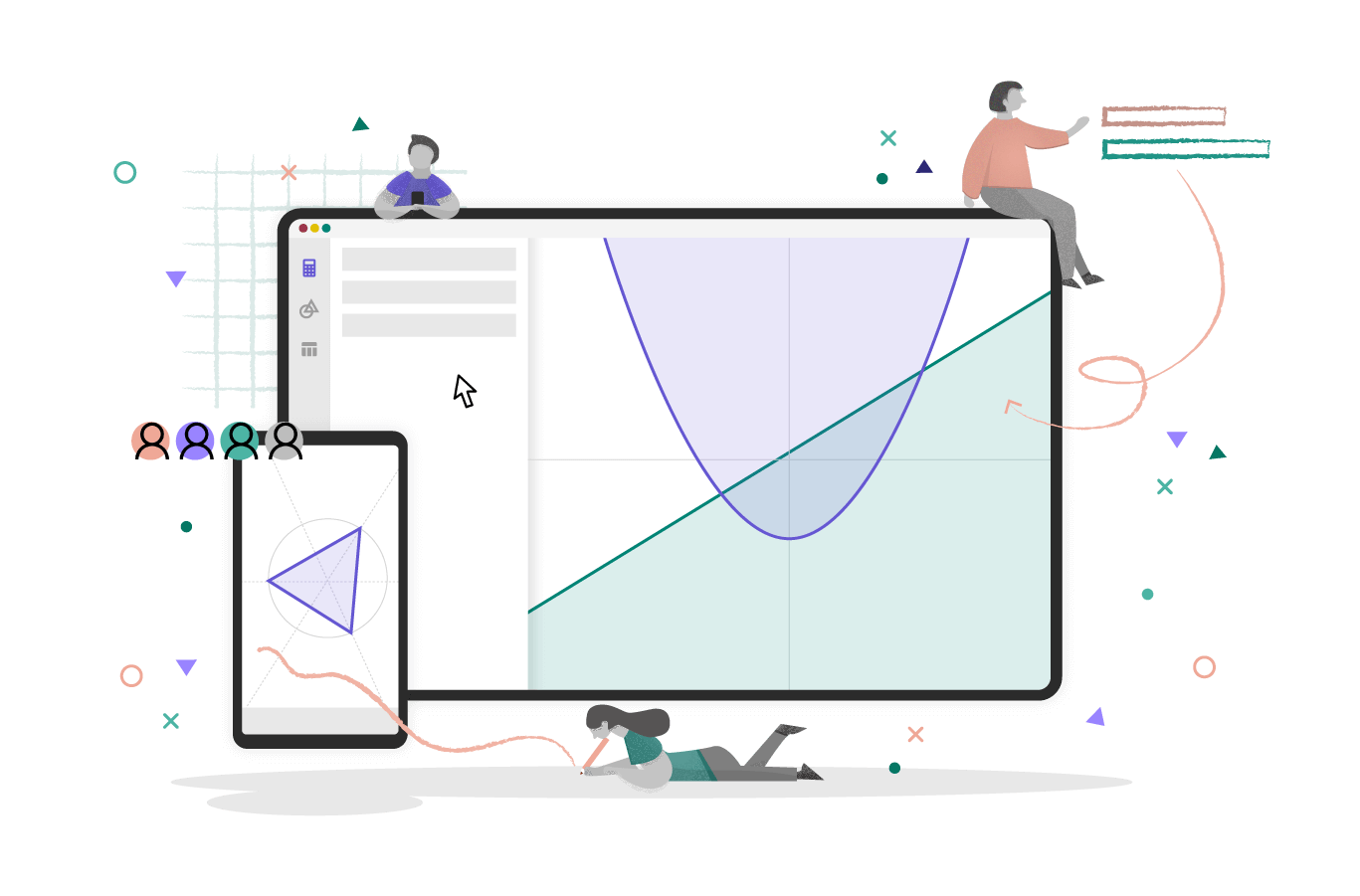
# 

**שם התלמידה:** רותי ברס  **ת"ז:** 211551668

**שם התלמידה:** פסי קשש **ת"ז:** 207069014

SuperQuick



[השג את תשומת הלב של הקוראים באמצעות ציטוט משמעותי מהמסמך, או השתמש בשטח זה כדי להדגיש נקודת מפתח. כדי למקם תיבת טקסט זו בכל מקום בעמוד, פשוט גרור אותה.]

**רינת אביטל**

**212735427**

**פרויקט גמר 2022**

רינת אביטל

# **תוכן עניינים**

[תוכן עניינים 3](#_Toc42553067)

[שלמי תודה 4](#_Toc42553068)

[מבוא 5](#_Toc42553069)

[תקציר 7](#_Toc42553070)

[מטרות ויעדים: 9](#_Toc42553071)

[מטרות המערכת: 9](#_Toc42553072)

[יעדי המערכת: 9](#_Toc42553073)

[תיחום המערכת: 10](#_Toc42553074)

[רעיונות לפיתוח עתידי 11](#_Toc42553075)

[מפרט טכני: 13](#_Toc42553076)

[סביבת פיתוח: 13](#_Toc42553077)

[עמדת משתמש: 13](#_Toc42553078)

[הסביבה הארגונית 13](#_Toc42553079)

[מבנה המערכת 14](#_Toc42553080)

[ניתוח מערכת 27](#_Toc42553081)

[מילון מושגים 28](#_Toc42553082)

[האלגוריתם המרכזי 30](#_Toc42553083)

[תחומי לוגיקה נוספים 67](#_Toc42553084)

[ממשק משתמש 72](#_Toc42553085)

[תצוגת המסכים 82](#_Toc42553086)

[מסקנות 91](#_Toc42553087)

[ביבליוגרפיה 93](#_Toc42553088)

[נספח 94](#_Toc42553089)

# **שלמי תודה**

ראשית, תודתנו **לבורא העולם** שהוביל אותנו צעד צעד בדרך הארוכה, משלב גיבוש הרעיון לפרויקט ועד להגשתו הסופית, ורק בעזרתו ית' הגענו עד הלום.

בנוסף, נודה לכל אלה שנתנו הרבה מכוחם, מזמנם ומתבונתם במהלך בניית הפרויקט:

**למנחה, המורה רבקה אדלשטיין** שבאכפתיות ובמסירות, סייעה לנו לאורך כל הדרך בהכנת הפרויקט, הנחתה, יעצה, בדקה, חיוותה דעה והובילה לפתרון בעיות שנראו בלתי פתירות, החל משלב התכנון, הביצוע ועד להגשה הסופית.

**למורה חני סירוקה,** שלימדה אותנו להתמודד גם עם בעיות מלחיצות.

**למורה מוריה שילר** המיוחדת על הסיוע הבלתי נלאה באנגולר בעזרת הניסיון הרב והידע הנרחב, מכל הלב, בכל שעה, בכל שלב מהבסיס ועד לסוף המורכב. וכל הבאגים, השאלות שאותרו וטופלו על ידה יעידו- אין לנו מילים!

**למרכזת המסלול, המורה שרה ברלין**, שמובילה במקצועיות ובכשרון את המגמה ולא חוסכת מאמץ להביא אותנו להישגים מקסימליים.

**לכל צוות רכזות ומורות סמינר הרב כהנא,** על המקצועיות הרבה וההשקעה לטווח רחוק.

**להורינו היקרים, לבני משפחותינו ולחברות הקבוצה**, על התמיכה, אוזן קשבת [לכל תהליך החישוב הארוך...] העידוד והפירגון לאורך כל הדרך, החל מתקופת הלימודים ועד להגשת פרויקט הגמר.

# **מבוא**

המטרה העיקרית בפרויקט גמר היא התלמדות. בשלב ראשון למידה עצמית מעמיקה של התחומים בהם נוגע הפרויקט, לאחר מכן, החלטה על דרך פעולה ומידול העבודה והשלב העיקרי – הפיתוח עצמו.

**התלמדות** – כיון שהאפליקציה שלנו מחייבת שימוש נרחב בתחומים שונים מעולם התכנות, בהם תחומים אליהם לא נחשפנו במהלך הלימודים, נדרשה מאתנו יכולת למידה עצמית גבוהה מאד והיא אכן הלכה והשתפרה במהלך הפיתוח כאשר נדרשנו להתגבר על בעיות שונות ולמצוא להן פתרונות מגוונים.

**בחירת הרעיון** – המטרה שעמדה לנגד עינינו בבחירת הרעיון לפרויקט היתה, ראשית כל, אלגוריתם חזק ומאתגר, ויחד עם זאת יעיל שימושי ונדרש שיעזור לכל אדם.

האפליקציה שלנו, פותרת בעיה יומיומית נפוצה, וכוללת אלגוריתם מורכב וחכם.

עריכת קניה בסופרמרקט גדול, עם שפע גירויים ומגוון מוצרים, מוכרת לרבים כחוויה מפוקפקת.

בעלי החנויות מנסים למשוך אותנו ולהביא לצריכה מיותרת. הם מגייסים לשם כך את הטכנולוגיה ומערכות תוכנה חדישות במטרה לפתות את הלקוח שנאבד בהמולה, מתקשה להתמקד בעצמו ובצרכיו ולא בשפע המפתה סביבו.

גם אם נאמץ את מדריכי הכלכלה הנבונה ונצא לקניה רק אחרי ארוחה טובה ועם אמצעי תשלום מוגבל אנו מסתכנים בשיטוט ארוך ומתיש כמעט כמו טיול שנתי אחרי יום עבודה.

אז זהו. הפעם הטכנולוגיה לצד הלקוח – הוא במרכז והחנות – סביבו. הוא מוזמן לערוך קניה מהירה ושווה שכולה חוויה.

האפליקציה שפיתחנו מזמינה את המשתמשים לקנות (רק מה שצריך!) בצורה יעילה ונעימה;

ראשית ברשימה מסודרת באמצעות חיפוש חכם ומהיר, כשתוך כדי כך גם מוצג מיקומם במפת החנות, ובנוסף גם בהקלקה על מפת החנות עם שרטוט מסלול איסוף מהיר.

**האפליקציהSuper Quick** , אפליקציה חדשנית לסיוע בעריכת קניה בצורה יעילה ונעימה. השירות המרכזי הינו הגשת מפה ללקוח עם המסלול הקצר ביותר שעליו ללכת. האפליקציה פותחה בטכנולוגיה המתקדמת ביותר ובממשק משתמש נוח וידידותי כדי למקסם את חווית המשתמש.

**התנסות ופיתוח –** במהלך הפיתוח השתפשפנו רבות בכתיבת קוד ומימוש רעיונות מופשטים. התנסות זו כללה פיתוח אלגוריתמים מתוחכמים לצירוף שטחים קרובים ואיתור נקודות גישה. בנוסף חקרנו לעומק את בעיית הנוסע המתמיד (TSP) ומגוון פתרונותיה, סיבוכיות החישובים ויעילותם.

נקודה משמעותית נוספת בפיתוח האפליקציה היא הגדרת היעדים. כפי שיוסבר, נדרשנו ללוגיקה חזקה ומורכבת כדי לחסוך בזמן ריצה. למעשה, בנינו מודל חישובי עצמאי שמביא, בס"ד, לתוצאות מטיבות אשר נותנות ללקוח דרך חכמה, קצרה ויעילה. כמו"כ התמודדנו עם בעיות אנגולריות רבות בהצלחה, והתוצאה אכן מדברת בעד עצמה.

# **תקציר**

**כללי**: האפליקציה Super Quick מאתרת מוצרים בחנות בצורה מהירה ויעילה.

כאשר לקוח עורך קניה בחנות גדולה, לעיתים הוא נתקל בקושי למצוא את מבוקשו. למרות השילוט הקיים בדר"כ בחנויות כגון אלו, הוא נאלץ להסתובב כה וכה ברחבי החנות כדי להגיע למחוז חפצו.

המערכת נבנתה ע"מ להקל על הלקוח בעריכת הקניה ולעבור אותה בשלום ואפילו בחיוך. האפליקציה מאפשרת לקונה למטב את חווית הקניה ולהפוך אותה לשונה ומהנה. בעזרתה יוכל לקנות את כל הדרוש לו בזמן קצר והגיוני מבלי להתבלבל ומבלי להאריך את הדרך ולטעות בכיוונים.

שימוש באפליקציה מתאים לקניה ממוקדת של מספר פריטים מוגדר.

תחילה הלקוח בוחר את החנות בה הוא רוצה לקנות. (במידה ומדובר במכשיר מסוג 'מסופון' השייך לעמדה בחנות – יוכל לדלג על שלב זה). לאחר מכן הלקוח בוחר את המוצרים לקניה, המערכת מאתרת לו אותם באמצעות השלמת מילים ומאגר שמות למוצרים. לבסוף המערכת מחשבת לו את הדרך הקצרה והיעילה ביותר והוא מקבל את מפת החנות עם שרטוט ברור כיצד עליו ללכת.

כמובן שהמערכת מאפשרת למנהל חנות להכניס את מפת החנות שלו לתוכנית. על המנהל לעדכן את מפת החנות שלו בהתאם למבנה ולשינויים הנוכחים, ולהגדיר את המוצרים הקיימים.

**הממשק**: האפליקציה פותחה באנגולר והושקעה מחשבה רבה בעיצוב נקי ונעים לעין, בנוחות השימוש ובפשטות ההפעלה כדי לספק למשתמש חוויית שימוש מושלמת. בין השאר שמנו דגש על ניצול פונקציונליות מעניינת ועיצוב נעים ומרענן.

**הפיתוח**: האפליקציה פותחה בסביבת העבודה .net בשפה c#. שפה מעניינת מלאה בפונקציונליות, נוחה וגמישה למתכנת. לאורך פיתוח האלגוריתם נחשפנו לאפשרויות פתרון רבות עבור כל שלבי התוכנית. כיון שהמערכת הינה אינטראקטיבית ותגובתה צריכה להיות מידית, הקפדנו לצמצם את זמן הריצה למינימום האפשרי. השקענו מחשבה רבה בבחירת הדרך היעילה והטובה ביותר להשגת המטרה תוך ניסיונות רבים. דרך פעולה זו התבטאה רבות במיפוי החנות, באלגוריתם קיבוץ האזורים וביצירת מסלול. כמובן שבדרך זו למדנו והרחבנו ידע בכל התחומים בפרויקט.

**הספר:** המעיין בספר זה ימצא תיאור נרחב של הפרויקט הן בפן הלוגי והן בפן הגרפי-חיצוני שישתלב במערכת הכללית. במהלך הספר ניתן להבחין בעקרונות התכנון של המערכת, שלבי החישוב, מדגם של אלגוריתמים הנכללים בפרויקט, תהליכים שקיימים במערכת, תרשימים ועוד. בנוסף מכיל הספר מדריך למשתמש בו נמצאים צילומי מסך והסברים כיצד ניתן להשתמש במגוון האפשרויות אותן מספקת האפליקציה.

חלק משמעותי נוסף בספר הוא הלמידה הרבה שהושקעה בפרויקט. לאלגוריתם המוגמר ודרך הפעולה הסופית שנבחרה קדמו למידה עמוקה של החומר ואפשרויות פתרון שונות שנוסו ונשללו במהלך העבודה עד לבחירת האלגוריתם המתאים ביותר.

# **מטרות ויעדים:**

בבניית המערכת יש לוודא שתהיה נוחה, פשוטה, וקלה לשימוש. בנוסף צריך שתהיה ברורה ומובנית למתכנת כך כשירצה להמשיך בפיתוח התכנה יוכל בקלות לעשות זאת. כמובן חשוב שתהיה יעילה ותעבוד כראוי.

בבואנו לבנות את מערכת SuperQuick הצבנו לעצמנו מטרות ולשם כך מספר יעדים:

## **מטרות המערכת:**

**1.** מיטוב חווית הקניה ועריכת רשימה בקלות.

**2.** חסכון בזמן, יעילות ומהירות באיסוף המוצרים בחנות ומניעת שיטוט מיותר בחיפוש אחר מוצר.

**3.** חסכון בכסף ע"י מניעת קניה מיותרת וסיוע בהתמקדות ברשימת מוצרים נצרכת.

**4.** הנגשת מוצרים אשר תוביל למשיכת לקוחות לתועלת בעל החנות.

**5.** נוחות וידידותיות למשתמש.

## **יעדי המערכת:**

**1.** בניית ממשק חכם לעריכת רשימת קניות.

**2.** הצגת מפת חנות עם מיקומי המוצרים.

**3.** הצגת מסלול איסוף אידיאלי במפת החנות.

**4.** יצירת המערכת כאפליקציה המספקת נוחות וידידותיות למשתמש.

# **תיחום המערכת:**

המערכת מטפלת ב:

המערכת מאפשרת ללקוח להזין רשימת קניות מתוך המוצרים הקיימים בחנות, מציגה את מיקומי המוצרים בחנות ומחשבת מסלול קניה אופטימלי.

המערכת תקבל את מפת החנות מבעל החנות בקובץ ,XML ותבצע המרה של נתוני החנות לנתונים טבלאיים.

המערכת מאפשרת לבעל החנות להוסיף מוצרים ולערוך אותם, לבחור מוצרים לחנות, להוסיף כינויים למוצרים ולמקם מוצרים בחנות.

המערכת אינה מטפלת ב:

המערכת אינה מתחייבת ל 100 אחוזי הצלחה במציאת מסלול קצר ביותר.

המערכת מניחה שקיים ממשק נוח לבעל החנות המטפל בבניית חנות והזנת הנתונים ומייצר קובץ XML המכיל את כל הנתונים הנדרשים. ממשק זה אינו במסגרת הפרויקט.

(אנו פיתחנו ממשק בסיסי לשימוש אישי למפתח האפליקציה בעיצוב מינימלי המאפשר הזנת נתונים והכנסת חנות ללא בדיקות תקינות, ואיננו מטפלות בתקינות הנתונים הנקלטים).

המערכת אינה אחראית לשינוי שמות ושאר הפרטים במוצרים הכלליים שמבצע בעל חנות מסוימת (היות שהמוצרים משותפים לכל החנויות).

# **רעיונות לפיתוח עתידי**

בשלבי תכנון המערכת הגדלנו ראש ולא הגבלנו את עצמנו בחשיבה וברעיונות. כתוצאה מכך הצטברו לנו מגוון רעיונות, אותם ניתן יהיה לפתח בעתיד, לחלקם הכנו תשתית מתאימה במבנה הנתונים.

* **עדכון מלאי:** המערכת תתממשק בזמן אמת עם המלאי הקיים בחנות כך שלקוח לא ילך לקחת מוצר שאינו קיים במלאי. לצורך זה נוסיף מאפיין 'מלאי' בטבלת קישור המוצרים למדפים בחנות.
* **קריאת ברקוד:** ע"י הזנת ברקוד (באמצעות קורא ברקודים או בהזנה ידנית) ניתן לראות את מיקום המוצר במפת החנות (מתאים לעובדים\סדרנים) או להוסיפו למוצרי החנות (עבור בעל חנות).
* **סטטיסטיקות:** פיתחנו כמה אלגוריתמים לחישוב מסלול אופטימלי הפועלים במקביל.

כיון שהמדובר הוא בחנויות שמבנן לא אמור להשתנות הרבה במהלך החיים, ניתן לתכנן את המערכת באופן שתוכל ללמוד מניסיונה ותאפשר לדעת מהם האלגוריתמים היעילים יותר ואלו פחות – עבור כל חנות, המידע ישמש גם להמשך הפיתוח.

ע"י שמירת נתונים סטטיסטיים ושימוש בהם נוכל לייעל את החישובים ולדייק את השימוש בשיטה הנכונה לכל חנות. ככל כשהמערכת תהיה ותיקה יותר היא תחכם ותהיה ממוקדת יותר.

* **שכלול Alias:** קינון היררכי של כינויי מוצרים, כך שכאשר מקשרים כינוי למוצר יודעים עליו דברים רבים. דוגמא: כשמשייכים ל'שוש' את ההורה 'במבה' יוכלו לדעת עליו שהוא מכיל בוטנים, חטיף, חטיף מלוח, ממתק, וכולי.

טבלת הכינויים קיימת, וקיים בטבלה מאפיין parent לאפשור ההיררכיה.

* **אבטחת נתוני המוצרים-** כדאי ליצור מערך אבטחה על שמות המוצרים, היות והנתונים משותפים לכל החנויות וכל אחד יכול ליצור חנות ולשבש את הנתונים. בטבלת Alias יש שדה IsPrivate שיאפשר לבעל חנות ליצור שמות פרטיים משלו ולהשתמש רק בהם.
* **עיצוב תלת מימדי-** במידה והמערכת תשתדרג ותאפשר ללקוח לראות את **המדפים** בסטנדים, נוכל להראות ללקוח באיזה מדף בדיוק נמצא המוצר, כי המוצרים ממוקמים במדפים. בממשק הנוכחי אין למדפים משמעות, אך קיימת תשתית במסד הנתונים להצגת מדפים.
* **בניית חנות-** כמובן, השידרוג הראשוני שנעשה יהיה בניית ממשק חכם נוח לבעל החנות. הממשק יצטרך לכלול אלגוריתם ליצירת נקודות גישה וקישור אוטומטי בין נקודות גישה לסטנדים כך שכל תהליך הזנת נתוני חנות יהיה קליל וקצר.

**מציאת נקודות גישה-**התחלנו לממש אלגוריתם בסיסי ליצירת נקודת גישה. אלגוריתם זה יובא בהמשך הספר, בפרק **'תחומי לוגיקה נוספים'**.

**אלגוריתם המוצא קשתות בחנות-** ז"א מקשר בין 2 נקודות גישה או בין נקודת גישה לסטנד, אלגוריתם זה ממומש במידה מסוימת בפונקציה למציאת נקודות גישה לנקודות התחלה, שם מימשנו חיפוש נקודות גישה לנקודה מסוימת בחנות. כדי להוסיף קשתות לחנות, נוכל להשתמש באלגוריתם זה.

בניית החנות לא בתחום הפרויקט, אך קיימים בידינו האלגוריתמים המוזכרים לעיל, עליהם נתבסס בעת כתיבת הלוגיקה המטפלת בבניית חנות.

# **מפרט טכני:**

## **סביבת פיתוח:**

**עמדת פיתוח:** מחשב dell

**מערכת הפעלה:** windows 10

**כלי התוכנה לפיתוח המערכת:** visual studio 2017, visual studio code

**שפות תוכנה:** c# תוך שימוש בטכנולוגית WebApi, Angular עבור האפליקציה.

## **עמדת משתמש:**

**חומרה:** Core i3 or higher, RAM 1024

**תוכנות:** Internet Explorer 7 and higher, or Firefox or chrome

**חיבור לרשת:** נדרש

# **הסביבה הארגונית**

**האפליקציה תשרת:**

**לקוח-** כל אחד יוכל להיעזר באפליקציה כדי לקנות בקלות ובמהירות ולחסוך כסף, זמן וכח.

**בעל חנות-** למשוך לקוחות באמצעות אפליקציה שובה לב ונוחה לשימוש.

**היקף הפרויקט:**

1200 שעות

# 

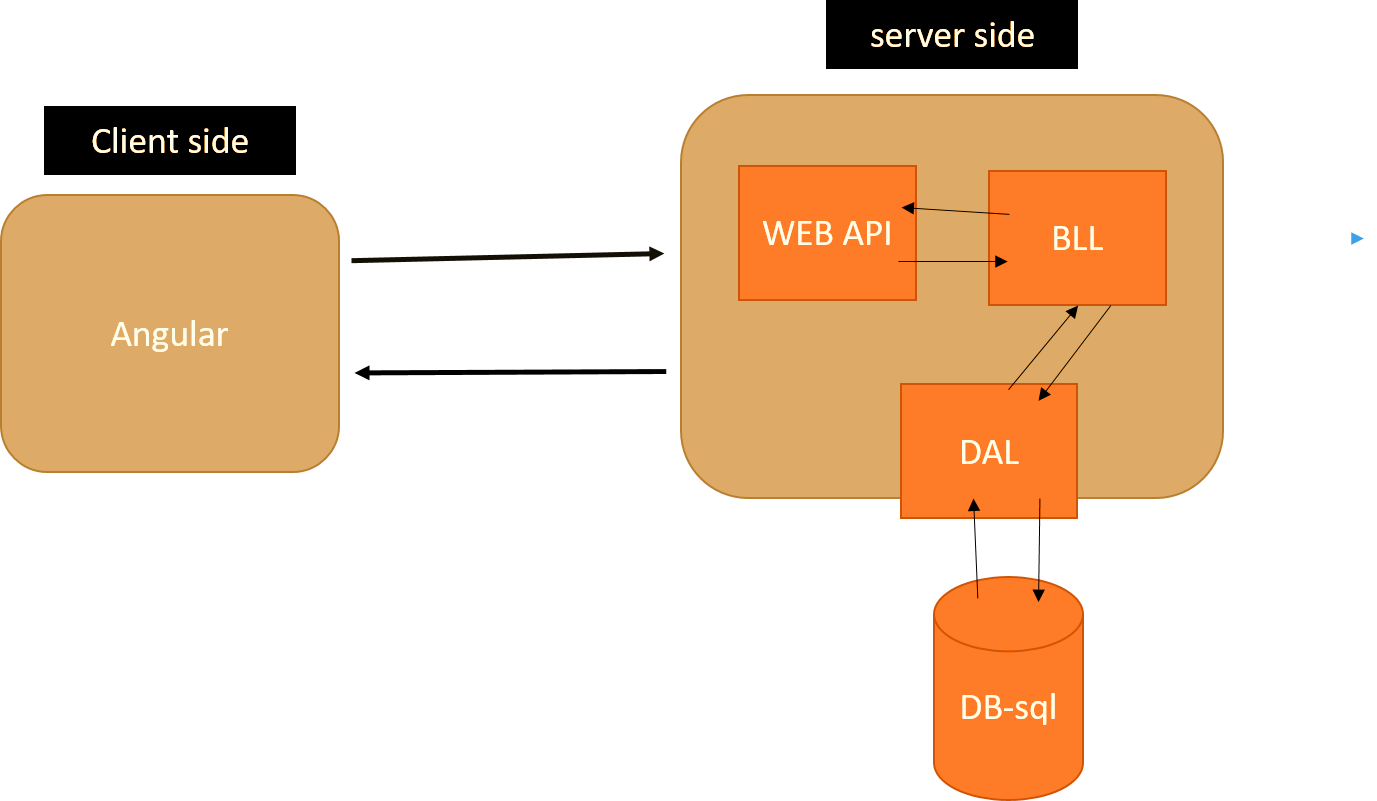
# **מבנה המערכת**

בניית המערכת נעשתה תוך שימוש בהפרדת רשויות מלאה ובניית היררכיה ברורה, על מנת למנוע שיבושים בנתונים ובצורת התצוגה. בנוסף, המבנה מאפשר שינוי תכנותי בעתיד, הן בצורת הגישה לנתונים והן בצורת התצוגה ללא הצורך במגע עם שכבות נוספות.

**המערכת מתחלקת לשני תחומים:**

האחד**:** **שרת** המבצע את הפעולות הנדרשות תוך התקשרות למסד הנתונים.

השני**:** **ממשק המשתמש** הבנוי בצורת אפליקציה המתקשרת לשרת לצורך ניהול האפליקציה תוך שליפת נתונים ממסד הנתונים והצגתם למשתמש בצורה נאה.

**זרימת המידע במערכת**

תרשים זה מתאר את מבנה הפרויקט, המכיל 3 חלקים:

**1.** שכבת מסד הנתונים (SQL)

**2.** צד שרת- c#

**3.** צד לקוח- האפליקציה

**צד השרת נבנה כמקובל כמודל שלוש השכבות:**

**1.** שכבת הישויות (Entity Framework) והגישה לבסיס הנתונים, ה **-DAL.**

בשכבה זו קיים מודל שנבנה ע"י טכנולוגיית EntityFrameWork, ובו מחלקה מקבילה לכל טבלה במסד הנתונים.

בנוסף, יש בו ספריית Partial למחלקות מסוימות, המאפשרת הרחבה למחלקות במודל. הוספנו הרחבות שנדרשו לצורך החישובים בשכבת הלוגיקה. כך הוספנו למחלקה פעולה בונה, הגדרנו מחדש את פונקציית Equals, ומימשנו במחלקה מסוימת את ממשק IComparable.

המחלקות בספריה זו בעלות שמות זהים לשמות המחלקות שבמודל, וכוללות בכותרת המחלקה את ההגדרה 'partial' וכך המערכת מזהה כי מדובר בהרחבה למחלקה קיימת.

**2.** שכבת הקוד ה**BLL**-

שכבה זו מכילה את הלוגיקה של הפרויקט. כמו כן, משמשת כמתווכת בין שכבת הDAL לשכבת ה-WebApi, בשליפת ובעדכון הנתונים.

הספריות שבחלק זה:

* **Logic-** כאן יהיו מחלקות שונות המטפלות בחישוב המסלול ובשאר חלקי הלוגיקה.
* **DTO-** ספריה זו מכילה את מחלקות מקבילות למבנה הנתונים כדי לקשר בין שכבת הDAL- וה-WebApi.
* **Utilities-** מחלקות עזר המסייעות לתקשורת בין השכבות

**3.** שכבת הקישור- **WebApi**

שכבה זו מתקשרת עם צד הלקוח, באמצעות קונטרולרים – בקרים, המקבלים פניות מהלקוח ומאחזרים מידע בהתאם.

בשיטה זו קיימת הפרדת ישויות מוחלטת וכל שכבה עומדת באופן עצמאי לחלוטין ומתקשרת רק עם השכבה שמעליה.

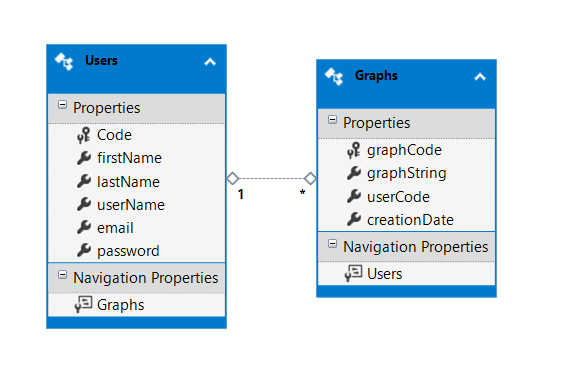
**צד הלקוח - ממשק המשתמש:**

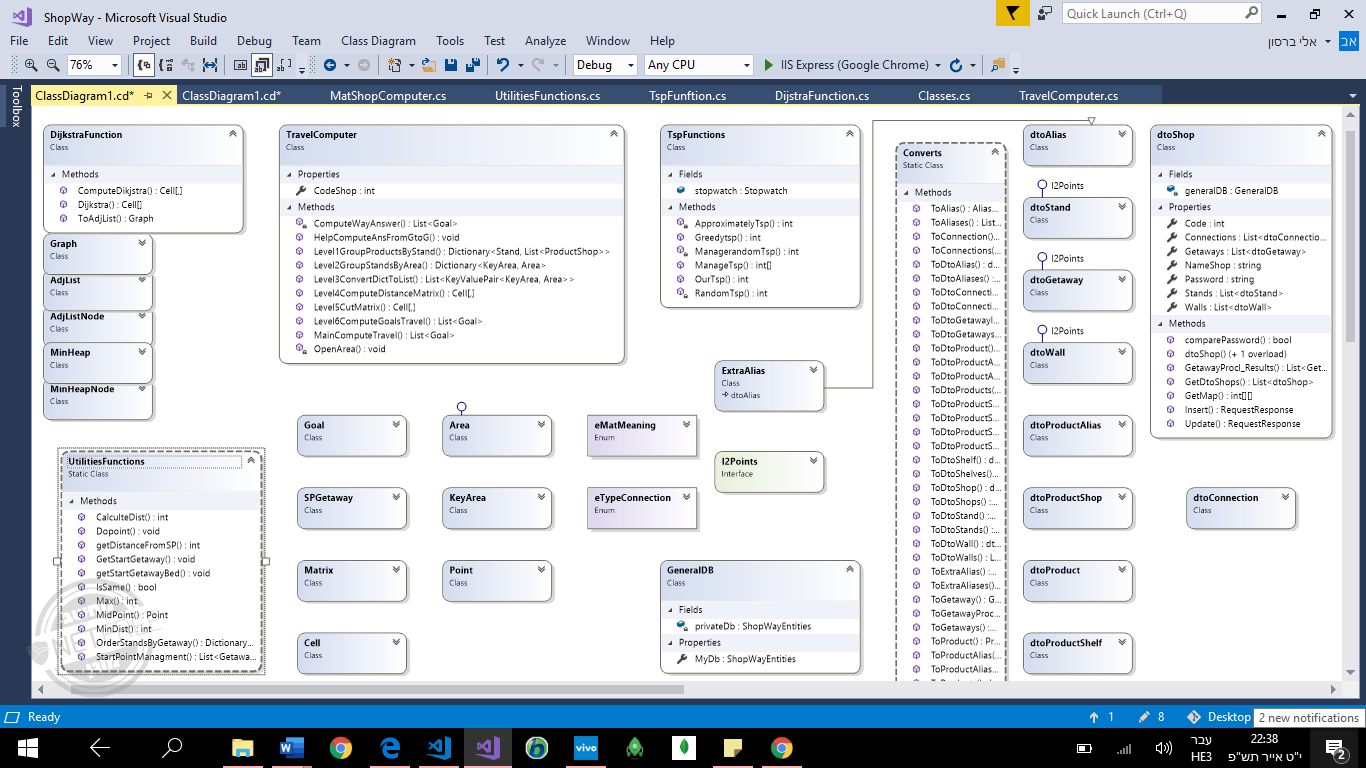
ממשק המשתמש נכתב ב Angular תוך שימוש ב8 Angular. מומש ב CSS, Html, TypeScript.

הממשק כולל את המסכים שיוצגו לבעל החנות ואת המסכים שיוצגו ללקוח בחנות.

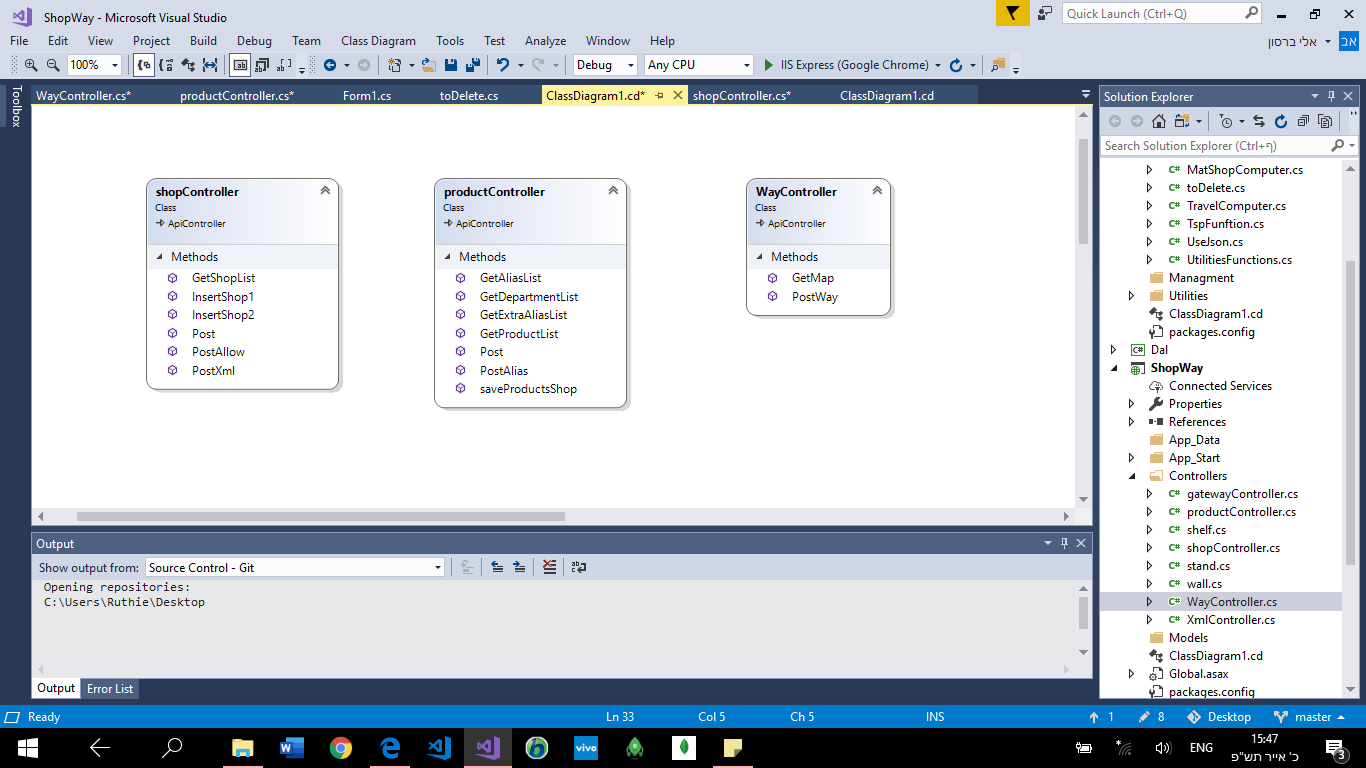
**Data Diagram**

להלן התרשימים המתארים את המחלקות מהן מורכב הפרויקט:

**שכבת ה-DAL, המחלקות להן קיימת הרחבה בספריית ה-partial:**

**תרשים המחלקות בשכבת ה-BLL**

**ה-Controls בשכבת ה-WebApi:**



**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, וכן שישמור נתונים שיבואו לידי שימוש בהמשך פיתוח המערכת, ע"מ לאפשר פיתוח ושדרוג של המערכת בלי זעזועים לחלקים הקיימים.

הנתונים נשמרים בצורה טבלאית בבסיס נתונים SQL. כל החנויות הרשומות במערכת שמורות בטבלת חנויות וכך גם כל המוצרים של החנות, הקירות, הסטנדים והמדפים, נקודות הגישה והקשתות.

מטריצת המרחקים הבסיסית של החנות מחושבת כשנוצרת החנות ונשמרת בקובץ XML.

כשבעל החנות מזין את חנותו הוא מצרף קובץ XML שמומר לאובייקט חנות וכך מתקבלים כל הנתונים על החנות. נתוני החנות נשמרים בטבלאות. הקובץ לא נשמר.

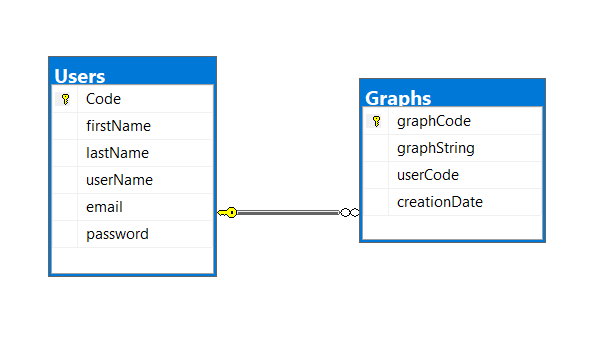
לאחר שהחנות נשמרת באפשרותו לערוך את הנתונים, לשנות מיקום מוצרים בסטנדים, להוסיף מוצרים ועוד.

**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 בהן השתמשנו:

****

**טבלאות**

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

**טבלת משתמשים- Users**

הטבלה מכילה את נתוני המשתמשים.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מפתחות | שם השדה | סוג השדה | תאור | שדה חובה |
| PK | Code | int | קוד | ✓ |
|  | firstName | varchar(50) | שם משתמש | ✓ |
|  | lastName | varchar(50) | משפחת המשתמש | ✓ |
|  | userName | varchar(50) | שם משתמש | ✓ |
|  | email | varchar(50) | מייל | ✓ |
|  | password | varchar(50) | ססמה | ✓ |
| FK  לטבלת נקודות גישה | \*CodeGetaway | int | קוד נקודת גישה שהיא פתח החנות | ✓ |

\*פתח החנות נשמר גם כ-Getaway. שמרנו בטבלה זו לכל חנות את הפתח שלה, וכך נוכל לחשב את התחלת המסלול מהפתח כאופציית ברירת מחדל [במידה ולא יבחר מיקום אחר בחנות להתחלת המסלול]

**טבלת גרפים- Graph**

טבלה המכילה גרפים.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מפתחות | שם השדה | סוג השדה | תאור | שדה חובה |
| PK | graphCode | int | קוד גרף | ✓ |
|  | graphString | varchar(50) | מחרוזת הגרף | ✓ |
|  | userCode | int | קוד משתמש | ✓ |
|  | creationDate | datetime | תאריך יצירה |  |

פרטי המוצר הם נתונים הרלווטים להצגה למשתמש [שם החברה, התמונה, והגודל אינם נצרכים לחישובים ומשמשים לתצוגה בלבד].

\*השדה גודל הינו מחרוזת המייצגת תיאור על גודל המוצר ויחידת המידה שלו, כמו "1 ליטר", "אריזה משפחתית", על פי החלטת בעל החנות המזין את הנתונים, ללא אכיפה.

\*\*השדה Src שומר את נתוני התמונה כמחרוזת המכילה את נתוני התמונה בביטים.

**דוגמא לזרימת המידע במערכת:**

לכל מחלקה יש מחלקה מקבילה בשרת ובלקוח.

לדוגמא מחלקת Graphs בשכבת הDAL:

public partial class Graphs

{

public long graphCode { get; set; }

public string graphString { get; set; }

public long userCode { get; set; }

public System.DateTime creationDate { get; set; }

public virtual Users Users { get; set; }

}

בנוסף, קיימת בספריית הDTO בשכבת הBLL מחלקת DtoGraph:

public class DtoGraph

{

public long graphCode { get; set; }

public string graphString { get; set; }

public long userCode { get; set; }

public DateTime creationDate { get; set; }

}

למעשה, המחלקה dtoShop מכילה באופן היררכי את כל נתוני החנות באמצעות רשימות אובייקטים המכילות אף הם תתי רשימות ואובייקטים:

Shop

1. Walls

1.1 Alias

1. Stands

2.1 Alias

2.2 Shelves

2.2.1 alias

2.2.2 ProductsShelves

2.2.2.1 Product

2.2.2.1.1 Alias

2.2.2.1.2 productAlias

2.1.2.1.2.1 Alias

1. Connections
2. Getaways

כדי להמיר אובייקטים משכבת DAL לאובייקטי הDTO, קיימת בשכבת הBLL בספריית Utilities מחלקת Converts.

לדוגמא, פונקציות במחלקת הConverts להמרת חנות:

public DtoGraph(Graphs g)

{

graphCode = (long)g.graphCode;

graphString = g.graphString;

userCode = (long)g.userCode;

creationDate = (DateTime)g.creationDate;

}

public static List<DtoGraph> DTOtoList(List<Graphs> list)

{

List<DtoGraph> DTOlist = new List<DtoGraph>();

foreach (var g in list)

{

DTOlist.Add(new DtoGraph(g));

}

return DTOlist;

}

public Graphs toTableEntity()

{

Graphs g = new Graphs();

g.graphCode = graphCode;

g.graphString = graphString;

g.userCode = userCode;

g.creationDate = creationDate;

return g;

}

כמובן, לכל מחלקה בצד השרת קיימת מחלקה מקבילה בצד הלקוח, באנגולר.

מחלקת Graph באנגולר, זהה למחלקת DtoGraph שבשרת:

export interface Graph {

    graphCode: string;

    graphString: string;

    userCode: number;

    creationDate: Date;

}

**אודות הספריה DTO:**

במודל שבשכבת הDAL קיימות המחלקות המכילות אובייקטים ורשימות של מחלקות המקושרות אליהם.

**לדוגמא:** בטבלת מדף קיים קוד סטנד, שהמדף משויך אליו. כתוצאה מכך מנגנון ה-EntityFrameWork הוסיף לטבלת מדף אובייקט מסוג סטנד, ובטבלת סטנד הוסיף רשימה של אובייקטים מסוג מדף. קיימת כאן הפניה מעגלית- המדף מצביע לסטנד והסטנד מצביע למדף. כל עוד מדובר בהפניות בלבד, הדבר אינו יוצר בעיה.

בזמן שליחת הנתונים מהשרת ללקוח מומרים האובייקטים לקובץ מסוג Json. הפעם לא נשלח האובייקט עם הפניות לאובייקטים המקושרים אליו, אלא נשלח נתוני האוביקט בפורמט Json ובתוכו נתוני האובייקט המקושר. כשנשלח מדף, יומר הסטנד שבתוכו לJson, וכן יומרו המדפים שבסטנד, וכך הלאה עד אין סוף, היות וההצבעות מעגליות. תתקבל שגיאה כי השרת לא הצליח לשלוח את הנתונים.

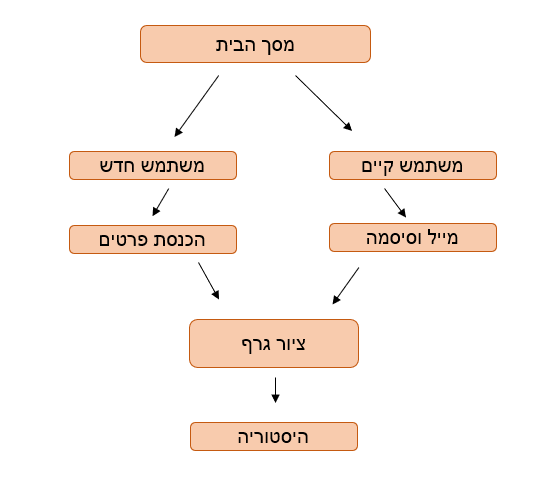
היינו מעוניינים לבטל את האובייקטים והרשימות שבמודל. אך, לא ניתן לשנות את מחלקות המודל שבשכבת הDAL, שהרי הן מיוצרות עפ"י בסיס הנתונים, ובכל עדכון מודל עלולות להיווצר מחדש ללא שמירת שינויי המתכנת. לכן, נשתמש בDTO.

DTO יכילו מחלקות מקבילות לכל מחלקה מהמודל, הן יוכלו להכיל אובייקטים השייכים למחלקה אחרת כל עוד לא יוצרו הפניות מעגליות.

כמו שהובא לעיל, ניתן לראות כיצד יצרנו היררכיה מסודרת כך שמחלקת Graphs מכילה את כל הנתונים השייכים לחנות, אך ללא מעגלים.

ע"י האובייקטים שהחנות מכילה נחסכו לנו שאילתות רבות, וקיבלנו את כל הנתונים בגישה ישירה ופשוטה. המרת הנתונים התבצעה באמצעות מחלקתConverts .

# **ניתוח מערכת**

# **מילון מושגים**

**להלן מספר הגדרות שיעשו סדר במושגים הקיימים בפרויקט:**



**מדף-** 'קומה' בעמודה שעליה מונחים המוצרים [כל מוצר בחנות ממוקם במדף אחד בלבד. במדף עשויים להיות כמה מוצרים].



**עמודה, סטנד, Stand-** 'בנין' המורכב מכמה מדפים. [כל מדף בחנות ממוקם בעמודה אחת בלבד, בעמודה ישנם, בדר"כ מספר מדפים].

**טור-** 'רחוב', הטור כולל מקום מעבר ללקוחות באמצע, כמה עמודות מימין וכמה עמודות משמאל.

הטורים אינם נשמרים במסד הנתונים, אלא רק נראים לעין באופן ויזואלי.

**נקודת גישה, שער, Getaway-** לכל טור ניתן להיכנס ממקום אחד או יותר. נסגור את הפתחים לטור בצורה מדומה, ונשמור לטור שער דמיוני, לצורך החישובים.

נקודת גישה היא דלת לטור של מוצרים, שא"א להגיע אליהם בלא מעבר דרכה (או דרך דלתות אחרות) ולכן השתמשנו במונח 'גישה'.

השם 'שער' ניתן להמחשה. למעשה חישבנו את אמצע הקטע של ה'שער' והשתמשנו בנקודה ולכן השתמשנו במונח 'נקודת גישה'.

**קשת, Connection-** גשר. יש שני סוגים של קשתות: 1-קשת המחברת בין 'נקודת גישה' לסטנד [לסטנד שבתוך הטור הסגור ע"י ה'נקודת גישה']. 2- קשת המחברת בין 'נקודת גישה' לנקודות גישה הסמוכות לו ונגישות לו. בקשת שמור המרחק בין המקור ליעד.

לכל סטנד ניצור קשתות עבור כל אחת מנקודות הגישה שלו. לכל נקודת גישה ניצור קשתות לכל אחת מנקודות הגישה האחרות שנגישות לו.

כמובן הקשת הינה לצורך החישובים בלבד ואינה נראית למשתמש, וכן כל נקודות הגישה.

**מטריצת מרחקים-** טבלה ריבועית, השומרת מרחקים עבור כל i j [הקטנים כמובן מגודל הטבלה].

mat[i][j] מכיל את המרחק מהיעד ה-i ליעד ה-j.

**פונקציית TSP-** פונקציה המקבלת מטריצת מרחקים שמכילה את כל המרחקים בין היעדים שאליהם מעוניינים להגיע, ומחשבת את הסדר האופטימלי לביקור ביעדים [מה קודם למה...] כך שסכום המרחקים שיעברו בהם יהיה אופטימלי. הפונקציה יכולה לכלול גם אילוץ יעד להתחלה ויעד לסיום.

**אזור, Area- 'חדר'**, יחידת שטח שרובה בדר"כ מוקף קיר, ולמעשה היא חלק מטור.

האזור, כמו טור, אינו נשמר במסד הנתונים אולם משמש רבות לחישוב וקיבוץ.

מטרת האזור לעזור לצמצום יעדים לחישוב במסלול. אזור יוכל לכלול כמה מוצרים, ולהיחשב כיחידת יעד אחת.

**הרעיון:** המוצרים שבאותו אזור, יאספו בטוח באותה פעימה ונחשבים כיעד אחד. "אותו אזור" מחשבים על פי נקודות הגישה. אם נכנס לחדר (קרי: אזור) לקחת דבר מה, ניקח כל אשר נרצה מחדר זה, ואז נצא מהחדר. אם יש יותר מדלת אחת לחדר מחלקים את החדר לחלקים כמספר הדלתות לפי קרבתם לדלת, כל חלק הוא חדר בפני עצמו.

# **האלגוריתם המרכזי**

**תאור המשימה ודרכי היישום:**

בשלב תכנון המערכת ובניית מסד הנתונים התבקשנו לבנות זאת בצורה הנכונה, המדויקת והממוקדת ביותר.

לכן חיפשנו את האלגוריתם היעיל ביותר לניווט בשטח מקורה, אלגוריתמים ושיטות שונות למיפוי ולשמירת השטח.

**המשימה:** ייצוג המשוואה באופן שנוכל לבצע בה חישובים מתמטיים

**האפשרויות:**

חקירת נושא פתירת משוואות פונקציונליות ושרטוטם הביאה אותי למסקנה שהדרך הטובה והיעילה ביותר היא ייצוג הפונקציה ע"י מבני נתונים מיוחד של רשימת פרמטרים לפונקציה.

חיפשתי סוגים שונים של אלגוריתמים שקשורים במתמטיקה ופתרון משוואות פונקציונליות אך לא מצאתי משהו שתואם את הבעיה שלי ולכן החלטתי ליצור בעצמי את האלגוריתם שמתאים כדי שאוכל לייצג את משוואת הפונקציה בצורה קלה ונכונה וכך אוכל ביתר קלות לגשת לכל אברי בפונקציה ולחקור אותה ולקבל תוצאות מדויקות.

**1.** ייצוג השטח באמצעות מטריצת אפסים ואחדות כאשר 1 מייצג קיר ו0 מייצג מעבר.

חישוב המסלול יתבצע ע"י אלגוריתם רקורסיבי שיחפש מסלול שמגיע לכל היעדים ויבחר את הדרך האופטימלית.

**החיסרון:** הסיבוכיות גבוהה מידי. (לא נוכל להסתפק בכך שמצאנו את הדרך, יהיה עלינו למצא את כל הדרכים האפשריות כדי להוכיח שנמצא המסלול המינימלי. מציאת כל המסלולים ברקורסיה העוברת תא אחר תא במטריצה אינה סיבוכיות המתאימה לממשק אינטראקטיבי).

**2.** נחשב מרחק בין כל מוצר למוצר בחנות ונשמור את כל המסלולים האפשריים.

**החיסרון**: לא מעשי ולא יעיל. החנות גדולה ויש אין סוף צירופים. הסיבוכיות אקספוננציאלית כשn- הוא מספר המוצרים בחנות.

**האפשרות הנבחרת:** חקירת הנושא הביאה אותנו למסקנה שהדרך הטובה והיעילה ביותר היא ייצוג החנות כגרף עם קודקודים וקשתות.

לכן נשמור לחנות נקודות גישה וקשתות, נחשב מרחקים בעזרת אלגוריתם דייקסטרה, נקבל מטריצת מרחקים ונחשב סדר מסלול ע"י TSP**.**

ערכנו עבודת חקר וקראנו חומרים רבים בנושא גרפים, התדיינו רבות באשר לדרך המימוש היעילה, [מטריצת סמיכויות לעומת רשימת סמיכויות, ערמת מינימום ומערך, וכו'] התייעצנו עם מומחים היאך לחשב מטריצת מרחקים ולהשתמש בפתרונות TSP מגוונים כדי לשמור על יעילות, דיוק ומהירות.

ניתן לסכם שלכל אורך הדרך עמדו לנו היצירתיות והחשיבה. ההכרח הוא אבי ההמצאה, ובזכותו נולדו המושגים החדשים (מוסברים במילון) הרעיון הוא לשלוט בשטח החנות ולחלק לאזורים כך שכל אזור הוא "ישר" ואינו כולל עיקולים. לצורך כך 'מתחנו חבלים' (וירטואלית כמובן) לאורך החנות ו'סגרנו אזורים'.

להלן מוצגות חלק מבעיות נוספות שהתעוררו בעת תכנון מבנה המערכת באופן זה, וכן הצעות שעלו ופתרונות שנקטנו בפועל.

**בעיות ופתרונותיהן:**

**א. שמירת מחרוזת של גרף הפונקציה לכדי מבני נתונים שישמר בתוכו נתוני הפונקציה והפיכתה לשימושית ופתירה.**

1. לשם כך צריך ליצור אובייקט "פרמטר" שבו מאוכסן כל איבר בפונקציה:

Value

Operator

Class

Parameter

Value – ערך האיבר

Operator – אופרטור (+/-)

Class – מעלה של האיבר (דרגה)

3

+

2

Parameter

לדוגמא:

1. על מנת שאוכל לאגד את כל רצף האיברים הקיימים לי אצטרך ליצור רשימת פרמטרים שתכיל את כל רצף האיברים הקיימים בפונקציה.

Value

Operator

Class

Value

Operator

Class

Value

Operator

Class

List< Parameter>

1. אך מכיוון שאני צריכה ליצור אובייקט משוואת פונקציה ולדעת פרטים נוספים חשובים אודות הפונקציה החדשה שנוצרת על כן ניצור אובייקט משוואה שבה נשמור את כל המידע הדרוש אודות הפונקציה.

Class

Count

Parameters

Equation

Value

Operator

Class

Value

Operator

Class

Value

Operator

Class

Class – מעלת האיבר הגדול ביותר מבין הפרמטרים

Count – מספר האיברים בפונקציה

Parameters – רשימת כל אברי הפונקציה הממוינים בסדר יורד



לדוגמא: עבור הפונקציה:

2

3

Parameters

Equation

3

+

2

4

-

0

5

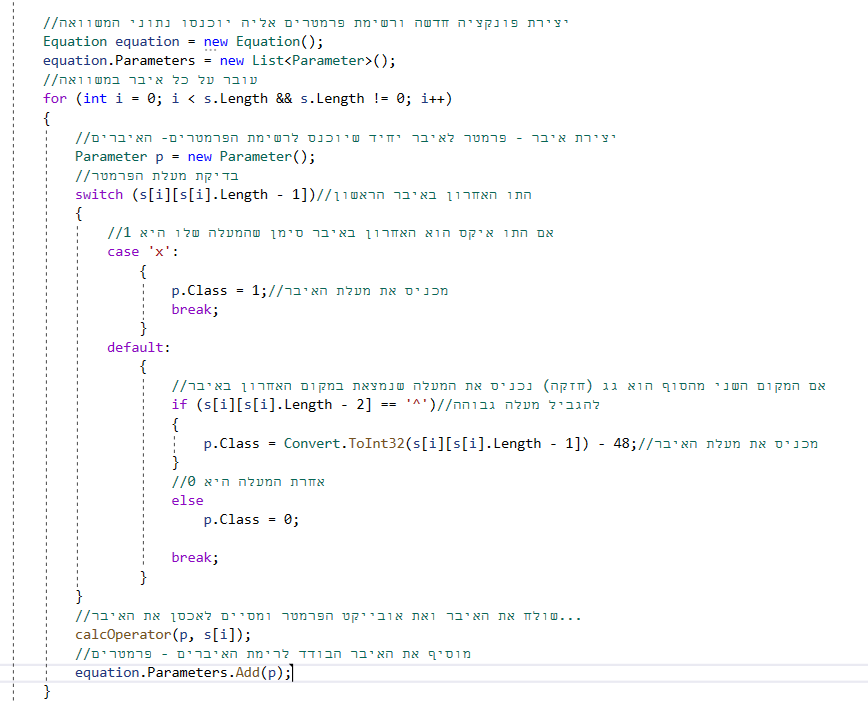
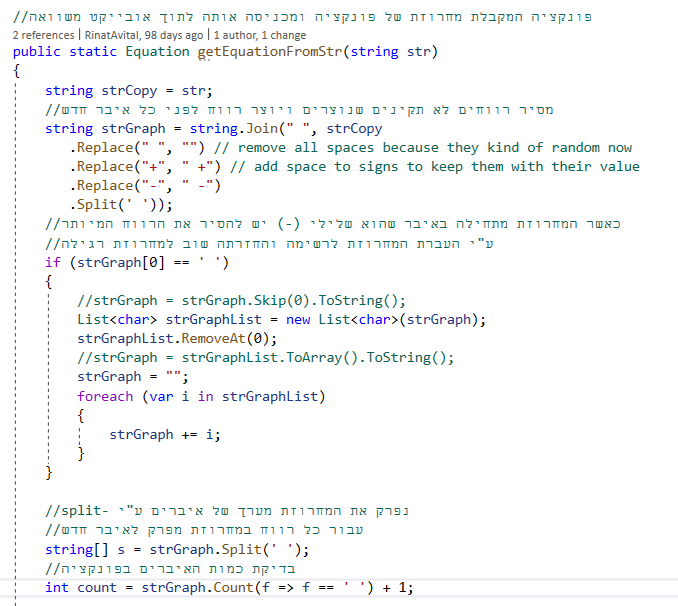
+

1

גם חישוב פונקציית הנגזרת מיוצגת ע"י האובייקט Equation שכמובן יכנס לתוכו ע"י חישוב אובייקט המשוואה שאפרט בהמשך.

אחרי שאני יודעת איכן יאוכסן כל המידע אודות הפונקציה נשאלת השאלה איך ממחרוזת פשוטה שהמשתמש הכניס למערכת אוכל לפרק את המשוואה לכדי מבני נתונים של אובייקט משוואה.

וכאן האלגוריתם העיקרי מגיע ותפקידו לעבור באופן שיטתי על כל איבר בפונקציה ובו לטפל באופן אישי.

****











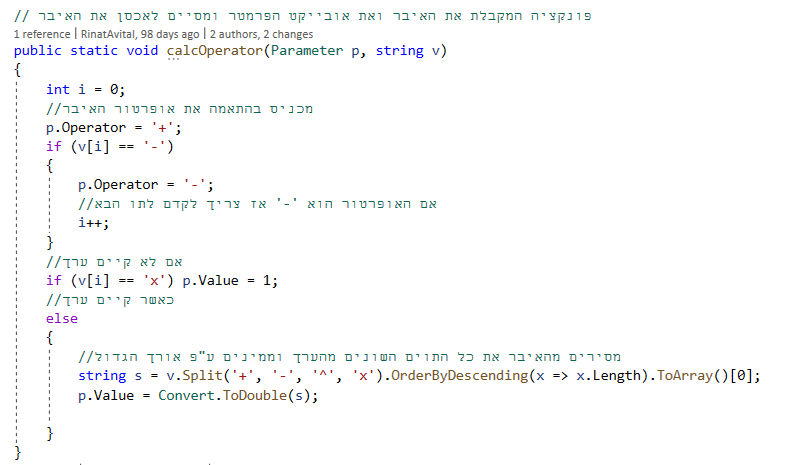
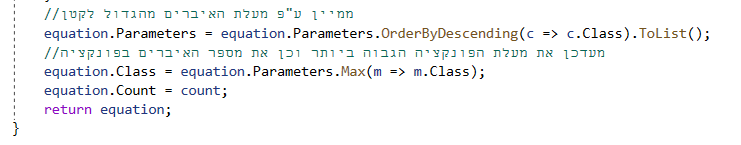








## 

****

ע"י שתי פונקציות אלו נפרק את המחרוזת הפשוטה של גרף הפונקציה ל אובייקט משוואה שבו מאוכסנים נתוני הפונקציה.

הפונקציה getEquationFromStr מקבלת מחרוזת משוואה ומסירה ממנה רווחים מיותרים ויוצרת ורווח לפני כל איבר חדש במשוואה שע"י כך נוכל לדעת איכן מתחיל איבר חדש ולפרק אותם למערך של איברים.

הפונקציה עוברת על כל איבר ואיבר במערך האיברים ובודקת אותו באופן אישי. וכן יוצרת אובייקט משוואה ובתוכה מוכלת רשימת פרמטרים – איברים שלתוכה נכניס לכל איבר חדש.

עוברים על כל איבר מהתו האחרון שלו ובודקים אם התו הוא ' X ', אם כן סימן שהמעלה שלו היא 1 ולכן נכניס 1 למעלת האיבר. אם לא נבדוק אם המעלה של האיבר גדולה מ – 1 או שהיא 0 שאז זה אומר שהאיבר הוא מספר בודד.

אם המקום השני מהסוף הוא גג (סימן מעלה – דרגה) אזי נדע שהאיבר האחרון באיבר הוא המעלה של האיבר. אחרת אם לא קיים גג סימן שאין מעלה באיבר והוא רק מספר בודד.

אחרי שחישבנו את מעלת האיבר נעבור לחשב את האופרטור ואת הערך של האיבר ע"י הפונקציה calcOperator. הפונקציה מקבלת את מחרוזת של האיבר ואת אובייקט האופרטור ומכניסה לאובייקט האיבר את האופרטור (+/-) בהתאמה.

אם מחרוזת האיבר במקום הראשון הוא ' X ' אזי ערך האיבר הוא – 1 אחרת נוריד מהמחרוזת ע"י split את כל התווים חוץ מתו מספרי ואז ממינים ולוקחים את המקום הראשון שהוא ערך האיבר. וכאן סיימנו להכניס לאובייקט האיבר את כל נתוני האיבר. וחוזרים בחזרה לפונקציה getEquationFromStr ששם מכניסים את אובייקט האיבר לתוך רשימת הפרמטרים של המשוואה. וכך עוברים על כל אברי המשוואה.

לבסוף כאשר רשימת הפרמטרים מלאה, ברצוני לבדוק את המעלה בגבוה ביותר ואת כמות האיברים משוואה. לכן נמיין את רשימת הפרמטרים ע"פ המעלה של האיברים מהגבוה לנמוך, וניקח את המעלה של האיבר הראשון אחרי המיון. וכן נכניס את כמות האיברים.

וכך יצרנו אובייקט משוואה המכיל מידע על המשוואה: מעלה הגבוה ביותר, כמות האיברים ורשימת פרמטרים – איברים.

**מחלקת TspFunctions מציאת המסלול האופטימלי**

**Traveling salesman problem - בעיית הסוכן הנוסע**

**הגדרה והסבר:**

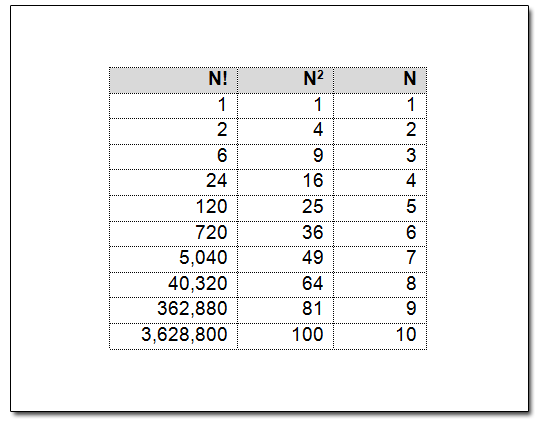
בעיית 'הסוכן הנוסע' היא בעיה ידועה בתורת הגרפים ובסיבוכיות, המיוצגת ע"י בעיה של סוכן נוסע. הסוכן צריך לעבור בערים רבות המחוברות בניהן ברשת כבישים, בעייתו של הסוכן הנוסע היא - בחירת המסלול הקצר ביותר העובר בין כל ערי היעד שלו.

הפתרון "הברוטלי" לבעיה זו, הוא בדיקת כל המסלולים האפשריים. אבל, בבעיה שבה יש n ערים, יש לבדוק מספר ענק של !n (עצרת) אפשרויות - מה שהופך שיטה זו ללא מעשית ולא יעילה גם בעזרת מחשבים רבי עוצמה.

פתרון הבעיה הוא קשה מבחינה אלגוריתמית, ועל פי רוב נעשה ניסיון למציאת פתרון הקרוב לאופטימלי, באמצעות אלגוריתם קירוב. אלגוריתם "חמדן" (סוג של אלגוריתם המעדיף את הפתרון המשתלם הניכר בטווח הבדיקה מידי) אף הוא משמש כדרך מעשית לפתרון שאינו מבטיח אופטימליות.

נדייק בניסוח הבעיה: גרף הוא מבנה מתמטי המאפשר להציג בעיות רבות ושונות מחיי היום-יום. יש בו קודקודים (או צמתים) וצלעות (או קשתות). דוגמאות אפשריות: מפת כבישים, רשת מסילות ברזל, רשת צינורות להולכת מים ומפות של מדינות. במקרה של הסוכן הנוסע נתונה מפה ובה מסומנות ערים. כל עיר היא קודקוד, בין כל שתי ערים יש כביש (שהוא צלע) ועליו כתוב ארכו. ברצוננו למצוא מסלול העובר דרך כל הערים אשר ארכו מינימאלי (מבין כל המסלולים האפשריים). זוהי בעיית חישוב: יש לחשב את המסלול הקצר ביותר העובר דרך כל הערים.  
דרך אחרת לניסוח הבעיה היא לא כבעיית חישוב אלא כבעיית הכרעה: נוסף לטבלת המרחקים (בין כל שתי ערים) הנתונה, נתון גם אורך המסלול המבוקש. השאלה היא האם קיים מסלול העובר דרך כל הערים, אשר ארכו אינו עולה על האורך הזה. התשובה המתבקשת פה היא "כן" או "לא".

לא קשה להבין שהבעיה, בשני ניסוחיה, פתירה, שהרי תמיד אפשר לעבור על כל המסלולים האפשריים. צריך רק לבחור תמורה (פרמוטציה) של הערים, ולעבור בין הערים לפי סידרן בתמורה. כך, אם מספר הערים הואn , אז יש בסך הכל יש n! מסלולים אפשריים (כי אסור לעבור בעיר אחת פעמיים) – זה מספר סופי. לכל מסלול כזה אפשר לחשב (והפעם בקלות) את ארכו – ולבחור את הקצר ביותר.

אם כן, מה הבעיה? הבעיה שמספר הפעולות גדול מדי – הוא גדול יותר מכל חזקה קבועה של N כאשר N הולך וגדל) .לכן האלגוריתם הזה אינו יעיל)

הטבלה הבאה מדגימה כיצד פונקציית העצרת גדלה הרבה יותר מהר מפונקציה ריבועית. פונקציית העצרת "מנצחת" גם כל חזקה קבועה אחרת.

בעיית הסוכן הנוסע, בגרסתה ההכרעתית (כלומר זאת ששואלת האם קיים מסלול שארכו פחות ממספר נתון) היא בעיה NP-קשה. מהי בעיה NP-קשה? מהם ראשי התיבות הללו? הדבר קשור ליעילות של אלגוריתמים.

יעילותו של אלגוריתם נמדדת לפי מספר פרמטרים. המרכזי שבהם הוא זמן הריצה – מספר הפעולות הנדרשות בתלות בגודל הקלט.

איך מודדים את גודל הקלט? במקרה של הסוכן הנוסע, גודל הקלט הוא מספר הערים (כלומר מספר הקודקודים). נהוג לומר שאלגוריתם הוא יעיל אם מספר הפעולות שהוא עושה הוא פולינומי בגודל הקלט, כלומר, מספר הפעולות קטן מגודל הקלט בחזקה קבועה. למחלקת בעיות ההכרעה שניתנות לפתרון במספר פולינומי של פעולות מחשב קוראים בשם P – קיצור שלPolynomial . מהן פעולות המחשב המותרות? מדובר בפעולות דטרמיניסטיות, כלומר אם נריץ תוכנית מחשב המשתמשת רק בפעולות דטרמיניסטיות פעמיים על אותו הקלט, מובטח שנגיע לאותו הפתרון. הייתכן אחרת? כן, אם נצרף למשפחת הפעולות גם ניחושים התמונה עשויה להשתנות. ואכן, בעיות הכרעה שניתן לפתור במספר פולינומי של פעולות, אבל בין הפעולות מותר גם לנחש ניחושים (מוצלחים) נקראות NP – Nondeterministic Polynomial כלומר, מותר במהלך האלגוריתם לקבל החלטות על סמך ניחושים, וקיימים ניחושים מוצלחים שיביאו אותנו לתוצאה המבוקשת. ההגדרה הזאת אולי נראית מסובכת ובלתי ברורה, אבל אפשר להסיק ממנה, די בקלות, אפיון אחר של המחלקה NP: בעיית הכרעה שייכת למחלקה NP אם, עם קצת עזרה מחבר, ניתן לוודא את נכונות פתרונה. הכוונה היא לכך שאם חבר טוען שהוא פתר בעיה כזאת (לחיוב) אז הוא יכול לספק לנו הוכחה, שאותה יהיה לנו קל לבדוק. לדוגמא, אם בבעיית הסוכן הנוסע יטען חברנו שיש לו מסלול העובר בכל הערים, והמסלול קצר יותר מהחסם הנתון, אז נבקש ממנו את המסלול ובקלי קלות (כלומר בזמן פולינומי) נוכל לבדוק את אורך המסלול ונשתכנע בצדקתו.

**סקירה על סוגי פתרונות TSP מוצעים:**

* [אלגוריתמים](https://www.hamichlol.org.il/%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9D) למציאת פתרון מדויק (יעבדו בזמן סביר רק עבור מספר קטן של ערים).
* תכנון אלגוריתמים [היוריסטים](https://www.hamichlol.org.il/%D7%94%D7%99%D7%95%D7%A8%D7%99%D7%A1%D7%98%D7%99%D7%A7%D7%94) למציאת קירובים טובים לפתרון, אך לא פתרונות מדויקים מוכחים.
* מציאת מקרים פרטיים של הבעיה להם קיימים פתרונות מדויקים.

למרות שהבעיה קשה לחישוב, כיום ידועות מספר [היוריסטיקות](https://www.hamichlol.org.il/%D7%94%D7%99%D7%95%D7%A8%D7%99%D7%A1%D7%98%D7%99%D7%A7%D7%94) לפתרונות מקורבים שלה.

דוגמה ל[אלגוריתם חמדן](https://www.hamichlol.org.il/%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9D_%D7%97%D7%9E%D7%93%D7%9F) המחפש קירוב למסלול הקצר ביותר היא "שיטת השכן הקרוב". בשיטה זו בכל צעד הסוכן יפנה לעיר הקרובה אליו ביותר שעדיין לא ביקר בה. בעזרת אלגוריתם זה ניתן לקבל במהירות מסלול קצר ויעיל. עבור מספר מסוים של ערים בפיזור אקראי לרוב יתקבל מסלול ארוך ב-25% מהמסלול הקצר ביותר. עם זאת, קיימים סידורי ערים מיוחדים שגורמים לאלגוריתם זה להניב את התוצאה הגרועה ביותר (!).

שיטות נוספות הן הגבלות חיפוש, בעת ריצת האלגוריתם מחפשים את היעד הבא לבקר בו. ניתן להגביל את החיפושים כדי שזמן הריצה יהיה סביר.

מגבלות חיפוש נפוצות: הגבלת זמן חיפוש, הגבלת חיפוש לכל איטרציה (שכן) ועוד.

**מימושי פתרונות TSP בפרויקט:**

בתרגום הבעיה לנתונינו, הסוכן הוא הקונה, והערים הם המוצרים. אם כי השוני הבולט הוא בפריסת השטח - מרחק בין ערים אינו דומה למרחק בין מוצרים בחנות.

הפועל היוצא מהבדל זה הוא הצורך בדיוק מושלם. כמו כן, ניתן לראות כי התכנית מתאימה לקלטים קטנים ופחות לרשימות מוצרים ארוכות, כך שניתן להשתמש בפונקציה המדויקת. עובדה זו השפיעה על ההחלטה לכתוב קוד עצמאי, נפרט את השיקולים שהביאו לקבלתה:

הרעיון של ניווט ומציאת מסלול קיים כבר ומיושם בהרבה מאד מערכות, וישנם אף שירותים חיצונים של חישובי מרחק (כגון ספרית orTools של google,tsplib.net העוטפת את ספרית TSPLIB) שממשים זאת ומאפשרים שימוש נוח בלא להזיע כמעט.

לאחר ניסויים ובדיקות נוכחנו לראות כי הפתרונות הנ"ל אינם מתאימים ואינם נצרכים עבורנו. שירותים אלו יפים ונכונים במערכות הנדרשות להם כגון חישובי TSP עם אילוצי קיבולת\זמן, כשהשטחים הנידונים הם מרחקי ערים/ארצות וכו'. אולם המערכת שלנו, לעומת זאת, עובדת על שטח פנים שאף אם מתפרס מאד - אינו חוצה מרחקים, לכן דרישתה העיקרית היא דיוק, ומקסימלי. מעבר לכך החישוב הוא TSP בסיסי ללא אילוצים, ועם הרבה רצון נוכל בהחלט להצליח להתמודד אתו בכוחות עצמינו.

לכן החלטנו לקחת את כתיבת אלגוריתם TSP יעיל כיעד בפרויקט, וניתן לומר שזהו יעד סופר מאתגר.

פרוט על המחלקה TspFunctions :

במחלקה זו קיימת פונקציה ManageTsp – הפונקציה הראשית. הנקראת ע"י מחלקת TravelComputer. היא מקבלת את מטריצת המרחקים של האזורים ומפעילה את פונקציות הקירוב השונות או לחילופין את הפונקציה המדויקת במידה ומדובר בעד 10 אזורים בלבד.

פונקציות הקירוב הן בעצם האלגוריתמים [היוריסטים](https://www.hamichlol.org.il/%D7%94%D7%99%D7%95%D7%A8%D7%99%D7%A1%D7%98%D7%99%D7%A7%D7%94) המוזכרים. כרגע השתמשנו ב-3 שיטות שונות, אולם מבנה המחלקה מאפשר דינמיות וניתן להוסיף צורות חישוב נוספות.

במקרה שתתקבל תוצאה זהה בכמה פונקציות קירוב, נעדיף את המסלול שחושב ע" ה- Greedytsp היות ומסלול זה, איסוף המוצר הקרוב ביותר בכל פעם, היא הנוחה והאינטואיטיבית ביותר למשתמש האנושי.

את הפונקציות מימשנו במקביליות כדי לקבל תוצאה מיטבית במינימום זמן, הרצה במקביליות התאפשרה ע"י שימוש ביכולות אסינכרוניות של תשתיות תוכנה המובנות ב.net כשכך מופעלות פונקציות הקירוב כולן בו זמנית, כל אחת מטופלת ע"י Task אחר.

**בואו תראו את זה קורה:**

public static int[] ManageTsp(Cell[,] matrixToTsp, int dest,int lenMat)

{//הפונקציה תפעיל סוכן נוסע מתאים

int lenResult = dest + 1;

bool[] visited = new bool[lenResult];

//the result array should be of type 'answer'.

int[] result = new int[dest+1];

int[] arr = new int[lenResult];

for (int i = 0; i < lenResult; i++)

{

result[i] = 0;

arr[i] = 0;

}

// Mark 0th node as visited

visited[0] = true;

int ans = int.MaxValue;

int curPos = 0, count = 1, cost = 0, startIndex=0; bool aprox = false;

//עבור מספר קטן

if (lenResult <= 10)

{

ans = OurTsp(matrixToTsp, visited,curPos, lenMat, count, cost, ans, result, arr,

aprox, dest);

return result;

}

else

{

int[] greedyWay = new int[lenResult], approximatelyWay = new int[lenResult],

randomWay = new int[lenResult], BestRandomWay=new int[lenResult];

//מקביליות

Task<int> greedyTask = Task<int>.Factory.StartNew(() => Greedytsp(matrixToTsp, lenMat,

startIndex, greedyWay, dest));

Task<int> approximateTask = Task<int>.Factory.StartNew(() =>

ApproximatelyTsp(matrixToTsp, visited,curPos, lenMat, count, cost, ans, arr,

startIndex, approximatelyWay));

Task<int> randomtask = Task<int>.Factory.StartNew(() => ManagerandomTsp(matrixToTsp,

lenMat, startIndex,dest, randomWay, BestRandomWay));

//החזרת האופטימלי ביותר

if (greedyTask.Result <= approximateTask.Result && greedyTask.Result <=

randomtask.Result)

return greedyWay;

if (randomtask.Result < approximateTask.Result)

return BestRandomWay;

return approximatelyWay;

}

}

הפונקציה הראשית יוצרת taskים – משימות כמספר פונקציות הקירוב, כשכל משימה אחראית על זימון פונקציה אחרת. הפונקציה הראשית תשווה את תוצאות הפונקציות ותבחר את האופטימלית מבניהם כך שהתוצאה הסופית תהיה יעילה ביותר.

הפונקציות ישתמשו בפונקציה BestCash לבחירת נקודת הקופה האופטימלית, הקרובה ביותר לאינדקס הקודם שנבחר. פונקציה זו עוברת בלולאה החל מהאינדקס dest המציין את אינדקס הקופה הראשונה, עד לסיום המעבר על המערך, שגודלו התקבל כ-lenMatrix.

public static int BestCash(int dest,int lenMatrix,int beforeIndex,Cell[,] matrixToTsp)

{

int sumMin = int.MaxValue, iMin = 0;

for (int k = dest; k < lenMatrix; k++)

{

if (matrixToTsp[beforeIndex, dest].distance < sumMin)

{ iMin = k; sumMin = matrixToTsp[beforeIndex, dest].distance; }

}

return iMin;

}

הפונקציה OurTsp היא המדויקת, כלומר מחזירה בהכרח תשובה אופטימלית. היא עוברת על כל המסלולים האפשריים (n!) ומחזירה את המסלול האופטימלי. כמובן ניתן לבצע זאת רק עבור קלט נמוך במיוחד. (10). בהמשך נציג את הקוד שלה. השם שבחרנו לה משדר את רגש הפטרונות והשייכות לדבר שכל כך טרחנו עליו..

**פונקציית קרוב 1:** הפתרון המוגבל;

הפונקציה ApproximatelyTsp מוגבלת בזמן חיפוש, ומבצעת את הפונקציה המדויקת במשך 10 שניות [היות והמערכת צריכה לספק ללקוח תשובה במהירות ככל האפשר]. הפונקציה תחזיר את התוצאה האופטימלית ביותר שהושגה במסגרת הגבלת הזמן.

הפונקציה OurTsp מקבלת משתנה בוליאני approx המציין האם הפונקציה תרוץ בצורה מוגבלת או לא.

public static Stopwatch stopwatch = new Stopwatch();

public static int OurTsp(Cell[,] MatrixToTsp, bool[] visited, int currPos,

int lenMat, int count, int cost, int ans, int[]

ResultIndexes, int[] arr, bool approx, int dest)

{

if (approx)

{

if (currPos == 0)

stopwatch.Restart();

else

if (stopwatch.Elapsed.Seconds >= 10)

return ans;

}

//אם זה החוליה האחרונה וגם יש גישה לחולית היעד, החוליה הנתונה

if (count == dest && MatrixToTsp[currPos, dest].distance > 0)

{

//בוחר את היעד הכי קרוב [הקופה שהכי קרובה למיקום שנמצא בו

int iMin = BestCash(dest, lenMat, currPos, MatrixToTsp);

cost+= MatrixToTsp[currPos, iMin].distance;

if (cost < ans)

{

ans = cost;

arr[dest] = iMin;

//תעתיק את המערך למערך התוצאה

for (int i = 0; i < dest+1; i++)

{

ResultIndexes[i] = arr[i];

}

}

return ans;

}

//תעבור על הטיולים של הרשימה הסמוכה לנקודה הנוכחית ותגדיל את המונה באחד ותתמחר עי גרף הסמיכויות

for (int i = 0; i <= dest; i++)

{

//אם זה לא החוליה הנוכחית וגם עוד לא עברו שם אז תעבור שם

if (i != dest && visited[i] == false && MatrixToTsp[currPos, i].distance > 0)

{

visited[i] = true;

arr[count] = i;

ans = OurTsp(MatrixToTsp, visited, i, lenMat, count + 1,

cost + Convert.ToInt32(MatrixToTsp[currPos, i].distance), ans, ResultIndexes,

arr, approx, dest);

// Mark ith node as unvisited

visited[i] = false;

}

}

return ans;

}

פונקציית ApproximatelyTsp המפעילה את OurTsp כשהמשתנה approx הוא חיובי.

private static int ApproximatelyTsp(Cell[,] matrixToTsp, bool[] visited, int currPos,

int lenMatrix, int count, int cost, int ans, int[] arr, int

startIndex, int[] approximatelyWay)

{

ans=OurTsp(matrixToTsp, visited, currPos, lenMatrix, count, cost, ans,arr, approximatelyWay,true,lenMatrix-1);

return ans;

}

**פונקציית קירוב 2:** הפתרון החמדני;

הפונקציה Greedytsp מממשת אלגוריתם חמדן בשיטת השכן הקרוב שהוזכרה.

public static int Greedytsp(Cell[,] matrixToTsp, int lenMatrix, int startIndex, int[]

resultIndex, int dest)

{

int lenResult = dest + 1;

int[] Moved = new int[lenResult];

Moved[startIndex] = 1;

int Path = 0;

int currentIndex = startIndex;

int min=0;

int minIndex = -1;

resultIndex[0] = startIndex;

for (int j = 0; j <dest-1; j++)

{

min = int.MaxValue;

for (int i = 0; i < dest; i++)

{

if (Moved[i] != 1 && matrixToTsp[currentIndex, i].distance != 0)

if (min > matrixToTsp[currentIndex, i].distance)

{

min = matrixToTsp[currentIndex, i].distance;

minIndex = i;

}

}

resultIndex[j + 1] = minIndex;

currentIndex = minIndex;

Moved[currentIndex] = 1;

Path += min;

}

//TODO: לבחור נקודת סיום אופטימלית

int iMin = BestCash(dest, lenMatrix, currentIndex, matrixToTsp);

resultIndex[dest] = iMin;

Path += matrixToTsp[currentIndex, iMin].distance;

return Path;

}

**פונקציית קירוב 3:** הפתרון הרנדומלי;

הפונקציה RandomTsp משתמשת גם היא ב Stopwatch להגבלת זמן. משימתה בעצם להגריל מסלולים אפשריים בטווח הזמן הקצוב כאשר כל הגרלה מושווית להגרלה שאחריה וזו האופטימלית יותר ממשיכה לסיבוב הבא.

הרציונל שעומד מאחורי רעיון הניחושים (שעשוי אולי להשמע די אוילי) הוא לספק פתרון המקביל-משלים את הפתרון המוגבל, ה-approximatelyTsp.

בשימוש ב- approximatelyTspיתבצע חיפוש מסלול אופטימלי לפי הסדר, תוך בדיקת האפשרויות הראשונות, עד לעצירה.

במידה והפתרון האופטימלי האמיתי מצוי דווקא לא לפי הסדר, יש סיכוי שלא יצליחו להגיע אליו מספיק מהר. לכן חיוני להשתמש באלגוריתם רנדומלי שאמנם הוא יוצא מחוץ לקופסא, אך יכול להשלים ולהגיע לפתרון אופטימלי שהפונקציה המוגבלת הקודמת לא הגיעה אליו. ואכן פתרונותיו יכולים להיטיב את התוצאות. הסבר מפורט על כך מובא לעיל אודות בעיות NP -קשות.

הפונקציה ManagerandomTsp מפעילה בלולאה את RandomTsp, דואגת לשמור את תוצאות המסלול המוגרל בה בכל פעם, ותמיד מחזיקה את המסלול הטוב ביותר שהוגרל עד כה. בדרך זו תוכל להחזיר בסיום הריצה (בתום הזמן) את המסלול האופטימלי.

private static int ManagerandomTsp(Cell[,] matrixToTsp, int lenMatrix, int

startIndex,int dest,int[] randomyWay,int [] BestRandomWay)

{

int ans = int.MaxValue, temp;

stopwatch.Restart();

while (stopwatch.Elapsed.Seconds < 10)

{

temp = RandomTsp(matrixToTsp, lenMatrix, startIndex,dest, randomyWay);

if (temp < ans)

{

ans = temp;

for (int i = 0; i < randomyWay.Length; i++)

{

BestRandomWay[i]=randomyWay[i];

}

}

}

return ans;

}

private static int RandomTsp(Cell[,]matrixToTsp, int lenMatrix, int startIndex,int

dest, int[] randomyWay)

{

for (int i = 0; i < dest+1; i++)

{

randomyWay[i] = -1;

}

int ans = 0;

Random rnd = new Random();

int matrixIndex=1;

randomyWay[0] = startIndex;

int randomIndex = startIndex, prev;

for(int i = 1; i < dest; i++)

{

prev = randomIndex;

while (randomyWay[randomIndex]!=-1)

randomIndex = rnd.Next(1, dest);

randomyWay[randomIndex] = matrixIndex++;

ans += matrixToTsp[prev, randomIndex].distance;

}

// לבחור נקודת סיום אופטימלית

int iMin = BestCash(dest, lenMatrix, randomIndex, matrixToTsp);

randomyWay[dest] = iMin;

ans += matrixToTsp[randomIndex, iMin].distance;

return ans;

}

# **תחומי לוגיקה נוספים**

**Alias**

כשלקוח מזין את רשימת הקניות שלו, הוא זקוק לממשק נוח, חכם ומהיר, שלא ידרוש ממנו מאמץ וזמן רב.

בממשק שפיתחנו הלקוח מתחיל להקיש את שם המוצר, ומקבל אופציות להשלמה שמסוננות לפי הטקסט שמקיש. הבעיה היא שהלקוח אינו יודע את הכינוי שבעלי החנות הגדירו לכל מוצר [הלקוח מקיש 'אבקת שוקו' אך שם המוצר הוא 'שוקולית'].

חיפוש חכם ושימוש בספריות שונות יכול אולי לפתור את העניין, אך היות שנקודה זו אינה נקודה מרכזית בפרויקט שלנו, אך בכל זאת נרצה לתת אפשרות לחיפוש נוח, הטלנו את התפקיד על בעלי החנויות, שאחראיים על הזנת הנתונים. נתונים אלו הם שיתופיים, כלומר, כלל בעלי החנויות יוכלו להשתמש בנתונים שהוזנו ע"י בעל חנות.

במסד הנתונים הוספנו טבלה בשם 'Alias', בטבלה זו קיימים כל שמות המוצרים. לכל מוצר במערכת יש שם ראשי, וכן יש רשימת כינויים נוספים.

כשלקוח בוחר חנות, התוכנית מבקשת מהשרת את כל הכינויים [הראשיים והמשניים] שקיימים בחנות. היות שמאגר הנתונים משותף, ניתן לשער שלכל מוצר ישנם כינויים רבים, וכך הלקוח כותב את הרשימה בשפתו והמערכת תוכל 'להבין' למה כוונתו.

**מחלקות**

כיון שהתכנית דינאמית, אפשרנו לבעל החנות למקם כל מוצר בכל מקום ללא הגבלת מחלקות. מבחינת המערכת- הטיפול במחלקות מיותר, אך יצרנו אותו לתצוגת המשתמש.

בטבלת Alias ישנו ערך parent המכיל עבור כל כינוי ילד קוד כינוי אחר, של אביו. הדבר מאפשר יכולות רבות- קינון היררכי של כינויים, כך שכאשר מקשרים כינוי למוצר יודעים עליו דברים רבים [כששייכתי ל'שוש' את ההורה 'במבה' אני יודעת עליו שהוא מכיל בוטנים, חטיף, חטיף מלוח, ממתק, וכו'].

אפשרויות אלו אנו נותנות כ'אופציה לפיתוח עתידי'. למעשה, הלקוח אינו חשוף לצורת שמירת הנתונים. מבחינת התצוגה קיימת הפרדה מוחלטת בין המחלקות לשאר שמות המוצרים.

כרגע הכינויים מתחלקים ל-2: כינויים שה-parent שלהם ריק - והם כינויים למחלקות, ובשאר הכינויים בparent יש קוד כינוי מהסוג הראשון.

בעל החנות יכול לשמור כינויים לקירות, לסטנדים ולמדפים. כינויים אלו יהיו מסוג 'מחלקה'. נשתמש במידע זה לתצוגת החנות. למחלקות קיים שדה 'Color' המאפשר צביעת המפה בצבעים המתאימים למחלקות. בנוסף, כשבעל החנות בוחר מוצרים שקיימים בחנות מתוך מאגר המוצרים, יכול לעבור בין המחלקות ולראות את המוצרים בצורה ממוינת ונוחה.

**ExtraAlias**

במבנה הנתונים ההיררכי המוצרים מכילים את הכינויים, אך הכינויים אינם 'יודעים' באיזה מוצר הם מופיעים. לכן, בשביל לחסוך שאילתות מרובות, השרת ישלח רשימת ExtraAlias במקום רשימת Alias.

המחלקה ExtraAliasהיא מחלקה היורשת מ Aliasאך מכילה גם רשימת מוצרים. לאחר שהלקוח יבחר כינוי ויבחר את המוצר מתוך רשימת המוצרים שיש להם את הכינוי הזה [או, יבחר כברירת מחדל המוצר הראשון], תתבצע שאילתה שתמצא את מיקום המוצר בחנות.

הפונקציה שמחשבת את הExtraAlias של החנות ומוסיפה להם את המוצרים:

public static List<ExtraAlias> GetAliasesShop(int codeShop)

{

GeneralDB generalDB = new GeneralDB();

Shop s = generalDB.MyDb.Shops.Where(x => x.Code == codeShop).First();

List<Product> products = s.ProductShops.Select(x=>x.Product).ToList();

List<ExtraAlias> extraAliases = new List<ExtraAlias>();

foreach (var item in products)

{

foreach (var item1 in item.ProductAlias)

{

extraAliases.Add(Converts.ToExtraAlias(item1.Alias));

}

extraAliases.Add(Converts.ToExtraAlias(item.Alias));

}

extraAliases.Distinct();

extraAliases = extraAliases

.GroupBy(p => p.Code)

.Select(g => g.First())

.ToList();

GetAliasesShopProduct(codeShop, extraAliases);

return extraAliases;

}

public static void GetAliasesShopProduct(int codeShop, List<ExtraAlias> extraAliases)

{

GeneralDB generalDB = new GeneralDB();

List<Product> products = generalDB.MyDb.Products.Where(x => x.ProductShops.Where(y => y.CodeShop == codeShop).Count() > 0).ToList();

List<dtoProduct> dtoProducts = Converts.ToDtoProducts(products);

ExtraAlias e;

int? codeAlias;

foreach (var product in dtoProducts)

{

codeAlias = product.CodeAlias;

e = extraAliases.Where(eAlias => eAlias.Code == codeAlias).First();

e.products.Add(product);

if(product.ProductAlias!=null)

product.ProductAlias.ForEach(productAlias =>extraAliases.Where(eAlias => eAlias.Code == productAlias.Alias.Code).ToList().First().products.Add(product));

}

}

המחלקה ExtraAlias מסייעת גם למסך בחירת מוצרים לחנות. חלוקת המוצרים למחלקות מסייעת לבעל החנות באיתור המוצרים הרלוונטים לו בקלות ונוחות.

גם הפעם השרת ישלח רשימה של כינויים מסוג ExtraAlias, כשהפעם הכינויים הם מחלקות בלבד, ולכל מחלקה רשימת מוצרים מהמאגר הכללי השייכים למחלקה.

public static List<ExtraAlias> GetDepartment()

{

GeneralDB generalDB = new GeneralDB();

List<Alias> aliases = generalDB.MyDb.Aliases.Where(x => x.IsMainCategory == true).ToList();

List<ExtraAlias> departments = Converts.ToExtraAliases(aliases);

List<dtoProduct> dtoProducts = Converts.ToDtoProducts(generalDB.MyDb.Products.ToList());

ExtraAlias e;

foreach (var item in dtoProducts)

{

if(item.Alias.Parent!=0)

{

e = departments.Where(x => x.Code == item.Alias.Parent).First();

e.products.Add(item);

}

}

return departments;

}

}

**ממשק I2Points**

לצורך החישובים כתבנו פונקציות לחישוב מרחק, אמצע קטע, המשמשות בד בבד סטנדים, נקודות גישה ואזורים. כל אחד טיפוס מחלקה אחר. המשותף היחיד בין מחלקות אלו הוא 2 הנקודות הקיימות בכולם.

לכן, הגדרנו את המחלקות dtoStand, dtoGetaway ,Area כמממשות את הממשק I2Points, מה שאפשר דינאמיות וחישוב מרחק בין אזור לנקודת גישה, בין סטנד לאזור, וכו'.

במקומות נוספים בקוד, ניתן לראות שהשתמשנו I2Points היות שלא ניתן לצפות מראש מאיזה טיפוס בדיוק יהיה.

public static int MinDist(I2Points s1, I2Points s2)

{

Point ps1 = MidPoint(s1.P1, s1.P2);

Point ps2 = MidPoint(s2.P1, s2.P2);

return CalculteDist(ps1, ps2);

}

פונקציה זו מחשבת מרחק בין 2 אובייקטים המממשים את I2Points ע"י חישוב אמצע קטע של כל אחד מהם, ואח"כ חישוב נוסחת מרחק ביניהם.

**מציאת נקודות גישה**

להלן אלגוריתם נסיוני היוצר נקודות גישה. האלגוריתם מטייל מ-4 קודקודיו של כל קיר לכל הכיוונים, במטריצה המייצגת את החנות [של אפסים ואחדות] על מנת למצא קיר נוסף מולו ולחבר ביניהם בנקודת גישה.

public static void FindGetaways(int [,] mat, dtoWall[] walls)

{

List<dtoGetaway> getaways = new List<dtoGetaway>();

foreach (var w in walls)

{

Point p1 = w.P1, p2 = w.P2, p3 = new Point(p2.X, p1.Y), p4 = new Point(p1.X, p2.Y);

Point? p1Up = GetGetawayPoint(p1, mat, eSide.Up);

if (p1Up != null) getaways.Add(AddGetaway(p1Up, p1));

Point? p2Up = GetGetawayPoint(p2, mat, eSide.Up);

if (p2Up != null) getaways.Add(AddGetaway(p2Up, p2));

Point? p3Down = GetGetawayPoint(p3, mat, eSide.Down);

if (p3Down != null) getaways.Add(AddGetaway(p3Down, p3));

Point? p4Down = GetGetawayPoint(p4, mat, eSide.Down);

if (p4Down != null) getaways.Add(AddGetaway(p4Down, p4));

Point? p2Right = GetGetawayPoint(p2, mat, eSide.Right);

if (p2Right != null) getaways.Add(AddGetaway(p2Right, p2));

Point? p4Right = GetGetawayPoint(p4, mat, eSide.Right);

if (p4Right != null) getaways.Add(AddGetaway(p4Right, p4));

Point? p1Left = GetGetawayPoint(p1, mat, eSide.Left);

if (p1Up != null) getaways.Add(AddGetaway(p1Left, p1));

}

}

public static dtoGetaway AddGetaway(Point? p1,Point? p2)

{

dtoGetaway g = new dtoGetaway();

g.P1 = p1;

g.P2 = p2;

return g;

}

private static Point? GetGetawayPoint(Point p, int[,] mat, eSide side)

{

//תחפש ממול לנקודת ההתחלה מקום שהוא לא ריק- קיר או וילון?

//וכן מנקודת הסוף

//go from i to j

int i;

int j;

int end;

Point pResult = new Point();

if (side == eSide.Up)

{

i = (int)p.X + 1;

j = (int)p.Y;

pResult.Y = p.Y;

pResult.X = i;

end = N;

while (i <= end && mat[i, j] != 1) i++;

pResult.X = i;

if (i == end)

return null;

}

else if (side == eSide.Down)

{

i = (int)p.X + 1;

j = (int)p.Y;

pResult.Y = p.Y;

pResult.X = i;

end = 0;

while (i >= end && mat[i, j] != 1) i--;

pResult.X = i;

if (i == end)

return null;

}

else if (side == eSide.Left)

{

i = (int)p.Y - 1;

j = (int)p.X;

pResult.Y = i;

pResult.X = p.X;

end = 0;

while (i >= 0 && mat[j, i] != 1) i--;

pResult.Y = i;

if (i == end)

return null;

}

else if (side == eSide.Right)

{

i = (int)p.Y;

j = (int)p.X;

pResult.X = p.X;

pResult.Y = i;

end = M;

while (i <= end && mat[j, i] != 1) i++;

pResult.Y = i;

if (i == end)

return null;

}

return pResult;

}

האלגוריתם החזיר תוצאות נחמדות, אך לא טיפל בהרבה מקרים יוצאי דופן.

למעשה, היות ולא טיפלנו במיפוי החנות ובממשק להוספת חנות ע"י בעל החנות, לא השתמשנו באלגוריתם זה.

# **ממשק משתמש**

צד הלקוח פותח באנגולר ובו כלול ממשק המשתמש עבור מיפוי החנות ועריכת קניה.

צד הלקוח משרת את בעלי החנויות המעוניינים ביצירת חנות וירטואלית ואת הלקוחות המעוניינים לייצר בקלות את רשימת הקניות שלהם ולאסוף את המוצרים באופן אופטימלי.

**החלקים העיקריים בפיתוח הממשק הם ארבעה:**

* כניסת המנהל לחנות באמצעות סיסמא ועריכת שינויים בפרטי החנות.
* הצגת מפת החנות ללקוח, וכן הצגת אייקונים במיקומים הנבחרים.
* עריכת רשימת קניות.
* שרטוט מסלול במפה.

**המסכים:**

מסך פתיחה ובחירת חנות, שם ניתן לבחור אם נכנסים כמנהל או כלקוח.

מסכי לקוח:

מסך לעריכת רשימה והצגת מפת חנות. מוצרי הרשימה מצוינים במפה ובעת לחיצה מוצג שרטוט מסלול.

מסכים לבעל חנות:

מסך התחברות באמצעות סיסמא.

מסך הוספת חנות.

מסך הוספת מוצרים חדשים למאגר מוצרים, הוספת מוצרים לחנות,

הוספת שמות למאגר שמות, הוספת שמות למוצרים בחנות.

**הצגת מפת החנות ללקוח**

אחד מיעדי הפרויקט החשובים הוא הצגת החנות בצורה יפה ומושכת.

הקדשנו זמן, מחשבה ומעוף וניסינו אפשרויות רבות. אחת מהן היתה שרטוט תלת ממדי שמאפשר טיול בתוך החנות – דרך המסך. מעשית אין בכך ענין כי הלקוח נמצא בחנות פיזית, וההדמיה המלאכותית מיותרת. עדיין זהו רעיון נחמד, במידה וניתן לביצוע.

במהלך הדרך חקרנו ממשקים של מציאות רבודה וin Door-, ועוד.

פגשנו גם בממשק להדמיה תלת ממדית הנקרא three.js. מדובר בספרית JavaScript לעיצוב 3D עם ביצועים עוצרי נשימה. בדוגמאות המובאות באתר הבית שלה מצאנו דוגמא שלכדה את תשומת ליבנו. לאחר שהתעמקנו גילנו ששימוש בספריה מביא לתוצאות יפות אך לא פרקטי למשתמש.

המסקנה העולה מתהליך היצירה הוא, כי בעיצוב - השמים הם הגבול, וניתן לשפר ולשפץ בלי סוף. עקב מגבלות הזמן, נסתפק בתצוגה פשוטה אך נוחה ונעימה לעין.

**השלבים בעיצוב:**

**1. טבלה**

תחילה, הפכנו את החנות למטריצה של אפסים ואחדות והצגנו אותם בטבלה.

את המטריצה חישבנו באמצעות מחלקת MatShopComputer שבשכבת הלוגיקה. להלן הקוד:

public static int[][] ComputeMat(Wall[] w, Stand[] s, Getaway[] g, int width, int

height)

{

//מצייר מטריצה מהקירות

int[][] mat = new int[width + 1][];

for (int i = 0; i < width+1; i++)

{

mat[i] = new int[height];

}

for (int i = 0; i < width; i++)

for (int j = 0; j < height; j++)

mat[i][ j] = 0;

int x1, x2, y1, y2;

//עובר על הקירות

foreach (var item in w)

{

x1 = Convert.ToInt32(item.X1);

x2 = Convert.ToInt32(item.X2);

y1 = Convert.ToInt32(item.Y1);

y2 = Convert.ToInt32(item.Y2);

for (int i = x1; i <= x2; i++)

for (int j = y1; j <= y2; j++)

mat[i][ j] = 1

}

//עובר על הסטנדים

Point p1;

Point p2;

foreach (var item in s)

{

if (item.Code == 104)

item.Code = item.Code;

p1 = new Point();

p2 = new Point();

if ((item.X1 + item.Y1) < (item.X2 + item.Y2))

{

p1.X = Convert.ToInt32(item.X1); p1.Y = Convert.ToInt32(item.Y1);

p2.X = Convert.ToInt32(item.X2); p2.Y = Convert.ToInt32(item.Y2);

}

else

{

p1.X = Convert.ToInt32(item.X2); p1.Y = Convert.ToInt32(item.Y2);

p2.X = Convert.ToInt32(item.X1); p2.Y = Convert.ToInt32(item.Y1);

}

for (int i = p1.X; i < p2.X + 1; i++)

{

for (int j = p1.Y; j < p2.Y + 1; j++)

{

mat[i][ j] = 2 + item.Code \* 100;

}

}

}

//עובר על הנקודות גישה

foreach (var item in g)

{

p1 = new Point() { X = Convert.ToInt32(item.X1), Y = Convert.ToInt32(item.Y1) };

p2 = new Point() { X = Convert.ToInt32(item.X2), Y = Convert.ToInt32(item.Y2) };

for (int i = p1.X; i < p2.X + 1; i++)

{

for (int j = p1.Y; j < p2.Y + 1; j++)

{

mat[i][j] = 3 + item.Code \* 100;

}

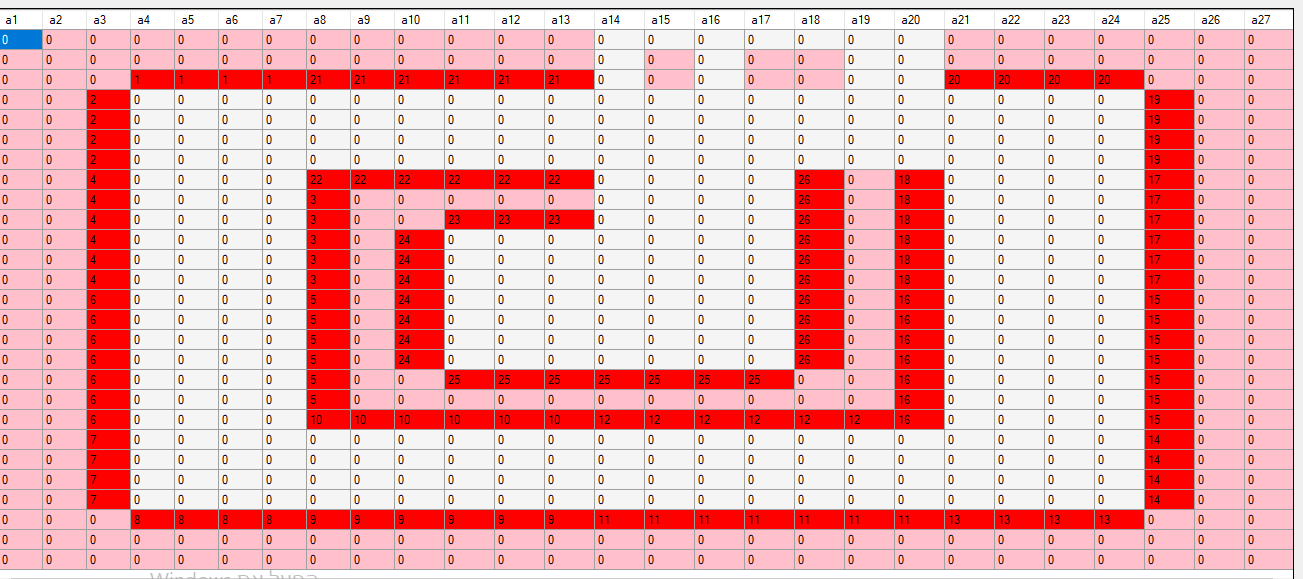
}

}

return mat;

}

שימו לב, בכל הדוגמאות מדובר באותה החנות. עיקבו אחר התמונות, וראו כיצד משתפרים הביצועים.

בתחילה הצגנו את החנות בDataGridViewבממשק WinForm לבדיקה

כשהתחלנו לפתח באנגולר יצרנו טבלה עם תגית table>>, כשצבע כל תא בא מקושר למטריצה שחושבה:

ערכי arrColor הם [white,grey,pink]

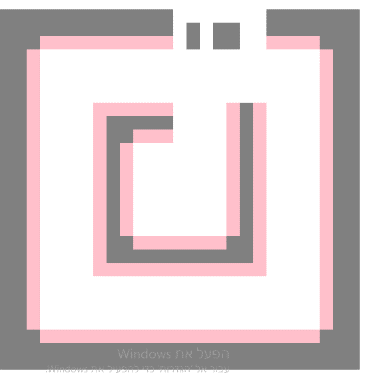
<table border="0">

<tr \*ngFor="let item of mat;let ind1=index" >

<td \*ngFor="let item1 of item;let ind=index" (click)="func(ind1,ind)" [ngStyle]="{'background-color':arrColor[item1%10]}" >

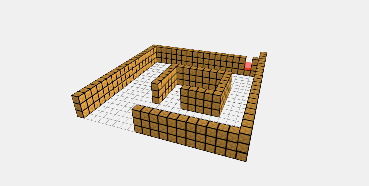
</td>

</tr>

</table>

התקבל השירטוט הבא:

**2. שימוש בספריית three.js**

בדוגמא שמצאנו, שירטטנו את החנות, וכך זה נראה:

(כמובן, לו היינו משתמשות בממשק היינו משדרגות את התצוגה שתראה כחנות. )

**3.** דרך הצגה נוספת היא תמונת רקע של החנות במבט מלמעלה עם שקיפות למפה המכילה מסלול. לא השתמשנו היות וקשה להציג מסלול דינאמי במפה סטטית. בשיטה זו האפשרויות לפיתוח והוספת פונקציונליות יהיו מאד מוגבלות.

**4. ציור חנות באמצעות קנווס**

בדרך הבאה ניתן להבחין בדינאמיות והביצועים המשופרים לקנווס לעומת הטבלה.

בדוגמא זו לא השתמשנו במטריצת אפסים ואחדות של החנות, אלא עברנו על הקירות והסטנדים וציירנו אותם בלולאה באמצעות פונקציה לציור ריבועים:

drawShop()

{

let s:Shop=this.db.getShop();

//drow walls:

// ------------------

this.ctx.fillStyle = "rgb(185, 185, 183)";

let ws:Wall[]=s.Walls;

for(let i:number=0; i<ws.length; i++)

{

let wal:Wall=ws[i];

let p1:Point=wal.P1;

let p2:Point=wal.P2;

let w=(p2.X-p1.X+1)\*this.mul;

let h=(p2.Y-p1.Y+1)\*this.mul;

this.ctx.fillRect (p1.X\*this.mul, p1.Y\*this.mul,w , h);

let squre:Square=new Square(this.ctx);

squre.draw(p1.X\*this.mul,p1.Y\*this.mul, w,h,"rgb(185, 185, 183)");

}

//drow stands

// ------------------

this.ctx.fillStyle = "rgb(146, 146, 146)";

for(let i:number=0; i<s.Stands.length; i++)

{

let st:Stand=s.Stands[i];

let p1:Point=st.P1;

let p2:Point=st.P2;

let w=(p2.X-p1.X+1)\*this.mul;

let h=(p2.Y-p1.Y+1)\*this.mul;

this.ctx.fillRect (p1.X\*this.mul,p1.Y\*this.mul, w,h );

// 3d

let squre:Square=new Square(this.ctx);

squre.draw(p1.X\*this.mul,p1.Y\*this.mul, w,h,"rgb(108, 109, 108)");

// stroke

this.ctx.strokeStyle ="rgb(108, 109, 108)";

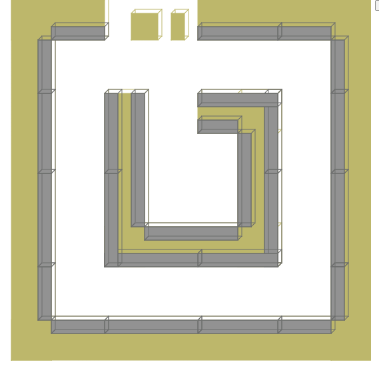
this.ctx.lineWidth =1;

this.ctx.strokeRect(p1.X\*this.mul,p1.Y\*this.mul, w,h);

}

}

התוצאה היא כזאת:



וזוהי הדרך הנבחרת, שמצאה חן בעינינו.

ציור החנות נעשה באמצעות אלמנט [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) מסוג HTMLCanvasElement.

קנווס הוא אזור מלבני בדף, המשמש כבד לציור. קיימות מתודות בעזרתן מציירים עליו.

המחלקה Square מציירת את הממד הנוסף (קוים צידיים) שמיטיב את התוצר:

export class Square {

constructor(private ctx: CanvasRenderingContext2D) {}

draw(x: number, y: number, a: number,b: number,color:string) {

this.ctx.strokeStyle =color;

this.ctx.lineWidth = 1;

this.ctx.strokeRect( x, y, a, b);

this.ctx.strokeRect(x-6,y-6,a,b);

this.ctx.beginPath();

this.ctx.moveTo(x,y);

this.ctx.lineTo(x-6,y-6);

this.ctx.moveTo(x+a,y);

this.ctx.lineTo(x-6+a,y-6);

this.ctx.moveTo(x,y+b);

this.ctx.lineTo(x-6,y-6+b);

this.ctx.moveTo(x+a,y+b);

this.ctx.lineTo(x-6+a,y-6+b);

this.ctx.closePath();

this.ctx.stroke();

}

}

ניתן להיווכח באמצעות מימוש המחלקה הזו, שגם שפת TypeScript היא מונחית עצמים בדיוק כמו שפות Server .

**שירטוט מסלול**

בפונקציה drawPath שרטוט המסלול נעשה באמצעות פונקציות ציור מובנות כ'התחל מסלול', 'עבור לנקודה', ועוד.

הפונקציה מקבלת רשימה של יעדים ומשרטטת מסלול על המסך:

drawPath(Coordinates:Goal[])

{

this.ctx.strokeStyle ="rgb(108, 109, 108)";

this.ctx.lineWidth =5;

this.ctx.beginPath();

this.ctx.moveTo(Coordinates[0].midPoint.X\*this.mul,Coordinates[0].midPoint.Y\*this.mul);

Coordinates.forEach(e =>

{

if(e.kind=='s')

this.ctx.strokeStyle ="rgb(108, 109, 108)";

else

this.ctx.strokeStyle ="rgb(100, 100, 0)";

this.ctx.lineTo(e.midPoint.X\*this.mul,e.midPoint.Y\*this.mul);

});

this.ctx.closePath();

this.ctx.stroke();

}

הפונקציה draw מציירת נקודה בעת לחיצה על המסך על מנת להראות ללקוח היכן לחץ את מיקומו.

draw(e){

let rect = this.canvas.nativeElement.getBoundingClientRect();

var posx = (e.clientX-rect.left)/this.mul;

var posy = (e.clientY-rect.top)/this.mul;

this.ctx.fillStyle = "#000000";

this.pStart.X=posx;

this.pStart.Y=posy;

this.onSelectPoint.emit(this.pStart);

}

מפת החנות ורשימת הקניות נמצאות בשני רכיבים נפרדים. כדי לממשק ביניהם ולאפשר תגובתיות הולמת, השתמשנו ב eventEmmiter. זהו רכיב של אנגולר המאפשר הרמת אירוע בדומה לאירועים מובנים כמו אירועי מקלדת וכו'. האירוע הוא בעצם פלט כלשהוא מן הקומפוננטה אל מחוצה לה, הנרשמים לאירוע מקבלים את נתוניו. פליטת המידע נעשית באמצעות הדקורטור output.

שתי הקומפוננטות מוכלות בקומפוננטת אב אחת, כשהן מעבירות דרכה את הנתונים.

כך זה נראה בhtml:

<div class="row">

<div class="column middle">

<app-using-canvas (onSelectPoint)="inputPoint($event)" >

</app-using-canvas>

</div>

<div class="column side">

<app-buying (onSelectPrduct)="inputProduct($event)"

(toShowPath)="sendPath($event)"></app-buying>

</div>

</div>

וכך בקוד ה typescript :

import { Component, OnInit, ViewChild } from '@angular/core';

import { Product } from '../../../classes/Product';

import { UsingCanvasComponent } from '../../using-canvas/using-canvas.component';

import { Goal } from '../../../classes/Goal';

import { Point } from 'src/app/classes/Point';

import { BuyingComponent } from '../buying/buying.component';

@Component({

selector: 'app-customer',

templateUrl: './customer.component.html',

styleUrls: ['./customer.component.css']

})

export class CustomerComponent implements OnInit

{

@ViewChild(UsingCanvasComponent,null) child: UsingCanvasComponent;

@ViewChild(BuyingComponent,null) childBuying: BuyingComponent;

selectedProducts:Product[]=[];

product:Product;

constructor() { }

inputProduct(e:Product)

{

this.child.findProduct(e);

}

inputPoint(e:Point)

{

this.childBuying.setPstart(e);

}

sendPath(coordinates:Goal[])

{

this.child.drawPath(coordinates);

}

}

עריכת הרשימה מעוררת אירוע השולח את המוצר שנוסף לרשימה:

export class BuyingComponent implements OnInit

{

@ViewChild('aliasInput',{static: false}) aliasInput: ElementRef<HTMLInputElement>;

@Output() onSelectPrduct:EventEmitter<Product>=new EventEmitter<Product>();

@Output() toShowPath:EventEmitter<Goal[]>=new EventEmitter<Goal[]>();

מפת החנות מאזינה לאירוע ובעקבותיו מציינת את מיקום המוצר במפה:

export class UsingCanvasComponent implements OnInit

{

@Input()

canvasInputProduct:Product;

@ViewChild('canvas', { static: true }) canvas:

ElementRef<HTMLCanvasElement>;

@Output() onSelectPoint:EventEmitter<Point>=new EventEmitter<Point>();

ctx: CanvasRenderingContext2D;

להלן הפוקנציה בקומפוננטת הקנווס המקבלת מוצר ויוצרת אייכון במקום המתאים:

findProduct(p:Product)

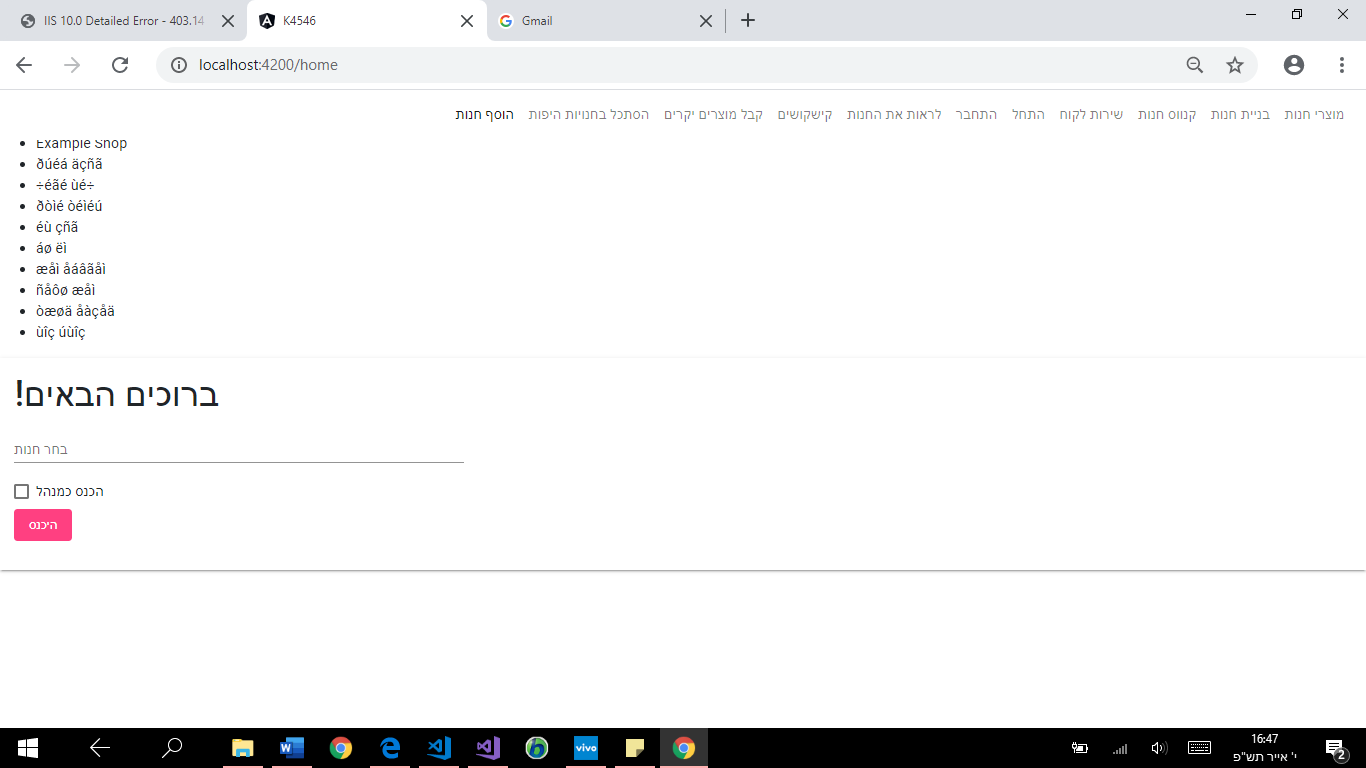
{

let stand=this.shop.Stands.filter(a=>a.Shelves.filter(b=>b.ProductShelves.filter(c=>c.CodeProduct==p.Code)[0])[0])[0];

this.drawIcon(stand.P1.X\*this.mul,stand.P1.Y\*this.mul,p.Alias.TextAlias);

}

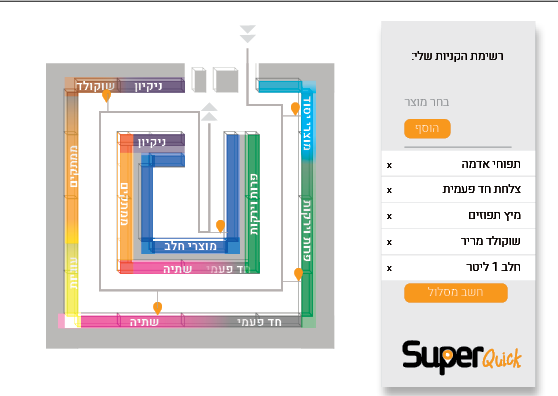
# **תצוגת המסכים**

1. **מסך כניסה ובחירת חנות** 

מסך הפתיחה כולל השלמה אוטומטית לחיפוש החנות, תיבת בחירה לכניסת מנהל, וכפתור לאישור.

באירוע הלחיצה של הכפתור ישנו ניווט באמצעות navigate לפי מצב תיבת הבחירה:

* עבור מנהל יוצג מסך התחברות.
* עבור לקוח יוצג מסך קנייה.
* במידה ולא נבחרה חנות יוצג מסך הוספת חנות חדשה.

1. **מסך לקוח, קנייה:** **עריכת רשימה ותצוגת מפה**

במסך הקניה 2 קומפוננטות:

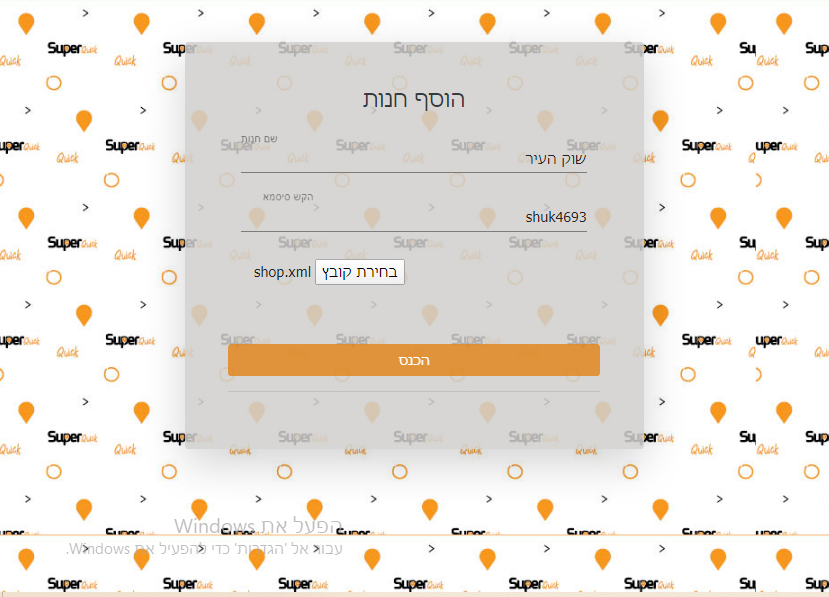
**האחת** בה עורכים רשימה. כוללת השלמה אוטומטית המציגה את כל המוצרים המתאימים לחיפוש שמזין הלקוח. לחיצה על המקש ENTER או על 'הוסף' מוסיפה את המוצר לרשימה ומאתרת את המוצר במפת החנות.

**השניה** מכילה את מפת החנות, בה מוצגים האיקונים במיקומי המוצרים הנבחרים לקניה.

בלחיצה על 'חשב מסלול' מתבקש הלקוח ללחוץ על המפה באזור בו הוא נמצא. לחיצה זו תפעיל את תהליך חישוב המסלול האופטימלי.

כשמסיים השרת את החישוב הוא מחזיר את המסלול, ועל המפה יוצג שרטוט המסלול האופטימלי.

1. **מסך הוספת חנות חדשה**



בעל חנות מזין את שם החנות והסיסמא הרצויה, ומעלה קובץ XML המכיל את נתוני החנות.

הטופס מאפשר שמירה רק בעת הזנת נתונים חוקיים. הסיסמא תהא מורכבת מ-3 תוים לפחות, שם החנות לא ישאר ריק.

נתוני החנות נשלחים מיד לשרת שם מתבצעת המרת הנתונים וחישוב מטריצת מרחקים לחנות.

1. **מסך התחברות לבעל חנות**



בעת הזנת סיסמא דאגנו לאבטחה נכונה, לצורך כך נשלחת הסיסמא + קוד החנות אל השרת, המחזיר ערך בוליאני כנכונות הסיסמא.

הקוד:

this.db.isTruePassword(this.textPassword).subscribe(x=>{

r= <RequrestResponse>x;

if(r.Result==true){

alert(r.Message);

this.router.navigate(["ProductToShop"]);

}

else

{

this.openDialog(r.Message,this.nameShop);

this.textPassword="";

}

});

כפי שנראה בקוד, במקרה של סיסמא שגויה יפתח 'דיאלוג'-פופאפ המתריע על סיסמא שגויה.

קוד הפונקציה הקוראת לפופאפ :

openDialog(message:string,nameShop:string): void

{

const dialogRef = this.dialog.open(InvalidPasswordPopUpComponent,

{

width: '250px',

data: {message,nameShop}

});

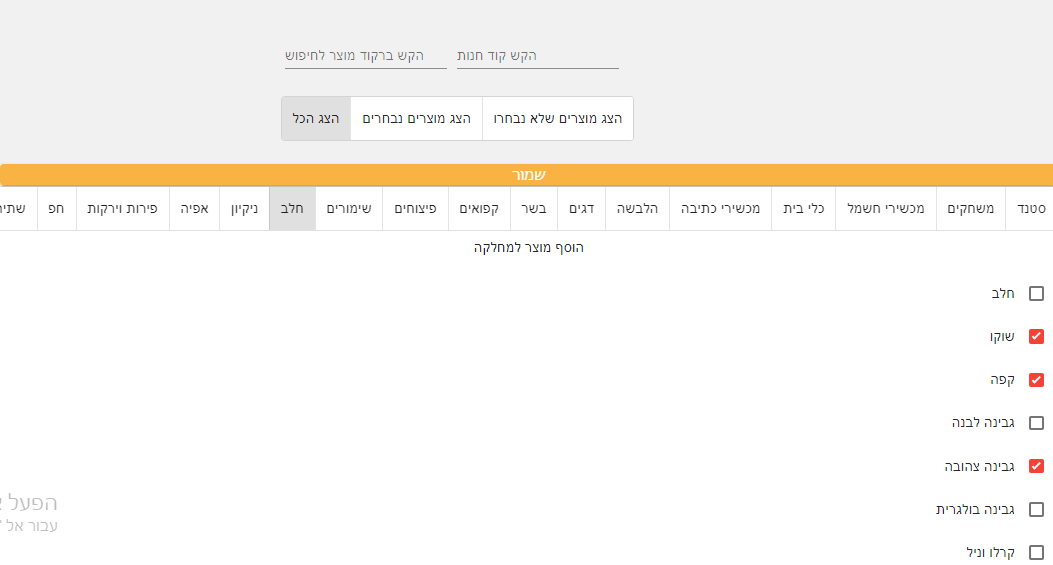
dialogRef.afterClosed().subscribe(result => {

console.log('The dialog was closed');

this.textPassword="";

});

}

**5. בחירת מוצרים** **לחנות**

במסך זה בעל החנות יסמן את המוצרים הקיימים בחנות, יוכל לערוך אותם, וכן יוכל להוסיף מוצרים חדשים.

המוצרים, ממוינים לפי מחלקות לנוחות המשתמש. כמו כן ניתן להגדיר את אופן סינון המוצרים- הצגת המוצרים שנבחרו, הצגת המוצרים שלא נבחרו, או, להציג את כל המוצרים.

בעת לחיצה על מוצר יפתח דיאלוג.

הקוד:

openDialog(p:Product,i:number): void

{

let b=true;

let changePicture=false;

let newProductAliases:Alias[]=[];

const dialogRef = this.dialog.open(AddProductPopUp1Component,

{

width: '50%',

data: {p: p, aliases: this.aliases,codeDep:this.departments[this.iDep].Code,

dataResult:{newProductAliases,b: b,changePicture:changePicture

}

}

});

dialogRef.afterClosed().subscribe(result =>

{

console.log('The dialog was closed');

var r = result; if(result!=undefined&&result.b!=undefined&&result.b==true)

{

this.selectP(p);

this.curProducts[i].flag=true;

.o.option.selected = true;

result.newProductAliases.forEach(x=>this.ToAddProductAliases.push(x)) ;

if(result.changePicture)

{ this.ToUpdateProduct.push(p);

}

}

else

{ this.curProducts[i].flag=false;

this.o.option.selected = false;

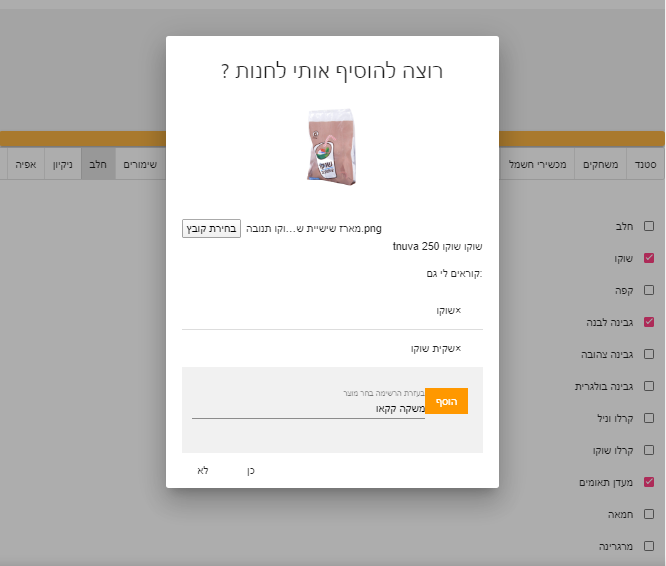
this.cancelP(p);

}

this.chooseDep(this.iDep);

});

}

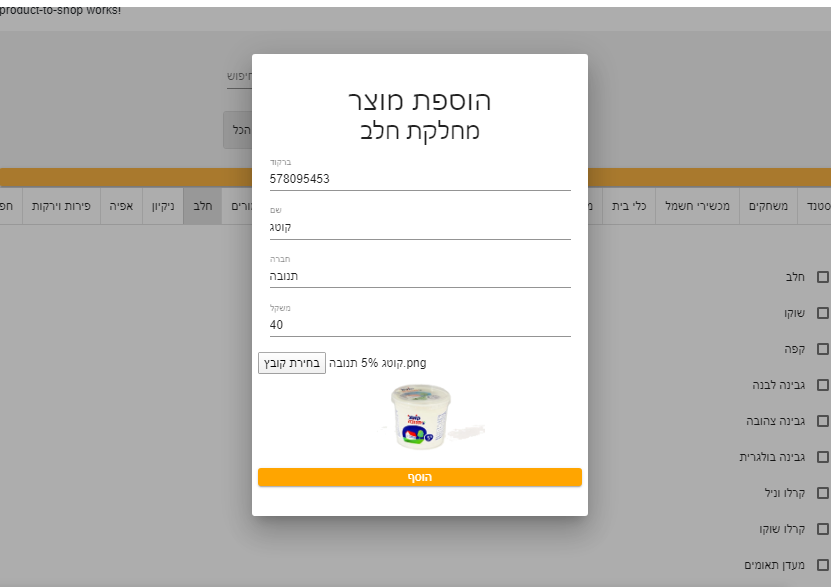


בחלונית זו, בעל החנות יוכל לשבץ/להחליף תמונה למוצר.

כמו כן, בעל החנות יוכל בקלות להגדיר כינויים נוספים למוצר מתוך המאגר/להוסיף כינויים חדשים למאגר הכינויים ולשייכם למוצר.

אם יבחר בעל החנות להוסיף את המוצר, ישמר המוצר ברשימת המוצרים של החנות ותיבת הבחירה של המוצר תסומן כחיובית. אחרת- תסומן כשלילי.

**6. הוספת מוצר למחלקה**



במסך זה ניתן להוסיף מוצר. שם המוצר ישמר בטבלת הכינויים- Alias ושדה הParent שלו יהיה מקושר לקוד המחלקה.

בשני המסכים האחרונים, הוספת מוצר למחלקה ובחירת מוצר חנות, ניתן לשבץ תמונה למוצר.

בלחיצה על "בחירת קובץ" יפתח למשתמש סייר הקבצים, והוא יוכל לבחור את התמונה המתאימה למוצר.

כשהמערכת מזהה שהתקבלה תמונה חדשה, היא ממירה את התמונה לקידוד טקסטואלי, הנשמר בשדה Src של המוצר, וכך נשמרת התמונה במסד הנתונים, כרצף תוים.

קוד הHTML של הכפתור 'בחירת קובץ':

<input type="file" (change)="handleFileSelect($event)" />

פונקציית handleFileSelect

handleFileSelect(evt){

var files = evt.target.files;

var file = files[0];

if (files && file)

{

var reader = new FileReader();

reader.onload =this.\_handleReaderLoaded.bind(this);

reader.readAsBinaryString(file);

}

}

\_handleReaderLoaded(readerEvt) {

let binaryString:string = readerEvt.target.result;

let base64textString:string= btoa(binaryString);

console.log(btoa(binaryString));

let textForPicture:string="data:image/png;base64, ";

this.img.nativeElement.src=this.textForPicture+ base64textString.toString();

this.data.p.Src=this.textForPicture+ base64textString.toString();

this.data.dataResult.changePicture=true;

}

# **מסקנות**

סוף כל סוף הגענו לזמן חתימת הספר, בו חשפנו טפח ממה שמאחורי הפרויקט וסקרנו במעט את תהליך ההתלמדות המופלא, רצוף העמל וההשקעה, עמקני ועשיר בגילויים כה מרתקים, בו הוספנו ידע והתנסות בתחומים רבים ומגוונים.

תחילה, תכנון הפרויקט ושלבי העבודה, נראה בעינינו כמשימה בלתי אפשרית. זאת בשל ההיקף, המורכבות ולחץ הזמן. חלקים מסוימים שתוכננו נפסלו מראש כי חששנו שלא נצליח לבצע אותם, אך בסופו של תהליך ניתן לומר שהכול אפשרי. כל שהיה נראה בלתי עביר, הפרויקט בכללותו, וכן חלקים מסוימים שנפסלו- הכול התברר כבר ביצוע. השכלנו להבין שעם הרבה מוטיבציה ומוסר עבודה, ניתן להשלים כל משימה, קשה ומורכבת ככל שתהיה.

בנוסף כמעט על כל צעד ושעל בפיתוח נתקלנו בתחומי ידע לא מוכרים, מורכבים, הרבה מעבר לרמת הידע שיש לנו. היתקלויות אלה שדרשו פתרונות, גרמו לנו להתנסות המון בלמידה עצמית.

ערכנו הכרות עם מגוון כלים וטכנולוגיות, גילינו שאין בעיה חסרת פתרון וגם לסטודנטיות חסרות ידע רחב ומקיף כמונו ישנה אפשרות להיכנס לנושא שהן לא מכירות, להתאמץ להבין ולמצוא פתרון.

ניתן לומר שכסטודנטיות ומתכנתות, הפרויקט הכניס אותנו לעולם היוצרים בכלל והמתכנתים בפרט. למדנו לתכנן מערכת ע"י ניתוח ואפיון הצרכים והאתגרים, למדנו לחשוב על כל הפרטים ופרטי הפרטים ועל כל הבעיות שיכולות לצוץ ולמצוא את הדרך היעילה ביותר להתמודד איתן. למדנו המון על דרכי חקירה, פיתוח וכתיבה, והרבה על הסיפוק העצום שביצירה.

את הפרויקט אפיינו באופן בולט בכתיבה עצמאית. את התוצר הנפלא יצרנו בעשר אצבעות, בנחישות ובאומץ. שברנו את הראש, כמעט כפשוטו:) לכלכנו הרבה את הידיים בכתיבה ומחיקה וכתיבה חוזרת. נאבקנו קשות במלחמת הבהירות ונטרול השגיאות. ובעצם, פיתחנו אפליקציה בעצמנו, מא' ועד ת'. וזו התנסות מדהימה. פיתוח עצמי הוא אולי לא הדרך הכי טובה להגיע למוצר בקלות ובלי להזיע, אולם במבט לאחור, את מטרת הלמידה השגנו גם השגנו, והכלים שרכשנו הם נכס עוצמתי ששווה את הכל ואותו קנינו לתמיד.

מיומנות נוספת אותה רכשנו במהלך העבודה על הפרויקט היא העבודה בזוגות, בצוות.

עבודת צוות דורשת רבות מכל שותף בה; חלוקת עבודה הוגנת, מוכנות להקשיב ולהכיל מהלכי חשיבה אחרים, ויתור על דעה אישית, ובאופן כללי מוכנות לבוא אחד לקראת השני להצלחת הפרויקט. בפרויקט שלנו לשיתוף הפעולה הפורה היה משמעות בעלת משקל ביצירת "מודל חשיבה" עצמאי, עמוק ויסודי, עם כללים משלו, עקרונות, הנחות יסוד ופעלים היוצאים מכך.

כך התנהלה עבודתנו על הפרויקט ואין ספק כי ניסיון זה תרם לנו רבות ובוודאי יועיל גם בעתיד בעבודות צוות.

את**Super Quick** אף פעם לא נגדיר כמושלם, כי תמיד נשכללו ונפתחו עוד ועוד, אך את הדרך שעשינו נסכם אחר אינספור שעות של עמל בחשיבה ובמעשה כי היה זה מדהים לחוות זאת, וסופר יעיל. הפקנו מפרויקט הגמר תועלת מרובה ותקוותנו שנסיון זה יסייע לנו בעתיד בעזרת ד'.

# **ביבליוגרפיה**

Stackoverflow

Angular material

Github

Bootstrap

w3schools

developer.mozilla

GeeksForGeeks

# **נספח**

במקביל לשלב הפיתוח, ביצענו מעקבים וכך מצאנו את הבאגים ושיפורים נדרשים.

המעקב היה יסודי ומעמיק, תיעדנו כל שלב כדי לאחוז ראש מה קורה.

המעקב להלן אינו מציג רק את האופן שבו המערכת אמורה לפעול, אלא **את הנתונים שחישב המחשב בפועל,** עפ"י חלוניות מעקב והצצה תוך כדי תהליך החישוב.

אז נתחיל, מקוות שהכל ברור. ו-שווה להשקיע במעקב, זה עשוי להיות מענין:

**נבחרו המוצרים:** שוגי, קורונפלקס, כריות, סוכריות אדומות, במבה, ביסלי, תן צ'אפ, צ'יפס מקסיקני, צ'יפס קידס, פופקורן, במבה אדומה, חטיף חדש, מים.

**בחנות הבאה:**

ירוק-קיר, גווני החום-כתום- סטנדים. בכל סטנד מספר המציין את הקוד. המספרים בקצות הטורים הם קודי נקודות הגישה [ליתר דיוק- האינדקס. שלפנו מהחנות את נקודות הגישה עם מספור אוטומטי לחנות, וכך ניתן לגשת למיקום נקודת גישה בטבלה, לפי האינדקס]. תכלת- נקודת ההתחלה.

פופקורן, במבה אדומה, חטיף

תן צאפ ציפס מקסיקני, ציפס קידס

שוגי, קורונפלקס, כריות

סוכריות במבה בסלי

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
|  |  |  |  |  |  |  |  |  | קופה | |  | קופה |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  | 20 |  |  |  |  |  |  |  | 21 |  |  |  |  | 1 |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 |
|  |  | 19 |  |  |  |  | 8 |  |  |  |  |  |  | 9 |  |  |  |  | 1 |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |  |  | 5 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  | 7 |  |  |  |  |  |  |  |  |  |  |  | 22 |  |  |  |  |  | 2 |  |  |  |  | 7 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 8 |
|  |  | 17 |  |  |  |  | 18 |  |  |  | 10 |  |  |  |  |  |  |  | 3 |  |  |  |  | 4 |  |  | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 14 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  | 6 |  |  | 15 |
|  |  | 15 |  |  |  |  | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 17 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 18 |
|  |  |  |  | 6 |  |  |  |  |  | 12 |  |  |  |  | 10 |  |  |  |  |  |  | 3 |  |  |  |  | 19 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 20 |
|  |  | 14 |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  | 4 |  |  |  |  | 7 |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 22 |
|  |  |  |  |  |  |  |  |  |  |  |  | מים |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 23 |
|  |  |  |  |  | 13 |  |  |  |  |  | 11 |  |  |  | 9 |  |  |  |  |  | 8 |  |  |  |  |  | 24 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 25 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 26 |

בהצצה לפרטי המוצרים שנבחרו, גילנו את מיקומם:

|  |  |  |  |
| --- | --- | --- | --- |
| שם מוצר | קוד מוצר | קוד מדף | קוד סטנד |
| שוגי | 7 | 7 | 3 |
| קורונפלקס | 8 | 8 | 3 |
| כריות | 9 | 9 | 3 |
| סוכריות אדומות | 10 | 10 | 4 |
| במבה | 11 | 11 | 4 |
| בסלי | 12 | 12 | 4 |
| תן צ'אפ | 13 | 13 | 5 |
| צ'יפס מקסיקני | 14 | 14 | 5 |
| צ'יפס קידס | 14 | 14 | 5 |
| פופקורן | 16 | 16 | 6 |
| במבה אדומה | 17 | 17 | 6 |
| דוריטוס | 18 | 18 | 6 |
| מים | 31 | 31 | 11 |

**שלב ראשון- ExtraStand**

המוצרים התקבצו לפי הסטנדים-

|  |  |  |  |
| --- | --- | --- | --- |
| קוד סטנד | קודי המוצרים | קודי נקודות גישה | נקודת גישה קרובה ביותר |
| 3 | 7,8,9 | 2,3 | 2 |
| 4 | 10,11,12 | 2,3 | 2 |
| 5 | 23,14,15 | 2,3 | 3 |
| 6 | 16,17,18 | 2,3 | 3 |
| 11 | 31 | 4,5 | 5 |

**שלב שני- productArea**

הסטנדים מתקבצים לפי נקודות גישה משותפות [כשנקודות הגישה הקרובות ביותר גם משותפות, ולכן סטנד 4 לא התקבץ עם סטנד 5].

|  |  |  |
| --- | --- | --- |
| קודי סטנד | קודי נקודות גישה | הכי קרוב |
| 3,4 | 2,3 | 2 |
| 5,6 | 2,3 | 3 |
| 11 | 4,5 | 5 |

**שלב שלישי- productAreaList**

המילון הופך לרשימה, מוסיפים את נקודת ההתחלה, מוצאים לה נקודות גישה, מוסיפים את הקופות- אזורי הסיום, ומחשבים לכל אזור ברשימה 2 נקודות קיצוניות- P1,P2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| קודי סטנד | קודי נקודות גישה | הכי קרוב | P1 | P2 |
| נקודת התחלה (8,22), אין קוד | 6,7 | 6 | (8,22) | (8,22) |
| 3,4 | 2,3 | 2 | (12,7) | (7,2) |
| 5,6 | 2,3 | 3 | (19,7) | (13,2) |
| 11 | 4,5 | 5 | (24,19) | (24,13) |
| קופה 1 | 8,9 | 8 | (1,16) | (2,17) |
| קופה 2 | 8,9 | 9 | (1,14) | (2,14) |

**שלב 4- חישוב מטריצת סמיכויות**.

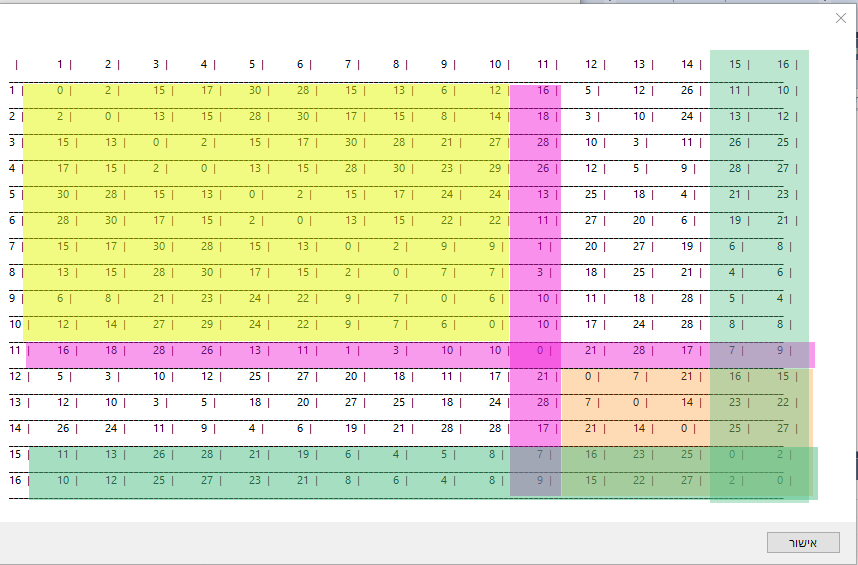
מטריצת סמיכויות של החנות, בעקבות הרצת אלגוריתם דיקסטרה: [להלן המרחקים]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 12 | 6 | 13 | 15 | 28 | 30 | 17 | 15 | 2 | 0 | 0 |
| 14 | 8 | 15 | 17 | 30 | 28 | 15 | 13 | 0 | 2 | 1 |
| 27 | 21 | 28 | 30 | 17 | 15 | 2 | 0 | 13 | 15 | 2 |
| 29 | 23 | 30 | 28 | 15 | 13 | 0 | 2 | 15 | 17 | 3 |
| 24 | 24 | 17 | 15 | 2 | 0 | 13 | 15 | 28 | 30 | 4 |
| 22 | 22 | 15 | 13 | 0 | 2 | 15 | 17 | 30 | 28 | 5 |
| 9 | 9 | 2 | 0 | 13 | 15 | 28 | 30 | 17 | 15 | 6 |
| 7 | 7 | 0 | 2 | 13 | 17 | 30 | 28 | 15 | 13 | 7 |
| 6 | 0 | 7 | 9 | 22 | 24 | 23 | 21 | 8 | 6 | 8 |
| 0 | 6 | 7 | 9 | 22 | 24 | 29 | 27 | 14 | 12 | 9 |

נרחיב את המטריצה הבסיסית:

השורה והעמודה 0 נשארו ריקות, להקל על הפיתוח.

לצורך המעקב, הצגנו messageBox המכיל את המרחקים שחושבו במטריצה הגדולה: צהוב- מטריצת המרחקים הבסיסית של החנות, מרחקים בין נקודות גישה, ורוד- נקודת ההתחלה כתום- המרחקים בין האזורים ירוק- נקודות הסיום, הקופות. אפשר לספור את המשבצות בטבלת החנות ולהשוות עם המרחק במטריצה.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
|  |  |  |  |  |  |  |  |  | קופה1 | |  | קופה2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  | 20 |  |  |  |  |  |  |  | 21 |  |  |  |  | 1 |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 |
|  |  | 19 |  |  |  |  | 8 |  |  |  |  |  |  | 9 |  |  |  |  | 1 |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |  |  | 5 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  | 7 |  |  |  |  |  |  |  |  |  |  |  | 22 |  |  |  |  |  | 2 |  |  |  |  | 7 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | אזור 12 | | | |  |  |  | 8 |
|  |  | 17 |  |  |  |  | 18 |  |  |  | 10 |  |  |  |  |  |  |  | 3 | 4 |  |  | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | אזור 13 | | | |  |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 14 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 | 6 |  |  | 15 |
|  |  | 15 |  |  |  |  | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 17 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 18 |
|  |  |  |  | 6 |  |  |  |  |  | 12 |  |  |  |  | 10 |  |  |  |  |  |  | 3 |  |  |  |  | 19 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 20 |
|  |  | 14 |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  | 4 |  |  |  |  | 7 |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 22 |
|  |  |  |  |  |  |  | אזור 14 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  | 23 |
|  |  |  |  |  | 13 |  |  |  |  |  | 11 |  |  |  | 9 |  |  |  |  |  | 8 |  |  |  |  |  | 24 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 25 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 26 |

וזו המטריצה שנשלחה לחישוב סדר המסלול בפונקציית ה-TSP:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| קופה 2 | קופה 1 | אזור 14 | אזור 13 | אזור 12 | נקודת התחלה |  |
| 9 | 7 | 17 | 28 | 21 | 0 | נקודת התחלה |
| 15 | 16 | 21 | 7 | 0 | 21 | אזור 12 |
| 22 | 23 | 14 | 0 | 7 | 28 | אזור 13 |
| 27 | 25 | 0 | 14 | 21 | 17 | אזור 14 |
| 2 | 0 | 25 | 23 | 16 | 7 | קופה 1 |
| 0 | 2 | 27 | 22 | 15 | 9 | קופה 2 |

כמובן, היות שמדובר במטריצה בגודל 6 חושבה פנקציית ה-TSP המדוייקת.

**הסדר שחזר:**

0,3,2,1,5

יש לצאת מנקודת ההתחלה ולאסוף מאזור 14,13,12 החל מ-14 ואח"כ להגיע לקופה.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
|  |  |  |  |  |  |  |  |  | קופה | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  | 20 |  |  |  |  |  |  |  |  | 21 |  |  |  |  | 1 |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 |
|  |  | 19 |  |  |  | 8 |  |  |  |  |  |  |  | 9 |  |  |  |  | 1 |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |  |  | 5 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  | 7 |  |  |  |  |  |  |  |  |  |  |  | 22 |  |  |  |  |  | 2 |  |  |  |  | 7 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | אזור 12 | | | |  |  |  | 8 |
|  |  | 17 |  |  |  |  | 18 |  |  |  | 10 |  |  |  |  |  |  |  | 3 | 4 |  |  | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | אזור 13 | | | |  |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 14 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 | 6 |  |  | 15 |
|  |  | 15 |  |  |  |  | 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 17 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 18 |
|  |  |  |  | 6 |  |  |  |  |  | 12 |  |  |  |  | 10 |  |  |  |  |  |  | 3 |  |  |  |  | 19 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 20 |
|  |  | 14 |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  | 4 |  |  |  |  | 7 |  |  | 21 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 22 |
|  |  |  |  |  |  |  | איזור 14 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  | 23 |
|  |  |  |  |  | 13 |  |  |  |  |  | 11 |  |  |  | 9 |  |  |  |  |  | 8 |  |  |  |  |  | 24 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 25 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 26 |

במפה מופיע שירטוט 2 מסלולים המתחילים בנקודת ההתחלה הכחולה ומסתיימים בקופה:

הקו הכתום- המסלול האופטימלי, הולך ראשית לאזור 14, מבצע פחות מסיבוב שלם.

הקו הכחול- המסלול הארוך יותר, הולך לאזור 12 בתחילה ומבצע קצת יותר מסיבוב.

ההבדל בין 2 המסלולים אינו גדול מאד, וכך נוכחנו שהחישוב נכון ומדויק.

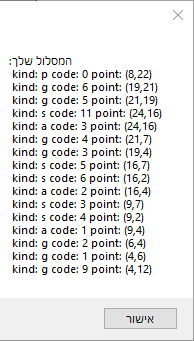
אחרי שיש את סדר האזורים, מחושב המסלול כולו, ז"א נוצרת רשימת מסלול, שתכיל מסלול מפורט [כל הנקודות בהן צריכים לעבור]. בין כל 2 אזורים יתווספו כל נקודות הגישה המקשרות ביניהם.

המסלול יצוייר בממשק המשתמש.

בשביל המעקב, הצגנו messageBox המציג את רשימת ה-goal שחושבה.

המאפיין kind מציין את סוג ה-goal: pנקודת התחלה, g נקודת גישה a כניסה לאזור s סטנד באזור.

המאפיין code מציין עבור נקודות הגישה והסטנדים את הקוד שלהם. לעומת זאת הקוד לאיזור ולנקודת התחלה אינם שמורים במסד הנתונים והקוד הוא פיקטיבי.

המאפיין pointמציין את הנקודה או נקודת האמצע, משמש לממשק המשתמש, בעזרתו יוכל הקנווס למתוח קו, מנקודה לנקודה.

את מערך ה- goal מקבל האנגולר, והמשתמש מקבל מסלול ברור לקניה, החל מהמקום בו הוא נמצא, עובר דרך כל המוצרים שבחר, ומשם- לקופה, ו-הבייתה, להמשך יום עמוס,

**תודה שקניתם איתנו, מקווים שהיה נחמד ו- !!!**