



משרד החינוך



בהנהלת הרב ח.א. כהנא
רח' אמרי ברוך 30 בני ברק



פרויקט גמר להנדסאים

הנדסת תוכנה | תש"פ

שם הפרויקט: SuperQuick

שם התלמידה: פסי קיש ת"ז: 207069014

שם התלמידה: רותי ברס ת"ז: 211551668



SuperQuick

פרויקט גמר 2020

קניה מליגה אחרת.

211551668
רותי בורס

207069014
פסי קשש

תוכן עניינים

3.....	תוכן עניינים
4.....	שלמי תודה
5.....	מבוא
7.....	תקציר
9.....	מטרות ויעדים:
9.....	מטרות המערכת:
9.....	יעדי המערכת:
10.....	תיחום המערכת:
11.....	רעיונות לפיתוח עתידי
13.....	מפרט טכני:
13.....	סביבת פיתוח:
13.....	עמדת משתמש:
13.....	הסביבה הארגונית
14.....	מבנה המערכת
27.....	ניתוח מערכת
28.....	מילון מושגים
30.....	האלגוריתם המרכזי
67.....	תחומי לוגיקה נוספים
72.....	ממשק משתמש
82.....	תצוגת המסכים
91.....	מסקנות
93.....	ביבליוגרפיה
94.....	נספח

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 התלמידות. בשלב ראשון למידה עצמית מעמיקה של התחומים בהם נוגע הפרויקט, לאחר מכן, החלטה על דרך פעולה ומידול העבודה והשלב העיקרי – הפיתוח עצמו.

התלמידות – 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 כחווייה 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 מזמן 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.

האפליקציה Super 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 בחיך. 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 'מספון' 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](#) לאפשר 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 המזכירים 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 בטכנולוגיית 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

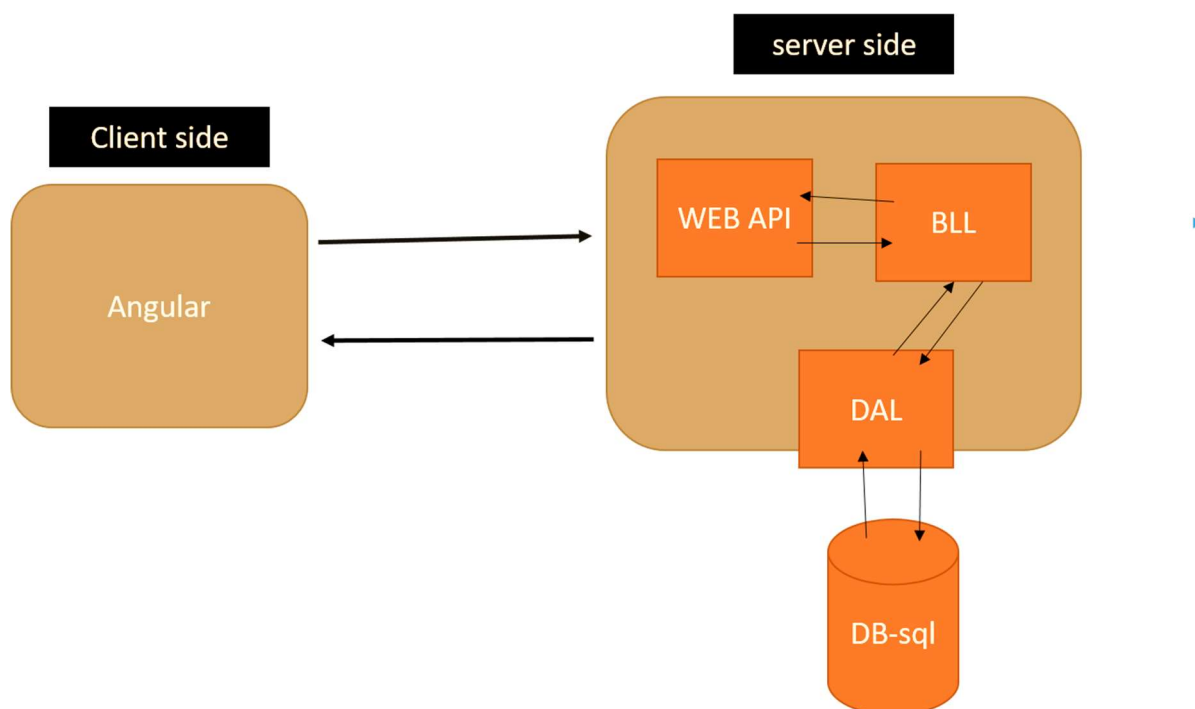
בניית המערכת נעשתה תוך שימוש בהפרדת רשויות מלאה ובניית היררכיה ברורה, על מנת למנוע שיבושים בנתונים ובצורת התצוגה. בנוסף, המבנה מאפשר שינוי תכנותי בעתיד, הן בצורת הגישה לנתונים והן בצורת התצוגה ללא הצורך במגע עם שכבות נוספות.

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 **Comparable**.

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 ה-DAL 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 ה-DAL 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:

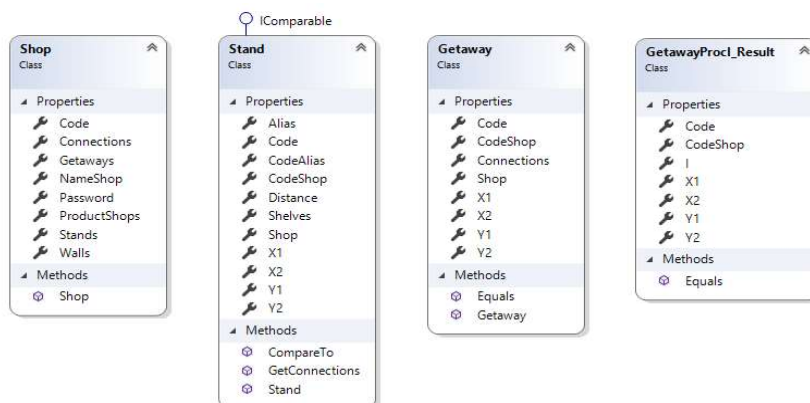
ממשק המשתמש נכתב ב Angular תוך שימוש ב Angular 8. מומש ב TypeScript, Html, CSS.

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

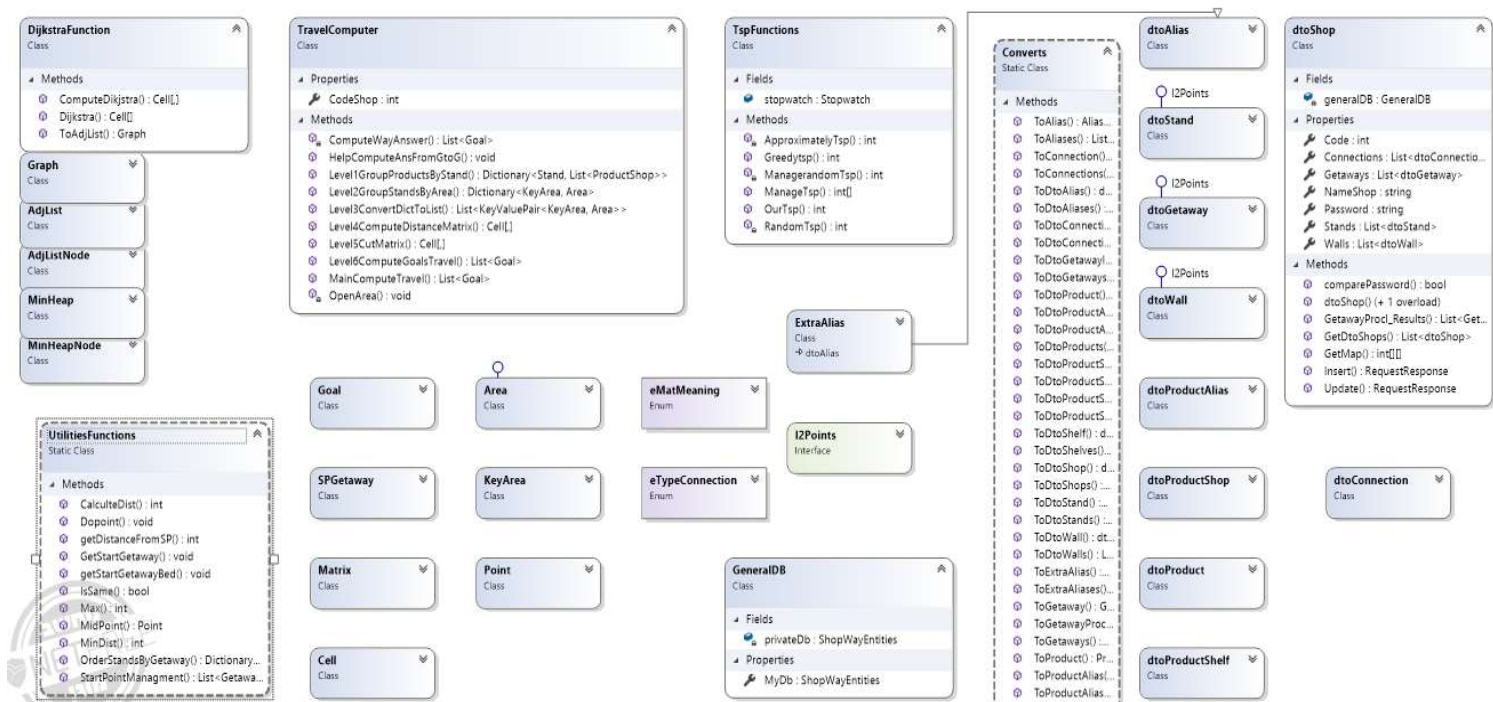
Data Diagram

להלן התרשימים המתארים את המחלקות מהן מורכב הפרויקט:

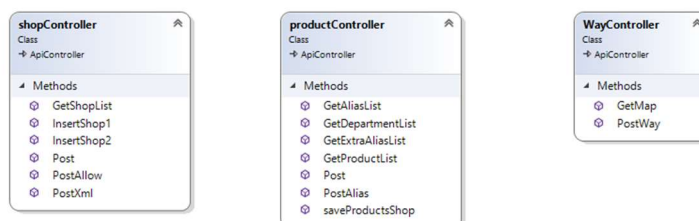
שכבת ה-DAL, המחלקות להן קיימת הרחבה בספריית ה-partial:



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.

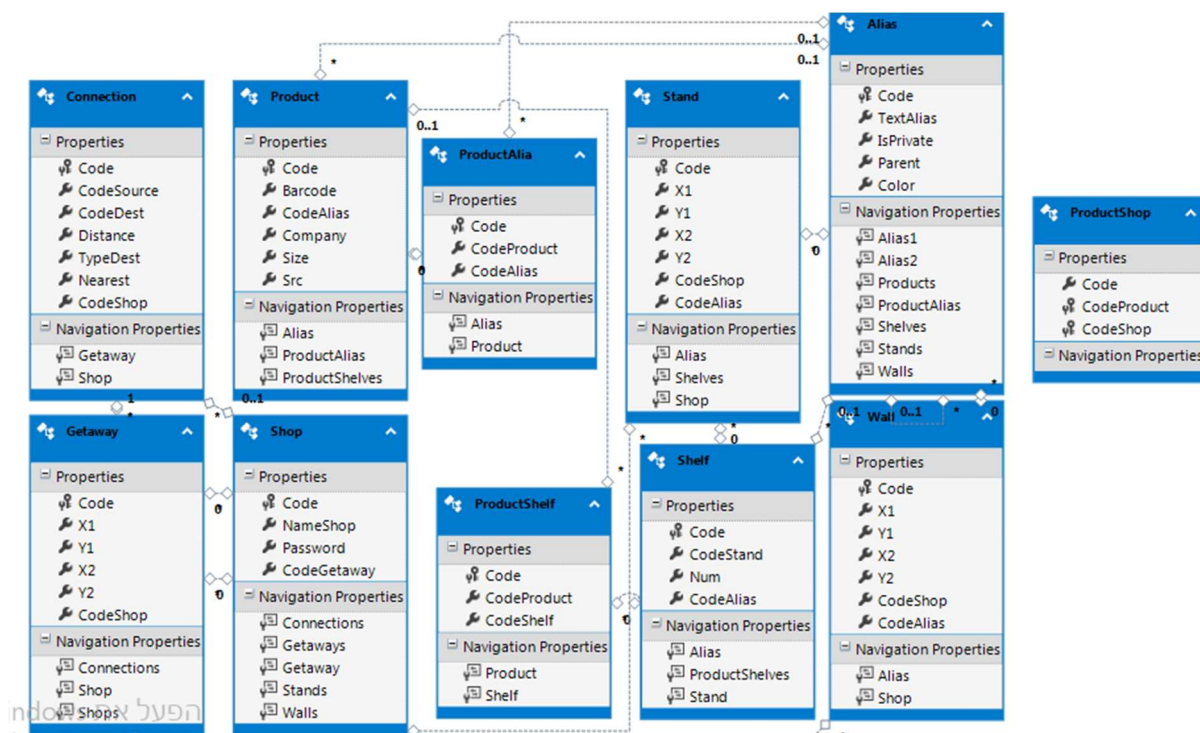
לאחר שהחנות נשמרת באפשרותו לערוך את הנתונים, לשנות מיקום מוצרים בסטנדים, להוסיף מוצרים ועוד.

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 SQL Server, ולפי השמות של השדות נשלפים הנתונים.

טבלת חנויות - shop

הטבלה מכילה את נתוני החנות הבסיסיים.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	שם חנות	string [20]	NameShop	
✓	סיסמת בעל החנות	string [20]	Password	
✓	קוד נקודת גישה שהיא פתח החנות	int	*CodeGetaway	FK לטבלת נקודות גישה

*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 מוצרים - Product

טבלה המכילה מוצרים.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	ברקוד מוצר	string[20]	Barcode	
✓	קוד שם מוצר	int	CodeAlias	FK מטבלת Alias
	חברה	string[20]	Company	
	גודל	string[20]	*Size	
	תמונת מוצר	string[max]	**Src	

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.

טבלת כינויים - Alias

טבלת המכילה מאגר שמות, כינויים. ביניהם כינויי שמות המוצרים, שמות המחלקות, ועוד כינויי מקומות בחנות [קיר, קופה, וכו'...]

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	טקסט הכינוי	string[30]	TextAlias	
✓	האם הכינוי פרטי	bool	*IsPrivate	
	קוד כינוי ההורה	int	**Parent	FK מטבלת ALIAS
	צבע	string[50]	***Color	

*השדה IsPrivate מאפשר תמיכה בהגנה על פרטיות הכיניים, **שדה Parent מאפשר יצירת היררכיה בכיניים ושימוש במחלקות. פרטנו עוד אודות שדות אלו בנושא 'אפשרויות פיתוח עתיד'.
 ***השדה Color מכיל את הצבע שנבחר כמתאים לכינוי, כך יוכלו לצבוע את הסטנדים בחנות בהתאם למה שמכילים [סטנד המכיל מוצרים ממחלקת שוקולד תצבע בגוון חום-שוקולד]. בשלב הנוכחי שדה זה נקלט כמחרוזת ואין בדיקת תקינות מיוחדת, בפיתוח בעתיד ניתן יהיה ליצור טבלת צבעים נפרדת ואפילו להשתמש באלגוריתם המקשר בין קטגורית מוצרים לצבעם המתאים ע"פ הגדרות ה'פסיכולוגיה של הצבעים'.

טבלת קירות - Wall

נתוני הקירות ושאר השטחים הסגורים למעבר בחנות

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	נתוני 2 נקודות קיצוניות בקיר	float	X1	
✓		float	Y1	
✓		float	X2	
✓		float	Y2	
✓	קוד חנות	int	CodeShop	FK לטבלת חנויות
	כינוי קיר	int	CodeAlias	FK לטבלת כינויים

טבלת סטנדים - Stand

נתוני הסטנדים בחנות.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	נתוני 2 נקודות קיצוניות בסטנד	float	X1	
✓		float	Y1	
✓		float	X2	

✓		float	Y2	
✓	קוד חנות	int	CodeShop	FK לטבלת חנויות
	כינוי סטנד	int	CodeAlias	FK לטבלת כינויים

בטבלאות קירות, סטנדים ומדפים קיים מאפיין 'CodeAlias' המאפשר יכולות רבות ודינאמיות לפרויקט הן בעיצוב והן בתמיכה במיפוי נרחב יותר, של מדפים מיוחדים או שטחים- קירות ייעודיים, ועוד כיד הדימיון הטובה...

טבלת מדפים- shelf

נתוני המדפים בסטנד.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	קוד סטנד	int	CodeStand	FK לטבלת סטנדים
✓	מספר מדף בסטנד	int	Num	
	כינוי מדף	int	CodeAlias	FK לטבלת כינויים

טבלת שערים, נקודות גישה- Getaway

הטבלה מכילה מאגר כניסות ופתחים לטורים בחנות.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	נתוני 2 נקודות קיצוניות בנקודת הגישה	float	X1	
✓		float	Y1	
✓		float	X2	
✓		float	Y2	
✓	קוד חנות	int	CodeShop	FK לטבלת חנויות

טבלת קשתות-Connection

הטבלה מכילה קישור בין נקודות הגישה והסטנדים.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	קוד מקור	int	CodeSource	FK לטבלת נקודות גישה
✓	קוד יעד	int	*CodeDest	
✓	מרחק	float	Distance	
✓	סוג יעד	int	TypeDest	
✓	האם הכי קרוב?	bit	**Nearest	
✓	קוד חנות	int	CodeShop	FK לטבלת חנויות

*השדה CodeDest מכיל קוד מטבלת Getaway או קוד מטבלת Stand משום שהקשת יכולה לחבר בין שני Getaway או בין Getaway ל- Stand [או בין Getaway ל-Wall כשניצור קיר מיוחד שאין בו סטנדים, כמו קופה]

משום כך, לא יכולנו להגדירו כמפתח זר. כדי להבטיח שיכנס קוד תקין, הוספנו בדיקה בעת ההוספה כי אכן הקוד קיים בסוג היעד שנבחר.

**השדה Nearest משמש לתהליך חישוב מסלול, וחוסך חישוב בכל הרצה 'איזה נקודת גישה קרובה ביותר עבור כל סטנד' [השדה משמש לקיבוץ איזורים כפי שיפורט].

טבלת כינויי מוצרים-ProductAlias

קישור כינוי למוצר, כדי לאפשר כינויים רבים לכל מוצר.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	קוד מוצר	int	CodeProduct	FK לטבלת מוצרים
✓	קוד כינוי	int	CodeAlias	FK לטבלת כינויים

עבור טבלה זו קיים אילוץ ייחודיות המונע רשומות כפולות, כך שלא ניתן לקשר כינוי מסוים למוצר יותר מפעם אחת.

טבלת מוצרי מדף - ProductShelf

קישור מוצר למדף בחנות.

שדה חובה	תאור	סוג השדה	שם השדה	מפתחות
✓	קוד	int	Code	PK
✓	קוד מוצר	int	CodeProduct	FK לטבלת מוצרים
✓	קוד מדף	int	CodeShelf	FK לטבלת מדפים

גם עבור טבלה זו קיים אילוץ ייחודיות המונע רשומות כפולות, כך שניתן לקשר פעם אחת בלבד את אותו מוצר לאותו מדף בחנות.

טבלת דמה של מוצרי חנות - ProductShop

מאגר מוצרים לחנות.

כשבנינו את מסד הנתונים רצינו לשמור טבלת קישור מוצר לחנות. כשניתחנו את מבנה הנתונים גילינו כי היא מיותרת, ואין צורך לקשר מוצר גם למדף וגם לחנות. ניתן לראות זאת בצורה מוחשית בהמשך, בהרחבה אודות מחלקת DtoShop, שם הצגנו את השמירה ההיררכית של הנתונים.

למרות זאת, היות וטבלה כזו מועילה לנוחות שלילת נתונים, יצרנו View המהווה תחליף לטבלה.

שם השדה	סוג השדה	תיאור
Code	int	קוד
CodeProduct	int	קוד מוצר
CodeShop	int	קוד חנות

דוגמא לזרימת המידע במערכת:

לכל מחלקה יש מחלקה מקבילה בשרת ובלקוח.

לדוגמא מחלקת Shop בשכבת DAL:

```
public partial class Shop
{
    public int Code { get; set; }
    public string NameShop { get; set; }
    public string Password { get; set; }
    public int CodeGetaway { get; set; }
    public virtual ICollection<Connection> Connections { get; set; }
    public virtual ICollection<Getaway> Getaways { get; set; }
    public virtual Getaway Getaway { get; set; }
    public virtual ICollection<Stand> Stands { get; set; }
    public virtual ICollection<Wall> Walls { get; set; }
}
```

בנוסף, קיימת בספריית DTO בשכבת BLL מחלקת dtoShop:

```
public class dtoShop
{
    public int Code { get; set; }
    public string NameShop { get; set; }
    public int CodeGetaway { get; set; }
    public List<dtoWall> Walls { get; set; }
    public List<dtoStand> Stands { get; set; }
    public List<dtoConnection> Connections { get; set; }
    public List<dtoGetaway> Getaways { 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

2. 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

3. Connections

4. Getaways

כדי להמיר אובייקטים משכבת DAL לאובייקטי DTO, קיימת בשכבת BLL בספריית Utilities מחלקת `Converts`.

לדוגמא, פונקציות במחלקת `Converts` להמרת חנות:

```
public static dtoShop ToDtoShop(Shop shop)
{
    // כל רשימה שממירים יש לבדוק האם אינה ריקה
    // משום שהרשימות של אובייקט הדאל צריכות המרה לטיפוס רשימה, ולא ניתן להמיר אובייקט ריק
    dtoShop dto = new dtoShop();
    dto.Code = shop.Code;
    dto.NameShop = shop.NameShop;
    dto.Stands = shop.Stands != null ? Converts.ToDtoStands(shop.Stands.ToList()) : null;
    dto.Walls = shop.Walls != null ? Converts.ToDtoWalls(shop.Walls.ToList()) : null;
    dto.Connections = shop.Connections != null ?
        Converts.ToDtoConnections(shop.Connections.ToList()) : null;
    dto.Getaways = shop.Stands != null ?
        Converts.ToDtoGetawaysI(dtoShop.GetawayProcI_Results(shop.Code)) : null;
    dto.CodeGetaway = Convert.ToInt32(shop.CodeGetaway);
    return dto;
}

public static Shop ToShop(dtoShop dtoShop, string password)
{
    return new Shop() { Code = dtoShop.Code, NameShop = dtoShop.NameShop, Password = password,
        Connections = Converts.ToConnections(dtoShop.Connections), Getaways = Converts.ToGetaways(dtoShop.Getaways),
        Walls = Converts.ToWalls(dtoShop.Walls), Stands = Converts.ToStands(dtoShop.Stands), CodeGetaway = dtoShop.CodeGetaway };
}

public static List<Shop> ToShops(List<dtoShop> dtoShops)
{
    if (dtoShops == null)
        return null;
    return dtoShops.Select(x => Converts.ToShop(x, "")).ToList();
}

public static List<dtoShop> ToDtoShops(List<Shop> shops)
{
    if (shops == null)
        return null;
    return shops.Select(x => Converts.ToDtoShop(x)).ToList();
}
```

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

מחלקת `Shop` באנגולר, זהה למחלקת `DtoShop` שבשרת:

```
export class Shop
{
    Code:number
    NameShop: string
    CodeGetaway:number
    Walls:Wall[]
    Stands:Stand[]
    Connections:Connection[]
    Getaways:Getaway[]
}
```

אודות הספריה DTO:

במודל שבשכבת ה-DAL 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 האובייקט 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.

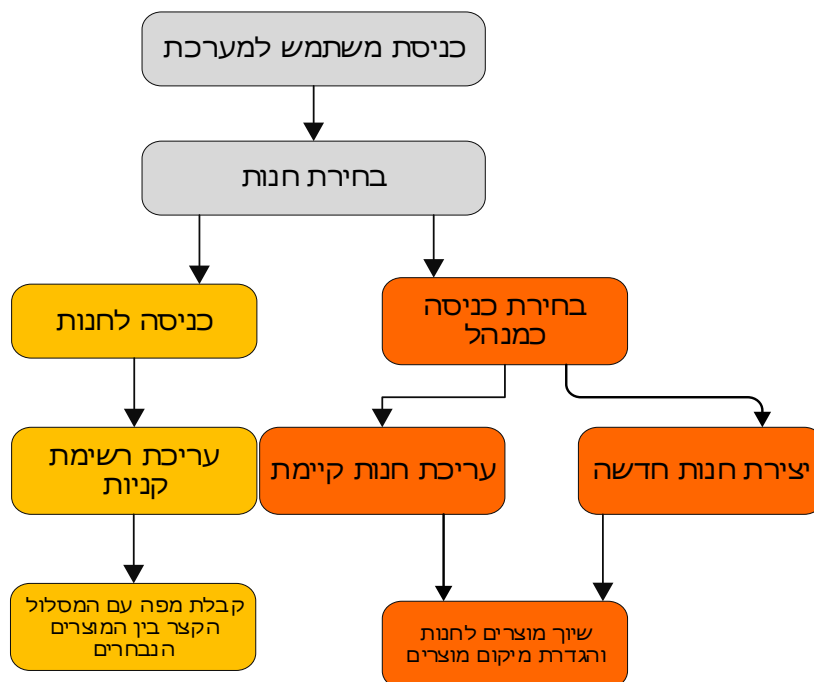
היינו מעוניינים לבטל את האובייקטים והרשימות שבמודל. אך, לא ניתן לשנות את מחלקות המודל שבשכבת ה-DAL, שהרי הן מיוצרות עפ"י בסיס הנתונים, ובכל עדכון מודל עלולות להיווצר מחדש ללא שמירת שינויי המתכנת. לכן, נשתמש ב-DTO.

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 Shop מכילה את כל הנתונים השייכים לחנות, אך ללא מעגלים.

ע"י האובייקטים שהחנות מכילה נחסכו לנו שאילות רבות, וקיבלנו את כל הנתונים בגישה ישירה ופשוטה. המרת הנתונים התבצעה באמצעות מחלקת **Converts**.

ניתוח מערכת

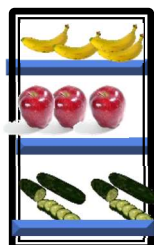


מילון מושגים

להלן מספר הגדרות שיעשו סדר במושגים הקיימים בפרויקט:

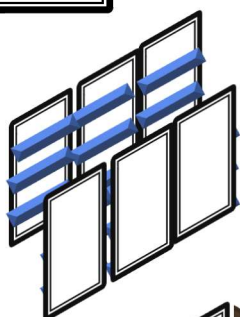


מדף - 'קומה' בעמודה שעליה מונחים המוצרים [כל מוצר בחנות ממוקם במדף אחד בלבד. במדף עשויים להיות כמה מוצרים].

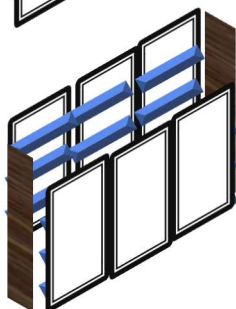


עמודה, סטנד, Stand - 'בנין' המורכב מכמה מדפים. [כל מדף בחנות ממוקם בעמודה אחת בלבד, בעמודה ישנם, בדרך"כ מספר מדפים].

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 וגישות 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 בדו"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 ומיפוי חנות באופן אידיאלי שיאפשר חישוב מסלול

האפשרויות:

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 כש-n 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 'מתחננו חבלים' (וירטואלית כמובן) לאורך החנות וסגרנו אזורים'.

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

בעיות ופתרונותיהן:

א. חישוב מרחקים בין שטחים

בשביל להפעיל פונקציית TSP יש ליצור מטריצת מרחקים מהחנות.

1. איך נמצא מרחק בין השטחים?

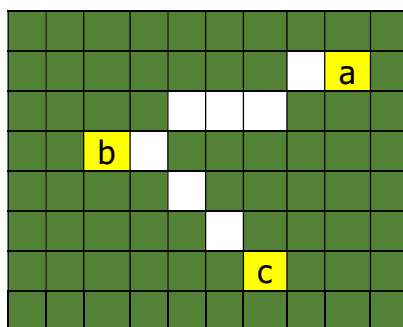
הפתרון הפשוט הוא למצוא מרחק מנקודה לנקודה סמוכה ע"י נוסחת מרחק פשוטה.

הבעיה: אין אפשרות ללכת בחנות מכל נקודה לכל נקודה על ידי נוסחת מרחק, כיון שבאמצע יש קירות שדרכם א"א לעבור.

הפתרון: נשתמש בנקודות גישה ובקשתות של סטנד. ע"י הקשתות נדע, מאיפה אפשר להיכנס מהסטנד לנקודת גישה ומה המרחק ביניהם. בכל פעם שרוצים להגיע למוצר מחשבים מאיזה מנקודות הגישה של הסטנד שלו כדאי להיכנס הפעם למרחק אופטימלי.

2. איך מחשבים מרחק בין קו לקו ובין סטנד לסטנד?

דוגמת חישוב מרחק בין נקודות: [בשטח הירוק אין מעבר]



אפשר לראות שהמרחק מ a ל b הוא 5, והמרחק מ b ל c הוא 3. בשביל להגיע מ a ל c נחפש נקודת גישה משותפת [מב אפשר לגשת גם ל a וגם ל c], נחשב מטריצת מרחקים disMat:

	a	B	C
A	0	5	8
B	5	0	3
C	8	3	0

וכך נחשב את המרחק מ A ל C: $\text{mat}[a,c], \text{mat}[c,a] = \text{mat}[a,b] + \text{mat}[b,c]$

דוגמא לחישוב מרחק בין 2 ישרים: [צהוב- ישר, ירוק- אין מעבר]

המרחק מ A ל b הוא 1 והמרחק מ C ל B הוא 1, B הוא נקודת גישה משותפת של A ו C (מ b אפשר לגשת גם ל a וגם ל c) אך אם נחשב כמו שחישבנו קודם ($\text{mat}[a,c], \text{mat}[c,a] = \text{mat}[a,b] + \text{mat}[b,c]$) נקבל תוצאה מוטעית, 2 במקום 4.

	a	B	C
A	0	1	2
B	1	0	1
C	2	1	0

הצעה: אם המרחק בין A ל C אינו ישר ועובר דרך B, נוסיף למרחק שחושב את האורך של B שעוברים דרכו: $\text{mat}[a,c], \text{mat}[c,a] = \text{mat}[a,b] + \text{mat}[b,c] + |b|$

הפרכה: בדוגמא זו ניתן לראות שבשביל להגיע מ A ל C לא עברנו בכל B, ולא מתאים לחשב את כולו.

למעשה, בנתוני החנות הנורמליים הפתרון קצת יותר מתאים, כי הסטנדים בחנות נפגשים בקצותיהם ולא נקודה אמצעית כלשהי אקראית, אך בצורת החישוב שלנו- אורך המסלול הוא סכום מרחקים במטריצת מרחקים בין האיזורים שעברו בהם, לא ניתן לחשב בצורה כזו.

הפתרון שלנו:

המרת חישוב מרחקי ישרים לחישוב מרחקי נקודות- לכל ישר נחשב את אמצע הקטע שלו ומשם נחשב מרחק לאמצע הקטע של הישר השני.

כמובן שחישוב כזה אינו מדויק לגמרי, וניתן להתווכח עליו. בחרנו בו כי ראינו שקשה לעקוב מהיכן הגענו ולאן אנחנו מגיעים ומה הקו שעברנו בדרך, ועלולות להיווצר טעויות רבות בחישוב.

באותה צורה נחשב מרחק בין קוים צמודים: נחשב חצי מהאורך של קו מקור + חצי מהאורך של קו יעד.

במבט ראשון המרחק בין 2 קוים צמודים הוא 0, כי לכאורה לא צריך ללכת שום מרחק לעבור בין סטנדים צמודים. למעשה חישוב המסלול מהקו הירוק לקו הצהוב [ע"י נקודות משותפות, המרחק מהקו הירוק לתכלת + המרחק מהקו התכלת לאדום... עד שמגיעים לקו הצהוב] מביא לתוצאה שגויה, כאילו שמעבר בטור בין 4 סטנדים מחוברים אורכו 0, כי לכאורה לא עברנו שום מרחק, כולם צמודים.

לכן המרחק בין הקו הירוק לקו התכלת, לדוגמא, יהיה מחצית אורך הקו התכלת+מחצית אורך הקו הירוק. כך גם שאר המרחקים בין 2 קווים צמודים, וחישוב המרחק בין הקו הירוק לקו הצהוב יהיה יותר מדויק.

ההיגיון: כשלוקוח ניצב מול סטנד, ניתן להניח שהוא נמצא באמצע הסטנד שהוא יחידה קצרה יחסית, ואדם אינו מקפיד לעמוד בקצה הסטנד כדי לקחת מוצר שנמצא סטנד.

ככל שהקו- (הסטנד) יהיה ארוך יותר, כך חוסר הדיוק יגדל. ככל שהקו יהיה קצר יותר, נוכל להסתכל עליו כנקודה אחת, נקודת האמצע שלו.

אנו מניחים בתוכנית, וכך נמליץ לבעל החנות, ליצור מדפים קצרים ולמקם בכל מדף מוצר אחד, משום שהתוכנית לא יורדת לרזולוציה של 'מיקום המוצר במדף', אלא רק נתון לכל מוצר באיזה מדף נמצא, ובשביל להגיע למוצר יש להגיע למדף, ובפסיעה אחת הלקוח הגיע לאמצע הסטנד וניצב מול אמצע המדף. כך הגענו למירב הדיוק בחישוב.

נקודה שניה לטיפול:

פונקציית TSP היחידה המדויקת ב-100% [שידועה לנו בינתיים] שייכת לסדר גודל אקספוננציאלי של $n!$, כש-N מייצג את מספר היעדים.

לאחר ניסויים ראינו שלא ניתן להריץ בזמן הגיוני 13 נקודות, ויותר מ-10 נקודות לוקח זמן לעיבוד. [מדדים לראות בעיניים את ההבדל בין 11 ל-12!] לכן הוחלט לא להפעיל אותה על יותר מ-10 נקודות, ולכן ניסינו 2 פתרונות:

1. צמצום נקודות- נקבץ סטנדים בעלי נקודות גישה זהות ונקודות גישה קרובות ביותר.
2. נשתמש בפונקציות קירוב.

פתרון 1 : צמצום הנקודות

נרצה לצמצם את מס' הנקודות, להקטין את מטריצת המרחקים שתשלח לTSP. לכן קיבצנו מספר מוצרים ביחד.

איך אפשר לקבץ מוצרים?

רעיון 1: נקבץ את כל המוצרים שבאותו טור. הגיוני שלקוח שנכנס לסטנד ייקח את כל המוצרים שצריך מסטנד זה.

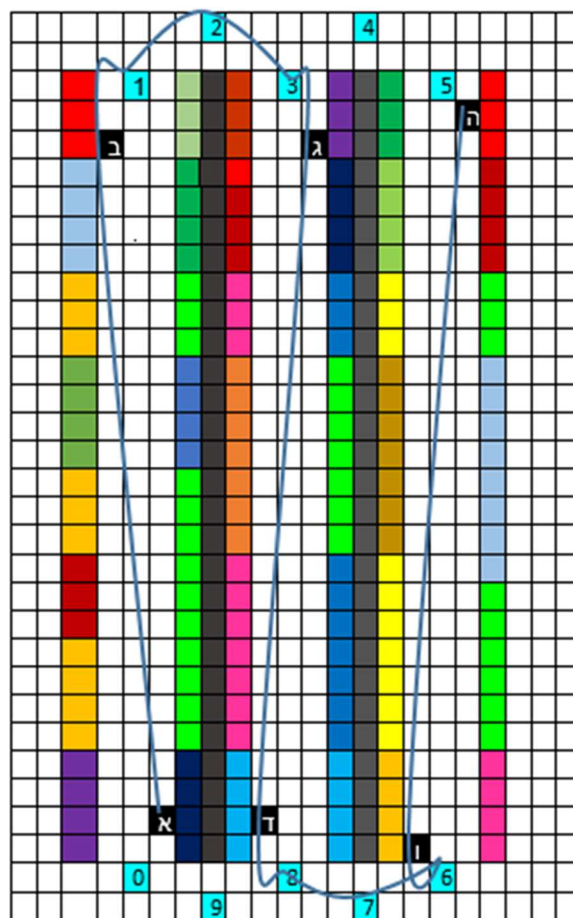
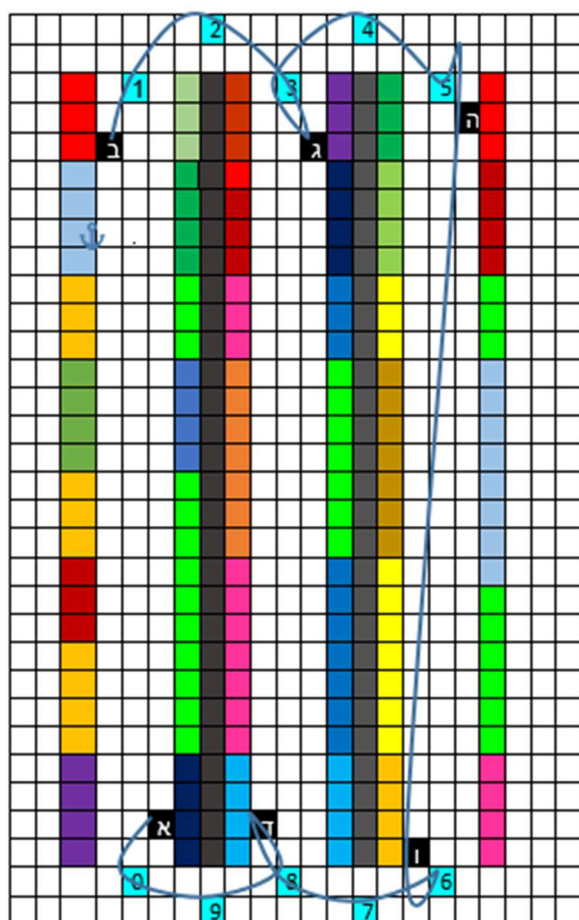
הפרכה: יתכנו סטנדים ארוכים מאד, שלא שווה ללקוח ללכת לאורך כל הטור.

המחשה: בשרטוט זה נקודות הגישה ממוספרות וצבועות בטורקיז, והיעדים מסומנים כאותיות ומושחרים. נניח שאין כאן נקודת התחלה ונקודת סיום.

בחנות זו כדי להגיע למוצרים א' ב' ג' ד' ה' ו' קיימים 2 אפשרויות מסלול:

2. ואפשר לאסוף תחילה את המוצרים שבקצה התחתון של הטבלה, א' ד' ו' ואח"כ למוצרים שבקצה העליון של הטבלה, ה' ג' ב'.

1. לאסוף מוצרים עבור כל טור בנפרד [הסדר: א,ב,ג,ד,ה]



בספירה ידנית אורך מסלול א' הוא כ-100 משבצות, ומסלול ב' כ-65 משבצות. ההפרשים ילכו ויגדלו ככל שהטורים יתארכו והפרופורציה בין רוחב כל סטנד לאורך הסטנד תשתנה, כלומר, בסטנדים צרים וארוכים כשצריכים מוצרים מהקצוות מסלול ב' מעגלי יהיה אופטימלי הרבה יותר.

ולכן לא נקבץ את כל המוצרים שבאותו טור לנקודה אחת במטריצה שתישלח לtsp.

רעיון 2 לצמצום יעדים: נקבץ את כל המוצרים שבאותו סטנד כי אין לכל מוצר את המיקום המדויק שלו בחנות אלא רק את המדף שבו הוא נמצא, והלקוח מגיע לכל המדפים שבסטנד בבת אחת, ולכן אכן נקבץ את כל המוצרים שבאותו סטנד.

רעיון 3: שדרוג: הרעיון שלנו למפות את החנות בצורה שמוצרים שמגיעים אליהם מאותו מקום, שנמצאים באותו טור, יתקבצו כנקודת יעד אחת, וכך נחסוך בנקודות.

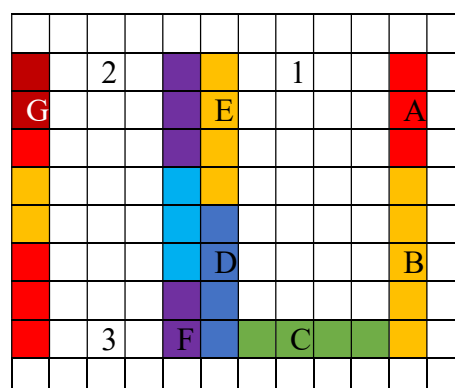
מה הם מוצרים שמגיעים אליהם מאותו מקום?

הטור ארוך, ויש לו 2 פתחים והרבה סטנדים, אך יש סטנדים שניתן לראות כי אם יכנסו לטור בשביל סטנד אחד, כבר יאספו בטוח גם מוצרים של הסטנד השני.

איך נחשב בדיוק אלו סטנדים יתקבצו?

הנחנו שסטנדים שיש להם אותן נקודות גישה ואותן נקודות גישה קרובות ביותר יכולות להיחשב כיעד אחד, כלומר- כשניגשים לאחת ייגשו גם לשנייה. קבוצת הסטנדים תקרא אזור.

תזכורת: המטרה לחשב סדר אספת המוצרים, ע"י פונקציית TSP. אם ידוע שלכל האזור הוא יעד אחד, אין צורך לחשב מכל סטנד בו לכל סטנד אחר שאינה באזור, כי הם קבוצה אחת שכשיגיע תורה יכנסו לכולה. לאחר חישוב הTSP יחושב סידור פנימי בתוך האזור, בסדר הגיוני.



בדוגמא הנ"ל הלקוח בחר מוצרים מכל הסטנדים שבטבלה, A,B,C,D,E,F,G. הסטנדים A,B,C,D,E יתקבצו כי כל נקודות הגישה שלהם- 1 משותפות, והיא גם הכי קרובה. F וG לא

יתקבצו למרות שנקודות הגישה שלהם 2,3 משותפות אך ל G 2 היא הקרובה ביותר ול F 3 היא הקרובה ביותר, ולכן לא מובטח שכשיכנסו ל F יכנסו גם ל G.

לא רלוונטי לדעת את המרחק מ G ל A ומ G ל B בשביל לדעת את הסדר, כי A ו B יהיו ביחד כשיגיע תור כל האזור שלהם, ולכן נחשב מרחק רק מהאזור שלהם לכל יעד אחר.

הבעיה בשימוש בקיבוץ לאזורים:

בנקודה הקודמת, חישוב מרחקים בין קוים, הזכרנו את הבעיה בחישוב מרחקים ע"י אמצע קטע של קוים ארוכים.

כזכור, כשחישבנו מרחק בין סטנד לנקודת גישה הסתבכנו כי זה מרחק בין קו לקו ולא בין נקודה לנקודה, ולכן חישבנו לכל קו את אמצע הקטע שלו וביניהם חישבנו מרחק, וזה כמעט מדויק בהנחה שהסטנדים קצרים יחסית וכשלקוח מגיע לסטנד הוא בעצם נמצא באמצע הסטנד.

אך אם נקבץ מספר סטנדים יחד ונקבל אזור, בשביל לחשב מרחק אליו גם נחשב את נקודת האמצע שלו [ע"י 2 הנקודות הקיצוניות שבו] ומרחק לנקודת גישה, אך כאן חוסר הדיוק גדל הרבה יותר.

לאחר עיון מעמיק התגלה כי צורה זו עלולה במקרי קצה להציע סדר מוטעה לאיסוף המוצרים!

יתכן שהחישוב ימליץ על קניה מעגלית, שבמקרה זה יחושב בשביל להכנס לאזור א' מחצית מאורך אזור א', ומרחק יציאה מאזור א- חצי מאורך אזור א', כשלמעשה הלקוח לא הלך רק מחצית מאורך האזור אלא הגיע לכל הסטנדים הקיצוניים שבאזור. בצורה כזו התכנית עשויה להציע ללקוח להכנס לטור לאזור א', לצאת במקום להמשיך מאזור א' לאזור ב' הצמוד לאזור א'.

המצב נדיר ובעיקרון החישוב הלא מדויק אינו משנה לרוב את בחירת המסלול, [תזכורת: המטרה בחישוב מטריצת המרחקים בשביל לקבל סדר אידיאלי של יעדים ואם תתקבל תוצאה נכונה לא משנה אם המרחקים יהיו מוטעים] אך יש לשים לב לנקודה זו.

לכן, אם לאחר קיבוץ המוצרים לסטנדים נקבל יותר מ-10 נקודות להגיע אליהם, נקבץ לאזורים. אחרת- אין צורך לקבץ, הפונקציה TSP תוכל לפעול בפרק זמן הגיוני.

פתרון 2 לבעיית Tsp: שימוש בפונקציות פחות מדויקות

בהנחה שהפרויקט שלנו מתאים לקניה ממוקדת [לקוח שרוצה לקנות 100 מוצרים מסתובב בכל החנות ולא מקיש אותם והולך לפי הנחיות האפליקציה...] נוכל להפעיל את פונקציית TSP המדויקת כי לרוב נקבל עד 10 יעדים – אזורים.

אם לא, לא נרשה לתוכנית שלנו לקרוס או לעבוד בזמן לא סביר! לכן נפעיל מספר פונקציות אחרות.

הגדרנו מחלקה `TspFunctions` עם פונקציה ראשית `ManageTsp` שמפעילה את פונקציות ה-TSP הקיימות.

במקרה שלאחר כל הצמצומים והקיבוצים נקבל יותר מ-10 נקודות, נשתמש בכמה בפונקציות קירוב שמנסות לחשב את הדרך הקצרה ביותר, תוך יישום פתרונות יצירתיים ואפקטיביים.

תיאור חלקי הלוגיקה

את מיפוי החנות וחישוב המסלול ניתן לחלק לשלושה שלבים, כפי שהם מתבטאים בשלושה מחלקות שונות:

- **DijkstraFunction** - המחלקה המחשבת את מטריצת המרחקים ע"י אלגוריתם הנקרא **דייקסטרה**
- **TravelComputer** - מחלקה המנהלת את אלגוריתם **חישוב המסלול**, מקבצת אזורים ומחזירה סדר יעדים אופטימלי לשרטוט.
- **TspFunction** - המחלקה המנהלת את אלגוריתמי **מציאת המסלול האופטימלי**.

מחלקות אלה מכילות פונקציות המשתמשות גם בפונקציות עזר ובמחלקות אחרות כפי שיפורט.

מחלקות עזר שהוגדרו לצורך החישוב:

Cell - טיפוס של תא במטריצת מרחקים.

KeyArea - טיפוס מפתח לקיבוץ אזורים.

Area - אזור, מכיל נתונים על האזור ומשתני עזר לחישוב.

Goal - נקודת יעד במסלול המוחזר.

מחלקת DijkstraFuntion דייקסטרה

המערכת נכנסת לפעולה כאשר בעל החנות מוסיף את חנותו ומזין את נתוני החנות. השרת מקבל קובץ XML, מבצע סירלוץ לאובייקט ושומר בשרת הנתונים. לאחר מכן מופעל אלגוריתם דייקסטרה ונשמרת מטריצת המרחקים הבסיסית של החנות.

סריאליזציה (serialization) היא תהליך של תרגום מבני נתונים ואובייקטים לפורמט שניתן לאחסן אותו בקובץ, ולאחר מכן, 'להקים אותו לתחייה' כאשר רצף הביטים שנוצר בתהליך נקרא חזרה בהתאם לפורמט הסריאליזציה.

השרת מקבל קובץ XML ומחלץ ממנו אובייקט Shop.

מטריצת המרחקים הבסיסית של החנות כוללת מרחק בין כל נקודת גישה בחנות לשאר הנקודות.

האלגוריתם דייקסטרה:

כדי לקבל מטריצת מרחקים של החנות יש לדעת את המרחק מכל קודקוד (נקודת גישה) לשאר הקודקודים. ישנם 2 אלגוריתמים מפורסמים המטפלים בכך:

1. פלויד ורשל – האלגוריתם מחשב מרחק מינימלי בין כל זוגות הקודקודים בגרף, בסיבוכיות v^3 כאשר V מייצג את מספר הקודקודים.

2. דייקסטרה – האלגוריתם מחשב מרחק מינימלי מקודקוד מקור בגרף לכל שאר הקודקודים בסיבוכיות של $ELOGV$ ונעזרת בערמת מינימום ובאלגוריתם ההקלה. כדי לקבל מטריצת מרחקים של כל הקודקודים יש להפעיל את דייקסטרה על כל קודקוד והסיבוכיות שמתקבלת: $ELOGV * V$

נשווה את סיבוכיות האלגוריתמים: $ELOGV * V$ לעומת v^3

$V^2 > ELOGV$ במקרה שהגרף לא שלם ואין קשת בין כל קודקוד, ולכן בחרנו להשתמש באלגוריתם דייקסטרה.

מימוש האלגוריתם כמובן לא קרה ביום אחד; תחילה שכתבנו בעצמינו את האלגוריתם מפסאוו-קוד כך שיתאים לנתונים.

להלן המחלקות הנדרשות, קודקוד, קשת, גרף:

```
public class vertex
{
    public int color;
    public double distance;
```

```

    public vertex p;
    public E [] neighbor;
}
public class E
{
    public vertex source;
    public vertex dest;
    public double weight;
}
public class G
{
    public vertex[] v;
    public E[] E;
}

```

יש להמיר את נקודות הגישה של החנות והקשתות כך שיתאימו למחלקות הנ"ל, בעזרת פונקציה להמרה:

```

public static G Converter(Getaway [] points)
{
    1. מקבלת נקודות מקור ונקודות יעד מחזירה גרף- ממירה את הטבלאות לנתונים מספריים בלבד
    vertex[] v=new vertex[90];
    E[] e=new E[90];
    int j = 0;
    foreach ( Getaway p in points)
    {
        vertex v1 = new vertex();
        E[] e1 = new E[p.Connections.Count()];
        int i = 0;
        foreach(Connection c in p.Connections)
        {
            e1[i++] = new E() { source = v1, weight =
                Convert.ToDouble(c.Distance) };
        }
        v1.neighbor = e1;
        v[j++] = v1;
    }
    G G = new G() { E = e, v = v };
    return G;
}

```

בפונקציה שלהלן מימשנו קוד דייקסטרה מופשט כפי שלמדנו בקורס גרפים, בהתאמה לשפת #C. זהו קוד האלגוריתם ההתחלתי [למעשה הוא עדיין לא מחזיר שום תוצאה]

```

public static void Dijkstra(G G,vertex s)
{
    2. מקבלת גרף, מקור ויעד ומחזירה את הדרך הקצרה ביותר ביניהם
    int i = 0;
    Init_single_source(G, s);
    List<vertex> S = null;
    vertex u;
    List<vertex> Q = null;
    foreach(vertex v in G.v)
    {
        Q.Add(v);
        while(Q!=null)
        {
            u= (vertex)(Q.Where(y=>y.distance==Q.Min(x=>x.distance)));
            Q.Remove(u);
            foreach (E e in v.neighbor)
                Relax(u, e.dest, e.weight);
        }
    }
}

```



```

    }
}
private static void Relax(vertex u, vertex v, double w)
{
    if (v.distance > u.distance + w)
    {
        v.distance += u.distance + w;
        v.p = u;
    }
}
private static void Init_single_source(G G, vertex s)
{
    foreach (vertex v in G.v)
    {
        v.color = 1;
        v.distance = 999;
        v.p = null;
    }
    s.distance = 0;
}

```

אולם קיימת כאן בעיה בנתונים: לקשתות יש קודקודים, ולכל קודקוד יש קשתות, בעת ההוספה יוצרים עבור כל קודקוד רשימת קשתות, וכל קשת נוצרת פעמיים עבור 2 קודקודים, מה שכמובן אינו הגיוני. כמו כן היה צורך בביצוע שינויים ותיקונים בקוד כדי שנוכל להשתמש בפונקציות. בחרנו להמשיך לחפש דרכים נוספות.

את הקוד הנכון מצאנו לבסוף באתר [Geeks For Geeks](#) אותו שיכללנו ושינינו לפי צרכינו. בנוסף המרנו את הנתונים שלנו מייצוגם הרגיל לייצוג כגרף כדרישת הפונקציה.

התוכנית מפעילה את הפונקציה הראשית במחלקה - [ComputeDijkstra](#). הפונקציה יוצרת גרף מנתוני החנות, כשהגרף מיוצג באמצעות רשימת סמיכויות לסיבוכיות מינימלית ושיפור זמן ריצה.

הנתונים נשמרים במטריצה מסוג [Cell](#). הנה התאור של מחלקה זו:

```

public class Cell
{
    public int i { get; set; } /*מקור*/
    public int j { get; set; } /*יעד*/
    public int distance { get; set; }
    public int parent { get; set; } /*צריך לבדוק איך האבא הגיע ליעד, משנים את המקור*/
}

```

פונקציית ניהול הדייקסטרה: עבור כל קודקוד יחושב מרחק ממנו לכל שאר הקודקודים.

פונקציה זו שולחת לפונקציית הדייקסטרה כל אחד מן הקודקודים כדי שתחשב לו מרחקיו עם הקודקודים האחרים

```

public static Cell[,] ComputeDijkstra(List<GetawayProcI_Result> getaways,
                                     List<Connection> connections)
{
    int numGetaways = getaways.Count() + 1;
}

```

```

Graph graph = ToAdjList(getaways, connections);
Cell[,] mat = new Cell[numGetaways, numGetaways]; // מטריצת המרחקים שתוחזר
Cell[] arr = new Cell[numGetaways]; // משתנה עזר המקבל את תוצאת הדייקסטרה ומועתק למטריצה
foreach (var item in giveaways)
{
    arr = Dijkstra(graph, item.Code);
    for (int j = 1; j < numGetaways; j++)
        mat[Convert.ToInt32(item.I), j] = arr[j];
}

return mat;
}

```

לטיפוס **Graph** יש קודקוד v , ומערך של **AdjList** שמייצג רשימת סמיכויות כלומר קשתות לקודקוד.

להלן המחלקות **Graph**, **AdjList**, **AdjNode** ופונקציה המוסיפה קשת לגרף:

```

public class Graph
{
    public int v;
    public AdjList[] array;
    public Graph() { }
    // A utility function that creates a graph of V vertices
    public Graph(int v)
    {
        this.v = v;
        // Create an array of adjacency lists. Size of array will be V
        this.array = new AdjList[v];
        // Initialize each adjacency list as empty by making head as NULL
        for (int i = 0; i < v; ++i)
            this.array[i] = new AdjList() { head = null };
    }

    public void addEdge(int src, int dest, int weight)
    {
        // Add an edge from src to dest. A new node is added to the adjacency
        // list of src. The node is added at the beginning
        AdjListNode newNode = new AdjListNode(dest, weight);
        newNode.next = this.array[src].head;
        this.array[src].head = newNode;

        // Since graph is undirected, add an edge from dest to src also
        newNode = new AdjListNode(src, weight);
        newNode.next = this.array[dest].head;
        this.array[dest].head = newNode;
    }

    // A ure to represent an adjacency list
    public class AdjList
    {
        // pointer to head node of list
        public AdjListNode head { get; set; }
        public AdjList() { }
        public int SetHead() { head = null; return 1; }
    }

    public class AdjListNode
    {

```

```

public int dest;
public int weight;
public AdjListNode next;

```

להלן הפונקציה הממירה את ה-Connections וה-Getaways של החנות לרשימת סמיכויות:

```

public static Graph ToAdjList(List<GetawayProcI_Result> points,
                               List<Connection> connections)
{
    //המרת קשתות וקדקודים לרשימת סמיכויות
    List<Connection> connectionsGetaway = connections.Where(x => x.TypeDest ==
        Convert.ToInt32(eTypeConnection.getaway)).ToList();
    int V = points.Count() + 1;
    Graph graph = new Graph(V);
    int i = 0;
    int num = points[0].Code;
    foreach (var item in connectionsGetaway)
    {
        int src = Convert.ToInt32(item.CodeSource);
        int dest = Convert.ToInt32(item.CodeDest);

        graph.addEdge(src, dest, Convert.ToInt32(item.Distance));
    }
    return graph;
}

```

```

public static Cell[] Dijkstra(Graph graph, int src)
{
    int vv = graph.v; // Get the number of vertices in graph
    Cell[] dist = new Cell[vv];
    // dist values used to pick minimum weight edge in cut
    for (int i = 0; i < vv; i++)
    {
        dist[i] = new Cell();
        dist[i].i = src;
        dist[i].j = i;
        dist[i].Parent = 0;
        dist[i].distance = int.MaxValue;
    }
    // minHeap represents set E
    MinHeap minHeap = new MinHeap(vv);

    // Initialize min heap with all vertices. dist value of all vertices
    for (int v = 0; v < vv; ++v)
    {
        dist[v].distance = int.MaxValue;
        minHeap.array[v] = new MinHeapNode(v, dist[v].distance);
        minHeap.pos[v] = v;
    }
    // Make dist value of src vertex as 0 so that it is extracted first
    minHeap.array[src] = new MinHeapNode(src, dist[src].distance);
    minHeap.pos[src] = src;
    dist[src].distance = 0;
    minHeap.decreaseKey(src, dist[src].distance);
    // Initially size of min heap is equal to V
    minHeap.size = vv;

    // In the followin loop, min heap contains all nodes
    // whose shortest distance is not yet finalized.
    while (!minHeap.isEmpty())
    {
        // Extract the vertex with minimum distance value
        MinHeapNode minHeapNode = minHeap.extractMin();
        int u = minHeapNode.v; // Store the extracted vertex number

        // Traverse through all adjacent vertices of u (the extracted
        // vertex) and update their distance values
        AdjListNode pCrawl = graph.array[u].head;
        while (pCrawl != null)
        {
            int v = pCrawl.dest;

            // If shortest distance to v is not finalized yet, and distance to
            // through u is less than its previously calculated distance
            if (minHeap.isInMinHeap(v) && dist[u].distance != int.MaxValue &&
                pCrawl.weight + dist[u].distance <
                dist[v].distance)
            {
                dist[v].distance = dist[u].distance + pCrawl.weight;
                dist[v].Parent = u;
                // update distance value in min heap also
                minHeap.decreaseKey(v, dist[v].distance);
            }
            pCrawl = pCrawl.next;
        }
    }
}

```

```
    }  
  }  
  return dist;  
}
```

שימו לב שהשתמשנו ב-**MinHeap** - ערימת מינימום התומכת בפעולת השליפה `extractMin` שסיבוכיותה $O(\log N)$ כידוע, זאת כדי לייעל את זמן הריצה.

אחר שבידינו מטריצת המרחקים של החנות, השמורה לנו, נוכל לחשב מסלולי קניה עבור לקוחות שיבקשו זאת.

מחלקת TravelComputer חישוב מסלול

במחלקה זו יחושב מסלול עבור הלקוח.

הלקוח נכנס לחנות, בוחר מוצרים לקניה, ומבקש מהמערכת לחשב את המסלול האופטימלי. לצורך כך עליו להזין את מיקומו בחנות (בעזרת התכנית) או לבחור להתחיל את הקניה מפתח החנות.

המוצרים נשלחים לשרת, לקונטרולר **WayController**, ששולח אותם למחלקה **TravelComputer** האחראית לחישוב המסלול בצורה מדורגת, בשישה שלבים:

שלב 1: קיבוץ מוצרים שבאותו סטנד באמצעות מילון

מוחזר: Dictionary<Stand, List<ProductShop>>

שלב 2: קיבוץ סטנדים באותו אזור

מוחזר: מילון Dictionary<KeyArea, Area>

שלב 3: מטפל בנקודת פתיחה וסיום ומסדר אזורים,

מוחזר: List<KeyValuePair<KeyArea, Area>>

שלב 4, 5: חישוב מטריצת סמיכויות

מוחזר: Cell[,]

שלב 6: חישוב מסלול אופטימלי,

מוחזר: List<Goal>

השרת מחזיר לאפליקציה את התוצאה, רשימת המוצרים של הלקוח מתמיינת לפי הסדר שחושב ומוצג המסלול האופטימלי המוצגת בצורה נאה וברורה למשתמש.

פירוט השלבים:

שלב 1: צירוף מוצרים באותו הסטנד

יש לקבץ את המוצרים שבאותו סטנד כי ברור שהם נלקחים בפעימה אחת.

הנה הפונקציה:

```

public static Dictionary<Stand, List<Product>> Level1GroupProductsByStand(Product[]
                                                                    products)
{
    #region function level1 מקבץ מוצרים לפי סטנד
    Dictionary<Stand, List<Product>> ExtraStand = new Dictionary<Stand, List<Product>>();
    //level 1: group the product by the shelves
    //key: The common stand, value: list of the product [in the shop]
    foreach (var item in products)
    {
        if (item.ProductShelves.Count == 0) continue;
        Stand s = item.ProductShelves.First().Shelf.Stand;
        Stand anotherKey = ExtraStand.Keys.FirstOrDefault(x => x.Equals(s));
        if (anotherKey==null)
            ExtraStand.Add(s, new List<Product>() { item });
        else //או
            ExtraStand[anotherKey].Add(item);
    }
    return ExtraStand;
}

```

שלב 2: קיבוץ סטנדים באותו אזור

הגדרנו מילון שהמפתח שלו הוא אובייקט מסוג **KeyArea**. טיפוס זה מכיל 2 רשימות:

1. נקודות גישה סמוכות
2. נקודות גישה קרובות ביותר

לאחר שהכרנו את המושג אזור, נבין כי אלו הפרמטרים הקובעים ייחודיות לאזור.

הערך במילון הוא אובייקט מסוג **Area**. טיפוס זה מכיל 3 מאפיינים:

1. רשימת מוצרים המסודרים במילון לפי סטנדים
 2. נקודת התחלת האזור
 3. נקודת סוף האזור
- אופן פעילות האלגוריתם:

```

public static Dictionary<KeyArea, Area> Level2GroupStandsByArea(Dictionary<Stand,
                                                                    List<ProductShop>> ExtraStand, List<GetawayProcI_Result> getaways)
{
    // function level2 מקבץ את הסטנדים לפי אזור

    //level 2: מקבץ את הסטנדים שבאותו אזור - כלומר יש להם אותם הנקודות גישה
    //key: המפתח הוא נקודות הגישה של האזור וגם הנקודות גישה הכי קרובים של האזור
    //value: הערך הוא אזור המכיל רשימת סטנדים, נקודת התחלת אזור, נקודת סוף אזור
    Dictionary<KeyArea, Area> productArea = new Dictionary<KeyArea, Area>();

    foreach (var productStand in ExtraStand)
    {
        KeyArea key = new KeyArea();
        Stand s = new Stand(); ;
        key.Nearestes = productStand.Key.GetConnections().Where(x => x.Nearest
                                                                    == true).Select(x => x.Getaway).ToList();
        List<Getaway> l= productStand.Key.GetConnections().Select(x =>
                                                                    x.Getaway).ToList();
        key.Getaways = Converts.ToGetawayProcResult(productStand.Key

```

```

        .GetConnections().Select(x => x.Getaway).ToList(),
        dtoShop.GetawayProcI_Results(CodeShop));
    KeyArea anotherKey = productArea.Keys.FirstOrDefault(x =>
        x.Equals(key));
    if (anotherKey == null)
    {
        Area ee = new Area();
        ee.ExtraStand[productStand.Key] = productStand.Value;
        productArea.Add(key, ee);
    }
    else
    {
        productArea[anotherKey].ExtraStand.Add(productStand.Key,
            productStand.Value);
    }
}
return productArea;
}

```

שלב 3: בשלב זה מעבירים את המילון לרשימה, ותוך כדי כך מחשבים את מאפייני האזור -שתי נקודות קיצוניות שלו - בעזרת הפונקציה [CalculatePoints](#).

כמו"כ מוסיפים לרשימת האזורים אזור התחלה ואזורי סיום.

אזורי סיום אלו הקופות הקיימות בחנות, (wall עם כינוי 'קופה'). נכניס את כולם לרשימה, ופוקנציית TSP תהיה אחראית לבחור את הקופה האופטימלית למסלול שיבחר.

```

public static Dictionary<KeyArea, Area> Level2GroupStandByArea(Dictionary<Stand,
List<Product>> ExtraStand, List<GetawayProcI_Result> getaways, Shop shop)
{
    #region function level2
    מקבץ את הסטנדים לפי איזור
    //level 2: מקבץ את הסטנדים שבאותו אזור - כלומר יש להם אותם השערים
    //key: המפתח הוא השערים של האזור וגם השערים הכי קרובים של האזור
    //value: הערך הוא איזור המכיל רשימת סטנדים, נקודת התחלה, נקודת סוף אזור
    Dictionary<KeyArea, Area> productArea = new Dictionary<KeyArea, Area>();
    List<Connection> connections;
    Stand stand;
    foreach (var productStand in ExtraStand)
    {
        KeyArea key = new KeyArea();
        stand = productStand.Key;
        connections = stand.GetConnections(shop);
        key.Nearestes = connections.Where(x => x.Nearest == true).Select(x =>
            x.Getaway).ToList();
        key.Getaways = Converts.ToGetawayProcResult(connections.Select(x =>
            x.Getaway).ToList(), dtoShop.GetawayProcI_Results(CodeShop));
        KeyArea anotherKey = productArea.Keys.FirstOrDefault(x => x.Equals(key));
        if (anotherKey == null)
        {
            Area ee = new Area();
            ee.ExtraStand[productStand.Key] = productStand.Value;
            productArea.Add(key, ee);
        }
        else
        {
            productArea[anotherKey].ExtraStand.Add(productStand.Key, productStand.Value);
        }
    }
}

```



```

}
return productArea;
}

```

מחלקת Area:

```

public class Area : I2Points
{
    public Area()
    {
        this.P1 = new Point();
        this.P2 = new Point();
        this.ExtraStand = new Dictionary<Stand, List<ProductShop>>();
    }
    public Point P1 { get; set; }
    public Point P2 { get; set; }
    public Dictionary<Stand, List<ProductShop>> ExtraStand { get; set; }
    public void calculatePoints()
    {
        int xS = Convert.ToInt32(ExtraStand.Keys.Min(x => x.X1));
        int yS = Convert.ToInt32(ExtraStand.Keys.Min(x => x.Y1));
        P1 = new Point() { X = xS, Y = yS };
        P2 = new Point()
        {
            X = Convert.ToInt32(ExtraStand.Keys.Max(y => y.X2)),
            Y = Convert.ToInt32(ExtraStand.Keys.Max(y => y.Y2))
        };
    }

    private void calculateDistances(Point closer)
    {
        Point mid;
        foreach (Stand s in ExtraStand.Keys)
        {
            mid = UtilitiesFunctions.MidPoint(new Point() { X =
Convert.ToInt32(s.X1), Y = Convert.ToInt32(s.Y1.ToString()) }, new Point { X =
Convert.ToInt32(s.X2), Y = Convert.ToInt32(s.Y2) });
            s.Distance = UtilitiesFunctions.CalculteDist(mid, closer);
        }
    }
}

```

בשלב זה עלינו לטפל בנקודת ההתחלה אותה בחר הלקוח. אזור נקודת ההתחלה אינו כולל מוצרים וכתוצאה מכך לא קיימות נקודות גישה השייכות לאזור. יש לחפש אותן באופן כלשהוא כדי שיוכלו להוסיף גם את אזור זה לרשימת היעדים במטריצת המרחקים.

קיימות שתי דרכים מרכזיות למציאת נקודות הגישה, כל אחת על יתרונותיה וחסרונותיה.

בשתי הדרכים יש להשתמש במטריצת החנות שמחושבת על פי נתוני הטבלאות ונעזרת ב-

enum של סוגי שטחים בחנות.

```

public enum eMatMeaning { empty, wall, stand, getaway }

```

תא במטריצה מייצג יחידת מידה (מטר) בחנות ומכיל את מס' מזהה הסוג, על פי ה- `eMatMeaning` ואת הקוד.

כלומר, אם במיקום מסוים יש קיר, ערך התא המייצג אותו במטריצה בספרת המאות יהיה קוד הקיר, ובספרת האחדות יהיה `eMatMeaning.wall=2`.

הדרך הראשונה אמנם נכונה אך תוצאותיה לא השביעו את רצוננו ולכן חיפשנו דרך אחרת שהיא הדרך השנייה כפי שיפורט.

הדרך הראשונה `CalculateGetwaysForPStart`:

בפונקציה זו מכניסים את נקודת ההתחלה לתור, ובתוך לולאה מוציאים מהתור, אם הנקודה שהוצאה היא בתוך טווח החנות בודקים אם קיימת גישה בנקודה זו במטריצת החנות. אם כן, מוסיפים אותה לרשימת הגישות של נקודת ההתחלה ואם לא מכניסים לסוף התור את כל הנקודות במיקומים הסמוכים לה. כך כל עוד התור מלא.

כך זה נראה בקוד:

```
private static void CalculateGetwaysForPStart()
{
    //מספר המציין שעברו במיקום זה במפת החנות
    const int Marker = 11;
    int[][] mat = ComputeMat(null, null, null, 0, 0);
    int[,] SecondMat = (mat.Clone()) as int[,];
    Point p = new Point() { X = 4, Y = 5 };
    Queue<Point> tor = new Queue<Point>();

    tor.Enqueue(p);
    while (tor != null)
    {
        if (tor.First().X < SecondMat.Length && tor.First().Y <
SecondMat.LongLength &&
            tor.First().X >= 0 && tor.First().Y >= 0 &&
SecondMat[tor.First().X, tor.First().Y] > 10 && SecondMat[tor.First().X,
tor.First().Y] != Marker)
        {
            p = tor.Dequeue();
            tor.Enqueue(new Point(p.X, p.Y - 1));
            tor.Enqueue(new Point(p.X, p.Y + 1));
            tor.Enqueue(new Point(p.X + 1, p.Y + 1));
            tor.Enqueue(new Point(p.X - 1, p.Y + 1));
            tor.Enqueue(new Point(p.X - 1, p.Y - 1));
            tor.Enqueue(new Point(p.X + 1, p.Y));
            tor.Enqueue(new Point(p.X - 1, p.Y));
            tor.Enqueue(new Point(p.X + 1, p.Y - 1));
            SecondMat[p.X, p.Y] = Marker;
            //if p is a gateway- enter it to the result array
        }
        else tor.Dequeue();
    }
}
```

לכאורה נראה שהכל טוב ויפה, אולם קיימת בעיה אחת – סריקה זו עלולה להגיע לסיבוכיות גבוהה ולכלול אף את כל החנות. לולאת הפונקציה מתבצעת כל עוד התור מלא ואין דרך לרוקן את התור מנקודות מיותרות שכבר נמצאה הגישה בכיוון. באיטרציה הבאה יוכנסו הנקודות הסמוכות לנקודות אלו, ואח"כ הסמוכות להן, כך עד גלישה משטח החנות, כאשר בשלב זה תופסק ההכנסה. כלומר, הפונקציה אמנם לא תיפול וכן תצליח לבצע משימתה בצורה מדויקת, אך הדרך ארוכה ולא משתלמת. קל יותר לבצע המרת נתוני החנות למטריצה ולחשב בדרך השנייה.

הדרך השנייה **GetStartGetaway**:

פונקציה זו מתקדמת במטריצת החנות לכל הכיוונים עד אשר מוצאת נקודת גישה או סטנד, אותם מוסיפה לרשימה אשר כל תא בה מכיל קוד וסוג. כך נמצא את נקודות הגישה של נקודת ההתחלה.

הוספת הסטנדים שנפגשים איתם באה לפתור את הבעיה שעלולה להיווצר במקרים מסוימים של מבנה חנות פחות סטנדרטי כשיש "מבואות" או טורים היוצרים מרחב סגור, וגם אם נחפש בארבעת הכיוונים לא נמצא את הפתח, כי אין פתח מול הנקודה. במקרה כזה, נשתמש בנקודות הגישה של הסטנד.

החיפוש נעשה באופן ממוקד: סריקה לכל כיוון, ובעת שמוצאים גישה או סטנד -מסתיימת הסריקה לכיוון ההוא.

הפונקציה **StartPointManagment** מפענחת את הקודים שהתקבלו ומחזירה רשימת נקודות גישה.

```
public static List<GetawayProcI_Result> StartPointManagment(Point pStart, int[][] matShop, List<GetawayProcI_Result> getaways)
{
    //נקודת התחלת הקניה
    List<int []> listAns= GetStartGetaway(matShop, pStart, matShop.Length, matShop[0].Length);
    List<GetawayProcI_Result> l = new List<GetawayProcI_Result>();
    List<Stand> stands = new List<Stand>();
    listAns.ForEach(z => l.Add(getaways.Where(d => d.Code == z[0] && z[1] == Convert.ToInt32( eMatMeaning.getaway)).FirstOrDefault()));
    l.RemoveAll(z => z == null);
    if (l == null)
    {
        int code= listAns.Where(x => x[1] == Convert.ToInt32(eMatMeaning.stand)).First()[0];
        Getaway g = stands.Where(y => y.Code == code).First().GetConnections().First().Getaway;
        l.Add(Converts.ToGetawayProcResult(g, getaways));
    }
    //החזרת אזור התחלה שיוסף לרשימת האזורים
    return l;
}

public static List<int[]> GetStartGetaway(int[][] mat, Point pStart, int n, int m)
{
    List<int[]> lis = new List<int[]>();
    //לך קדימה, אחורה, ימינה, שמאלה
}
```

```

int i, j;
i = pStart.X; j = pStart.Y;
while (i < n && mat[i][j] == 0) i++;
if (i < n)
    if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.getaway))
        lis.Add(new int[2] { mat[i][j] / 100, 3 });
    else if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.stand))
        lis.Add(new int[2] { mat[i][j] / 100, 2 });
i = pStart.X;
while (i >= 0 && mat[i][j] == 0) i--;
if (i >= 0)
    if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.getaway))
        lis.Add(new int[2] { mat[i][j] / 100, 3 });
    else if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.stand))
        lis.Add(new int[2] { mat[i][j] / 100, 2 });
i = pStart.X;

while (j < m && mat[i][j] == 0) j++;
if (j < m)
    if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.getaway))
        lis.Add(new int[2] { mat[i][j] / 100, 3 });
    else if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.stand))
        lis.Add(new int[2] { mat[i][j] / 100, 2 });
j = pStart.Y;
while (j >= 0 && mat[i][j] == 0) j--;
if (j >= 0)
    if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.getaway))
        lis.Add(new int[2] { mat[i][j] / 100, 3 });
    else if (mat[i][j] % 100 == Convert.ToInt32(eMatMeaning.stand))
        lis.Add(new int[2] { mat[i][j] / 100, 2 });
return lis;
}

```

מצאנו את נקודות הגישה של אזור ההתחלה,

לאחר שלב זה, האזורים כולם מוכנים.

שלב 4 ו-5: מילוי מטריצת סמיכויות לכל האזורים

בשלב זה קיימת בידינו מטריצת מרחקים של החנות שחושבה כבר בעת טעינת חנות ומכילה מרחקים בין כל נקודות הגישה של החנות.

כמו"כ בידינו רשימה המכילה את כל האזורים בהם יש לבקר במהלך קנייה זו.

נותר לנו לחשב את המרחקים בין כל האזורים בעזרת הידוע לנו אודות המרחקים בין נקודות הגישה. אזור הוא מתחם המוקף נקודות גישה, כך שמאזור לאזור מגיעים דרך נקודת גישה (יש יוצאים מכלל זה – אזורים מחוברים שאין ביניהם נקודת גישה) אם כך מילוי מטריצת המרחקים יהא כמפורט כאן.

תחילה נעתיק את מטריצת המרחקים של נקודות הגישה בחנות:

```
public static Cell[,] Level4ComputeDistanceMatrix(List<KeyValuePair<KeyArea,
Area>> productAreaList,
Cell[,] baseDistanceMatrix, Point pStart, List<Wall> cashes)
{
    #region level4 סמיכויות של החנות גם אזורים
    int numCashes = cashes.Count();
    // שלב 4 מרחיבים את טבלת המרחקים הבסיסית של החנות
    // תוך התחשבות בכך ש'אזור' ההתחלה הוא ה'אזור' הראשון של טבלת החנות
    int LenBaseDistanceMatrix = baseDistanceMatrix.GetLength(1);
    int lenBig = LenBaseDistanceMatrix + productAreaList.Count();
    Cell[,] BigMatrix = new Cell[lenBig, lenBig];
    // מעתיקים את המטריצה של הדייקסטרה למטריצה הגדולה
    for (int i = 0; i < LenBaseDistanceMatrix; i++)
        for (int j = 0; j < LenBaseDistanceMatrix; j++)
            BigMatrix[i, j] = baseDistanceMatrix[i, j];
    }
```

נעבור בלולאה על כל האזורים, ונחשב לכל אחד את המרחק ממנו לכל נקודות הגישה. כך:

ראשית ממלאים את המרחק מהאזור לכל נקודות הגישה שסמוכות לו. מרחק זה מחושב כפי נוסחת מרחקים באופן פשוט.

```
// עובר על האזורים, א, ב, ג, ...
for (int i = LenBaseDistanceMatrix; i < lenBig; i++)
{
    // מילוי מטריצה מאזור לנקודות הגישה שלו
    // עובר על רשימת נקודות הגישה של האזור ומשבץ מרחק במטריצה
    foreach (GetawayProcI_Result gateway in productAreaList[i -
        LenBaseDistanceMatrix].Key.Getaways)
    {
        Cell c = new Cell();
        // עבור האזור הראשון או האחרון המרחק מתמלא באופן ייחודי, מלשאר האזורים
        // או אתה מנקודות הסיום || או שאתה נקודת התחלה
        if (i == LenBaseDistanceMatrix || i >= lenBig - numCashes && i < lenBig)
        {
            Point p;
            if (i == LenBaseDistanceMatrix)
                p = pStart;
            else
                p = gateway.Point;
            c.Distance = 1;
        }
        else
        {
            // מרחק מנקודות הגישה
            int minDistance = int.MaxValue;
            foreach (Point p in gateway.Points)
            {
                int distance = Math.Abs(i - LenBaseDistanceMatrix) + Math.Abs(p.X - pStart.X) + Math.Abs(p.Y - pStart.Y);
                if (distance < minDistance)
                    minDistance = distance;
            }
            c.Distance = minDistance;
        }
        BigMatrix[i, i] = c;
    }
}
```

```

    {
        dtoWall dtoWall = Converts.ToDtoWall(cashes[i - lenBig + numCashes]);
        p = UtilitiesFunctions.MidPoint(dtoWall.P1, dtoWall.P2);
        c.distance = UtilitiesFunctions.getDistanceFromSP(getway, p);
    }
}
else // אם זה אזור רגיל ולא נקודת התחלה/סיום
    c.distance = UtilitiesFunctions.MinDist(productAreaList[i -
        LenBaseDistanceMatrix].Valu, Converts.ToDtoGetawayI(getway));

c.i = i;
c.j = Convert.ToInt32(getway.I);
c.Parent = Convert.ToInt32(getway.I); /*כי יש לי קשת אליו, כי זה גיטואי שלי*/
BigMatrix[i, Convert.ToInt32(getway.I)] = c;
//לצד המקביל
BigMatrix[Convert.ToInt32(getway.I), i] = c;
}

```

כעת נותר למלאות מרחק מהאזור לנקודות הגישה שאינן סמוכות לו, זרות לו.

לכן נעבור בלולאה על כל הנקודות ובתוכה לולאה שעוברת על הגישות הסמוכות, הגישות שלו, ובכל איטרציה נשאל: האם קצר יותר ללכת מהאזור לגישה הזרה דרך הגישה הזו שלו, או שדרך גישה אחרת של האזור עדיפה? כך נגיע בסוף לגישה הטובה ביותר. הקוד:

```

#region מילוי מטריצה מהאזור לכל נקודות הגישה הזרות
for (int j = 1; j < LenBaseDistanceMatrix; j++)
{
    //בודקת איך הכי קצר להגיע אליה מנקודות הגישה שלי
    double min = int.MaxValue;
    Cell minCell = new Cell();
    minCell.i = i;
    minCell.j = j;
    //אם אתה נקודת גישה שלי תמשיך כי שחישבתי קודם
    if (BigMatrix[i, j] != null)
        continue;
    else //אם אתה לא אחד מנקודות הגישה שלי
        foreach (GetawayProcI_Result getway in productAreaList[i -
            LenBaseDistanceMatrix].Key.Getaways)
        {
            //המרחק א' לבי+המרחק מבי לסי
            if (BigMatrix[Convert.ToInt32(getway.I), j].distance + BigMatrix[i,
                Convert.ToInt32(getway.I)].distance < min)
            {
                min = BigMatrix[Convert.ToInt32(getway.I), j].distance + BigMatrix[i,
                    Convert.ToInt32(getway.I)].distance;
                minCell.distance = Convert.ToInt32(min);
                minCell.Parent = Convert.ToInt32(getway.I);
            }
        }
    BigMatrix[i, j] = minCell;
    BigMatrix[j, i] = minCell; //לצד המקביל
}
}
#endregion
// תמלא את המרחק שלך לעצמך, 0
Cell c2 = new Cell();
c2.distance = 0;
c2.Parent = 0;
c2.i = i;
c2.j = i;

```

```
BigMatrix[i, i] = c2;
}
//בשלב זה המטריצה הגדולה מלאה בחציה. חושב המרחק מכל אזור לכל נקודות הגישה. יש לחשב מרחק בין אזור לאזור
```

אחר שמלאנו לכל אזור את מרחקו לכל נקודות הגישה, הגיע הזמן למלא מרחק מכל אזור לכל אזור.

אם יש לאזור המקור (ממנו באים) ולאזור היעד (אליו הולכים) את אותן הגישות אות הוא כי האזורים מחוברים ויש למדוד את המרחק ביניהם.

אחרת עוברים בלולאה על כל הגישות של אזור המקור, ומבררים בכל איטרציה האם כדאי ללכת דרך גישה זו לאזור היעד [שהרי קיים באזור היעד את המרחק לכל נקודות הגישה].

```
//להגיע מאזור לאזור, א,ב,ג,
for (int i = LenBaseDistanceMatrix; i < lenBig; i++)
{
    for (int j = LenBaseDistanceMatrix; j < lenBig; j++)
    {
        if (i != j && BigMatrix[i,j]==null)
        {
            int min = int.MaxValue;
            Cell c = new Cell();
            c.i = i;
            c.j = j;
            // במקרה שאזורים אלו בעלי אותם נקודות גישה - זה אומר שאפשר לגשת ישירות ביניהם. נחשב אמצע קטע של אזור ראשון ואמצע קטע של
            // אזור שני ופשוט נחשב מרחק
            if (UtilitiesFunctions.IsSame(productAreaList[i - LenBaseDistanceMatrix].Key.Getaways,
                productAreaList[j - LenBaseDistanceMatrix].Key.Getaways))
            {
                //אם כל נקודת גישה של א היא גם נקודת גישה של ב/
                //א=i
                //ב=j
                c.Parent = i;
                //להגיע מאזור לאזור כשהם צמודים ולא צריך לעבור דרך נקודת גישה/
                //אם אחד מהם זה נקודת התחלה אין לו ערך, נחשב לחוד/
                I2Points a = productAreaList[i - LenBaseDistanceMatrix].Value;
                I2Points b = productAreaList[j - LenBaseDistanceMatrix].Value;
                //אם האזור הוא מנקודות הסיום || אם האזור הוא נקודת התחלה/
                if(i==LenBaseDistanceMatrix || i >= lenBig - numCashes && i < lenBig)
                {
                    //אם זה נקודת התחלה/
                    if(i == LenBaseDistanceMatrix)
                    {
                        if (a == null)
                            a = new dtoStand() { P1 = pStart, P2 = pStart };
                        if(b==null) b = new dtoStand() { P1 = pStart, P2 = pStart };
                    }
                    //אם זה מנקודות הסיום/
                    else if(i >= lenBig - numCashes && i < lenBig)
                    {
                        if (a == null)
                            a = Converts.ToDtoWall(cashes[i - lenBig + numCashes]);
                        if (b == null) b = Converts.ToDtoWall(cashes[j - lenBig + numCashes]);
                    }
                }
                c.distance = UtilitiesFunctions.MinDist(a, b);
                BigMatrix[i, j] = c;
                Cell c2 = new Cell() { i = j, j = i, distance = c.distance, Parent = j };
                BigMatrix[j, i] = c2;
            }
            //אם האזורים אינם בעלי אותם נקודות גישה, תעבור על נקודות הגישה של המקור, של אי/
            else
            {
                foreach (GetawayProcI_Result gateway in productAreaList[i -
```

```

        LenBaseDistanceMatrix].Key.Getaways)
    {
        if (BigMatrix[j, Convert.ToInt32(getway.I)].distance + BigMatrix[i,
            Convert.ToInt32(getway.I)].distance < min)
        {
            c.Parent = Convert.ToInt32( getway.I);
            min = BigMatrix[j, Convert.ToInt32(getway.I)].distance + BigMatrix[i,
                Convert.ToInt32(getway.I)].distance;
            c.distance = min;
        }
    }
    BigMatrix[i, j] = c;
    //לצד המקביל
    Cell c2 = new Cell() { i = j, j = i, distance = c.distance, Parent = c.Parent };
    BigMatrix[j, i] = c2;
}
}
}
return BigMatrix;
}

```

אחר שמלאנו מרחק מכל אזור לאזורים האחרים לא נזדקק למרחקי נקודות הגישה. נשתמש רק בטבלת המרחקים של האזורים.

```

public static Cell[,] Level5CutMatrix(Cell[,] matrix, int countAreas, int matrixLen)
{ //function level5 צור מטריצת סמיכויות מהאזורים בלבד
    int from = matrixLen - countAreas;
    Cell[,] AreasMatrix = new Cell[countAreas, countAreas];
    for (int i = from; i < matrixLen; i++)
    {
        for (int j = from; j < matrixLen; j++)
            AreasMatrix[i - from, j - from] = matrix[i, j];
    }
    return AreasMatrix;
}

```

שלב 6: בשלב זה שולחים את האזורים לחישוב הסדר האופטימלי ע"י מחלקת [TspFunctions](#), ומקבלים במערך את האזורים לפי סדר.

כעת יש לחשב 2 דברים:

1. חישוב סדר פנימי בתוך האזור, סדר מעבר בסטנדים שבאזור

2. חישוב המסלול בין האזורים.

יש לקרוא ממטריצת המרחקים, משדה ה[Parent](#) את המסלול בין אזור לאזור, את נקודות הגישה שיש לעבור בהם כדי להגיע מהמקור ליעד. לכך השתמשנו בפונקציה [RqorsibitHelpComputeAnsFromGtoG](#)

הקוד המובא עובר מאזור לאזור עפ"י הסדר האופטימלי שחושב, ומסדר ברשימה את כל הגישות והסטנדים בהם צריך לעבור, לפי הסדר, וכך יוכלו לשרטט את המסלול.


```

private static List<Goal> ComputeWayAnswer(Cell[,] BigMatrix, int[] order, int
numAreas, int numGetaways, int startIndex, List<KeyValuePair<KeyArea, Area>>
productAreaList, List<GetawayProcI_Result> getawayProcI_Results, Point pStart)
{
    List<Goal> way = new List<Goal>();
    List<Goal> AddWay = new List<Goal>();
    int from, to;
    int gFrom, gTo;
    //בכך בזאת מוסיפים את האזור הראשון שהוא רק התחלה ללא מוצרים
    way.Add(new Goal('p', startIndex, null, pStart));
    Goal w; //אזור להיות ריק-
    //עוברים על כל האזורים
    for (int i = 0; i < numAreas; i++)
    {
        from = order[i] + numGetaways + 1;
        to = order[i + 1] + numGetaways + 1;
        gFrom = BigMatrix[from, to].Parent;
        if (gFrom == 0) continue;
        //אם לא מגיעים ישירות מאזור לאזור אלא עוברים דרך נקודות גישה
        if (!(gFrom == from || gFrom == to))
        {
            gTo = BigMatrix[gFrom, to].Parent;
            if (gFrom == gTo) { gTo = gFrom; gFrom = BigMatrix[from, gFrom].Parent; }
            AddWay.Add(new Goal('g', gFrom, null, null));
            HelpComputeAnsFromGtoG(BigMatrix, gFrom, gTo, AddWay, false, 0);
            AddWay.Add(new Goal('g', gTo, null, null));
        }
        way.AddRange(AddWay);
        AddWay.Clear();
        //הגענו לאזור היעד
        Area area = productAreaList[order[i+1]].Value;
        if (area != null) {
            w = new Goal('a', to - numGetaways -
                        1, null, UtilitiesFunctions.MidPoint(area.P1, area.P2));
            OpenArea(way.Last(), w.num, productAreaList, way, getawayProcI_Results);
            way.Add(w);
        }
    }
    return way;
}

```

הפונקציה [HelpComputeAnsFromGtoG](#) היא הרקורסיבית

```

public static void HelpComputeAnsFromGtoG(Cell[,] BigMatrix, int from, int to,
List<Goal> answers, bool b, int reelTo)
{
    int cur = BigMatrix[from, to].Parent;
    if (cur == from || cur == to || cur == 0)
        return;
    else
    {
        HelpComputeAnsFromGtoG(BigMatrix, from, cur, answers, false, 0);
        answers.Add(new Goal('g', cur, null, null));
    }
}

```

הפונקציה [OpenArea](#) היא הפותחת בפועל כל אזור ומסדרת לפי הסדר את הסטנדים. כמו כן, מחשבת נקודה עבור כל סטנד, כדי שהאפליקציה תדע כיצד לצייר את המסלול.

```
private static void OpenArea(Goal gw, int num, List<KeyValuePair<KeyArea, Area>> productAreaList,
                             List<Goal> way, List<GetawayProcI_Result> getawayProcI_Results)
{
    GetawayProcI_Result g =getawayProcI_Results.Where(x => x.Code == gw.num).First();
    List<ProductShop> products = new List<ProductShop>();
    //למיון את הסטנדים לפי המרחק מנקודת גישה
    Area area = productAreaList[num].Value;
    if (area != null)
    {
        List<int> orderStand = area.sortStands(Converts.ToDtoGetawayI(g));
        foreach (var item in orderStand)
        {
            var curExtraStand = area.ExtraStand.Where(x => x.Key.Code == item).First();
            Point p = UtilitiesFunctions.MidPoint(new Point(Convert.ToInt32(
                curExtraStand.Key.X1.ToString()),
                Convert.ToInt32(curExtraStand.Key.Y1.ToString()), new
                Point(Convert.ToInt32(curExtraStand.Key.X2.ToString()),
                Convert.ToInt32(curExtraStand.Key.Y2.ToString())));
            way.Add(new Goal('s', item, Converts.ToDtoProductShops( curExtraStand.Value) ,p));
        }
    }
}
```

מיון הסטנדים שבאזור:

```
public List<int> SortStands(I2Points g)
{
    dtoStand stand;
    List<int> arr = new List<int>
    {
        Capacity = ExtraStand.Count()
    };
    Point midOfG = UtilitiesFunctions.MidPoint(g.P1, g.P2);
    Point closerer;
    if (UtilitiesFunctions.CalculteDist(midOfG, P1) < UtilitiesFunctions.CalculteDist(midOfG, P2))
        closerer = P1;
    else
        closerer = P2;
    for (int i = 0; i < ExtraStand.Count(); i++)
    {
        /* מחשב מרחק לכל סטנד */
        calculateDistances(closerer);
        ExtraStand.Keys.ToList().Sort();
        stand = Converts.ToDtoStand(ExtraStand.Keys.Where(x => !arr.Contains(x.Code)).First());
        arr.Add(stand.Code);
        closerer = UtilitiesFunctions.MidPoint(stand.P1, stand.P2);
    }
    return arr;
}
```

למעשה, התוצאה המתקבלת היא רשימה מסוג **Goal**, כלומר, רשימת יעדי המסלול, המוחזרת דרך הקונטרולר אל ממשק האגולר המצייר ללקוח את המסלול בצורה פשוטה (תיאור בהמשך תחת הכותרת "ממשק")

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 מיד) 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 סידור בתמורה. כך, אם מספר הערים הוא n , אז יש בסך הכל $n!$ 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)

$N!$	N^2	N
1	1	1
2	4	2
6	9	3
24	16	4
120	25	5
720	36	6
5,040	49	7
40,320	64	8
362,880	81	9
3,628,800	100	10

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 של Polynomial. מהן פעולות המחשב המותרות? מדובר בפעולות דטרמיניסטיות, כלומר אם נרץ 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 (לחייב) 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 למציאת פתרון מדויק (יעבדו בזמן סביר רק עבור מספר קטן של ערים).

- תכנון אלגוריתמים היוריסטים 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 לפתרונות מקורבים שלה.

דוגמה לאלגוריתם חמדן 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 החיפוש 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 חיצוניים של חישובי מרחק (כגון ספריט orTools של google, tsplib.net העוטפת את ספריט 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 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"י 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 task – 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 משרד 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)
    {
        // בחר 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 ותמחקר 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[]
                                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-approximatelyTsp יתבצע 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 זמן 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 שוקי' 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 שהזנו 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 ב-parent 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 [ExtraAlias](#) היא מחלקה היורשת מ-[Alias](#) אך 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 [ExtraAlias](#) 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 **Area**, **dtoStand**, **dtoGetaway** כממשות 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 CalulteDist(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 (p1Left != 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 ו-Door, in, ועוד.

פגשנו גם בממשק להדמיה תלת ממדית הנקרא three.js. מדובר בספריית JavaScript לעיצוב D3 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
    }
}
```

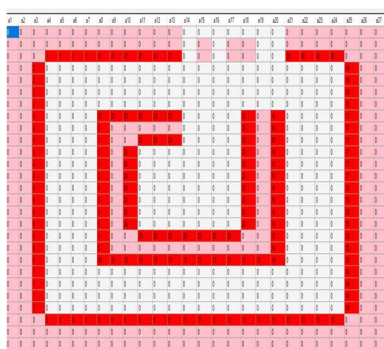
```

}
// 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 בDataGridview בממשק WinForm לבדיקה



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

ערכי `arrColor` הם `[white, grey, pink]`

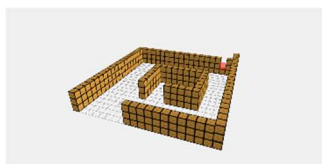
```
<table border="0">
  <tr *ngFor="let item of mat;let ind1=index" >
    <td *ngFor="let item1 of item;let ind=index" (click)="func(ind1,ind)"
[ngStyle]="{'background-color':arrColor[item1%10]}" >
      </td>
    </tr>
  </table>
```

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();
    //draw 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 squire:Square=new Square(this.ctx);
        squire.draw(p1.X*this.mul,p1.Y*this.mul, w,h,"rgb(185, 185, 183)");
    }

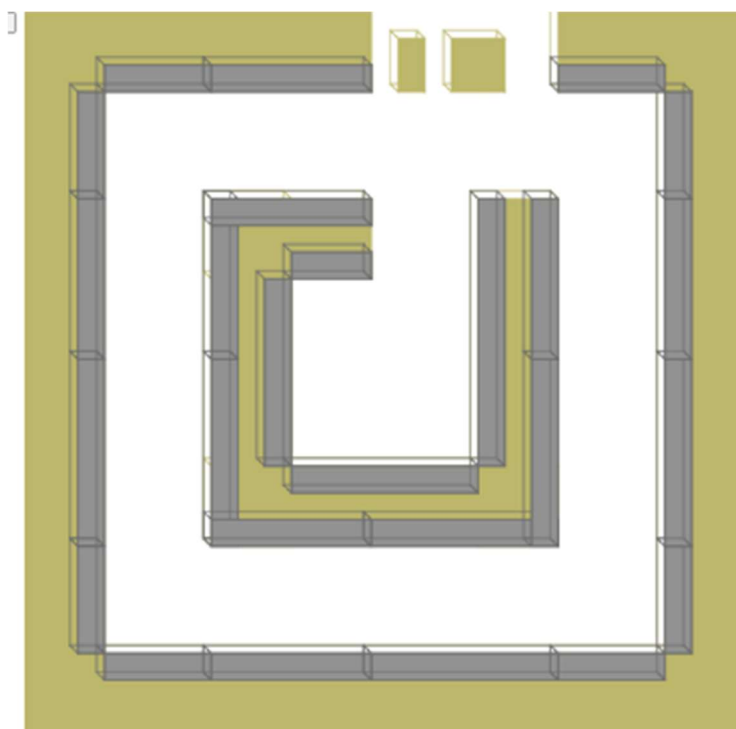
    //draw stands
    // -----
    this.ctx.fillStyle = "rgb(146, 146, 146)";
    for(let i:number=0; i<s.Stands.length; i++)
    {
        let st:Stand=s.Stands[i];
        let p1:Point=st.P1;
        let p2:Point=st.P2;
        let w=(p2.X-p1.X+1)*this.mul;
        let h=(p2.Y-p1.Y+1)*this.mul;
        this.ctx.fillRect (p1.X*this.mul,p1.Y*this.mul, w,h );
        // 3d
        let squire:Square=new Square(this.ctx);
        squire.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>](#) 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 `eventEmitter`. 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($event)" >
    </app-using-canvas>
  </div>
  <div class="column side">
    <app-buying (onSelectProduct)="inputProduct($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:

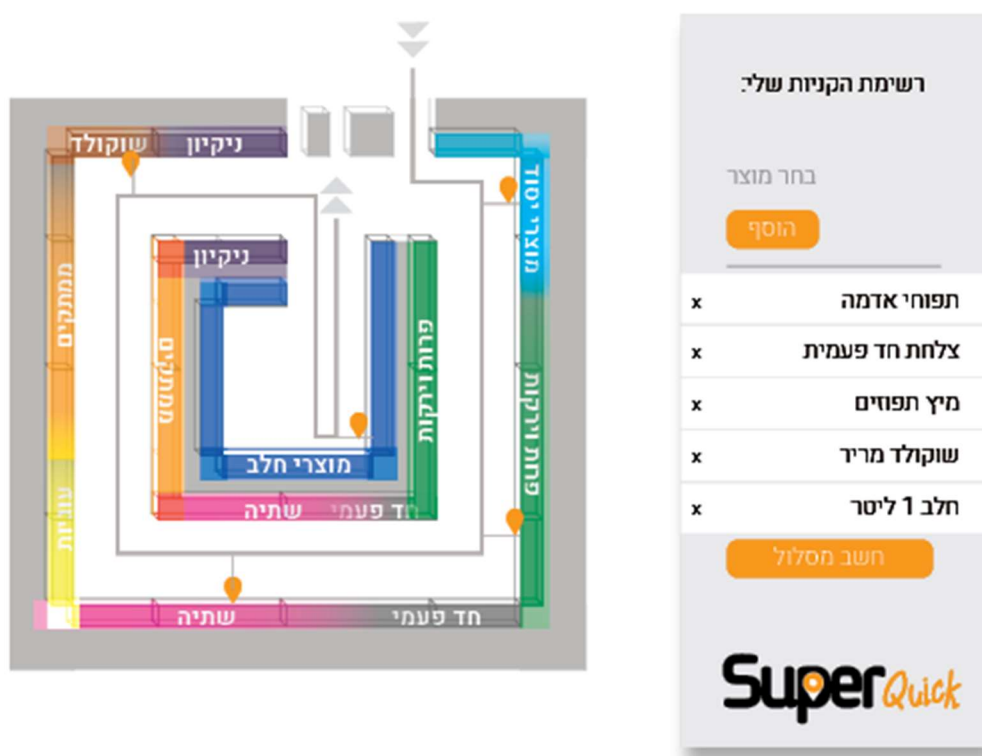
- עבור מנהל יוצג מסך התחברות.
- עבור לקוח יוצג מסך קנייה.
- במידה ולא נבחרה חנות יוצג מסך הוספת חנות חדשה.

2. 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.

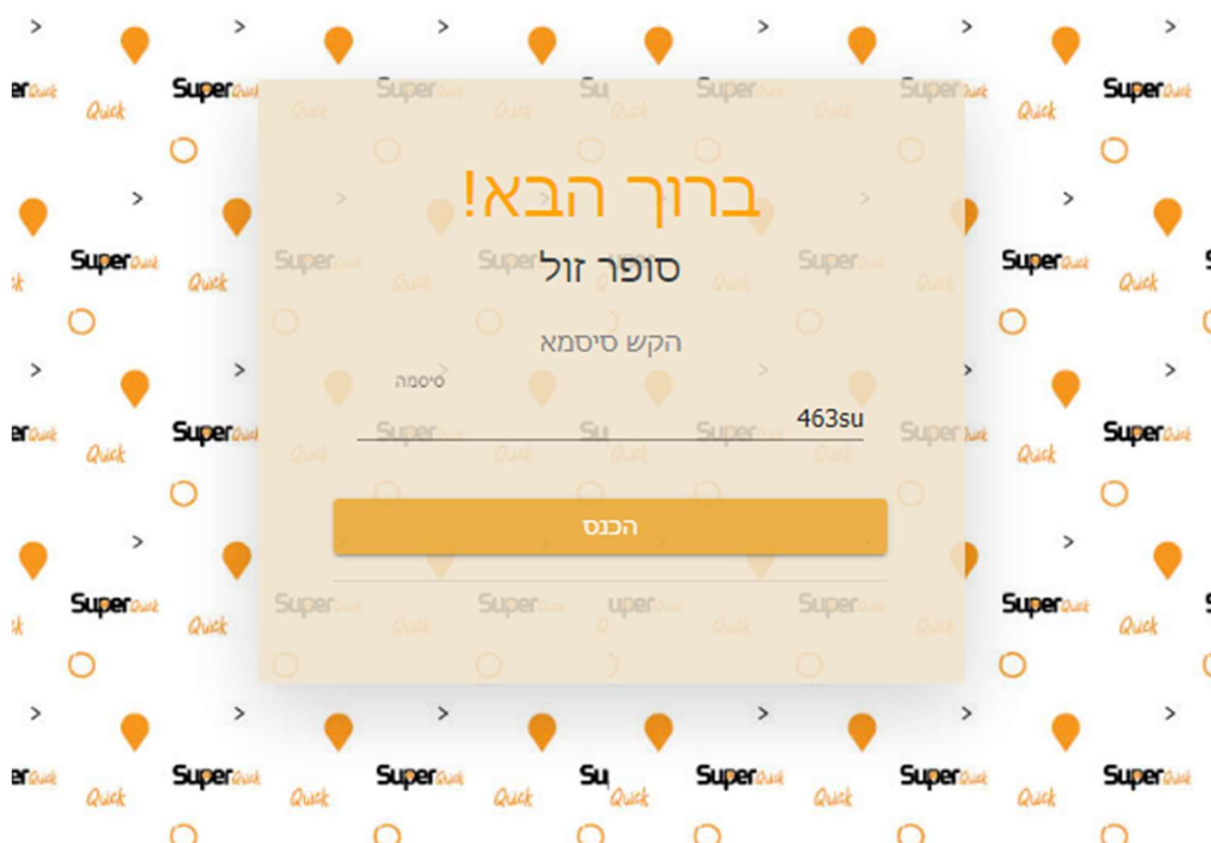


בלחיצה על 'חשב מסלול' מתבקש הלקוח ללחוץ על המפה באזור בו הוא נמצא. לחיצה זו תפעיל את תהליך חישוב המסלול האופטימלי. כשמסיים השרת את החישוב הוא מחזיר את המסלול, ועל המפה יוצג שרטוט המסלול האופטימלי.

3. 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 תווים לפחות, שם החנות לא ישאר ריק. נתוני החנות נשלחים מיד לשרת שם מתבצעת המרת הנתונים וחישוב מטריצת מרחקים לחנות.

4. 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= <RequirestResponse>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

הקש ברקוד מוצר לחיפוש

הקש קוד חנות

הצג מוצרים נבחרים

הצג מוצרים שלא נבחרו

הצג הכל

שומר

סטנד	משחקים	מכשירי חשמל	כלי בית	מכשירי כתיבה	הלבשה	דגים	בשר	קפואים	פיצוחים	שימורים	חלב	ניקיון	אפיה	פירות וירקות	חפ	שתי
------	--------	-------------	---------	--------------	-------	------	-----	--------	---------	---------	-----	--------	------	--------------	----	-----

הוסף מוצר למחלקה

חלב ☐

שוקו ☒

קפה ☒

גבינה לבנה ☐

גבינה צהובה ☒

גבינה בולגרית ☐

קרלז וניל ☐

הפעל!
עבור אל'

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

המוצרים, ממיינים 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);
});
}

```

שימורים

חלב

ניקיון


אפיה

סטנד

משחקים

מכשירי חשמל

רוצה להוסיף אותי לחנות ?



בחירת קובץ

מארח שישית ש...וקו תגובה.png

שוקו שוקו 250 tnuva

קוראים לי גם

אשוקו

אשקית שוקו

מחזרת הרשימה בחר מוצר

הוסף

משקה קקאו

כן

לא

חלב

שוקו

קפה

גבינה לבנה

גבינה צהובה

גבינה בולגרית

קרלו וניל

קרלו שוקו

מעדן תאומים

חמאה

מרגרינה

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 [ParentId](#) 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 תווים.

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 אפינו 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'. זו 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 חלונות מעקב והצצה תוך כדי תהליך החישוב.

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

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.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0																											
1																											
2																											
3																											
4																											
5																											
6																											
7																											
8																											
9																											
10																											
11																											
12																											
13																											
14																											
15																											
16																											
17																											
18																											
19																											
20																											
21																											
22																											
23																											
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 דיקסטר: [להלן המרחקים]

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

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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	2	15	17	30	28	15	13	6	12	16	5	12	26	11	10
2	2	0	13	15	28	30	17	15	8	14	18	3	10	24	13	12
3	15	13	0	2	15	17	30	28	21	27	28	10	3	11	26	25
4	17	15	2	0	13	15	28	30	23	29	26	12	5	9	28	27
5	30	28	15	13	0	2	15	17	24	24	13	25	18	4	21	23
6	28	30	17	15	2	0	13	15	22	22	11	27	20	6	19	21
7	15	17	30	28	15	13	0	2	9	9	1	20	27	19	6	8
8	13	15	28	30	17	15	2	0	7	7	3	18	25	21	4	6
9	6	8	21	23	24	22	9	7	0	6	10	11	18	28	5	4
10	12	14	27	29	24	22	9	7	6	0	10	17	24	28	8	8
11	16	18	28	26	13	11	1	3	10	10	0	21	28	17	7	9
12	5	3	10	12	25	27	20	18	11	17	21	0	7	21	16	15
13	12	10	3	5	18	20	27	25	18	24	28	7	0	14	23	22
14	26	24	11	9	4	6	19	21	28	28	17	21	14	0	25	27
15	11	13	26	28	21	19	6	4	5	8	7	16	23	25	0	2
16	10	12	25	27	23	21	8	6	4	8	9	15	22	27	2	0

אישור

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

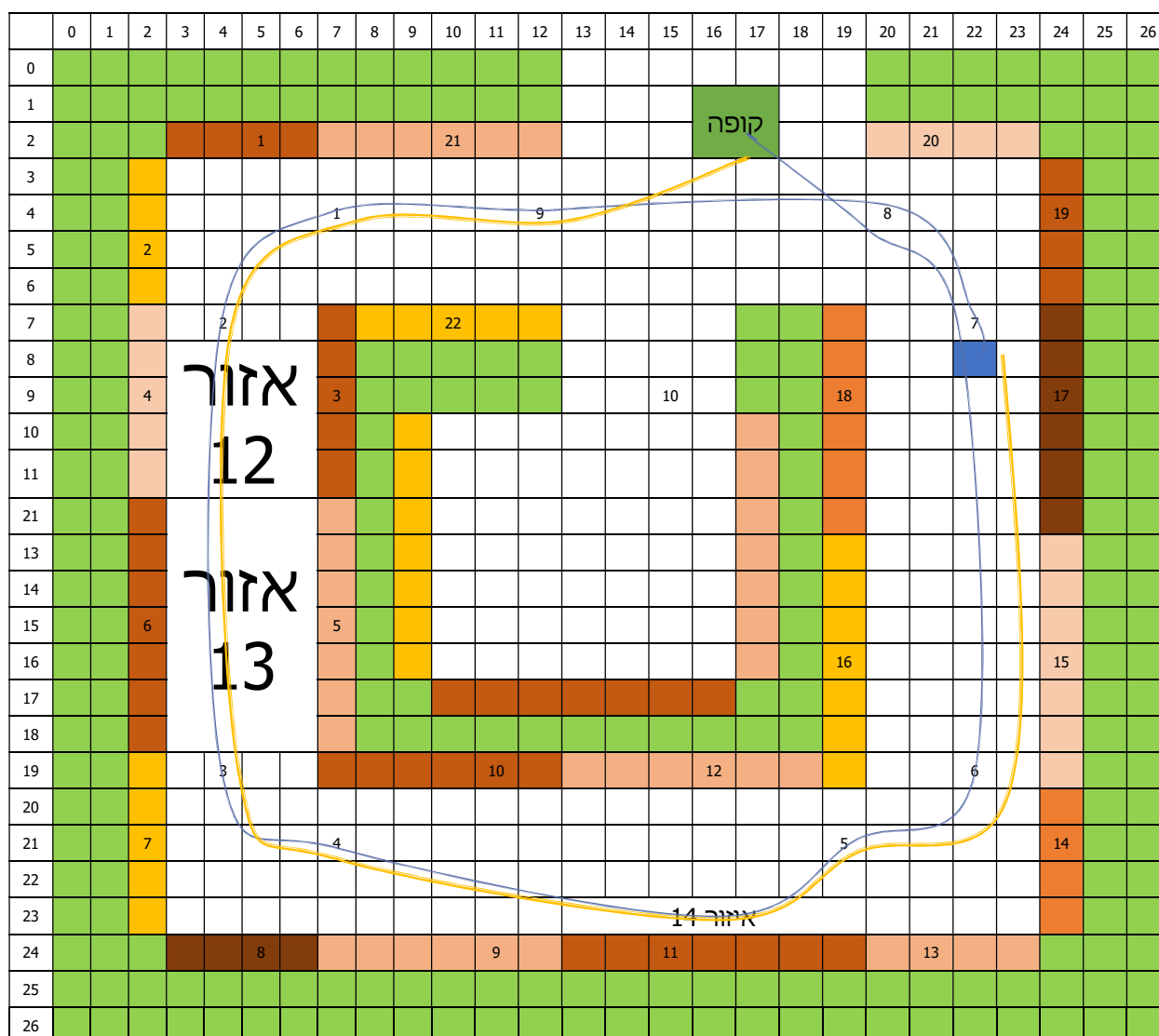
וזו המטריצה שנשלחה לחישוב סדר המסלול בפונקציית ה-TSP:

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

כמובן, היות שמדובר במטריצה בגודל 6 חושבה פונקציית ה-TSP המדויקת.

הסדר שחזר:

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



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.

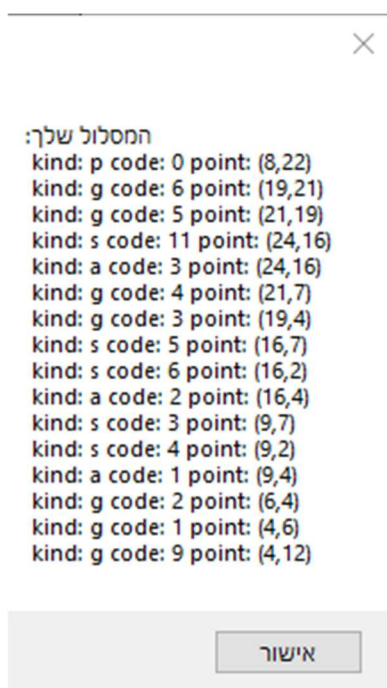
המסלול יצויר בממשק המשתמש.

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

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

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

המאפיין point מציין את הנקודה או נקודת האמצע, משמש לממשק המשתמש, בעזרתו יוכל הקנוס 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- SuperQuick !!!