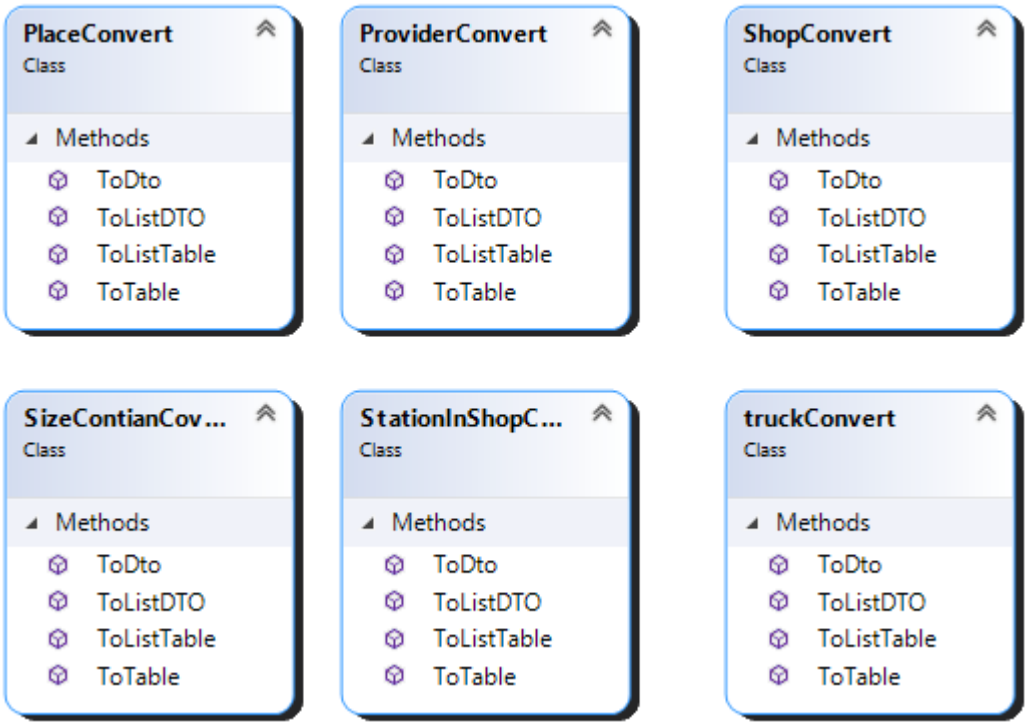
מחלקות ה-DAL



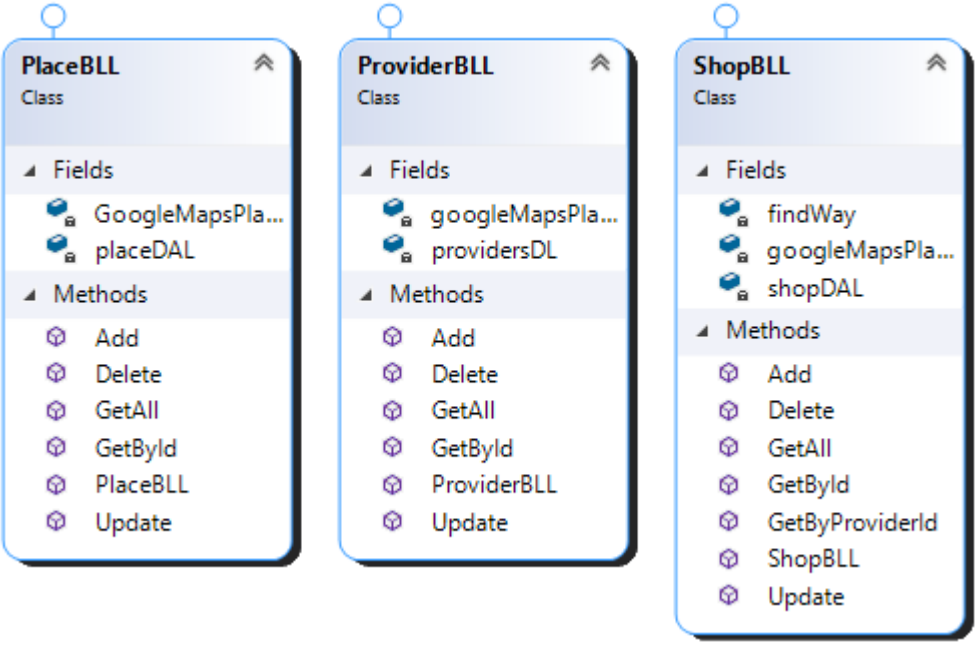
מחלקות ה- model

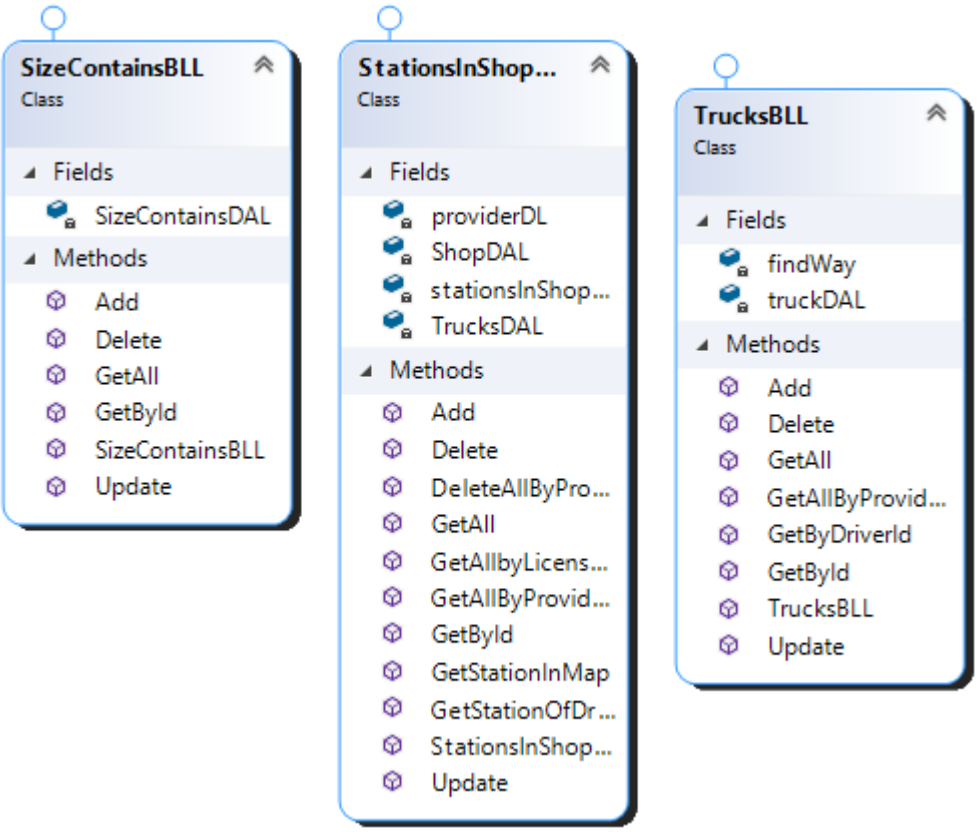


מחלקות הconvert

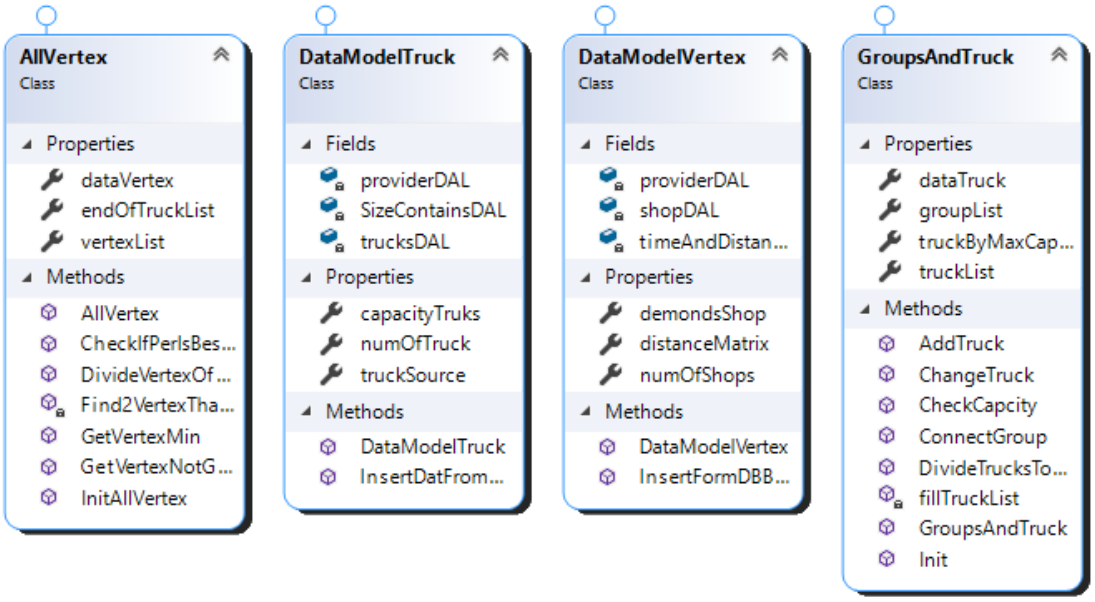

PlaceConvert
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable
ProviderConvert
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable
ShopConvert
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable
SizeContianCov...
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable
StationInShopC...
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable
truckConvert
Class
Methods
ToDto
ToListDTO
ToListTable
ToTable

מחלקות ה-BLL

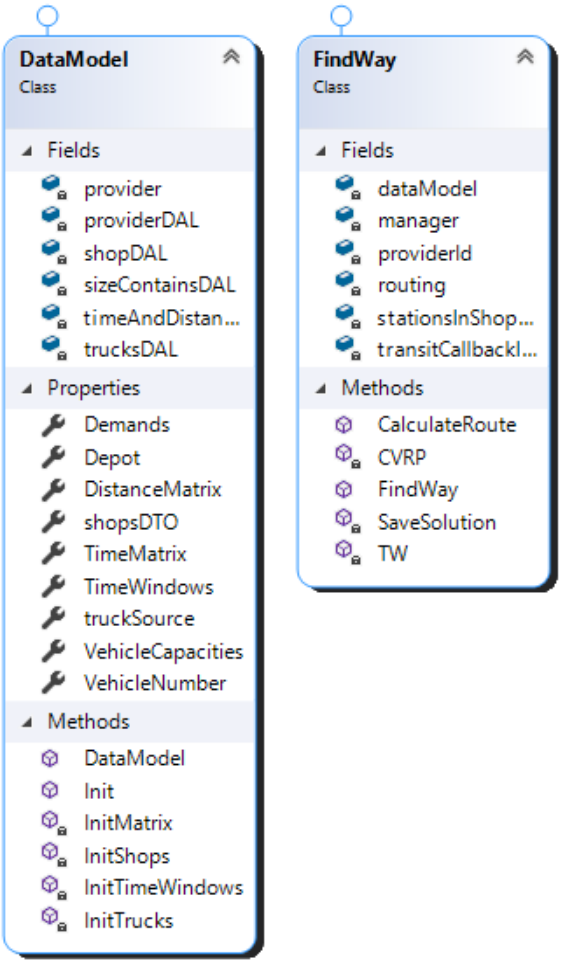

PlaceBLL
Class
Fields
GoogleMapsPla...
placeDAL
Methods
Add
Delete
GetAll
GetById
PlaceBLL
Update
ProviderBLL
Class
Fields
googleMapsPla...
providersDL
Methods
Add
Delete
GetAll
GetById
ProviderBLL
Update
ShopBLL
Class
Fields
findWay
googleMapsPla...
shopDAL
Methods
Add
Delete
GetAll
GetById
GetByProviderId
ShopBLL
Update



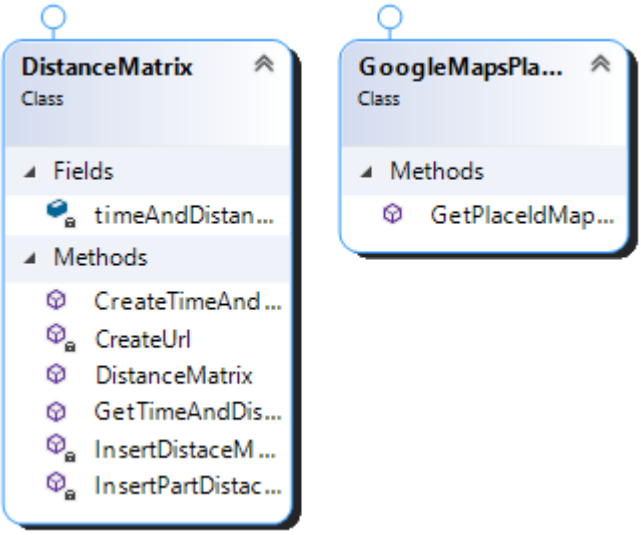
מחלקות הקוד



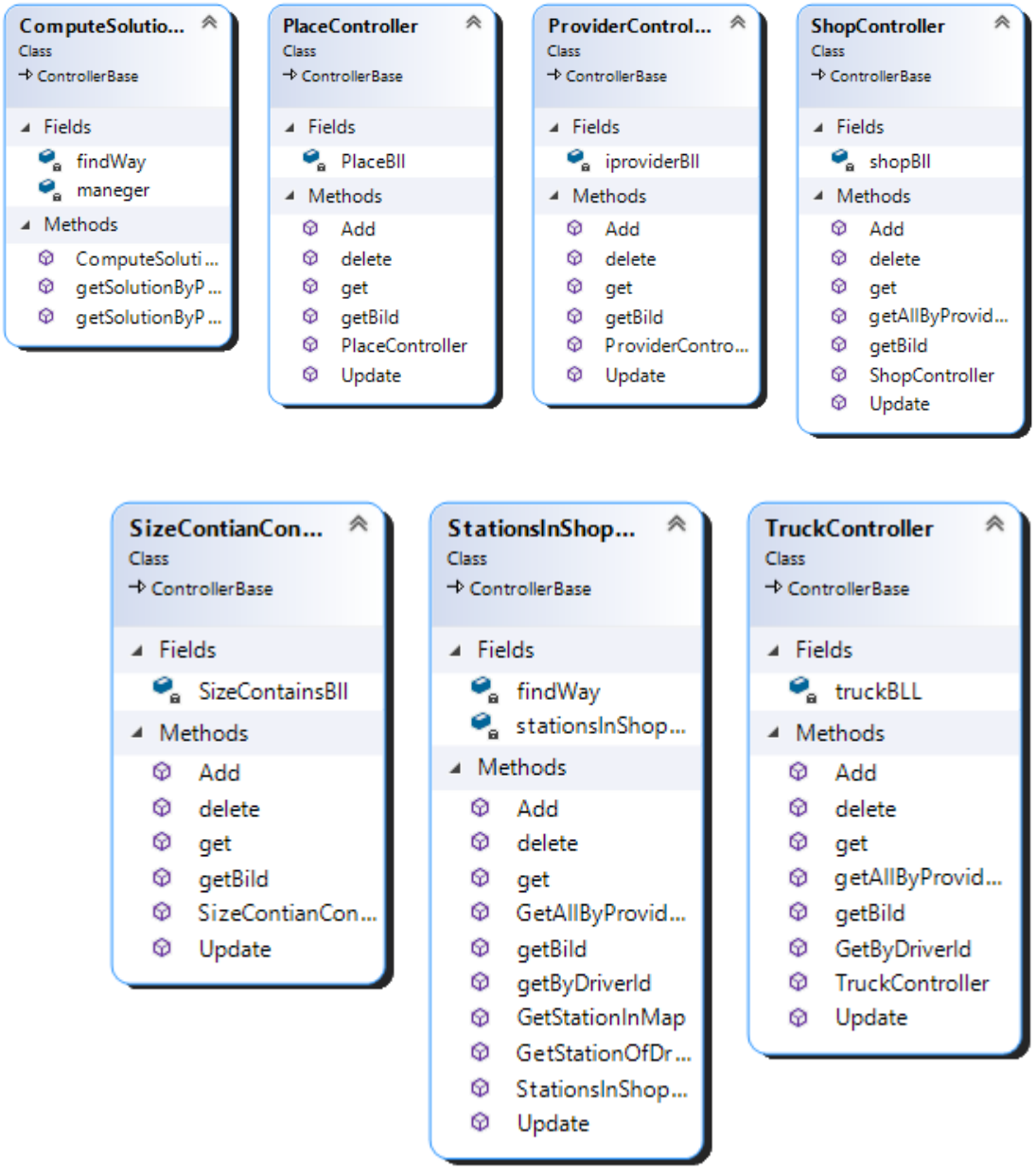
מחלקות OR TOOLS



מחלקות ה-google maps



מחלקות ה- controller



תיאור המחלקות

מחלקות ה DataTransferObject -

מחלקות אלה, הן שתחזורנה לצד הלקוח, ולא האובייקטים של DB .

- מחלקות הקוד- מייצגות אובייקטים לשימוש האלגוריתם
- מחלקת ה-google maps אובייקט המכיל מטריצת מרחקים
- מחלקות הצגת המפה -אובייקט הנצרך לשם הצגת המפה ב Client

מחלקות ה DAL -

מחלקות אלו שואבות מידע ממסד הנתונים ,הבקשות למידע מגיעות משכבת ה BLL - שהיא מפנה את הבקשה מהלקוח למחלקה המתאימה בDAL– .כל מחלקה מממשת ממשק ששם כתובות כל הפונקציות של המחלקה.

מחלקות אלו משתמשות בפורמט שהלקוח לא מכיר לכן צריך לעשות המרה.

- **מחלקות הconvert** מבצעות את ההמרה בין DAL לDTO
- **מחלקות ה- model** הטבלאות הבסיסיות של מסד הנתונים

מחלקות ה BLL –

מחלקות אלו מקשרות בין הבקשות מהלקוח לבין השליפות ממסד הנתונים – הנעשות במחלקות ה.DAL -

כל מחלקה מממשת ממשק שבו כתובות כל הפונקציות של המחלקה .כאשר מגיעה בקשה מה-controller, הבקשה תקרא לפעולה הרצויה מהממשק והיא תפנה אל המחלקה המתאימה לצורך ביצועה של הפעולה .

- **מחלקות הקוד** במחלקות אלו מתבצע הקוד של האלגוריתם
- **מחלקות OR TOOLS** מבצעות את השימוש בספרייה
- **מחלקות ה-google maps** מבצעות את הקריאות ל google maps

האלגוריתם המרכזי

הגדרת הבעיה האלגוריתמית:

מציאת מסלול עבור כל משאית, כך שכל חנות תקבל את כמות הסחורה הנדרשת עבורה, בזמן המבוקש, וכן משך זמן הנסיעה יתקצר ככל האפשר.

תהליך הפתרון:

בשלב ראשון, כדי לדעת את המרחק בין החנויות , מתבצעת קריאה לגוגל מפות, בקריאה נשלחות מחרוזות place_id המייצגות את כתובות החניות וכן את כתובת המפעל ומתקבלת מטריצת מרחקים המייצגת את המרחק מכל חנות לכל חנות.

הערה: כאשר מוסיפים חנות או מעדכנים את כתובת הפעל מתבצעת קריאה ל google maps כדי לקבל את מחרוזת place_id.

הפונקציה הבאה מקבלת כתובת ומחזירה את אובייקט מקום הכולל place_id וקואורדינטות.

```
public PlacesDTO GetPlaceIdMapsByAddress(string address)
        {
            address += " ישראל";
            string                                  url                                =
"https://maps.googleapis.com/maps/api/geocode/json?address="    +    address    +
"&key="+ keyGoogleMaps.key;
            string html = string.Empty;
            HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
            request.AutomaticDecompression = DecompressionMethods.GZip;



            using             (HttpWebResponse             response             =
(HttpWebResponse)request.GetResponse())
            using (Stream stream = response.GetResponseStream())
            using (StreamReader reader = new StreamReader(stream))
            {
                html = reader.ReadToEnd();
            }
            dynamic d2 = JsonConvert.DeserializeObject(html);
            PlacesDTO p = new PlacesDTO();
            p.IdMaps = d2.results[0].place_id;
            p.Lang = d2.results[0].geometry.location.lng;
            p.Lat = d2.results[0].geometry.location.lat;
            return p;
        }
```

פונקציות אלו מבצעות את הקריאה ל-google maps ויוצרות מטריצת מרחקים.

הממשק של google maps לא מאפשר ליצור מטריצה שגודלה עולה על 100 מקומות.

בחישוב פשוט מתברר, שכאשר יהיו מעל 9 חנויות לספק מסוים לא תהיה אפשרות ליצור עבורו מטריצת מרחקים. לכן במקרה כזה, ישלחו כמה קריאת ל google-maps ותיבנה מהם מטריצת מרחקים שלמה.

CreateTimeAndDistanceMatrix(string depotAddress, List<string> addressShop)

הפונקציה הזו מקבלת את כתובת המחסן ואת רשימת כתובות המשאיות ויוצרת מטריצת מרחקים

קוראת לפונקציה

InsertDistaceMatrixAndTimeMatrix(string[] addresses)

פונקציה זו מקבלת מערך של כתובות ויוצרת ממנו מטריצת מרחקים

קוראת לפונקציה

InsertPartDistaceMatrixAndTimeMatrix(string[] origin_addresses, string[] dest_addresses)

פונקציה זו מקבלת מערך של כתובות מקור ומערך של כתובות יעד ויוצרת מטריצת מרחקים מתאימה בהנחה שגודל המטריצה לא יעלה על 100.

קוראת לפונקציה

CreateUrl(string[] origin_addresses, string[] dest_addresses)

פונקציה זו מקבלת מערך של כתובות מקור ומערך של כתובות יעד ויוצרת כתובת url תואמת לשליחה ל google-maps.

פירוט הפונקציות:

```
private static string CreateUrl(string[] origin_addresses, string[]
dest_addresses)
        {
            string url =
"https://maps.googleapis.com/maps/api/distancematrix/json?";
            string orgins = "";
            string dest = "";
            foreach (string item in origin_addresses)
            {
                orgins += "place_id:" + item + "|";
            }
            orgins = orgins.Substring(0, orgins.Length - 1);
            foreach (string item in dest_addresses)
            {
                dest += "place_id:" + item + "|";
            }
            dest = dest.Substring(0, dest.Length - 1);
            url += " &destinations=" + dest + "&origins=" + orgins +
"&key="+keyGoogleMaps.key;
            return url;
        }
        private void InsertPartDistaceMatrixAndTimeMatrix(string[]
origin_addresses, string[] dest_addresses)
        {

            string html;
            if (origin_addresses.Length * dest_addresses.Length > 100)
                return;
            HttpWebRequest request =
(HttpWebRequest)WebRequest.Create(CreateUrl(origin_addresses, dest_addresses));
            request.AutomaticDecompression = DecompressionMethods.GZip;

            using (HttpWebResponse response =
(HttpWebResponse)request.GetResponse())
            using (Stream stream = response.GetResponseStream())
            using (StreamReader reader = new StreamReader(stream))
            {
                html = reader.ReadToEnd();
            }
            dynamic d2 = JsonConvert.DeserializeObject(html);
```

```
            for (int i = 0; i < dest_addresses.Length; i++)
            {
                List<int> rowInMatrix = new List<int>();
                for (int j = 0; j < origin_addresses.Length; j++)
                {
rowInMatrix.Add(Convert.ToInt32(d2.rows[j].elements[i].duration.value));
                    //
MessageBox.Show(Convert.ToInt32(d2.rows[i].elements[j].distance.value));
                }
                timeAndDistanceMatrix.durationMatrix.Add(rowInMatrix);
            }
            for (int i = 0; i < dest_addresses.Length; i++)
            {
                List<int> rowInMatrix = new List<int>();
                for (int j = 0; j < origin_addresses.Length; j++)
                {
rowInMatrix.Add(Convert.ToInt32(d2.rows[j].elements[i].distance.value));
                    //
MessageBox.Show(Convert.ToInt32(d2.rows[i].elements[j].distance.value));
                }
                timeAndDistanceMatrix.distanceMatrix.Add(rowInMatrix);
            }


        }
```

```
        private void InsertDistaceMatrixAndTimeMatrix(string[] addresses)
        {
            int max_elements = 100;
            int numAddress = addresses.Length;
            int numColumn = numAddress;
            if (numColumn > max_elements)
                return;
            int numRows = max_elements / numColumn;
            //double numRows_ = (double)max_elements / numColumn;
            //if (numRows_ != (int)numRows_)
            //    numRows_++;
            //int numRows = (int)numRows_;
            int numSend = numAddress / numRows;
            int numRowInEndSend = numAddress - numSend * numRows;
            string[] destion_adress;
            for (int i = 0; i < numSend; i++)
            {
                destion_adress = new string[numRows];

                Array.Copy(addresses, i * numRows, destion_adress, 0, numRows);
                InsertPartDistaceMatrixAndTimeMatrix(addresses,
destion_adress);
            }
            if (numRowInEndSend == 0)
                return;
            destion_adress = new string[numRowInEndSend];

            Array.Copy(addresses, destion_adress, numRowInEndSend);
            InsertPartDistaceMatrixAndTimeMatrix(addresses, destion_adress);
        }
```

```
        // מקבל את כתובת המחסן וכתובות החנויות
        public void CreateTimeAndDistanceMatrix(string depotAddress,
List<string> addressShop)
        {
            //באינדקס 0 יהיה שמור כתובת המחסן
            timeAndDistanceMatrix.addresses.Add(depotAddress);
            timeAndDistanceMatrix.addresses.AddRange(addressShop);

InsertDistaceMatrixAndTimeMatrix(timeAndDistanceMatrix.addresses.ToArray());

        }
```

מחזירה את מטריצת המרחקים שנוצרה

```
        public TimeAndDistanceMatrix GetTimeAndDistanceMatrix()
        {
            return timeAndDistanceMatrix;
        }
```

לאחר שיש לנו מטריצת מרחקים וכן מטריצת זמנים (נעשתה באותו האופן) ניתן כבר להתחיל בחישוב עצמו.

אפשר להעביר את הבעיה לגרף כאשר כל צומת מייצג חנות, צומת המקור S מייצג את המחסן ממנו יוצאות המשאיות. נעביר קשתות מכל צומת לכל צומת בגרף (גרף קליקה) כאשר משקל כל קשת (Ui,Vj) יקבע לפי יחידות הזמן הנדרשות למעבר בין החנויות אותן מייצגות הצמתים Ui ו Vj .

כעת ניתן להגדיר את הבעיה כמציאת מסלול המילטוני (מסלול היוצא מ-S ועובר בכל צומת פעם אחת ולבסוף חוזר לצומת S). כאשר הוא מתחלק למספר מסלולים (מספר המשאיות המשתתפות בהובלה) על כל המסלולים יחד לכסות את כל צמתי הגרף. כל מסלול הוא מעגלי, יוצא מהמחסן וחוזר למחסן.

תחילה נראה את האלגוריתם הסופי, בהמשך מצורפת דוגמא המסייעת להבין את המשמעות באופן מעשי יותר.

לצורך פתרון הבעיה נגדיר:

K -מספר המשאיות המשתתפות בהובלה

קצה מסלול -הקודקוד בגרף שמהווה כרגע את סיום המסלול.

קדקוד שחור-קדקוד שלא שייך לקבוצה מסוימת.

קצה מסלול פסול- קצה מסלול שמוגדר כרגע ככזה שאסור למתוח ממנו קשתות.(כביכול הקשתות שיוצאות ממנו מחוקות).

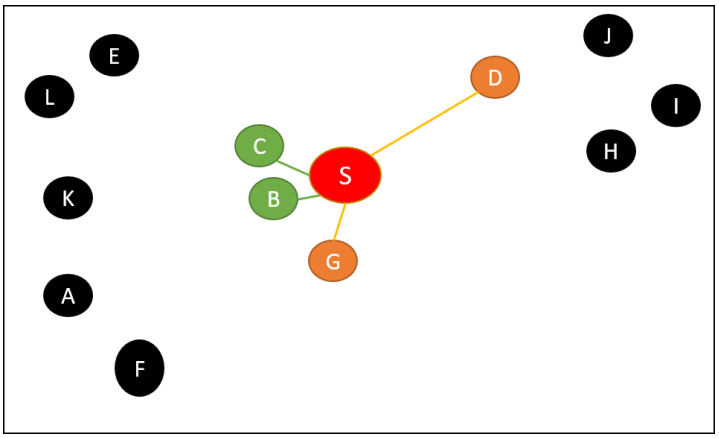
הערה: המילים הוספת קשת או מתיחת קשת מתייחסת להוספת הקשת למסלול ולא לגרף.

האלגוריתם:

1. בשלב ראשון נמצא את 2k הקודקודים הקרובים ביותר ל-S , נמתח אליהם קשתות ונשמור בקבוצה D
2. בקבוצה D נחפש בכל שלב 2 קודקודים קרובים ביותר שעדיין לא נבחרו, נבחר אותם ונסמן אותם כזוג, נפתח עבורם קבוצה חדשה ,כל קדקוד מהם יהווה קצה מסלול עבור הקבוצה. לאחר שכל הקודקודים התחלקו לקבוצות, נבדוק האם קיים קדקוד כלשהו u ב-D כך שהקודקוד הקרוב אליו ביותר מבין הקודקודים ב-D אינו הקודקוד שבקבוצה שלו.
3. אם נמצא קדקוד כזה, נמחק אותו מ-D , נחפש את הקודקוד הבא בגרף שהכי קרוב ל-S,נוסיף אותו לקבוצה D , נמחק את החלוקה שיצרנו ונחזור לשלב 2.
4. כל עוד קיימים קודקודים שחורים:
   - 4.1. נמצא את הקודקוד השחור u שמרחקו מאחד מקצות המסלול v הוא המינימלי והוא לא נמצא ב'רשימה השחורה' של הקבוצה של v .
   - 4.2. אם הוספת קדקוד u לקבוצה של v, תגרום לכך שלא תהיה משאית המסוגלת לספק סחורה לכל החנויות בקבוצה , נוסיף את u ל 'רשימה השחורה' של הקבוצה של v ונחזור לשלב 5.1.
   - 4.3. אם קדקוד הקצה הקרוב ביותר ל-u איננו שייך לקבוצה של קדקוד v נגדיר את קדקוד v כ 'קדקוד פסול' ונחזור לשלב 5.1.
   - 4.4. נמתח קשת (u,v), נעדכן את u כשייך לקבוצה של v וכקצה מסלול במקום v.
   - 4.5. נבטל את כל הגדרות ה 'קודקודים הפסולים'.
5. כעת קיבלנו 2k מסלולים המכסים את כל קודקודי הגרף, כאשר כל 2 קודקודים שייכים לקבוצה נפרדת, לכן נעבור על כל הקבוצות ונחבר את 2 המסלולים למסלול אחד כאשר אחד מהם 'יתהפך' כך שהמחסן ייהפך ליעד במקום למקור.

לדוגמא באיור להלן אנו מניחים שקיימות 2 משאיות ולכן מתחנו 4 קשתות לקודקודים B,C,D,G

כיוון שקודקודים B ו C הם הקרובים ביותר הם יהוו קבוצה אחת וקודקודים D ו-G יהוו קבוצה שניה (בהתאם לשלבים 1 ו-2).

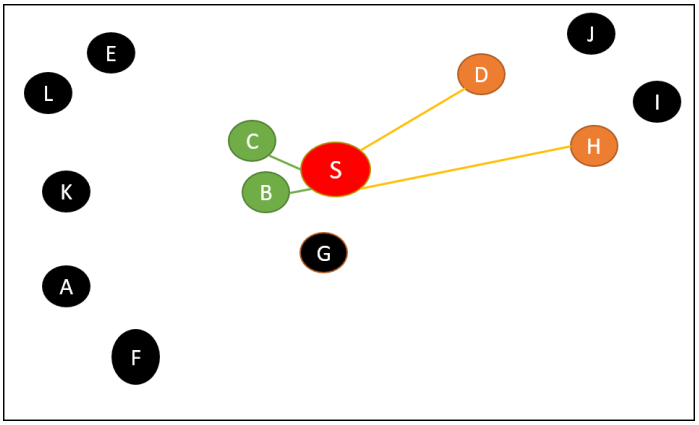


כעת נעבור (בהתאם לשלב 3) ונחפש קדקוד שהקודקוד הקרוב אליו ביותר מבין הקודקודים שנמצאו, לא שייך לקבוצה שלו.

קדקוד G עונה על התנאי כיוון שמבין קצות המסלול, קדקוד B הוא הקרוב אליו ביותר אבל קדקוד B לא נמצא באותה קבוצה יחד עם קדקוד G.

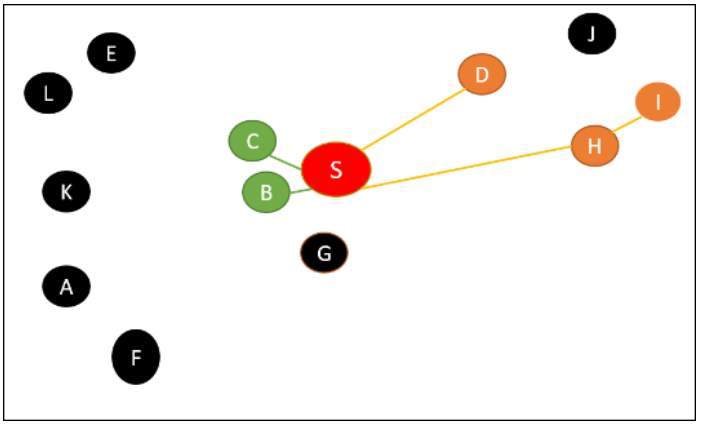
לכן נעבור לשלב 4 ונחפש את הקודקוד הבא בגרף שהכי קרוב ל-S, נוסיף אותו לקבוצה D , נמחק את החלוקה שיצרנו ונחזור לשלב 2.

לאחר שנבצע את שלב 2 (חלוקת הקודקודים שנבחרו לקבוצות) הגרף יראה כך:

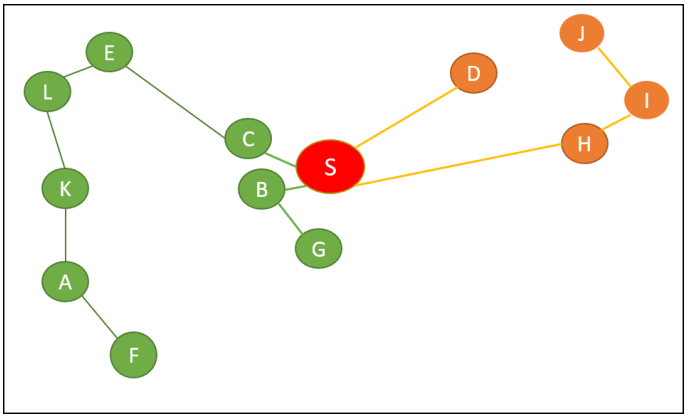


כעת התנאי בשלב 3 לא מתקיים ולכן נעבור לשלב 5 . נמצא קדקוד קרוב ביותר לאחד מקצות המסלול, נבדוק שהוא עומד בקיבולת המשאית ונצרף אותו למסלול.

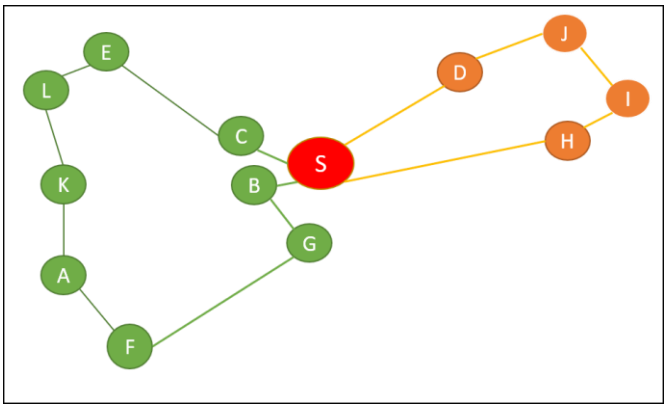
בדוגמא שלפנינו מצאנו את קדקוד I כקרוב ביותר כעת הגרף יראה כך:



נמשיך באותו אופן עד שלא יהיו קודקודים שחורים בגרף.



נחבר את תתי המסלולים בהתאם לשלב 6



קיבלנו K מסלולים מעגליים המכסים את כל צמתי הגרף.

יתרונות:

- זמן ריצה סביר $O(n^2)$
- קבלת פתרון קירוב אופטימלי

חסרונות:

- במקרים מסוימים הפתרון שיתקבל יהיה רחוק מאוד מפתרון אופטימלי
- האלגוריתם לא מתייחס לאילוצי הזמן.

המימוש בקוד:

מחלקת DataModelTruck

אחראית על שליפת נתוני המשאיות ממסד הנתונים

```
public class DataModelTruck : DataModelTruckInterface
    {
        //אובייקט המייצג את טבלת המשאיות במסד הנתונים
        ITrucksBLL truckBll;
        //אובייקט המייצג את טבלת הגבלות הקיבול של המשאיות במסד הנתונים
        ISizeContainsBLL SizeContainsBLL;
        //אובייקט המייצג את טבלת הספקים במסד הנתונים
        IproviderBLL providerBLL;
        //רשימת יחידות הסחורה שכל משאית מסוגלת לשאת
        public List<int> capacityTruks { get; set; }
        //רשימת המשאיות
        public List<TrucksDTO> truckSource { get; set; }
        //מספר המשאיות
        public int numOfTruck { get; set; }
        public DataModelTruck(ITrucksBLL trucksBLL, ISizeContainsBLL
sizeContainsBLL, IproviderBLL iproviderBLL)
        {
            this.truckBll = truckBll;
            this.SizeContainsBLL = sizeContainsBLL;
            this.providerBLL = providerBLL;
            numOfTruck = 2;
            capacityTruks = new List<int>();

        }

        // שולפת את נתוני המשאיות הרלוונטיים ממסד הנתונים
        public void InsertDatFromDBByProviderId(string providerId)
        {
            truckSource =
truckBll.GetAll().FindAll(t=>t.ProviderId==providerId);
            numOfTruck = truckSource.Count;
            //כמה סחורה נכנסת בכל ארגז
            int MaxGoodsInCrate =
providerBLL.GetById(providerId).MaxGoodsInCrate;
            //כמה ארגזים נכנסים במשאית בהתאם למגבלות המשקל והנפח
            int countOfCrateInTruck;
            ProviderDTO provider =providerBLL.GetById(providerId);

            //כמה סחורה נכנסת בכל משאית
            int maxCapacityWeigth;
            int maxCapacityVolume;
            //הגבלות הגודל של המשאית
            SizeContainDTO sizeContain;
            foreach (TrucksDTO truck in truckSource)
            {
                sizeContain = SizeContainsBLL.GetById(truck.SizeId);
```

```
                maxCapacityWeigth =
(int)(Math.Floor(sizeContain.WeightSize / provider.CrateWeight) *
MaxGoodsInCrate);
                maxCapacityVolume = sizeContain.VolumeSqmr /
provider.CrateVolume*MaxGoodsInCrate;

capacityTruks.Add(Math.Max(maxCapacityWeigth,maxCapacityVolume));
            }
        }
```

מחלקת DataModelVertex

אחראית על שליפת נתוני החנויות ממסד הנתונים ויצירת מטריצת מרחקים מתאימה

```
public class DataModelVertex:DataModelVertexInterface
    {
        //אובייקט המייצג את טבלת החנויות במסד הנתונים
        IShopBLL shopBLL;
        //אובייקט המייצג את טבלת הספקים במסד הנתונים
        IproviderBLL providerBll;
        //אובייקט המייצר מטריצת מרחקים
        DistanceMatrixInterface timeAndDistanceMatrix;

        public List<List<int>> distanceMatrix { get; set; }

        //רשימת המייצגת את יחידות הסחורה שכל חנות הזמינה
        public List<int> demondsShop { get; set; }

        //מספר החנויות
        public int numOfShops { get; set; }
        public DataModelVertex(IShopBLL shopBLL, IproviderBLL
providerBLL,DistanceMatrixInterface distanceMatrix)
        {
            this.shopBLL = shopBLL;
            this.providerBll = providerBLL;
            this.timeAndDistanceMatrix = distanceMatrix;

            demondsShop = new List<int>();
        }
        //שולפת את נתוני החנויות ממסד הנתונים
        public void InsertFormDBByProvider(string providerId)
        {
            List<ShopsDTO> shopsDTO = shopBLL.GetAll().FindAll(s =>
s.ProviderId == providerId);
            foreach (ShopsDTO shop in shopsDTO)
            {
                demondsShop.Add(shop.GoodsRequired);

            }
            numOfShops = shopsDTO.Count;
            //מערך המייצג את כתובות החנויות
            List<string> x = (from shop in shopsDTO select
shop.placeGoogleMapsId).ToList();
            //יצירת מטריצת מרחקים

timeAndDistanceMatrix.CreateTimeAndDistanceMatrix(providerBll.GetById(pr
oviderId).placeGoogleMapsId, x);
            distanceMatrix =
timeAndDistanceMatrix.GetTimeAndDistanceMatrix().distanceMatrix;
        }
```

```
    }
```

מחלקת Group

מייצגת קבוצת קודקודים המהווים 2 מסלולים בגרף.

```
public class Group
    {
        public Group(int id)
        {
            this.id = id;
            truckId = -1;
            vertexIdList1 = new List<int>();
            vertexIdList2 = new List<int>();
            blackList = new List<int>();
        }

        //קוד קבוצה ממוספר מ-0 ומעלה
        public int id { get; set; }

        // המשאית שתיקח את המסלול הזה (עשוי להשתנות במהלך הריצה
        public int truckId { get; set; }

        //מייצג את כמות הסחורה שהמשאית סוחבת כרגע
        //זה בעצם סכום הדרישות של כל החניות שבשתי הרשימות  של הקודקודים
        public int truckContents { get; set; }

        //רשימת הקודקודים בתת מסלול אחד כאשר הקודקוד האחרון ברשימה מהווה קצה מסלול
        //הקודקודים מאוחסנים ברשימה באופן שקודקוד 0 נמצא במקום ה-0 וכו
        public List<int> vertexIdList1 { get; set; }

        //רשימת הקודקודים בתת מסלול שני כאשר הקודקוד האחרון ברשימה מהווה קצה מסלול
        //הקודקודים מאוחסנים ברשימה באופן שקודקוד 0 נמצא במקום ה-0 וכו
        public List<int> vertexIdList2 { get; set; }

        // הרשימה השחורה של הקבוצה. מייצגת את הקודקודים שלא יכולים להצטרף לקבוצה בגלל מגבלות
קיבולת
        public List<int> blackList { get; set; }

        //פונקציה שמחזירה אמת האם הקודקוד שקיבלה הוא "קודקוד קצה" , אחרת מחזירה שקר.
        public bool IsEndOfTrack(int vertexId)
        {
            return vertexIdList1[vertexIdList1.Count - 1] == vertexId ||
vertexIdList2[vertexIdList2.Count - 1] == vertexId;
        }
    }
```

מחלקת Truck

מייצגת משאית

```
  public class Truck
    {
        //הקוד המקורי של המשאית, מספר הרישוי שלה
        public string truckIdSource { get; set; }
        //קוד שניתן כעת לצורך האלגוריתם
        public int id { get; set; }

        //קוד הקבוצה אליה משתייכת המשאית
        public int groupId { get; set; }
```

```
        //הקיבולת של המשאית
        public int capacity { get; set; }
    }
```

מחלקת Vertex

מייצגת צומת בגרף (חנות)

```
  public class Vertex
    {
        //הקוד של הקודקוד זה בעצם האינדקס שלו במטריצת המרחקים ובמטריצת הזמנים
        //כאשר האינדקס 0 שייך למחסן
        public int id { get; set; }
        //החנות אותה הקודקוד מייצג
        public int shpoId { get; set; }
        //כמות הסחורה שהחנות דורשת לקבל
        public int demand { get; set; }
        //קוד הקבוצה שהקודקוד משתייך אליה אם הוא עוד לא שייך לאף קבוצה הערך יהיה
        //groupId=-1
        public int groupId { get; set; }

        //האם הקודקוד כבר נבחר
        public bool selected { get; set; }

  }
```

מחלקת AllVertex

```
public class AllVertexInterface
    {
        public List<Vertex> vertexList { get; set; }
        public List<int> endOfTruckList { get; set; }
        public DataModelVertexInterface dataVertex { get; set; }
        public void InitAllVertex(string providerId);
        public void DivideVertexOfGroup(int newVertex);
        public int GetVertexMin();
        public bool CheckIfPerIsBestClosed(int vertexId, int newVertex);
        public int GetVertexNotGroup();
        private bool Find2VertexThatMinDistance(int groupId);
    }
```

פירוט הפונקציות

הפונקציה InitAllVertex

מאתחלת את נתוני הקודקודים

```
        public void InitAllVertex(string providerId)
        {
            //שליפת הנתונים מהמסד
            dataVertex.InsertFormDBByProvider(providerId);

            //מיון הקודקודים לפי הזמנות החנויות
            dataVertex.demondsShop.Sort();
            vertexList = new List<Vertex>();

            //הוספת המחסן לרשימת הקודקודים
            vertexList.Add(new Vertex(0, 0, 0));

            //הוספת שאר הקודקודים לגרף
```

```
            for (int i = 1; i < dataVertex.demondsShop.Count; i++)
            {
                vertexList.Add(new Vertex(i, i, dataVertex.demondsShop[i]));
            }
        }
```

הפונקציה GetVertexMin

מביאה קודקוד שמרחקו מהמחסן הוא הכי קטן שלא נבחר עדיין

```
        public int GetVertexMin()
        {
            int min = Int16.MaxValue;
            int vertexMin = 0;
            for (int i = 1; i < vertexList.Count; i++)
            {
                if (!vertexList[i].selected && dataVertex.distanceMatrix[0][i]
< min)
                {
                    min = dataVertex.distanceMatrix[0][i];
                    vertexMin = i;
                }
            }
            vertexList[vertexMin].selected = true;
            return vertexMin;
        }
```

הפונקציה Find2VertexThatMinDistance

מוצאת זוג קודקודים קרובים ביותר מתוך רשימת קצות המסלול

שעדיין לא שייכים לקבוצה מסוימת .

```
        private bool Find2VertexThatMinDistance(int groupId)
        {
            int minDistance = Int16.MaxValue;
            int vertexSource = 0;
            int vertexDest = 0;
            foreach (int i in endOfTruckList)
            {


                if (vertexList[i].groupId == -1)
                {
                    foreach (int j in endOfTruckList)
                    {

                        if (vertexList[j].groupId == -1 &&
dataVertex.distanceMatrix[i][j] < minDistance && i != j)
                        {
                            minDistance = dataVertex.distanceMatrix[i][j];
                            vertexSource = i;
                            vertexDest = j;
                        }
                    }

                }

            }
            if (minDistance == Int16.MaxValue)
                return false;
```

```
            vertexList[vertexSource].groupId = groupId;
            vertexList[vertexDest].groupId = groupId;



            return true;

        }
```

הפונקציה CheckIfPerIsBestClosed

הפונקציה בודקת עבור קדקוד מסוים האם הקודקוד הקרוב אליו ביותר מבין קצות המסלול, לא נמצא בקבוצה שלו.

```
        public bool CheckIfPerIsBestClosed(int vertexId, int newVertex)
        {
            int per = endOfTruckList.FirstOrDefault(v => vertexList[v].groupId
== vertexList[vertexId].groupId && vertexId != v);
            foreach (int someVertex in endOfTruckList)
            {
                if (dataVertex.distanceMatrix[someVertex][newVertex] <
dataVertex.distanceMatrix[per][newVertex] && someVertex != vertexId &&
someVertex != vertexId)
                    return true;
            }
            return false;
        }
```

הפונקציה GetVertexNotGroup

מחזירה קודקוד שלא שייך לשום קבוצה אם אין קודקוד כזה, תחזיר 1-

```
        public int GetVertexNotGroup()
        {
            Vertex ver = vertexList.FirstOrDefault(v => v.groupId == -1 && v.id
!= 0);
            if (ver == null)
                return -1;
            return ver.id;
        }
```

הפונקציה DivideVertexOfGroup

חלוקת הקודקודים שנבחרו לקבוצות של 2,2

```
        public void DivideVertexOfGroup(int newVertex)
        {
            int count = 0;
            //עבור כל קבוצה
            //מחפש 2 קודקודים קרובים ביותר
            //שעדיין לא שייכים לשום קבוצה ומעדכן אותם כשייכים לקבוצה
            while (Find2VertexThatMinDistance(count++)) ;
            foreach (int i in endOfTruckList)
            {
                //- האם הקדקוד הקרוב ביותר אליו מבין קצות המסלול
                //לא נמצא בקבוצה שלו
                if (CheckIfPerIsBestClosed(i, i))
                {
                    //בדיקה האם לא מוסר קודקוד שעכשיו נוסף(כדי למנוע מצב של קיפאון)
                    if (i != newVertex)
                    {
```

```
                        vertexList[i].groupId = -1;
                        endOfTruckList.Remove(i);
                        newVertex = GetVertexMin();
                        endOfTruckList.Add(newVertex);
                        vertexList[i].selected = false;
                    }
                    else
                    {
                        //מצא את הקודקוד הקרוב ביותר ברשימת קצות המסלול
                        int distuncevertexMin = endOfTruckList.FindAll((v1) =>
v1 != i).Min((v1) => dataVertex.distanceMatrix[i][v1]);
                        int vertexMin = endOfTruckList.FirstOrDefault((v) =>
dataVertex.distanceMatrix[i][v] == distuncevertexMin);

                        endOfTruckList.Remove(vertexMin);
                        newVertex = GetVertexMin();
                        endOfTruckList.Add(newVertex);
                        vertexList[vertexMin].selected = false;
                        vertexList[vertexMin].groupId = -1;
                    }
                    //מחיקת החלוקה לקבוצות
                    foreach (int v in endOfTruckList)
                    {
                        vertexList[v].groupId = -1;
                    }

                    //חלק לקבוצות מחדש
                    DivideVertexOfGroup(newVertex);
                    return;

                }
            }
        }
```

מחלקת GroupsAndTruck

```
   public class GroupsAndTruck:GroupsAndTruckInterface
    {
        //רשימת הגבלות קיבולת של המשאיות
        public DataModelTruckInterface dataTruck { get; set; }
        //שומר משאית כלשהי שהקיבולת שלה היא הגדולה ביותר
        public Truck truckByMaxCapacity { get; set; }

        //רשימת הקבוצות -כמספר המשאיות
        public List<Group> groupList { get; set; }

        //רשימת המשאיות
        public List<Truck> truckList { get; set; }
        public GroupsAndTruck(DataModelTruckInterface dataTruck)
        {
            this.dataTruck = dataTruck;
        }

        //
        public void Init(string providerId)
        {
            dataTruck.InsertDatFromDBByProviderId(providerId);
            fillTruckList();
            int maxCapacity = truckList.Max(t => t.capacity);
```

```
            truckByMaxCapacity = truckList.FirstOrDefault(t => t.capacity ==
maxCapacity);
        }
        //ממלא את רשימת המשאיות בהתאם לנתונים ב-capacityTruks
        private void fillTruckList()
        {

            truckList = new List<Truck>();
            for (int i = 0; i < dataTruck.capacityTruks.Count; i++)
            {
                truckList.Add(new Truck(i, dataTruck.capacityTruks[i]));
            }
        }
        public bool CheckCapcity(int demand, int groupId)
        {
            if (groupList[groupId].truckContents + demand <=
truckList[groupList[groupId].truckId].capacity)
                return true;
            return DivideTrucksToGroup();
        }
        //מחלקת את המשאיות לקבוצות כל משאית לקבוצה
        public bool DivideTrucksToGroup()
        {
            List<Group> l = new List<Group>(groupList);

            l.Sort((g1, g2) => g1.truckContents - g2.truckContents);
            for (int i = 0; i < dataTruck.capacityTruks.Count - 1; i++)
            {
                if (l[i].truckContents > truckList[i].capacity)
                    return false;

            }
            for (int i = 0; i < dataTruck.capacityTruks.Count; i++)
            {

                truckList[i].groupId = l[i].id;
                groupList[l[i].id].truckId = i;
            }
            return true;
        }
        //(כי המשאיות לא הספיקו) במקרה שנוספה משאית דמה
        //    הוספתי כל פעם את המשאית שהקיבולת שלה היא הגדולה ביותר
        // כעת נרצה לבדוק עבור כל משאית דמה כמה היא מכילה בפועל ולתת לה את המשאית הקטנה ביותר
המתאימה עבורה
        //כך שהמסלולים הכפולים יתחלקו באופן שווה בין המשאיות
       public void ChangeTruck()
        {

            foreach (var item in
truckList.FindAll(v=>v.truckIdSource==truckByMaxCapacity.truckIdSource))
            {
                foreach (var truck in truckList)
                {

                }
            }
        }
        //מאחדת את 2 תת המסלולים למסלול אחד סופי
        //ניתן להוסיף התחשבות בזמנים
        public void ConnectGroup()
        {
```

```
            foreach (Group group in groupList)
            {
                if (group.vertexIdList1 == null)
                    continue;
                group.vertexIdList2.Reverse();
                group.vertexIdList1.AddRange(group.vertexIdList2);
                group.vertexIdList2 = null;
                //מצא את המשאית השייכת לקבוצה הנ"ל//
                Truck currentTruck = truckList.FirstOrDefault(t => t.groupId ==
group.id);
                //בדוק האם המשאית הזאת קיימת פעמיים או יותר ברשימת המשאיות//
                Truck truck2= truckList.FirstOrDefault(t => t.truckIdSource ==
currentTruck.truckIdSource&&t.id!=currentTruck.id);
                while(truck2!=null)
                {
                    Group g = groupList[truck2.groupId];
                    g.vertexIdList2.Reverse();
                    g.vertexIdList1.AddRange(group.vertexIdList2);
                    g.vertexIdList2 = null;
                    group.vertexIdList1.AddRange(g.vertexIdList1);
                    g.vertexIdList1 = null;
                    truckList.Remove(truck2);
                    truck2 = truckList.FirstOrDefault(t => t.truckIdSource ==
currentTruck.truckIdSource && t.id != currentTruck.id);
                }
            }
        }
        //הוספת משאית למקרה שהקיבולת לא מספיקה//
        public void AddTruck()
        {

            Truck newTruck = new Truck(truckList.Count,
truckByMaxCapacity.capacity);
            newTruck.truckIdSource = truckByMaxCapacity.truckIdSource;
            newTruck.groupId = -1;

            truckList.Add(newTruck);
        }

    }
```

מחלקת Maneger

```
public class Maneger:ManegerInterface
    {

        //עבור איזה ספק מתבצע החישוב הנ"ל//
        public string providerId { get; set; }
        //רשימת הקבוצות והמשאיות שלהן//
        public GroupsAndTruckInterface groupsAndTruck { get; set; }

        //רשימת הקדקודים//
        public AllVertexInterface allVertex { get; set; }

        //רשימת התחנות כדי להכניס את הנתונים בסיום החישוב//
        IStationsInShopDAL stationsInShopDAL;



        public Maneger(IStationsInShopDAL stationsInShopDAL, AllVertexInterface
allVertex, GroupsAndTruckInterface groupsAndTruck)
        {
```

```
            this.stationsInShopDAL = stationsInShopDAL;
            this.allVertex = allVertex;
            this.groupsAndTruck = groupsAndTruck;

        }




        public bool CumputeByProvider(string providerId)
        {

            try
            {
                this.providerId = providerId;
                allVertex.InitAllVertex(providerId);
                groupsAndTruck.Init(providerId);
                Start();
                while (allVertex.GetVertexNotGroup() != -1)
                {  //לטפל המקרה קצה שבו כל הקודקודים פסולים
                    AddVertexToGroup(new List<bool>(new
bool[allVertex.vertexList.Count]), 1, false);
                }
                groupsAndTruck.ConnectGroup();
                InsertSolutionToDB();
                return true;
            }
            catch (Exception ex)
            {
                return false;
            }



        }
        //מוסיפה קדקוד למסלול
        // שומר כמה פעמים זימנתי את הפונקציה count
        private int AddVertexToGroup(List<bool> vertxeIsFiled, int count, bool
isAllvertxFailed)
        {

            int vertexClosed;
            int distanceClose;
            int minVertexClosed = 0;
            int minDistanceClose = Int16.MaxValue;
            int sorceVertexMin = 0;
            if (count == allVertex.vertexList.Count)
                return 0;
            // =;
            //list<int> vertexclosedlist = new list<int>(data.numofshops+1);
            //vertexclosedlist.add(int16.maxvalue);
            //  ללולאה שעוברת כל כל הצמתים ומוצאת את הצומת הקרובה ביותר לכל צומת שמהווה קצה
מסלול
            //0 אם לא נמצא קדקוד להוסיף תחזיר
            foreach (var i in allVertex.endOfTruckList)
            {
                //אם הקדקוד הנוכחי הוא קצה מסלול והוא לא קודקוד פסול
                if (vertxeIsFiled[i] == false)
                {
                    var results = FindVertexClosed(i);
```

```
                    vertexClosed = results.Item1;
                    distanceClose = results.Item2;
                    if (distanceClose < minDistanceClose)
                    {
                        minDistanceClose = distanceClose;
                        minVertexClosed = vertexClosed;
                        sorceVertexMin = i;
                    }
                }


            }
            if (minVertexClosed == Int16.MaxValue)
                //לטפל המקרה קצה שבו כל הקודקודים פסולים
                return AddVertexToGroup(new List<bool>(new
bool[allVertex.vertexList.Count]), count + 1, true);
            allVertex.vertexList[minVertexClosed].groupId =
allVertex.vertexList[sorceVertexMin].groupId;
            if (!isAllvertxFailed &&
allVertex.CheckIfPerIsBestClosed(sorceVertexMin, minVertexClosed))
            {
                vertxeIsFiled[sorceVertexMin] = true;
                allVertex.vertexList[minVertexClosed].groupId = -1;
                return AddVertexToGroup(vertxeIsFiled, count + 1, false);

            }
            else
            {
                if
(groupsAndTruck.CheckCapcity(allVertex.vertexList[minVertexClosed].demand,
allVertex.vertexList[sorceVertexMin].groupId))
                {
                    if
(groupsAndTruck.groupList[allVertex.vertexList[sorceVertexMin].groupId].vertexI
dList1.FirstOrDefault(v => v == sorceVertexMin) != 0)

groupsAndTruck.groupList[allVertex.vertexList[sorceVertexMin].groupId].vertexId
List1.Add(minVertexClosed);
                    else

groupsAndTruck.groupList[allVertex.vertexList[sorceVertexMin].groupId].vertexId
List2.Add(minVertexClosed);

                    allVertex.endOfTruckList.Remove(sorceVertexMin);
                    allVertex.endOfTruckList.Add(minVertexClosed);
                    return 1;
                }
                else
                {

groupsAndTruck.groupList[allVertex.vertexList[sorceVertexMin].groupId].blackLis
t.Add(sorceVertexMin);
                    allVertex.vertexList[minVertexClosed].groupId = -1;
                    return AddVertexToGroup(vertxeIsFiled, count + 1, false);
                }
            }
        }
        //בודק האם יש משאית מתאימה שתספיק לכל התכולה שבקבוצה יחד עם הדרישה הנוספת
```

```
        // מוצאת קדקוד קרוב ביותר לקצה המסלול שנשלח שעדיין לא שייך לשום קבוצה ולא נמצא ברשימה
השחורה
        // של הקבוצה של קצה המסלול הנשלח
        //0 אם אין קודקוד כזה מחזירה
        private (int, int) FindVertexClosed(int vertexId)
        {
            int minDistance = Int16.MaxValue;
            int vertexClosed = 0;



            for (int i = 1; i < allVertex.vertexList.Count; i++)
            {
                if (allVertex.vertexList[i].groupId == -1 &&
allVertex.dataVertex.distanceMatrix[i][vertexId] < minDistance)
                    if
(groupsAndTruck.groupList[allVertex.vertexList[vertexId].groupId].blackList.Fir
stOrDefault(v => v == i) == 0)
                    {
                        minDistance =
allVertex.dataVertex.distanceMatrix[i][vertexId];
                        vertexClosed = i;
                    }
            }
            return (vertexClosed, minDistance);

        }


        //הפונקציה מוצאת קודקודים קרובים למחסן ומשייכת כל 2 קודקודים לקבוצה מסוימת
        private void Start()
        {
            //ישמור את קבוצת הקודקודים העתידים להוות קצה מסלול
            allVertex.endOfTruckList = new List<int>();

            int i;
            //פתיחת המסלולים
            for (i = 0; i < groupsAndTruck.dataTruck.numOfTruck * 2; i++)
            {
                allVertex.endOfTruckList.Add(allVertex.GetVertexMin());
            }
            allVertex.DivideVertexOfGroup(0);

            //נסה לאתחל משאית עבור כל קבוצה
            //אם לא הצלחת
            if (!InitGroup())
            {
                //(חלק מהמשאיות יבצעו מספר מסלולים) הוסף משאית דמה למשאיות
                groupsAndTruck.AddTruck();
                Start();
            }


        }
```

```
        // מאתחלת את הקבוצות לפי הקודקודים שנבחרו לכל קבוצה ומתאימה משאית בעלת קיבולת מספקת
לכל קבוצה
        //להוסיף מקרי קצה כאשר אין משאית
        private bool InitGroup()
        {
            groupsAndTruck.groupList = new List<Group>();
            Group group;
            foreach (int item in allVertex.endOfTruckList)
            {
                //מציאת הקבוצה אליה הקודקוד שייך
                group = groupsAndTruck.groupList.FirstOrDefault(g => g.id ==
allVertex.vertexList[item].groupId);

                //אם הקבוצה שהוא שייך אליה לא קיימת
                if (group == null)
                {
                    //צור קבוצה חדשה
                    group = new Group(allVertex.vertexList[item].groupId);

                    //הוסף אותו לרשימת הקודקודים הראשונה של הקבוצה
                    group.vertexIdList1.Add(item);
                    group.truckContents = allVertex.vertexList[item].demand;
                    groupsAndTruck.groupList.Add(group);
                }
                //אם הקבוצה שהוא שייך אליה כבר קיימת
                else
                {
                    group.vertexIdList2.Add(item);

                    //הוסף את הקיבולת לתכולת המשאית
                    group.truckContents += allVertex.vertexList[item].demand;
                }

            }

            //למיין את רשימת הקבוצות לפי הקוד שלהם
            //כך שהקבוצה ה-0 תהיה במקום ה-0 במערך
            groupsAndTruck.groupList.Sort((g1, g2) => g1.id - g2.id);
            //מחלק את המשאיות בין הקבוצות

            //אם הפונקציה מחזירה שקר
            //קרא לפונקציה מקרי קצה
            if (!groupsAndTruck.DivideTrucksToGroup()) { return false; }
            return true;

        }
        //מכניסה את הנתונים הסופיים לטבלת התחנות
        private void InsertSolutionToDB()
        {
            int count = 0;
            foreach (Group group in groupsAndTruck.groupList)
            {
                count = 0;
                foreach (int shop in group.vertexIdList1)
                {
                    StationsInShopDTO station = new StationsInShopDTO();
                    station.ProviderId = providerId;
                    Truck t = groupsAndTruck.truckList.FirstOrDefault(t => t.id
== group.truckId);
                    if (t != null)
                        station.LicensePlateTruck = t.truckIdSource;
```

```
                    station.ShopId = allVertex.vertexList[shop].shpoId;
                    station.StationSerialNumber = count;
                    stationsInShopDAL.Add(station);
                    count++;
                }
            }

        }


    }
```

הפונקציה CumputeByProvider

מבצעת את החישוב עבור ספק מסוים ושומרת אותו במסד

```
public bool CumputeByProvider(string providerId)
        {

            try
            {
                this.providerId = providerId;
                allVertex.InitAllVertex(providerId);
                groupsAndTruck.Init(providerId);
                Start();
                while (allVertex.GetVertexNotGroup() != -1)
                {
                    AddVertexToGroup(new List<bool>(new
bool[allVertex.vertexList.Count]), 1, false);
                }
                groupsAndTruck.ConnectGroup();
                InsertSolutionToDB();
                return true;
            }
            catch (Exception ex)
            {
                return false;
            }



        }

            for (int i = 1; i < dataVertex.demondsShop.Count; i++)
            {
                vertexList.Add(new Vertex(i, i, dataVertex.demondsShop[i]));
            }
        }
```

כיוון שהאלגוריתם עדיין לא מושלם, הפרויקט משתמש בספרייה ייעודית לחישוב המסלולים האופטימליים.

OR-Tools

OR-Tools היא תוכנת קוד פתוח *לאופטימיזציה קומבינטורית* ,המבקשת למצוא את הפתרון הטוב ביותר לבעיה מתוך מערך גדול מאוד של פתרונות אפשריים .הנה כמה דוגמאות לבעיות ש OR-Tools-פותר:

- ניתוב רכב: מצא מסלולים אופטימליים עבור ציי רכב שאוספים ומחלקים חבילות בהינתן אילוצים (למשל, "משאית זו לא יכולה להחזיק יותר מ-20,000 פאונד" או "כל המשלוחים חייבים להתבצע תוך חלון של שעתיים").
- תזמון: מצא את לוח הזמנים האופטימלי עבור קבוצה מורכבת של משימות, שחלקן צריכות להתבצע לפני אחרות, על קבוצה קבועה של מכונות או משאבים אחרים.
- אריזת פחים: ארוז כמה שיותר חפצים בגדלים שונים למספר קבוע של פחים בעלי קיבולת מקסימלית.

ברוב המקרים, לבעיות כמו אלה יש מספר עצום של פתרונות אפשריים - רבים מדי מכדי שמחשב יוכל לחפש בכולם .כדי להתגבר על כך OR-Tools, משתמשת באלגוריתמים חדישים על מנת לצמצם את מערך החיפוש, על מנת למצוא פתרון אופטימלי (או קרוב לאופטימלי).

איך זה נראה בפועל?

שלב ראשון- בניית אובייקט הנתונים DataModel

לשם בניית הגרף עבור חישוב המסלול, יש צורך במספר רב של פרמטרים.

מחלקת dataModel בונה את הפרמטרים הדרושים בהתאם לנתונים, ומכילה בתוכה את כל הפרמטרים הנדרשים לפותר של orTools .

הנתונים כוללים :

מטריצת זמנים-הזמן הנדרש להגעה מכל חנות לכל חנות וכן למחסן.

מטריצת חלונות זמן- של החנויות.

מערך הדרישות של החנויות-מספר יחידות הסחורה המוזמנות מכל חנות.

משתני המחלקה

```
public class DataModel:IDataModel
    {
        //מטריצת מרחקים

        public long[,] DistanceMatrix { get; set; }
        //מטריצת זמנים
```

```
        public long[,] TimeMatrix { get; set; }
        //מטריצת חלונות זמן
        public long[,] TimeWindows { get; set; }
        //מספר המשאיות המשתתפות בהובלה
        public int VehicleNumber { get; set; }
        //אינדקס המחסן במטריצות
        public int Depot { get; set; }
        //מערך הדרישות של החנויות
        public long[] Demands { get; set; }
        public List<TrucksDTO> truckSource { get; set; }
        public List<ShopsDTO> shopsDTO { get; set; }
        //מערך הקיבולות של המשאיות
        public long[] VehicleCapacities { get; set; }
        private ProviderDTO provider;
        private IShopDAL shopDAL;
        private DistanceMatrixInterface timeAndDistanceMatrix;
        private IProviderDL providerDAL;

        private ITrucksDAL trucksDAL;
        private ISizeContainDAL sizeContainsDAL;
```

אתחול המחלקה

```
        public void Init(string providerId)
        {
            provider = providerDAL.GetById(providerId);
            if (provider == null)
                return;
            Depot = 0;
            InitShops();
            InitMatrix();
            InitTimeWindows();
            InitTrucks();
        }
```

הכנסת נתוני החנויות ממסד הנתונים

```
        private void InitShops()
        {
            //מילוי מערך הדרישות של החנויות
           shopsDTO = shopDAL.GetAll().FindAll(s => s.ProviderId ==
provider.ProviderId);
            Demands = new long[shopsDTO.Count+1];
            Demands[0] = 0;
            int i = 1;
            foreach (ShopsDTO shop in shopsDTO)
            {
                Demands[i++] = shop.GoodsRequired;
            }
        }
```

יצירת מטריצת זמנים מתאימה

```
        private void InitMatrix()
        {
            //מילוי מטריצת מרחקים ומטריצת זמנים
            List<string> x = (from shop in shopsDTO select
shop.placeGoogleMapsId).ToList();//רשימת הכתובות של החניות

timeAndDistanceMatrix.CreateTimeAndDistanceMatrix(provider.placeGoogleMapsId,
x);//google mpasיצירת המטריצות דרך
האוביקט
timeAndDistanceMatrix.GetTimeAndDistanceMatrix().durationMatrix
//list<list<int>> מחזיר
            //or-tools כדי שיתאים לספרייה  long[,] המטרה היא להמיר אותו ל-
```

```
            List<List<int>> timeMatrixInt =
timeAndDistanceMatrix.GetTimeAndDistanceMatrix().durationMatrix;
            TimeMatrix = new long[timeMatrixInt.Count, timeMatrixInt[0].Count];
            DistanceMatrix = new long[timeMatrixInt.Count,
timeMatrixInt[0].Count];
            for (int j = 0; j < TimeMatrix.GetLength(0); j++)
            {
                for (int k = 0; k < TimeMatrix.GetLength(1); k++)
                {
                    TimeMatrix[j, k] = Convert.ToInt64(timeMatrixInt[j][k]);
                    DistanceMatrix[j, k] =
Convert.ToInt64(timeMatrixInt[j][k]);
                }
            }
        }
```

הגדרת חלונות הזמן

```
      private void InitTimeWindows()
        {
            //מילוי מטריצת חלונות הזמן
            //חלונות הזמן דורשים מימד מסוים
            //אנחנו נשתמש במימד של שניות שעת היציאה תהיה מאותחלת ל-0//
            //ונבדוק עבור כל שעה את מספר השניות משעת היציאה ועד אליה//
            TimeWindows = new long[shopsDTO.Count + 1, 2];
            TimeSpan timeStart = provider.LeavingTime;
            ////אתחול המחסנב-0 כזמן התחלה וזמן סיום לא מוגבל


            TimeWindows[0, 0] = Convert.ToInt64(timeStart.TotalSeconds);
            TimeWindows[0, 1] = long.MaxValue;
            int i = 1;
            foreach (ShopsDTO shop in shopsDTO)
            {

                TimeWindows[i, 0] =
Convert.ToInt64(((TimeSpan)shop.HourDayStart).TotalSeconds);
                TimeWindows[i++, 1] =
Convert.ToInt64(((TimeSpan)shop.HourDayFinish).TotalSeconds);
            }
        }
```

הכנסת נתוני המשאיות ממסד הנתונים

```
        private void InitTrucks()
        {
            truckSource = trucksDAL.GetAll().FindAll(t => t.ProviderId ==
provider.ProviderId);
            VehicleNumber = truckSource.Count;
            //כמה סחורה נכנסת בכל ארגז
            int MaxGoodsInCrate = provider.MaxGoodsInCrate;
            //כמה ארגזים נכנסים במשאית בהתאם למגבלות המשקל והנפח
            int countOfCrateInTruck;
            //כמה סחורה נכנסת בכל משאית
            int capacity;
            int maxCapacityWeigth;
            int maxCapacityVolume;
            //הגבלות הגודל של המשאית
            SizeContainDTO sizeContain;
            VehicleCapacities = new long[truckSource.Count];
            int i = 0;
            foreach (TrucksDTO truck in truckSource)
            {
                sizeContain = sizeContainsDAL.GetById(truck.SizeId);
```

```
                maxCapacityWeigth = (int)(Math.Floor(sizeContain.WeightSize /
provider.CrateWeight) * MaxGoodsInCrate);
                maxCapacityVolume = sizeContain.VolumeSqmr /
provider.CrateVolume * MaxGoodsInCrate;
                VehicleCapacities[i++] = Math.Min(maxCapacityWeigth,
maxCapacityVolume);
            }
        }
    }
```

שלב שני- בניית הגרף

מחלקת Find Way

מחלקה זו, מהווה את היסוד למערכת כולה. מופע ממחלקה זו, יכיל את תוצאות השיבוץ והמסלול כולו .

מחלקה זו מכילה גם כן, אובייקט מסוג מחלקת dataModel המכיל את אילוצי הרכבים, ההזמנות ואת מטריצת המרחקים המחושבת .

הפותר של tools-or מכיל בתוכו מספר מחלקות חשובות:

routingIndexManager- מחלקה הכוללת את כל נקודות המסלול: נקודת התחלה, סיום וכן את יתר נקודות העצירה.

RoutingModel -מחלקה זו מכילה עצם מהמחלקה routingIndexManager ואת הנתונים אודות משקל הקשתות אילוצי הזמן, קיבולת הרכבים וכו'

לאחר מכן הפותר מחשב את המסלול האופטימאלי,

משך הזמן של מציאת המסלול האופטימלי הוגבל-10 שניות, לאחר שנמצא שבפרק זמן זה הפותר מצליח למצוא פתרון, אם הוא קיים.

```
private RoutingIndexManager manager;
        private RoutingModel routing;
        private IDataModel dataModel;
        private IStationsInShopDAL stationsInShopDAL;
        int transitCallbackIndex;
        string providerId;
        public void CalculateRoute(string providerId)
        {
            this.providerId = providerId;
            dataModel.Init(providerId);
            //פונקציה לחישוב מסלול
            manager = new RoutingIndexManager(dataModel.
           TimeMatrix.GetLength(0),//מספר החנויות הכולל
           dataModel.VehicleNumber,//מספר המשאיות
           dataModel.Depot);
            routing = new RoutingModel(manager);
            //יצירת קשתות עם עלות של המרחק מנקודה לנקודה
            transitCallbackIndex = routing.RegisterTransitCallback(
            (long fromIndex, long toIndex) =>
            {
```

```
                var fromNode = manager.IndexToNode(fromIndex);
                var toNode = manager.IndexToNode(toIndex);
                return dataModel.TimeMatrix[fromNode, toNode];
            });
            //הוספת עלות הזמן
            routing.SetArcCostEvaluatorOfAllVehicles(transitCallbackIndex);

            CVRP(); //הוספת מימד קיבולת על עלות הקשתות
            TW();//הוספת אילוצי חלונות זמן

            RoutingSearchParameters searchParameters =

operations_research_constraint_solver.DefaultRoutingSearchParameters();
            searchParameters.FirstSolutionStrategy =
            FirstSolutionStrategy.Types.Value.PathCheapestArc;
            //מגביל את זמן החישוב של הפותר כדי שלא נגיע לסיבוכיות גבוהה במקרה בו אין פתרון
            searchParameters.TimeLimit = new Duration { Seconds = 10 };
            Assignment solution =
routing.SolveWithParameters(searchParameters);
             SaveSolution(routing, manager, solution);
        }
```

הפונקציה CVRP -מוסיפה את מימד הקיבולת לקשתות

```
private void CVRP()
        {
            //הוספת אילוצי נפח ומשקל
            long x = routing.Size();
            int demandWeightCallbackIndex =
routing.RegisterUnaryTransitCallback(
            (long fromIndex) =>
            {
                var fromNode = manager.IndexToNode(fromIndex);
                return dataModel.Demands[fromNode];
            }
            );
            routing.AddDimensionWithVehicleCapacity(
            demandWeightCallbackIndex, 0, // null capacity slack
            dataModel.VehicleCapacities, // vehicle maximum capacities
            false, // start cumul to zero
            "Capacity"); ;
            //שם עלות על הקשתות לאלוצי נפח//
            int demandVolumeCallbackIndex =
routing.RegisterUnaryTransitCallback(
            (long fromIndex) =>
            {
                // Convert from routing variable Index to demand NodeIndex.
                var fromNode = manager.IndexToNode(fromIndex);
                return dataModel.Demands[fromNode];
            }
            );
            routing.AddDimensionWithVehicleCapacity(
            demandVolumeCallbackIndex, 0, // null capacity slack
            dataModel.VehicleCapacities, // vehicle maximum capacities
            true, // start cumul to zero
            "Capacity");
        }
```

הפונקציה TW -מוסיפה אילוצי חלונות זמן

```
private void TW()
        {
```

```
            //הוספת אילוצי חלונות זמן//
            routing.AddDimension(
            transitCallbackIndex, // transit callback
            500, // allow waiting time
            100000, // vehicle maximum capacitiess
            false, // 0-מ האם להתחיל בכל מקום
            "Time");
            RoutingDimension timeDimension =
routing.GetMutableDimension("Time");
            // הוסף אילוצי חלון זמן עבור כל מיקום מלבד המחסן
            for (int i = 1; i < dataModel.TimeWindows.GetLength(0); ++i)
            {
                long index = manager.NodeToIndex(i);
                timeDimension.CumulVar(index).SetRange(
                dataModel.TimeWindows[i, 0],//זמן התחלה
                dataModel.TimeWindows[i, 1]);//זמן סיום
            }
            //(זמן התחלה וסיום של כל משאית) הוסף את אילוצי חלון הזמן של המחסן
            for (int i = 0; i < dataModel.VehicleNumber; ++i)
            {
                long index = routing.Start(i);
                timeDimension.CumulVar(index).SetRange(
                dataModel.TimeWindows[0, 0],
                dataModel.TimeWindows[0, 1]);
            }
            // להכניס לבעיה את נתוני חלונות הזמן שהכנסנו כדי למצוא פתאון העונה עליהם
            for (int i = 0; i < dataModel.VehicleNumber; ++i)
            {
                routing.AddVariableMinimizedByFinalizer(
                timeDimension.CumulVar(routing.Start(i)));
                routing.AddVariableMinimizedByFinalizer(
                timeDimension.CumulVar(routing.End(i)));
            }
        }
```

שלב שלישי -שמירת תוצאות החישוב

קבלת תוצאות החישוב מהספרייה ועדכון טבלת התחנות כך שתייצג את תוצאות החישוב הנוכחי.

פונקציית SaveSolution

מחלצת את תוצאות המסלול מקובץ json שהפותר יצר .

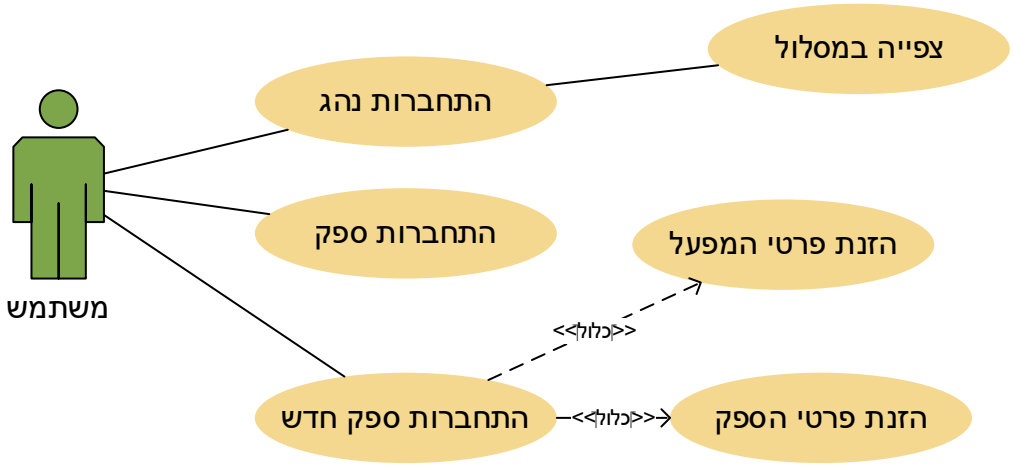
הנתונים נשמרים כרשימה לא מסודרת, ולכן יש צורך למצוא כל את המיקום של כל נקודה באמצעות הפונקציה indexToNode . תוצאות השיבוץ המחושב ישמרו בטבלת התחנות.

```
private void SaveSolution(RoutingModel routing, RoutingIndexManager manager,
Assignment solution)
        {

            stationsInShopDAL.DeleteAllByProviderId(providerId);
            if (solution == null)
                return;

            int StationSerialNumber;

            RoutingDimension timeDimension =
routing.GetMutableDimension("Time");
```

```
            long totalTime = 1;
            for (int i = 0; i < dataModel.VehicleNumber; ++i)
            {

                //האינדקס של המסלול של המשאית הנוכחית
                var index = routing.Start(i);
                StationSerialNumber = 0;
                while (routing.IsEnd(index) == false)
                {

                        var timeVar = timeDimension.CumulVar(index);
                    if (manager.IndexToNode(index) != 0)
                    {
                        StationsInShopDTO newStation = new StationsInShopDTO();
                        newStation.ProviderId = providerId;
                        newStation.LicensePlateTruck =
dataModel.truckSource[i].LicensePlate;

                        newStation.ShopId =
dataModel.shopsDTO[manager.IndexToNode(index) - 1].ShopId;
                        newStation.TimeStart =
TimeSpan.FromSeconds(solution.Min(timeVar));
                        newStation.TimeFinish =
TimeSpan.FromSeconds(solution.Max(timeVar));
                        newStation.StationSerialNumber = StationSerialNumber++;
                        stationsInShopDAL.Add(newStation);
                    }

                    index = solution.Value(routing.NextVar(index));
                }
                var endTimeVar = timeDimension.CumulVar(index);

                totalTime += solution.Min(endTimeVar);
            }

        }
```

ניתוח ותרשים UML / CASE USE של המערכת המוצעת

תרשים CASE USE התחברות משתמש



שם ה use case	התחברות נהג
תיאור קצר:	נהג קיים מתחבר למערכת
תנאי קודם:	הנהג רשום במערכת
שחקנים מעורבים:	נהג
הזנק:	הרצון להתחבר למערכת.
תנאי סיום:	הנהג התחבר בהצלחה.
תדירות:	פעם ביום

שם ה use case	התחברות ספק
תיאור קצר:	ספק קיים מתחבר למערכת
תנאי קודם:	הספק קיים במערכת
שחקנים מעורבים:	ספק
הזנק:	הרצון להתחבר למערכת.
תנאי סיום:	הספק התחבר בהצלחה.
תדירות:	פעם בשבוע

שם ה use case	התחברות ספק
תיאור קצר:	ספק קיים מתחבר למערכת
תנאי קודם:	הספק קיים במערכת
שחקנים מעורבים:	ספק
הזנק:	הרצון להתחבר למערכת.
תנאי סיום:	הספק התחבר בהצלחה.
תדירות:	פעם בשבוע

שם ה use case	התחברות ספק חדש
תיאור קצר:	ספק חדש מתחבר למערכת
תנאי קודם:	הספק לא קיים במערכת
שחקנים מעורבים:	ספק
הזנק:	הרצון להצטרף למערכת.
תנאי סיום:	הספק התחבר בהצלחה.
תדירות:	פעם בחודש

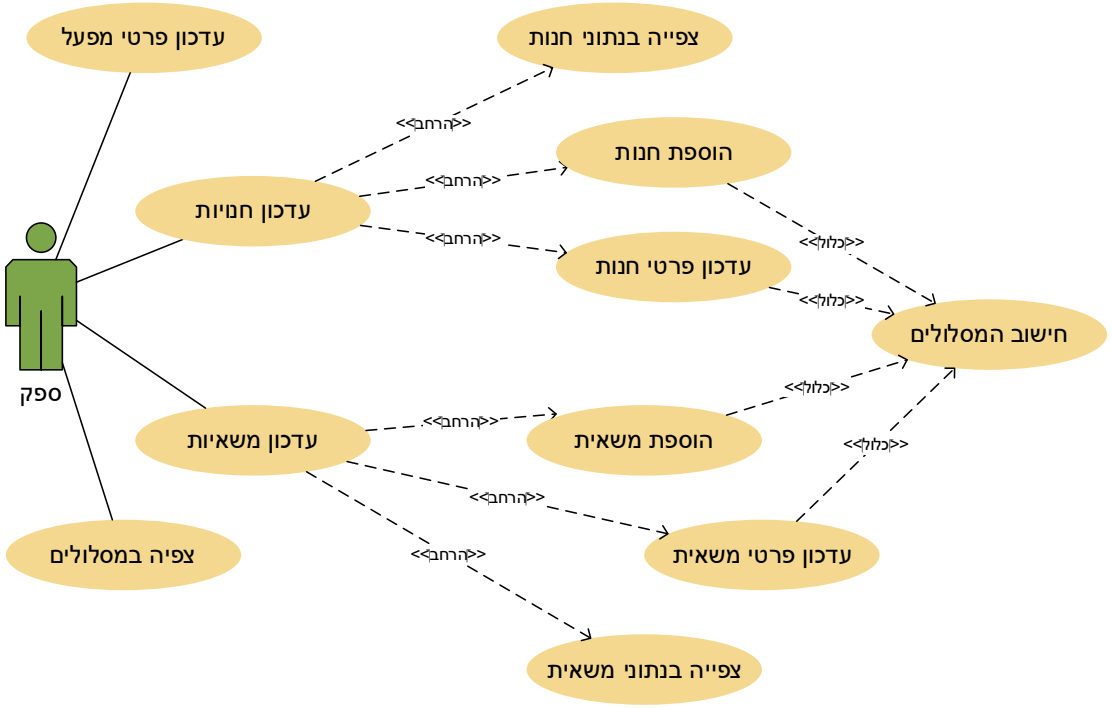
שם ה use case	התחברות ספק חדש-> הזנת פרטי ספק
תיאור קצר:	הספק מזין את פרטיו האישיים
תנאי קודם:	הספק התחבר כעת בפעם הראשונה למערכת
שחקנים מעורבים:	ספק
הזנק:	הרצון להצטרף למערכת.
תנאי סיום:	פרטי הספק נשמרו בהצלחה
תדירות:	פעם בחודש

שם ה use case	התחברות ספק חדש-> הזנת פרטי מפעל
תיאור קצר:	הספק מזין את פרטי המפעל

תנאי קודם:	הספק התחבר כעת בפעם הראשונה למערכת
שחקנים מעורבים:	ספק
הזנק:	הרצון להצטרף למערכת.
תנאי סיום:	פרטי המפעל נשמרו בהצלחה והספק נוסף למערכת.
תדירות:	פעם בחודש

שם ה use case	**צפייה במסלול**
תיאור קצר:	הנהג צופה בפרטי המסלול
תנאי קודם:	הנהג רשום במערכת
שחקנים מעורבים:	נהג
הזנק:	הרצון לצפות במסלול.
תנאי סיום:	המסלול הוצג בהצלחה
תדירות:	פעם ביום

תרשים use case פעילות הספק



שם ה use case	**עדכון פרטי מפעל**
תיאור קצר:	הספק מעדכן את פרטי המפעל
תנאי קודם:	הספק כבר קיים במערכת
שחקנים מעורבים:	ספק
הזנק:	חלק מפרטי המפעל השתנו ויש צורך בעדכון המערכת
תנאי סיום:	פרטי המפעל התעדכנו בהצלחה.
תדירות:	פעם בשנה

שם ה use case	חנויות-> צפייה בנתוני חנות
תיאור קצר:	הספק צופה בנתוני חנות מסוימת
תנאי קודם:	החנות קיימת במערכת ושייכת לספק הנוכחי
שחקנים מעורבים:	ספק
הזנק:	הרצון לצפות בפרטי החנות
תנאי סיום:	
תדירות:	פעם בשבוע

שם ה use case	חנויות-> הוספת חנות
תיאור קצר:	הספק מוסיף חנות חדשה למערכת
תנאי קודם:	החנות לא קיימת במערכת.
שחקנים מעורבים:	ספק
הזנק:	חנות חדשה הצטרפה ללקוחות הספק
תנאי סיום:	החנות נוספה בהצלחה למערכת
תדירות:	פעם במספר חודשים

שם ה use case	חנויות-> עדכון פרטי חנות
תיאור קצר:	הספק מעדכן את פרטי חנות מסוימת
תנאי קודם:	החנות קיימת במערכת ושייכת לספק הנוכחי
שחקנים מעורבים:	ספק
הזנק:	נתוני החנות השתנו ויש צורך לעדכנם
תנאי סיום:	פרטי החנות התעדכנו בהצלחה למערכת
תדירות:	פעם במספר חודשים

שם ה use case	משאיות-> צפייה בנתוני משאית
תיאור קצר:	הספק צופה בנתוני משאית מסוימת
תנאי קודם:	המשאית קיימת במערכת ושייכת לספק הנוכחי
שחקנים מעורבים:	ספק
הזנק:	הרצון לצפות בפרטי המשאית
תנאי סיום:	
תדירות:	פעם בשבוע

שם ה use case	משאיות-> הוספת משאית
תיאור קצר:	הספק מוסיף משאית חדשה למערכת
תנאי קודם:	המשאית לא קיימת במערכת.
שחקנים מעורבים:	ספק
הזנק:	משאית חדשה הצטרפה לשירותי ההובלה של הספק
תנאי סיום:	המשאית נוספה בהצלחה למערכת
תדירות:	פעם במספר חודשים

שם ה use case	חנויות-> עדכון פרטי חנות
תיאור קצר:	הספק מעדכן את פרטי משאית מסוימת
תנאי קודם:	המשאית קיימת במערכת ושייכת לספק הנוכחי
שחקנים מעורבים:	ספק
הזנק:	נתוני המשאית השתנו ויש צורך לעדכנם
תנאי סיום:	פרטי המשאית התעדכנו בהצלחה למערכת
תדירות:	פעם במספר חודשים

שם ה use case	חישוב מסלול
תיאור קצר:	המערכת מחשבת מחדש את המסלולים עבור כל משאית.
תנאי קודם:	קיימות חנויות המזמינות סחורה מהספק הנוכחי
שחקנים מעורבים:	המערכת
הזנק:	חל שינוי בחלק מהנתונים המשפיעים על החישוב.
תנאי סיום:	המסלולים החדשים נשמרו בהצלחה במערכת
תדירות:	פעם בחודש

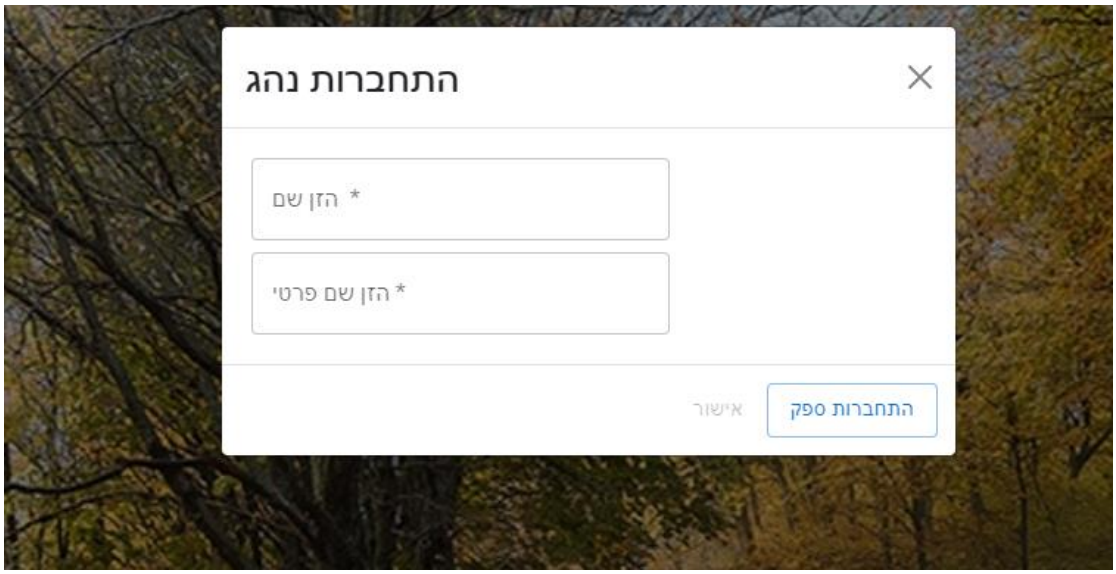
מדריך למשתמש

משתמש נהג

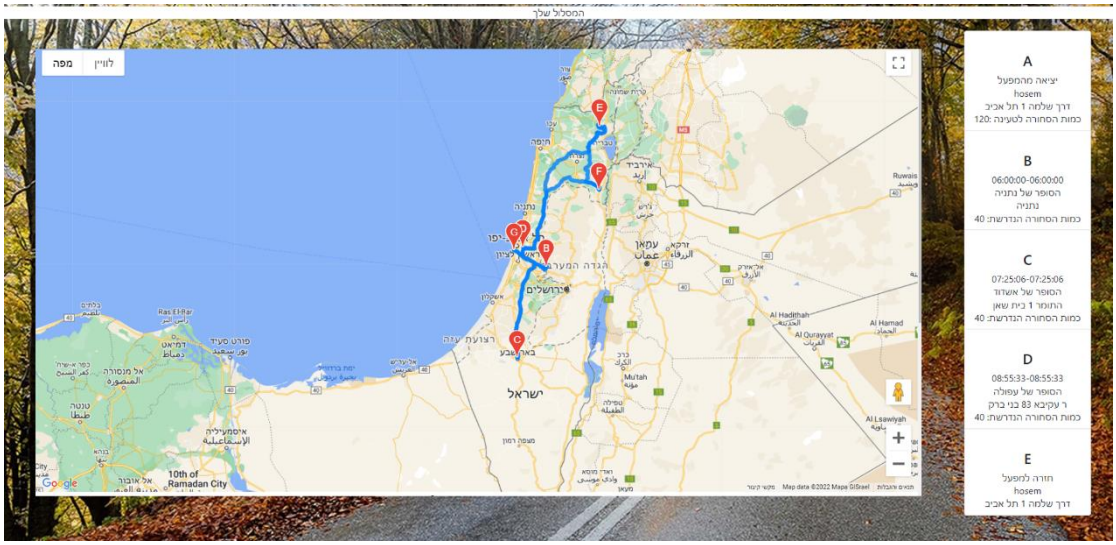
מסך התחברות

כדי להתחבר למערכת עליך להזין שם, ת.ז. וסיסמא ובסיום לבחור אישור

אם אתה ספק לחץ על התחברות ספק ועבור להדרכה על משמש ספק.



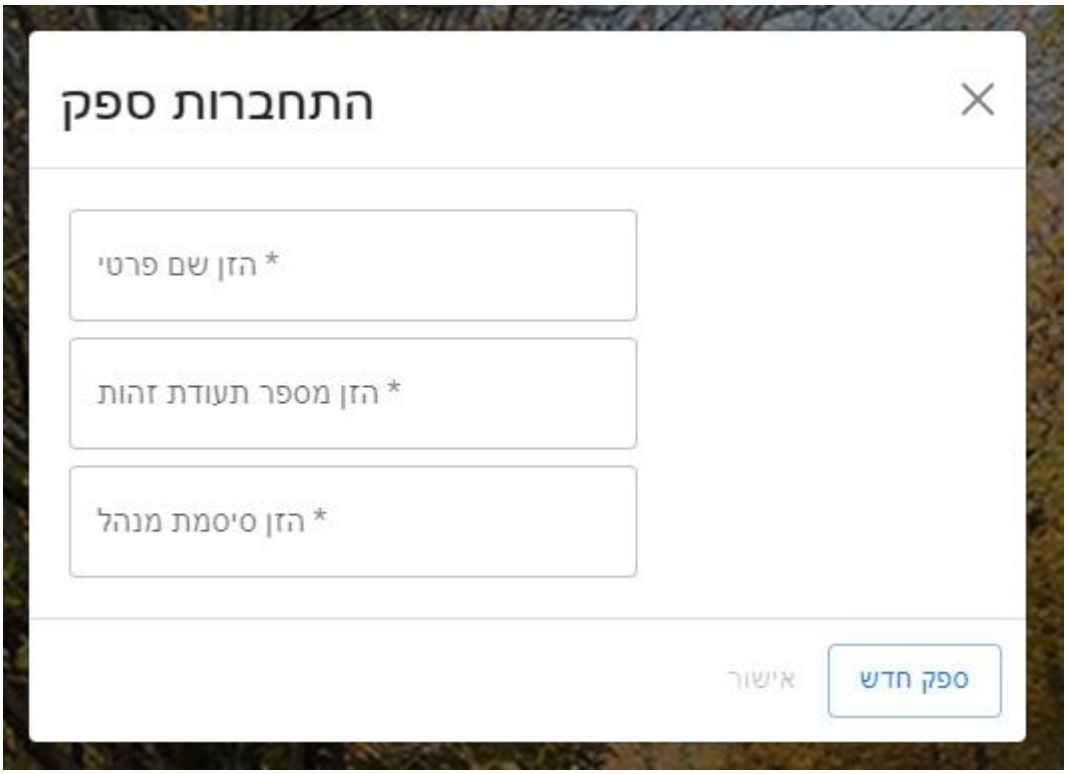
מיד לאחר שהתחברת למערכת, תוכל לצפות במסלול המתוכנן עבורך, בכמות הסחורה אותה עליך לטעון במשאית, בשעות המדויקות בהן תבקר בכל חנות וכו'



משתמש ספק

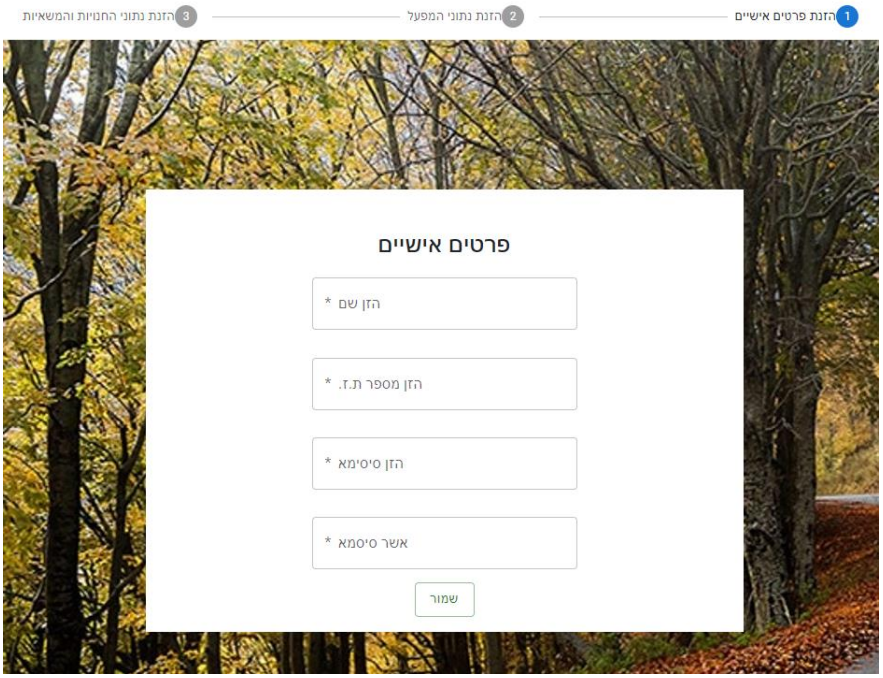
מסך התחברות

כדי להתחבר למערכת עליך להזין שם, ת.ז. וסיסמא ובסיום לבחור אישור



אם זו כניסתך הראשונה למערכת לחץ על "ספק חדש" ויפתח בפניך המסך הבא:

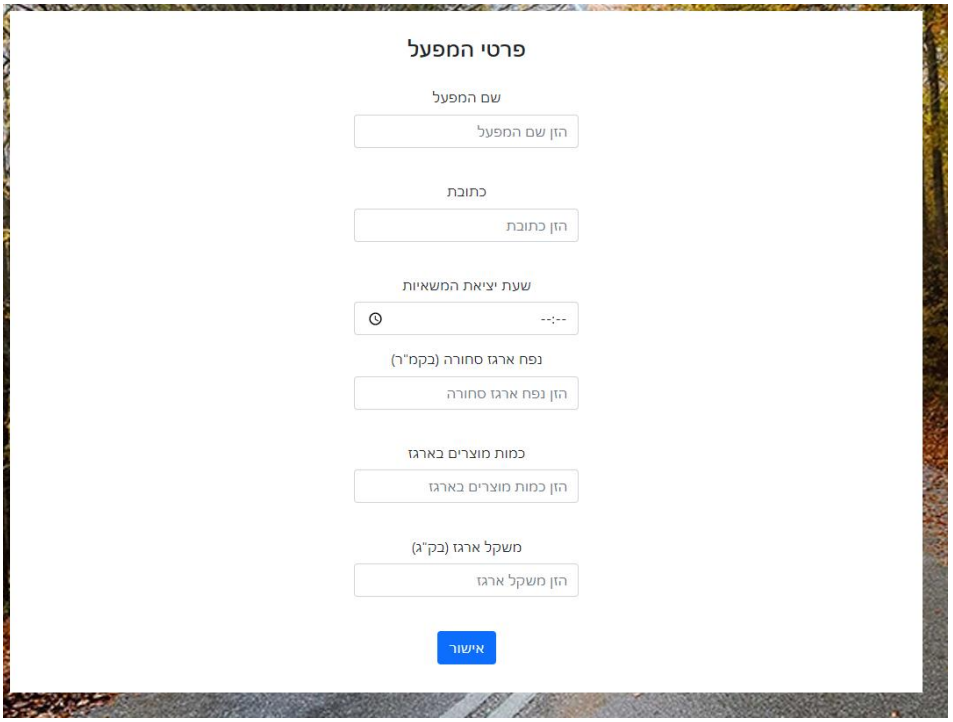
מסך פרטים אישיים



כאן עליך להזין את פרטיך האישיים ובסיום ללחוץ על שמור.

לאחר מכן, יפתח בפניך המסך הבא:

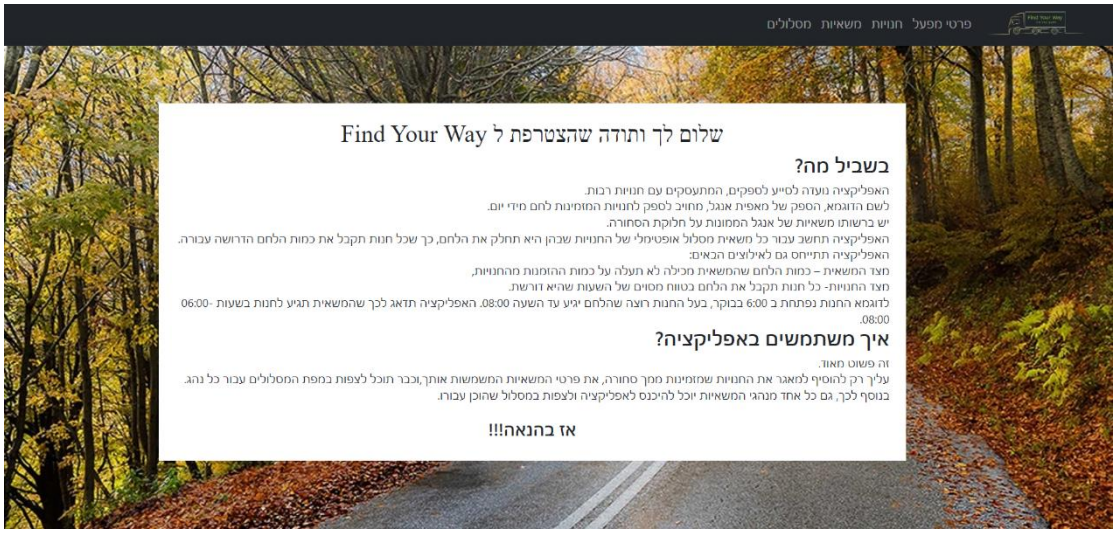
מסך פרטי המפעל



גם כאן עליך להזין את פרטי המפעל שלך ללחוץ על אישור ו...זהו! גמרנו להגדיר את הפרטים הבסיסיים שלך כעת יפתח בפניך המסך כשל ספק קיים.

אם כבר התחברת למערכת, לא שכחנו אותך, מיד בהתחלה יפתח לפניך המסך הבא:

מסך אודות



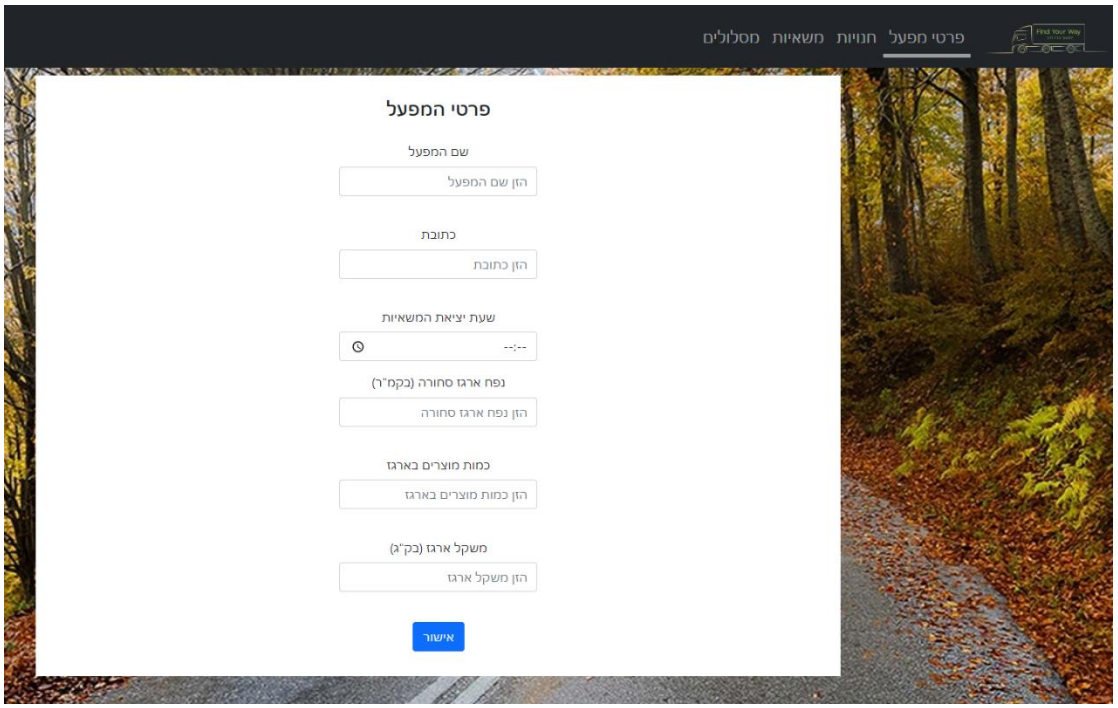
במסך זה ישנו פירוט נרחב על אופן השימוש והצורך עליו עונה האפליקציה.

מכאן אתה יכול לנוע בחופשיות לכל אחד מן המסכים הבאים:

- פרטי מפעל
- חנויות
- משאיות
- מסלולים

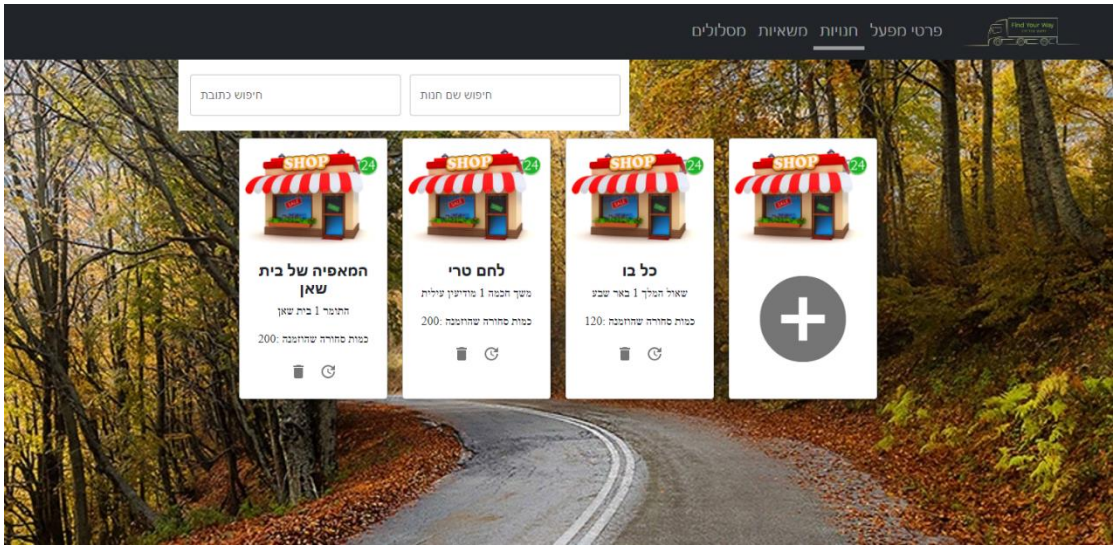
מסך פרטי מפעל

כאן תוכל לשנות את פרטי המפעל והסחורה, באם תידרש לכך.

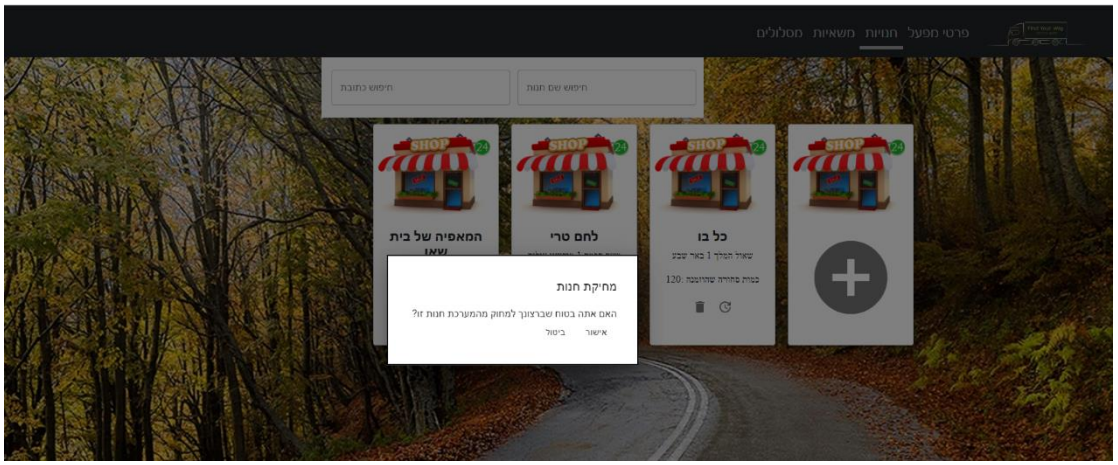


מסך חנויות

כאן תוכל לצפות בפרטי החנויות להן אתה מספק מוצרים באופן קבוע, לשנות את פרטיהם וכן למחוק ולהוסיף חנויות.

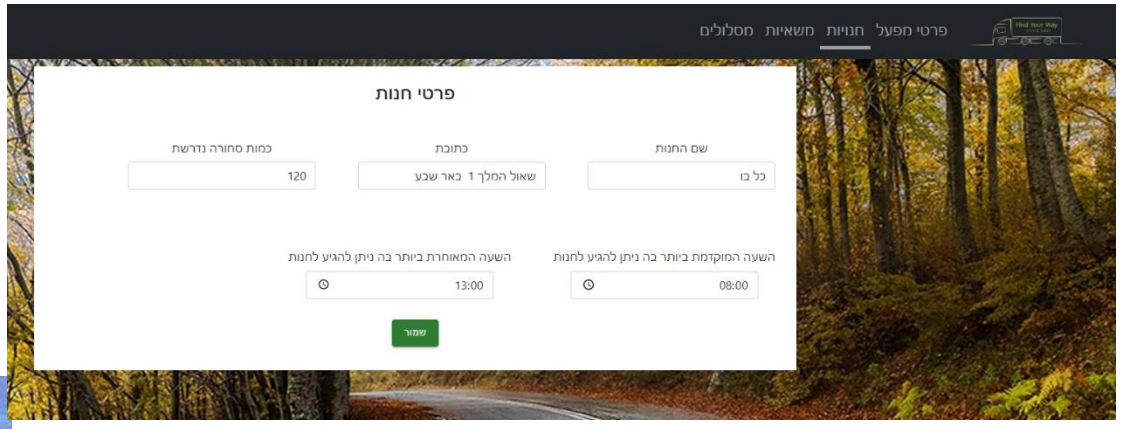


אם תרצה למחוק חנות עליך ללחוץ על סמל המחיקה ,תיפתח בפניך ההודעה הבאה:



לחץ על אישור והחנות תימחק.

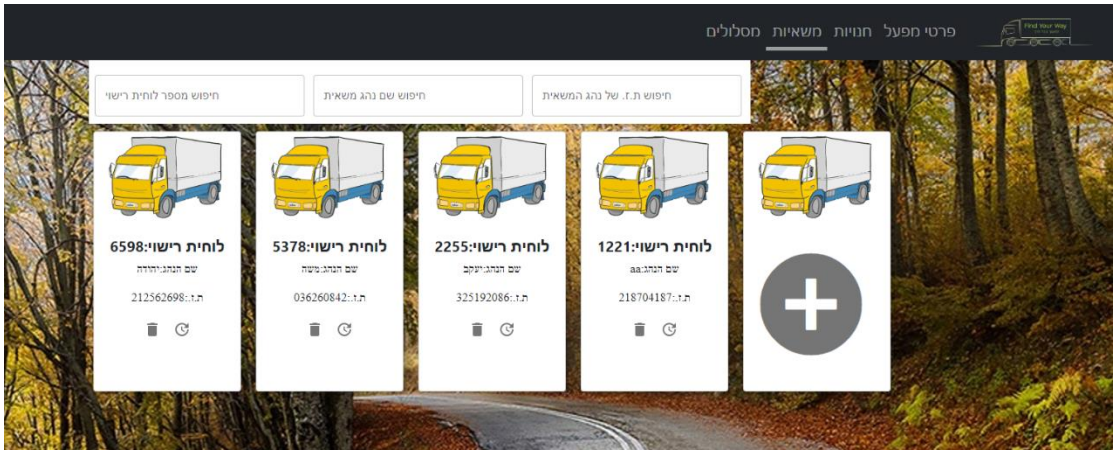
אם ברצונך לשנות את פרטי אחת החנויות לחץ על סמל העדכון, יפתח בפניך המסך הבא:



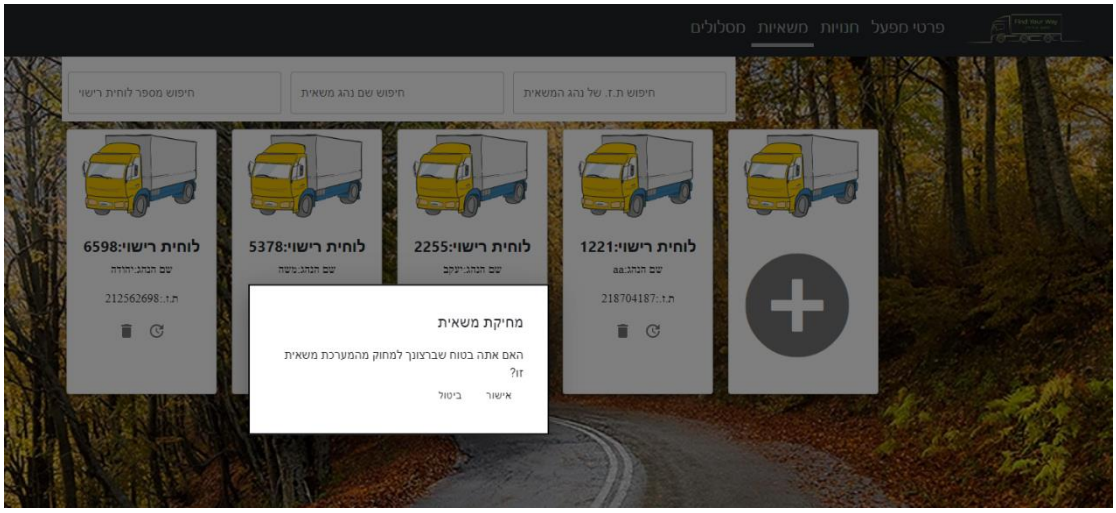
הזן את השינויים הנדרשים לך ולחץ "שמור"

מסך משאיות

כאן תוכל לצפות בפרטי המשאיות העומדות לרשותך, לשנות את פרטיהם וכן למחוק ולהוסיף משאיות.

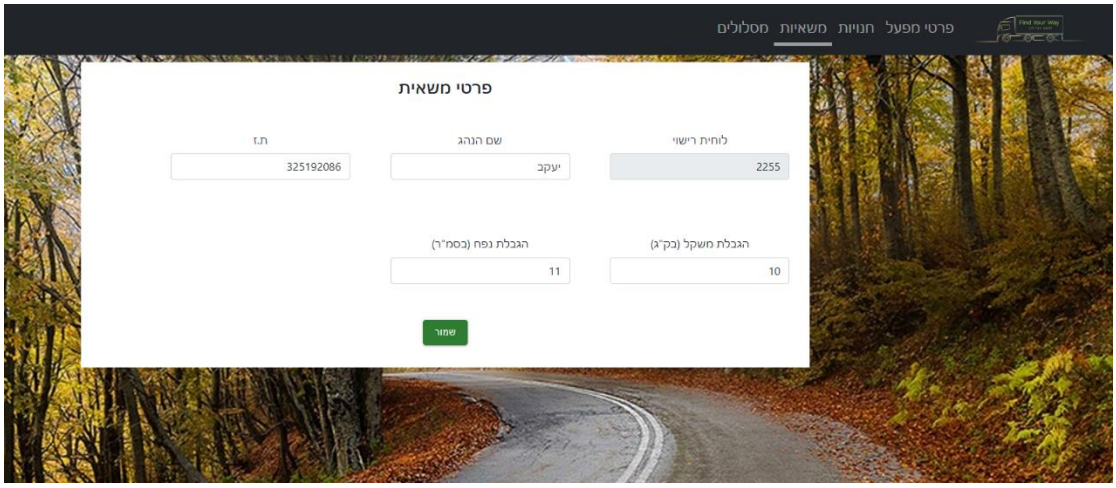


אם תרצה למחוק משאית עליך ללחוץ על סמל המחיקה ,תיפתח בפניך ההודעה הבאה:



לחץ על אישור והמשאית תימחק.

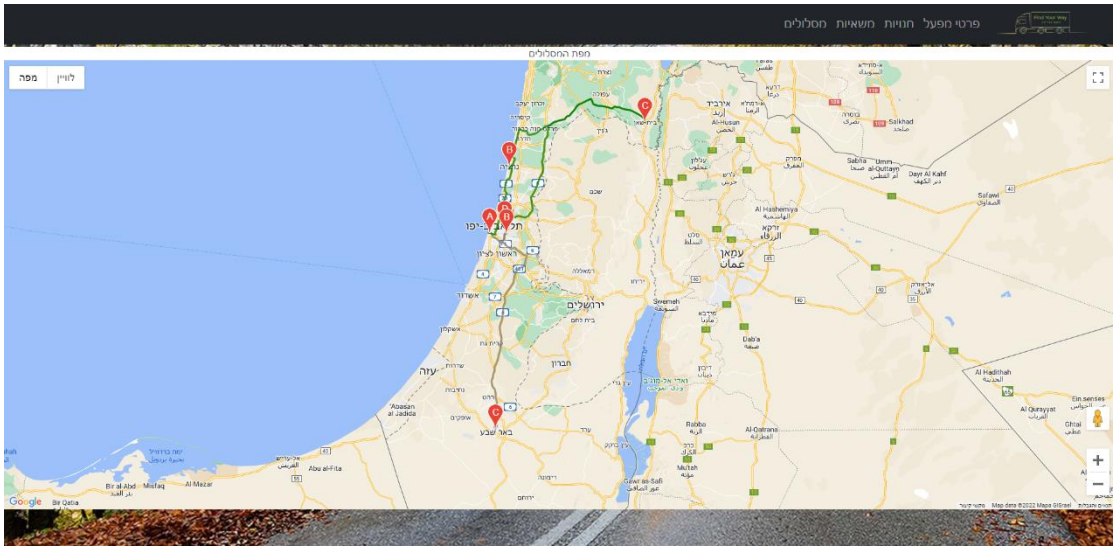
אם ברצונך לשנות את פרטי אחת המשאיות לחץ על סמל העדכון, יפתח בפניך המסך הבא:



הזן את השינויים הנדרשים לך ולחץ "שמור"

מסך מסלולים

כאן תוכל לצפות במפת המסלולים המכסה את כל החנויות ועומדת באילוצים שנקבעו בדרך האופטימלית.



בדיקות והערכה

המערכת מנהלת מאגר חנויות/ משאיות וכו" בצורה נוחה וברורה לשימוש.

המערכת מציגה לספק את קבוצת המסלולים האופטימליים העומדים הן באילוצי הקיבולת והן באילוצי הזמן.

המערכת מציגה לנהג את המסלול הסופי שעליו לבצע, בתוספת פרוט על כל תחנה.

ניתוח יעילות

הפתרון הראשוני שעלה במוחי היה בסיבוכיות של O((m*n)!), כאשר n מתייחס לכמות היעדים בהם על המשאיות לעבור ו m מתייחס למספר המשאיות.

וכן גם לאחר חיפוש בחומרים וברשת הבנתי שזוהי הסיבוכיות של הפתרונות הידועים.לבסוף פתרתי את הבעיה ע"י שימוש בספריית or tools הפותרת את הבעיה בתוך פחות מ- 10 שניות!

אבטחת מידע

אבטחת המידע הינה חלק חשוב וניכר בפרויקט, שכן איננו רוצים שמשתמשים בלתי מזוהים יכנסו למערכת ויגעו בנתונים,על כן הכניסה למערכת לספקים הינה ע"י הקשת שם משתמש ת.ז. וסיסמה שנבחרו בעת הצטרפות למערכת

ואימותם עם הנתונים השמורים בשרת, ורק לאחר האימות מעבר לניווט המתאים.

הכניסה לנהגים גם היא תתאפשר באמצעות שם ות.ז.

וכן ישנה הפרדה בין הרשאת נהג להרשאת ספק, הנהג לא יוכל לשנות כלל את הנתונים, באפשרותו רק לצפות במסלול שהוכן עבורו.

מסקנות

מלכתחילה הפרויקט היה נראה לי גדול ממדים.

מציאת אלגוריתם יעיל בסיבוכיותו, שימוש בפונקציות של גוגל מפות, התחשבות באילוצים רבים , הצגת המסלול במפה וכו"...

החלטתי לפרק את העבודה לשלבים קטנים ולעבוד אחד אחד, וכך גיליתי שככל שמפרקים את העבודה היא יותר פשוטה וברורה, הן לי והן למתכנתים אחרים שינסו להבין את הקוד, וכן שתכנון מוקדם ועשייה עקבית בס"ד מביאים אל התוצר המוגמר.

תוך כדי העבודה על הפרויקט נחשפתי ללמידת חומרים ושימוש בהם באופן עצמאי,נדרשתי לחפש ולנסות, עד שעבד, וכשלא עבד להבין למה ואיך.

מאד נהניתי מההישגים שהגעתי אליהם דרך הלמידה העצמית, הופתעתי מכך שזו מיומנות נרכשת וניתנת לשיפור.

ובכלל ביצוע החומר באופן מעשי מביא לידיעה והפנמה שלא מושגות בשום דרך אחרת.

לסיכום: דרך הפרויקט רכשתי בס"ד ידע וכלים רבים שיעזרו וישמשו אותי גם בעתיד.

ביבליוגרפיה

- Google maps
- W3School
- Npm
- StackOverFlow
- Mui
- react-bootstrap
- Developers .google optimizationrouting vrp
- Altexsoft
- Frontiersin
- Hindawi

-סוף-