



פרויקט גמר 2022

תהל לוי | 212697676

תוכן

1.	מבוא / תקציר	6
2.2	תהליך המחקר	7
2.3	סקירת ספרות	7
2.	מטרות ויעדים	8
3.	אתגרים	9
4.	מדדי הצלחה	9
5.	תיאור המצב הקיים	10
6.	רקע תאורטי	10
7.	ניתוח חלופות מערכת	11
8.	תיאור החלופה הנבחרת והנימוקים לבחירה	12
9.	אפיון המערכת	15
9.1	ניתוח דרישות המערכת	15
9.2	מודול המערכת	15
9.3	אפיון פונקציונאלי	16
9.4	ביצועים עיקריים	17
9.5	אילוצים	17
10.	תיאור הארכיטקטורה	17
10.1	הארכיטקטורה של הפתרון המוצע בפורמט של Design level Down-Top	17
10.2	תיאור הרכיבים בפתרון	18
	תיאור פרוטוקולי התקשורת	19
	שרת – לקוח	19
11.	ניתוח ותרשים use case של המערכת המוצעת	20
	רשימת use case	21
	תיאור ה-use case העיקריים של המערכת	21
	מבני נתונים בהם משתמשים בפרויקט	22
11.1	תרשים מחלקות	23
	תיאור המחלקות	24
12.	תיאור התוכנה	25
13.	אלגוריתמים מרכזיים	25
14.	קוד האלגוריתם	26
15.	תיאור מסד הנתונים	34
	פירוט הטבלאות ב- Data Base	34
16.	מדריך למשתמש	37
16.1	תיאור המסכים	37

38.....	מדריך למשתמש	
39.....	צילומי מסכים	16.2
42.....	בדיקות והערכה	17
42.....	ניתוח יעילות	18
42.....	אבטחת מידע	19
43.....	מסקנות	20
43.....	פיתוח עתידי	21

הצעת פרויקט

סמל מוסד: 560342

שם מכללה: סמינר תורני בית יעקב.

שם הסטודנט: תהל לוי.

ת"ז הסטודנט:

212697676.

שם הפרויקט: EazyShop

תיאור הפרויקט:

האפליקציה תאפשר כניסה למשתמש חדש, משתמש קיים, מלקטים ומנהל.

המשתמש החדש יתבקש להכניס פרטים אישיים כגון: שם וטלפון ולאחר שירשם במערכת יוכל לבצע רשימת קניות.

ביצוע הרשימה יתבצע בשני אופנים או על ידי חיפוש מוצרים חופשי או על ידי כניסה לקטגוריות. לאחר שהמשתמש יסיים לערוך את רשימת הקניות האפליקציה תחשב לו את המסלול הקצר ביותר בתוך הסופר.

בסיום הקניה המשתמש יוכל לבחור אם לשמור את הרשימה שערך או לא במידה וזו קניה חד פעמית ולא נשנית.

המשתמש הקיים יוכל לערוך רשימת קניות חדשה או לבחור מבין רשימות קודמות שבחר לשמור במערכת וכן לערוך בהם שינויים.

המלקטים עבודתם היא לאסוף מוצרים מהמדפים, לפי ההזמנה של לקוח מסוים. ולארוז אותם בצורה טובה. זה יסייע להם בכך שהם יוכלו להכניס את הרשימות שעליהם ללקוט ואפליקציה תאחד להם את הרשימות הדומות למסלול אחד וקצר. ובכך תחסוך להם זמן רב בעבודתם.

המנהל יכנס למערכת ויוכל לעדכן שינויים שנעשו בסופר.

הגדרת הבעיה האלגוריתמית:

חישוב מסלול יעיל וקצר תוך התחשבות באילוצים השונים.

בתחילה הוגדרה הבעיה כבעיית "הסוכן הנוסע" אך לאחר הבנה של תוכן האלגוריתם הגענו למסקנה שבעיה זו אינה פתירה משום שיעילות זמן הריצה שלה היא אין סופית. ולכן האלגוריתם המקורב ביותר לפיתרון זה הוא greedy Dijkstra.

אלגוריתם זה פותר את הבעיה של מסלולים קצרים ביותר ממקור יחיד עבור גרף מכוון וממושקל, עבור משקולות אי שליליים בלבד.

תוצאת האלגוריתם של דייקסטרה זהה לתוצאת אלגוריתם בלמן-פורד אך אלגוריתם בלמן-פורד פועל גם על גרפים הכוללים קשתות שמשקלן שלילי (שאצלי אין צורך בכך משום שזו מפה ואין משקולות שליליים). כמו כן זמן הריצה של אלגוריתם דייקסטרה מהיר יותר.

הבעיה העיקרית היא מציאת הצמתים וחישוב הקשתות מתוך המפה של המבנה ותירגומה לקוד אך בעיה זו אכן פתירה.

האלגוריתם יופעל לאחר שנמצא עבורו 2 מסלולים דומים ברשימותיהם. ויאלץ לסנכרן בין 2 המסלולים ומציאת המסלול הקצר ביותר ובכך יסייע גם למלקטים בעבודתם וגם לצרכים הרוצים לבצע שני קניות במקביל.

הבעיה הנוספת שעומדת לרשותנו הינה מציאת מיקום בתוך מבנה. תכולת המבנים הגדולים כיום מכילה ברזל שיוצר עיוותים במציאת המיקום.

ובכן קיימות מס' גישות לפתרון הניווט במבנה סגור:

הגישה הראשונה מדברת על שימוש באותות רדיו קיימים, כגון אותות סלולריים או אותות Wi-Fi בכדי לחשב את המיקום הנוכחי.

הגישה השנייה היא להתחיל עם מיקום ידוע המבוסס על לווייני ה-GPS ובעת שהמכשיר נכנס לתוך הבניין, להתחיל לעקוב אחרי מיקומו של המכשיר באמצעות חיישן התנועה המובנה בו.

הגישה השלישית היא הוספת מכשירים חדשים, המבוססים על גלי רדיו אשר ניתן להשתמש בהם בקלות וביעילות בכדי לקבוע את המיקום שלכם באיזורים מסויימים אשר ברצונכם לכסות.

הגישה הרביעית היא השימוש בגישות איזוטריות אחרות בכדי לקבוע את המיקום הגישות הללו עדיין לא נפרסו ביישומים מעשיים והם עדיין לא פתורות.

בניגוד לגישות האחרות המסתמכות על קישוריות בלוטות Wi-Fi, או סממנים פיזיים הפזורים בבית העסק הגישה השניה הינה הפתירה ביותר והזולה ביותר לפיתוח, משום שחיישן זה נמצא כיום בכל סמארטפון. על המשתמש בסך הכול להכניס את מיקומו הראשוני. בנוסף גישה זו הינה המדויקת ביותר ברמת דיוק של מטר אחד בלבד ולכן נפתור על פי הגישה הזו.

רקע תאורטי בתחום הפרויקט:

לכולנו יצא יותר מפעם אחת להיכנס לסופר שאנחנו לא מכירים ולערוך בו קניות. דבר זה עולה לנו זמן רב תוך חזרה על מקומות שביקרנו בהם וחוסר מציאת המוצרים המבוקשים. מחקרים שונים מראים שכ-30% מהקונים יוצאים מן החנות מבלי שרכשו לפחות מוצר אחד שהתכוונו לקנות, בין אם לא מצאו אותו או בין בגלל שמיהרו. הפתרון לכך הוא חישוב המסלול הקצר ביותר תוך התחשבות בכל האילוצים של המשתמש.

תהליכים עיקריים בפרויקט:

1. המערכת תחשב את מפת הסופר.
2. המערכת תאפשר ללקוח לקבל את המסלול הקצר על פי רשימת הקניות הנתונה.
3. המערכת תחשב ללקוח את סכום הקניה שלו בכל שלב של הקניה.
4. המערכת תמליץ ללקוח על המבצעים בסופר.
5. המערכת תאחד בין שני מסלולים למסלול אחד קצר ביותר על פי דמיון בין הרשימות.

תיאור הטכנולוגיה:

צד שרת: Web-API

שפת תכנות בצד השרת: C#

צד לקוח:

שפת תכנות בצד לקוח: אנגולר

מסד נתונים: SQL

חתימת הסטודנט:

חתימת הרכז המגמה:

אישור משרד החינוך:

1. מבוא / תקציר

1.1 הרקע לפרויקט

בפרויקט הגמר מטרת העל היא התנסות ולמידה עצמית אך לצד זה עבודה על נושא מעניין שיסייע לך ליהנות מהעבודה ומהתהליך.

בבחירת הרעיון לפרויקט התייעצתי עם אנשים רבים לגבי רעיון שיסייע לכלל הציבור. חיפשתי רעיון חדשני, רעיון שאין אותו כיום, נורא רציתי להקל על אנשים ועל אורח החיים שלהם. ולאחר מחשבות רבות הגעתי להחלטה לעשות מסלול מקוצר בסופר. הגעתי לרעיון הזה משום שלפני כמה שנים עברנו דירה והגעתי לסופר חדש שלא הכרתי כלל ושמתי לב שלוקח לי זמן רב למצוא כל מוצר ואני חוזרת המון פעמים למקומות שכבר ביקרתי בהם ולכן חשבתי איך עדיין לא פיתחו דבר כזה של waze בסופר. באותה תקופה הייתי במגמת תכנות בתיכון והתייעצתי עם המורה על הרעיון לגביי פרוייקט גמר של כיתה יב אבל המורה אמרה שזה מידי מסובך ולכן דחיתי את הרעיון לעכשיו. וכעת אני יישמתי אותו.

בחרתי דווקא ברעיון זה לפרויקט גמר משום שקיים בו אלגוריתם מאתגר וחכם ובנוסף למדנו על **Dijkstra** במבני נתונים והתחברתי מאוד לרעיון.

האפליקציה שלי מסייעת לצרכן בכך שחוסכת לו זמן רב בחיפוש המוצרים שלו ובחזרה על מקומות רבים שהיה בהם כבר, כמו כן לבזבז כסף אדיר. שהרי מטרת בעלי הסופרים היא שהלקוחות יוסיפו לעגלה שלהם מבצעים או מוצרים נוספים שלא בהכרח זקוקים להם ולכן כשהלקוח מזין את רשימת הקניות שלו מוצג לו מסלול מדויק שמאפשר לו לבצע את הקנייה שלו במדויק ללא הוספה של מוצרים שלרוב מיותרים עבורו. כמו כן ביצוע הקנייה במינימום של זמן משום שהמפה מציגה לו את הנקודות המדויקות שבהם עליו לעבור בזמן הקצר ביותר.

את האפליקציה שייצרתי יצרו רבים לפניי אך בEasyShop ישנו חידוש על פני שאר האפליקציות. EasyShop מסייעת למלקטים שעובדים בסופר.

המלקטים אלו עובדים של הסופר שעובדים לרוב עם אופצייה של קניות באינטרנט. הם מקבלים מספר רשימות ועליהם לארוז ללקוח את מוצריו, בעצם לעשות עבור הלקוח את הקנייה. אם נחשוב על זה לרגע יוצא שאותו המלקט חוזר על אותם מקומות שביקר בהם מספר רב של פעמים אז מדוע שלא נחסוך לו את העבודה ואותו מוצר שנמצא בכמה רשימות יתאחד לו בביקור אחד ויחיד באותו המדף. ולכן האפליקציה מאחדת מספר רשימות ומאפשרת למלקטים לעבוד על כמה הזמנות בו זמנית ולחסוך להם זמן רב בעבודתם והספק יעיל.

האלגוריתם שעזר לי ביותר לצורך יישום רעיון זה הוא **Dijkstra**. אלגוריתם זה מטרתו היא מציאת מסלול קצר מנקודה לנקודה. מכיוון שניתן למצוא באמצעות אלגוריתם זה בזמן זהה, את המסלולים המהירים **לכל** הנקודות בגרף, הוא נקרא לעיתים **מציאת המסלולים הקלים מנקודה**.

Dijkstra עובד על גרף ממושקל וחיובי, הצמדת משקל לקשתות בגרף מאפשרת למדל בעיות מעניינות רבות כגון מציאת המרחק הקצר בגרף בין שני צמתים ומציאת כל המרחקים הקצרים ביותר בגרף. ולכן מפת הסופר תהווה בעצם גרף בצורה שבה יהיו צמתים וקשתות. קשתות אלו הם המרחקים מנקודה לנקודה. משקל הקשתות יהיה תמיד חיובי משום שמדובר במפה של סופר. ולכן האלגוריתם שהוזכר לעיל יכול לעבוד על גרף כזה ולסייע בחישוב במסלול הקצר.

לאפליקציה קוראים EazyShop. השם שלה מבטא את המטרה שלה, פשוט לסייע למשתמש בקניה קלה ומהירה.

2.2 תהליך המחקר

לאחר הבחירה בנושא חשבתי על איזה סופר לעשות את הפרויקט. הבחירה נבעה ממספר שיקולים ראשית העדפתי לעבוד על סופר שאני מכירה ויודעת היכן נמצאים המוצרים. כמו כן בסופר שיסיעה לי בהשגת מפות וכדומה ולכן בחרתי בסופר בירושלים היכן שאני נוהגת לעשות קניות מידי שבוע כמו כן חברה שלי היא חלק מצוות הניהול של הסופר ולכן ביקשתי ממנה עזרה במציאת מפות קטגוריות ועוד.

לאחר בחירת הסופר ומציאת המפה קראתי חומרים על אנשים שפיתחו דברים דומים וכך בניתי תבנית לאיך שאני רוצה שהפרויקט יעבוד ומה הוא יכול.

לאחר מכן חיפשתי אלגוריתמים על חיפוש מסלול מקוצר וקראתי על מגוון הסוגים וההבדלים ביניהם והבנתי שאני צריכה אלגוריתם שיעבוד רק על משקל קשתות חיובי. כמו כן בתחילה הגדרתי את הבעיה כבעיית הסוכן הנוסע שעושה את אותה הפעולה של Dijkstra. אך הסיבוכיות זמן ריצה שלו היא אין סופית ולכן בחרתי בDijkstra.

2.3 סקירת ספרות

על מנת למצוא חומרים שונים הנצרכים לי בפרויקט נעזרתי רבות במנוע החיפוש של google. אך בין האתרים שנמצאו שימושיים עבורי ביותר לכתיבת הפרויקט הן ברקע התאורטי של הנושא והן בנושאים המקצועיים והתכנותיים:

ברקע תיאורטי :

- <https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction>

בכתיבת האלגוריתם:

- <https://stackoverflow.com/>

- <https://www.w3schools.com/>

- <https://angular.io/>

2. מטרות ויעדים

לצורך עבודה רצופה על הפרויקט הצבתי לעצמי מטרות ויעדים שאני רוצה להגיע אליהם לצד מטרות אישיות שרציתי לקדם:

המטרות:

- לחסוך לצרכנים זמן יקר
- לחסוך להם כסף בכך שייצמדו לרשימה ברורה ומתכוננת מראש
- לסייע למלקטים בעבודתם
- משיכת לקוחות לסופר על ידי שימוש בטכנולוגיה חדשנית ומתקדמת
- ליצור ממשק נוח וקל ולמשתמש.

מטרות אישיות:

- רכישת ידע והכרה נרחבת בשימוש בטכנולוגית angular.
- לימוד נוסף בנושא האלתגוריתמים ויישומם בפרויקט שלי.
- לימוד נוסף והתנסות נוספת ב-c#.
- הכרת טעויות ואופן הטיפול בהם.
- התמודדות עם למידה עצמית והגשת פרוייקט עצמאי.

היעדים:

- המערכת תבנה מפה על פי רשימת הלקוח.
 - המערכת תאחד מספר רשימות לרשימה אחת ולאחר מכן תמיר אותה למפה.
- מטרת העל שלי הייתה להגיע לתוצאה סופית מוכנה וליצור אפליקציה שאכן יכולה לצאת לשיווק.

3. אתגרים

כתיבת הפרויקט העמידה בפני אתגרים רבים שסייעו לי רבות בלמידה על כיווני חשיבה שונים והתמודדות עם אתגרים מגוונים שלא הכרתי קודם.

כגון: יצירת המפה בעצמה.

לצורך יצירת המפה היה עלי להפוך את מפת הסופר לגרף ולצורך כך להחליט מיהם הצמתיים וכיצד אני מחשבת את הקשתות. ולמדתי שצומת חייב להיות דבר שלא חוזר על עצמו בשום מקום נוסף ולכן שללתי מספר אופציות והחלטתי שהצמתיים יהיו העמודות של הסופר והקשתות יהוו את המרחק בין עמודה לעמודה. אבל לאחר העמקה נוספת הבנתי כי אין עניין בכך שהנקודות יהיו כל כך צפופות עד לרמה שאני אצטרך שני נקודות עצירה בשני מוצרים שהמרחק בניהם הוא פחות מצעד אחד

ונסביר על פי דוגמא:

אם המשתמש הכניס שני מוצרים שמונחים בעמודות סמוכות אני לא יצטרך ליצור לכך שני נקודות נפרדות אלא נקודה אחת מרכזית לשניהם. ולכן החלטתי שיהיו לי בערך 3 נקודות עצירה בטור שהם: כניסה לטור, אמצע טור ויציאה מהטור. וכך אם למשתמש יהיו שני מוצרים סמוכים הוא יופנה לנקודה אחת מרוכזת בטור.

אתגר נוסף שניתקלתי בו היה הצעת הפרויקט. ההצעה שלי לא קיבלה אישור בהתחלה משום שהנושא היה קיים במקומות רבים והייתי צריכה לחשוב על כיוון שונה וחדש ולכן הוספתי את המלקטים מה שהוסיף צבע ואתגר נוסף.

בנוסף בתחילת הפרוייקט רציתי ממש להפוך את מפת הסופר לinput שמשתנה הכוונה היא שכל מפה שהייתי מכניסה למערכת הייתה פועלת בצורה טובה ומחשבת מסלול קצר. אך נתקלתי בקשיים רבים עם עניין זה והחלטתי לעבוד על מפה של מקום אחד ולכן בניית Data Basen שלי לקחה זמן רב כי מיספרתי את כל עמודות הסופר הטורים והצמתיים.

אתגר נוסף היה ההיתקלות בשגיאות באופן גורף אך גם מתוך קושי זה למדתי להכיר טעויות רבות ואת צורת הפתרון שלהן.

4. מדדי הצלחה

- ✓ הכנסת פירטי משתמש בצורה נכונה תוך שמירת מיקומם הנכון בDB.
- ✓ הצגת מסלול ללקוח התואם לרשימת המוצרים שהכניס והקצר ביותר.
- ✓ איחוד רשימות המלקט לרשימה אחת והפעלת על פונקציה זו בצורה נכונה.
- ✓ ממשק נוח ויפה למשתמש תוך שימוש בcanvas ליצירת המסלול.

בנוסף היה לי מדד הצלחה אישי חששתי נורא מפרויקט זה ובהתחלה לא האמנתי שאצליח להגיע למוצר מוגמר וברוך השם הצלחתי ולכן אמונה ביכולותי היתה מטרה בפני עצמה.

5. תיאור המצב הקיים

כיום המצב הוא שהאפליקציה קיימת בשוק במגוון מקומות בחלקם שימושית יותר ובחלק פחות.

לאחר בירור אצל אנשים שונים הבנתי מה היתרונות של כל אפליקציה ומה החסרונות ולכן בפרויקט שלי יש סיוע גם למלקט שזה חידוש על פני האפליקציות האחרות שיש סיוע גם לעובד ולא רק ללקוח במיוחד שבשנים האחרונות מתפתח מאוד העניין של קניות אונליין ועבודת המלקטים תופסת נפח רב בשוק העבודה. ולכן EazyShop תוכל לסייע רבות לאנשים העוסקים בתחום זה.

6. רקע תאורטי

עבור חישוב מסלול לאיסוף המוצרים שנקנו יש צורך להשתמש באלגוריתם למציאת מסלולים קצרים.

בבחירת האלגוריתם המתאים יש לשים לב לכמה דברים חשובים:

- האלגוריתם אמור בסופו של דבר למצוא מסלול שיאסוף את כל המוצרים.
- האלגוריתם צריך לחשב מסלול שיהיה קצר כמה שניתן.
- סיבוכיות זמן הריצה של האלגוריתם יהיה בעל סיבוכיות זמן שאינה גדולה.

7. ניתוח חלופות מערכת

היו אפשרויות שונות ליישום רעיון זה:

- אלגוריתם בלמן-פורד

חסרונות: סיבוכיות הזמן של אלגוריתם בלמן-פורד גדולה $E \cdot V(O)$

כמו כן עובד גם עם משקולות שליליים- מה שאצלנו יצור תוצאות שאינן נכונות.

מכיוון שהאלגוריתם שלנו אמור לעבור על מפה, ובמפה כל המשקלות של הקשתות (המרחק ממוצר למוצר) הינם חיוביים בלבד.

- אלגוריתם DAG

יתרונותיו: סיבוכיות הזמן שלו יעילה יותר משאר האלגוריתמים $E+V(O)$.

חסרונותיו: אלגוריתם DAG עובד רק על גרף ממושקל ומכוון ללא מעגלים, ובגרף שלנו אי אפשר לדעת מראש האם יהיו מעגלים בגרף שיווצר ע"י נתוני המשתמש. המשתמש עלול לבחור מוצרים שאכן יצרו לנו מעגל בסופר ועלול להיצטרך ללכת בעיגול ולכן אלגוריתם זה אינו תואם לפרויקט שלי.

- מטריצה

פתרון נוסף שנפסל הוא ייצוג מפת הסופר 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).

8. תיאור החלופה הנבחרת והנימוקים לבחירה.

האפשרות הנבחרת: חקירת הנושא הביאה אותי 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.

האלגוריתם **Dijkstra** כפי שהוסבר לעיל הרעיון המרכזי של אלגוריתם זה הינו מציאת מסלול קצר ביותר **ממקור יחיד** עבור גרף מכוון.

לצורך מימוש האלגוריתם בניתי מטריצת מרחקים מהחנות. ועכשיו השאלה המתבקשת הינה כיצד מחשבין מרחק בין צומת לצומת(בין מוצר למוצר)? הפתרון הפשוט הוא למצוא מרחק מנקודה לנקודה סמוכה ע"י נוסחת מרחק פשוטה.

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

לצורך הפתרון בניתי Date Based טבלה של צמתים שכנים – Neighboring Nodes. טבלה זו הינה טבלה בקשר של רבים לרבים ושני השדות בה הינם מפתחות ראשיים, תפקידה היא לשמור נתונים לכל צומת איזה צמתים שכנים יש לה. ונוצרה על מנת לממש את האלגוריתם וכיצד?

ברגע שהלקוח יכניס רשימת מוצרים לבחירתו אנחנו נוציא מכל מוצר את מספר העמודה בו הוא נמצא לאחר מכן נפנה לטבלת עמודות ומשם נוציא האם מוצר זה נמצא בתחילת העמודה או בסופה (הכול מתבסס על מידע שהוכנס לDB). משם נפנה לטבלת טורים ונוציא את קוד הטור על פי ההתחלה או הסוף בתיאום. ורק לאחר מכן נוכל להגיע אל מספר הצומת. מספר צומת זה יהווה את הצומת הקרובה ביותר למוצר שנבחר.

לאחר שמצאנו את מספר הצומת נחשב מרחק מהצומת הנוכחית לכל הצמתים השכנים לה בעזרת הטבלה של צמתים שכנים שצויינה לעיל.

את המרחקים נשמור בList ונכניס אותם למטריצה שתשמור לי מה המרחק בין כל צומת לצמתים השכנים לה.

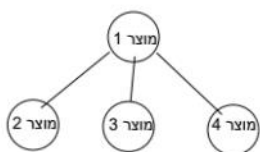
את המטריצה נשלח ל**Dijkstra**.

Dijkstra קיימת לנו בעיה שהאלגוריתם לא מתחייב לאסוף את כל המוצרים מתוך הרשימה שקיבל (הכוונה לעבור בכל הקודקודים) וזה יתואר באיור 1.

ולכן הפתרון שמוצע הוא עץ פורש מינימלי. אלגוריתם זה מוצא פרישה של כל הקודקודים בגרף כאשר הקשתות שמחברות ביניהן הן קשתות בעלות משקל מינימלי וללא מעגלים.

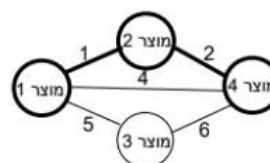
במקרה שלנו- מוצא דרך מכל מוצר שנאסף אל כל מוצר בצורה הקצרה ביותר בלי לעבור במוצר פעמיים. אך שוב, לאחר חקירת האלגוריתם ובדיקה מדוקדקת, ראיתי שהאלגוריתם אומנם נותן מסלול בין מוצר אחד לשני, אך לא נותן מסלול אחד, בין כל במוצרים. (איור 2)

איור 2: עץ פורש מינימלי



במקרה זה רואים שהאלגוריתם אומנם מחזיר פרישה מינימלית של הגרף, אולם כפי שאפשר להיווכח, ישנו מסלול מכל מוצר אל כל מוצר, אך אין מסלול יחיד בין כל המוצרים יחד.

איור 1: Dijkstra



במקרה זה רואים שהאלגוריתם אומנם יחזיר מסלול קצר אך לא כולל את כל המוצרים. דבר שלא נותן מענה למטרה.

לאחר שהבנתי שהאלגוריתמים אשר הוזכרו לעיל אינם נותנים את המענה המבוקש למציאת מסלול קצר בין כל המוצרים, גיליתי שבעיה זו היא בעיה מוכרת בקרב המתכנתים והיא נקראת "בעיית הסוכן הנוסע". זוהי בעיה שכבר עשרות שנים מנסים למצוא לה פתרון יעיל אך לא עלה בידיהם של העוסקים בדבר לפתור אותה אלה רק בצורה נאיבית בה מחשבים את כל המסלולים האפשריים בעבור כל מוצר וכך בוחרים את המסלול הקצר מבין כולם. סיבוכיות זמן הריצה של השימוש הנאיבי הזה גדולה מאוד $O(n!)$. אולם הם הצליחו למצוא קרובים לבעיה אשר אינם נותנים בהכרח פתרון אופטימלי לבעיה, אך נותנים פתרון טוב מספיק בזמן ריצה קצר בהרבה מהפתרון האופטימלי אך הלא יעיל...

קרוב אחד הוא שימוש באלגוריתם Dijkstra חמדני.

בתמצית ניתן לסכם את פעולת האלגוריתם כך:

עבור כל קודקוד מסומן האם ביקרו בו או לא ומה מרחקו מקודקוד המקור אותו נסמן ב-A.

הלולאה עובדת כך:

כל עוד נותרו קודקודים שלא ביקרנו בהם:

- מסמנים את X בתור הקודקוד הנוכחי (באיטרציה הראשונה זהו קודקוד המקור A).

עבור כל קודקוד y שהוא שכן של x ועדיין לא ביקרנו בו:

- Y מעודכן, כך שמרחקו יהיה שווה לערך המינימלי בין שני ערכים:

בין מרחקו הנוכחי, לבין משקל הקשת המחברת בין x לבין y ועוד המרחק בין A ל-x.

- בוחרים קודקוד x חדש בתור הקודקוד שמרחקו בשלב הזה מצומת המקור A הוא הקצר ביותר מבין כל הקודקודים בגרף שטרם ביקרנו בהם.

האלגוריתם מסתיים כאשר קודקוד x החדש הוא היעד או כאשר ביקרנו בכל הקודקודים.

דייקסטרה על כלל הצמתים יוצר לי לכל צומת:

מהי צומת המקור שלו (מאיפה יצאה הקשת), ומה המרחק מהמקור לצומת הנתונה.

ולכן יופעל דייקסטרה על כל זוג צמתים בנפרד לדוגמא:

דייקסטרה B < A יחזיר שצומת המקור היא A והמרחק הוא 3.

דייקסטרה על D < A יחזיר שצומת

דייקסטרה על F < A וכן הלאה.

לאחר מעבר על כל הצמתים שמחוברות ל-A הדייקסטרה יבחר את $D < A$ כקשת הקצרה ביותר. ולאחר מעבר על הגרף כולו יתקבל מהפונקציה המסלול הקצר ביותר והאפשרי מבין הצמתים הנתונים.

סיבוכיות זמן הריצה של אלגוריתם זה הינה $O(n^2)(E+V)$ זוהי סיבוכיות גדולה יחסית אך בהתחשב בכך שהאלגוריתם השני שעושה את הפעולה הזו הינו הסוכן הנוסע וסיבוכיות הזמן שלו היא אין עצרת $O(n!)$ ולכן וודאי שהסיבוכיות של האלגוריתם שלי נמוכה בהרבה.

- לאחר הפעלת האלגוריתם וקבלת המסלול המקוצר נבחר את קודקודי המסלול כנקודות ונציג אותם בצורה וויזואלית על המסך ללקוח כנקודות שצריך לעבור בהם לצורך מסלולו.

לצורך סדר הנקודות בהם הלקוח צריך לעבור השתמשנו באלגוריתם נוסף אלגוריתם זה נקרא **אלגוריתם הסוכן הנוסע (TSP)** אלגוריתם זה הוזכר לעיל בשל סיבוכיות זמן הריצה הבלתי יעילה שלו ולכן השתמשנו בו אך בשיפור משמעותי.

ניתן להציג את בעיית הסוכן הנוסע כגרף ממושקל לא מכוון, בו כל עיר היא צומת, דרך בין עיר לעיר היא קשת, והמרחק בין כל שתי ערים מצוין על ידי משקל הקשת המחברת ביניהן. בצורה כזו ניתן לשאול מהו המסלול עם המשקל הנמוך ביותר, שיצא ויחזור לאותו צומת לאחר שעבר בכל צומת אחר פעם אחת בדיוק. לרוב הבעיה תוצג על גבי גרף שלם (גרף בו כל זוג צמתים מחוברים בקשת). אם לא קיימת דרך בין שתי ערים ניתן להוסיף לגרף קשת עם משקל גבוה מאוד ובכך להשלים את הגרף לגרף שלם מבלי לפגוע בפתרון האופטימלי.

פתרון פשוט לבעיה הוא בדיקת כל התמורות האפשריות וחיפוש המסלול הקצר ביותר. פתרון זה מצריך בדיקה של $n!$ אפשרויות (עצרת של מספר הערים), ולכן סיבוכיות זמן הריצה במקרה כזה תהיה $O(n!)$. חישוב כזה הופך מהר מאוד לבלתי מעשי (לבלתי יעיל, במינוח של מדעי המחשב). דוגמה להמחשת הבעיה: אם ישנן 70 ערים בהן על הסוכן לבקר, יצטרך הסוכן לבחון $70!$ מסלולי נסיעה אפשריים, אשר מהווים מספר רב יותר ממספר האטומים ביקום כולו. ולכן אלגוריתם זה גם לא בנחר לפרויקט עקב זמן הריצה הביילתי יעיל שלו. לצורך שימוש באלגוריתם זה ניסתי לצמצם את זמן הריצה שלו ולצורך כך בניתי רעיון שהמוצרים 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. וההסבר הוא כך: ב data base בנינו את המפה בצורה שונה שבה כשלוקח מעוניין לאסוף מספר מוצרים מאותו אזור הוא יופנה לצומת מרכזית בהתחלת הטיור באמצע הטיור ובסוף הטיור וכל יצטמצם זמן הריצה של אלגוריתם זה שבמקום שהסיבוכיות תהיה על פי מספר המוצרים של הלקוח היא תהיה על פי האזורים שבהם המוצרים ממוקמים.

9. אפיון המערכת

סביבת פיתוח :

חומרה: מעבד i7 RAM 4GB.

עמדת פיתוח: מחשב hp.

מערכת הפעלה: Windows 10.

שפת תוכנה: c# תוך שימוש בטכנולוגיות Angular, WebApi.

כלי תוכנה לפיתוח המערכת: Microsoft visual studio 2019, vs code.

מסד נתונים: SqlServer.

עמדת משתמש מינימלית:

חומרה: מעבד i3 RAM 4GB.

- מערכת הפעלה: windows 10 ומעלה.
- חיבור לרשת: נדרש.
- תוכנות: chrome.

9.1. ניתוח דרישות המערכת.

דרישות בהן המערכת צריכה לעמוד:

- כתיבה בסטנדרטים מקצועיים.
- מחשוב השרות ללקוח.
- כתיבת הקוד בסיבוכיות היעילה ביותר.
- ממשק נוח וידידותי למשתמש.
- תגובה מהירה ככל שניתן למשתמש.

9.2. מודול המערכת.

- העלאת רשימת מוצרים של משתמש.
- הוצאה של מספר עמודה מכל מוצר מהדאטה בייס.
- בדיקה בטבלת עמודות האם המוצר בתחילת עמודה או בסוף.
- על פי העמודה לגשת לתחילת הטור המקושר לעמודה או לסופו.
- ומשם נעבור למספר הצומת בטבלת הצמתים.
- לאחר מכן בדיקת הצמתים השכנים ומציאת המרחק לצומת הבא הקצר ביותר.
- חיבור של כל הצמתים ביחד והצגת מפה למשתמש.

9.3. אפיון פונקציונאלי

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.

פירוט פונקציות עיקריות ותפקידן:

`Public static List<Route> CreateRoute(List<Nodes> ln)`

הפונקציה מחשבת את אורך הקשתות בין צומת לצומת באופן כזה: הפונקציה מקבלת רשימה של צמתים ועוברת ב `foreach` על כל צומת בנפרד.

לכל צומת היא בודקת את כל שכניו ומחשבת מרחק ביניהם על ידי משוואת `distance`. את התוצאה היא שומרת בתוך רשימה של קשתות שתכיל את כל המרחקים מצומת נתונה לצמתים השכנים שלו.

`Public static List<Nodes> ConvertProductToNodes(List<Products> lp)` -

הפונקציה ממירה רשימת מוצרים לרשימת צמתים באופן הבא:

הפונקציה מקבלת רשימת מוצרים ובודקת לכל מוצר מה העמודה שלו ומתוך טבלת עמודות האם העמודה עומדת בתחילה או בסוף טור. על פי המידע הזה אני פונה לתחילת הטור ומשם לצומת שמציינת את תחילת הטור הנוכחי. כך לכל המוצרים עד שנוצרת לי רשימת צמתים על פי המוצרים שהתקבלו.

`public static Cell[] Dijkstra(Graph graph, int src)`

פונקציה זו אחראית על מימוש האלגוריתם **דייקסטרה** כמובן בעזרת פונקציות נוספות שיפורטו בהמשך בהרחבה אך פונקציה זו מקבלת גרף וקודקוד ומחשבת את המסלול הקצר אליו ומחזירה מטריצה. מטריצה זו תפקידה לשמור את המרחק בין כל צומת לצומת ולכן נשלח בהתחלה לפונקציה זו את כל מפת הסופר עם קודקוד הכניסה וכך יחושבו לנו כל המרחקים בין כל הצמתים ולאחר מכן נשלח לה רק את הצמתים שהלקוח מעוניין לבקר בהם.

9.4. ביצועים עיקריים

- הרשמה/ התחברות לאפליקציה.
- חישוב מסלול קצר לאיסוף המוצרים.
- חיבור מספר רשימותיו של המלקט לרשימה אחת ויצירת מסלול.

9.5. אילוצים

- המערכת יכולה לפעול רק על סופר אחד. (אושר עד שמגר)
- המערכת חייבת עדכון במידה ומשנים מיקום של מוצרים בסופר.

10. תיאור הארכיטקטורה

10.1. הארכיטקטורה של הפתרון המוצע בפורמט

של Design level Down-Top

הפרויקט מחולק לשכבות:

הפרויקט מחולק ל 2 חלקים:

- צד שרת: הנכתב בשפת c#. בטכנולוגיית WebApi.
- צד לקוח: הנכתב בשפת Angular ובטכנולוגיית Script Type, Html.

תיאור צד השרת :

צד השרת מחולק כמקובל לשכבות :

שכבת ה - DAL

שכבת ה - BL

החלוקה לשכבות נועדה להפריד באופן מוחלט בין הלוגיקה של הפרויקט לבין הנתונים עצמם.

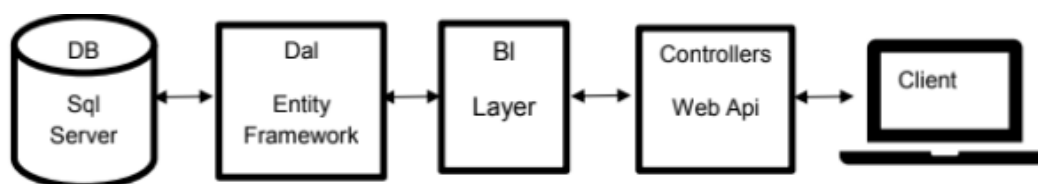
הפרדה זו מאפשרת לבצע שינויים בכל אחת מהשכבות בלי תלות ובלי זעזועים בשכבות האחרות.

פירוט:

שכבת ה DAL: היא השכבה דרכה ניגשים לנתונים היושבים ב DB היא מכילה מחלקות המייצגות את בסיס הנתונים וכן פונקציות נחוצות לתפעולו . פעולות ההתקשרות עם בסיס הנתונים נעשו בטכנולוגיית Entity framework. טכנולוגיה זו היא המקובלת לטיפול בבסיס הנתונים כאשר היא מנתחת את בסיס הנתונים, בונה מחלקות לייצוג הטבלאות ומספקת רשימות מלאות בנתוני בסיס הנתונים.

שכבת ה BL : שכבה שבה כתובה כל הלוגיקה של הפרויקט.

שכבת ה-Models : שכבה זו מכילה מחלקות המתארות את הנתונים ובמבנה זה מעבירים את הנתונים בין השכבות. מטרת שכבה זו היא למנוע תלות של שכבת ה-BL במבנה בסיס הנתונים .



שכבת ה-BL מכילה פונקציות המרה מטיפוס הנתונים של בסיס הנתונים לטיפוס הנתונים של שכבת ה-Models ולהיפך, וכך מיוצגים הנתונים בכל הפרויקט.

בנוסף קימות מתודות השרת המחצינות את הפעולות שניתן לבצע בשרת. מתודות אלו משתמשות ב-BL ומופעלות ע"י שכבת ה-GUI בצד הלקוח. מודל זה אמנם גורם טרחה טכנית לא מעטה בכתיבת הקוד ובתכנון הפונקציות אבל מספק קוד נקי, קל להבנה ונוח לשינויים ולשדרוגים.

10.2. תיאור הרכיבים בפתרון

1. מסד הנתונים הבנוי מטבלאות וקשרי גומלין ביניהן.
2. שכבת הגישה לנתונים באמצעות Framework Entity.
4. שכבת ה-bl בה כתובים האלגוריתם.
5. Api Web-פרוטוקול התקשורת בין צד הלקוח וצד השרת.
6. צד לקוח: angular, TypeScript

תיאור פרוטוקולי התקשורת

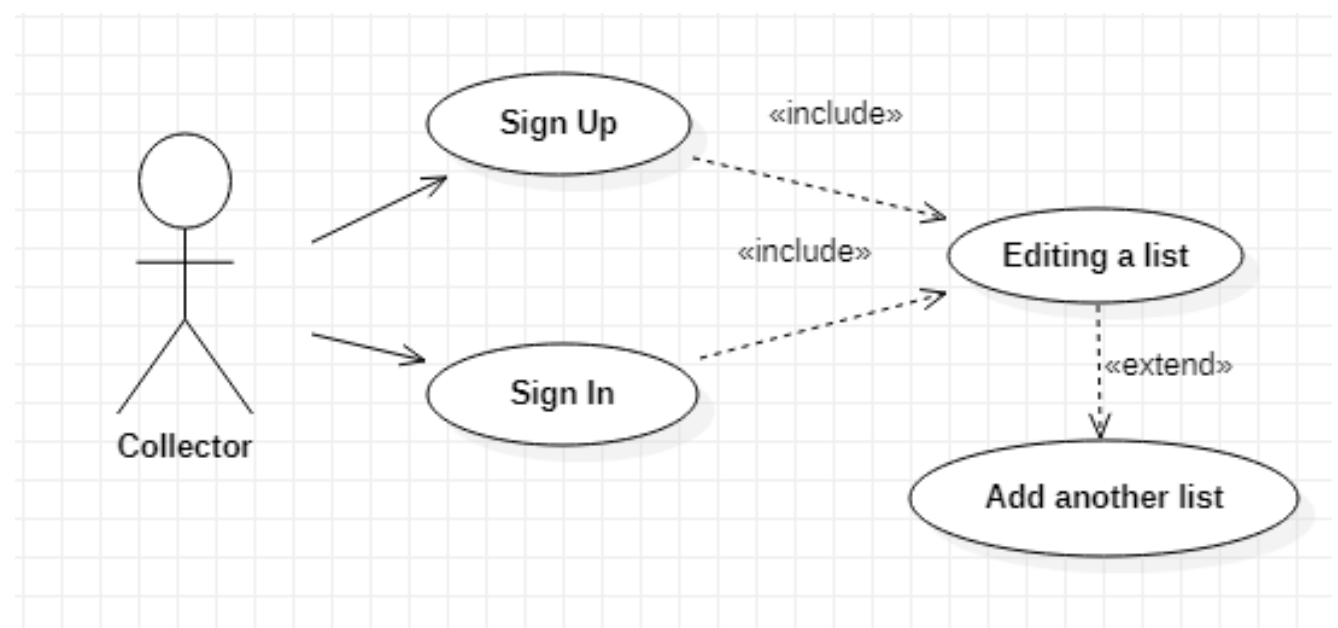
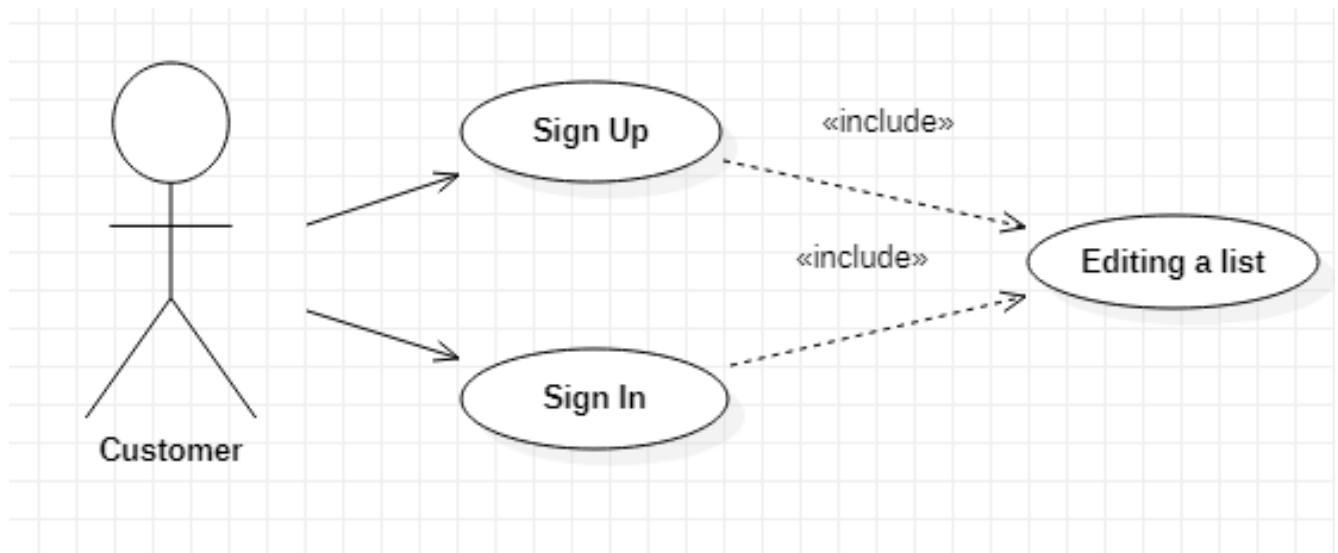
http – הוא פרוטוקול תקשורת שנועד להעברת דפי HTML ואובייקטים ברשת האינטרנט.

שרת – לקוח

צד השרת נכתב בטכנולוגיית WebApi ובשפת c#.

צד הלקוח נכתב בשפת Angular בטכנולוגיית CSS, typescript, Html

11. ניתוח ותרשים use case של המערכת המוצעת.



רשימת use case

- הרשמות/ התחברות לאפליקציה.
- צפייה במוצרים שבמערכת.
- בחירת מוצרים והכנסתם לעגלת קניות.
- מסלול לאיסוף המוצרים.

תיאור ה- use case העיקריים של המערכת

UC1

- **UC1 :Identifier**
- Sign in: **Name**
- **Description**: משתמש חדש מכניס פרטים למערכת.
- **Actors**: Customer
- **Frequency**: בכל פעם שמשתמש חדש נכנס.
- **Pre-Conditions**: פרטים בסיסיים.
- **Post-Conditions**: פרטי המשתמש נשמרים במערכת.
- **action of course Basic**: המשתמש מופנה לרשימת מוצרים.
- **action of course Alternate**: משתמש שלא התחבר למערכת יופנה למסך הרשמה.

UC2

- **UC2 :Identifier**
- Sign Up: **Name**
- **Description**: משתמש קיים מאמת את פרטיו במערכת.
- **Actors**: Collector , Customer
- **Frequency**: בכל פעם שמשתמש קיים נכנס.
- **Pre-Conditions**: פרטים בסיסיים.
- **Post-Conditions**: פרטי המשתמש מתאמתים במערכת.
- **action of course Basic**: המשתמש מופנה לרשימת מוצרים.
- **action of course Alternate**: משתמש שלא התחבר למערכת יופנה למסך הרשמה.

UC3

- **UC3 :Identifier**
- Editing a list: **Name**
- **Description**: המשתמש עורך רשימת קניות על ידי בחירת מוצרים והוספת לרשימה. בחירת המוצרים תתבצע על פי קטגוריה.
- **Actors**: Collector, Customer
- **Frequency**: בכל פעם שמשתמש יכנס למערכת.
- **Pre-Conditions**: הרשמה/התחברות.
- **Basic action of course**: המלקט יוכל להוסיף רשימה נוספת.

מבני נתונים בהם משתמשים בפרויקט

בפרויקט שלי היה שימוש במבני נתונים של:

List - הרשימה מאפשרת שמירה של נתונים בצורה סדרתית בלי חובה להגדיר את הגודל. ישנן גם פעולות רבות שמאד קל להפעיל אותם על רשימה בשפת C#.

השתמשי במבנה נתונים זה במספר מקומות לדוגמא בפונקציה שהופכת בין מוצרים לצמתים פונקציה זו מקבלת רשימת מוצרים.

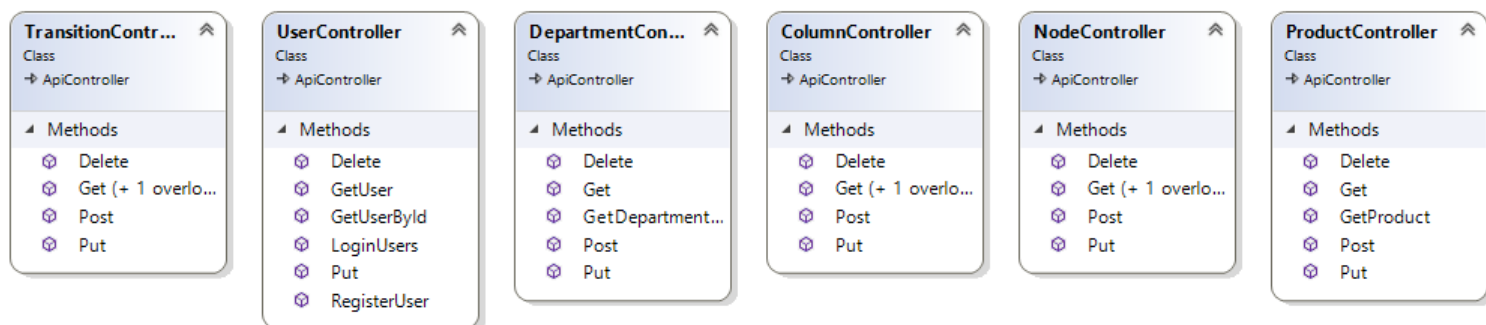
כמן כן באלגוריתם הדייקסטרה ישנו שימוש:

Queue על מנת לדעת על איזה קודקודים נותר לעבור.

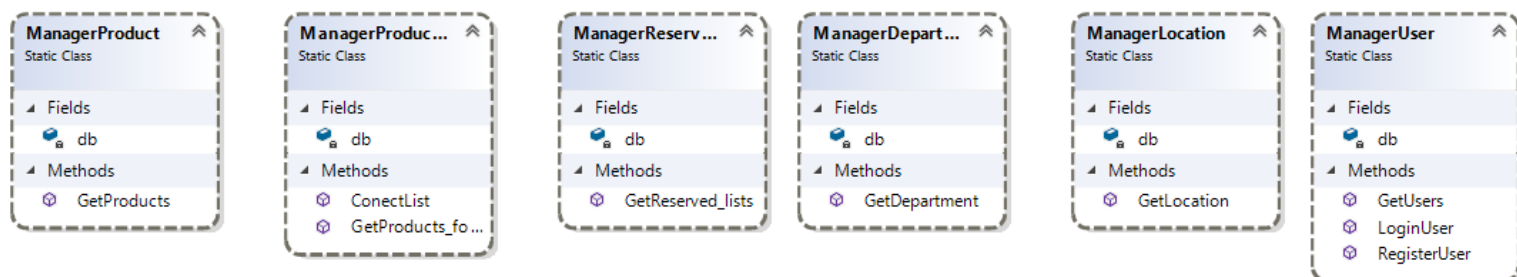
11.1. תרשים מחלקות

צילום של פירוט המחלקות:

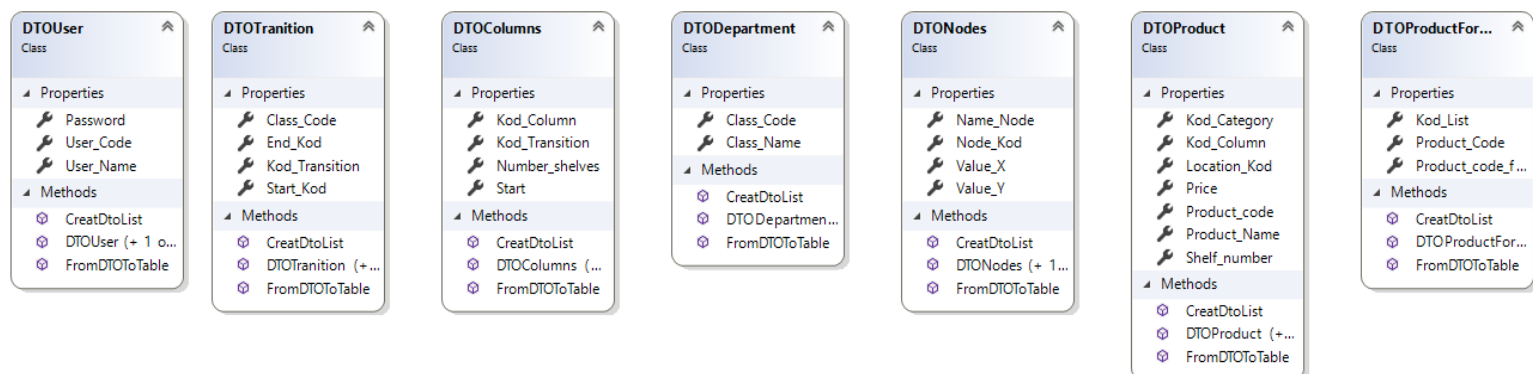
שכבת ה-API:



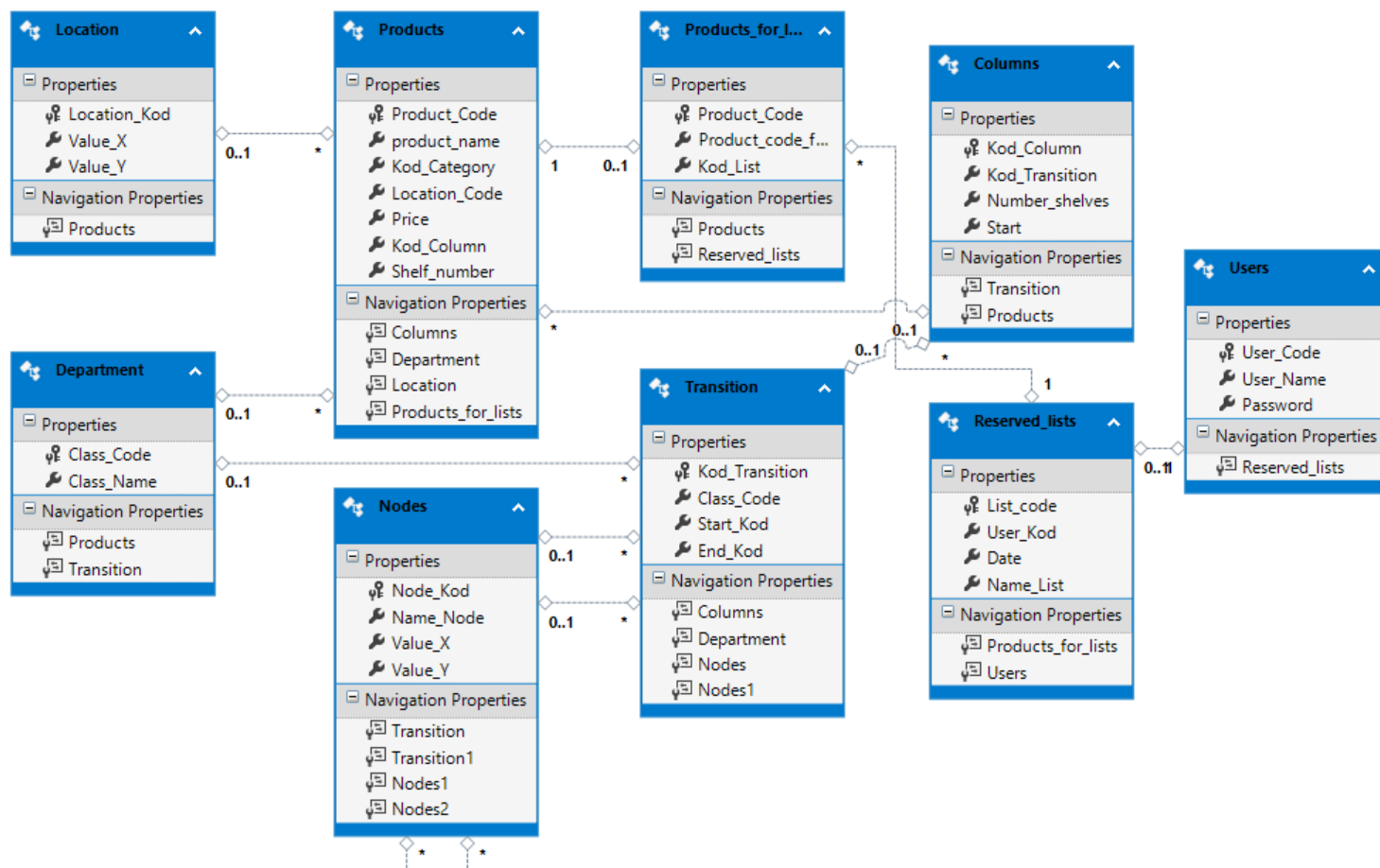
שכבת ה-BL:



שכבת ה-DTO:



שכבת ה-DAL:



תיאור המחלקות

- **שכבת API**
ממשקי API הם מגננים המאפשרים לשני רכיבי תוכנה לתקשר זה עם זה באמצעות סט של הגדרות ופרוטוקולים.
- **שכבת הלוגיקה העסקית (BL-Business Logic)**
השכבה שאמונה על הלוגיקה של המערכת. עוסקת בעיבוד המידע, בחישובים שונים ושליחתו לשכבת התצוגה. בשכבה זו נממש את הפונקציונליות של המערכת.
- **שכבת DAL**
שכבה זו מורכבת ממקור נתונים (Data Source). מקור הנתונים שלי היה SQL.
- **שכבת התצוגה**
שכבה זו מהווה את צד הלקוח השכבה הגבוהה ביותר משום שאחראית על הוויזואליות של המערכת ויישום של האלגוריתמיקה בצורה נוחה ונכונה למשתמש. בנוסף עיצוב נכון ומושך של מערכת יעניין לקוחות גם אם התוכן אינו נצרך עבורם.

12. תיאור התוכנה

סביבת עבודה:

Visual Studio Code, Microsoft Visual Studio

שפות תכנות:

צד השרת נכתב ב#c# בטכנולוגית WebApi.

צד הלקוח נכתב בשפות:

. Angular ,typescript ,css ,Html- בטכנולוגית

13. אלגוריתמים מרכזיים

- מספור העמודות והטורים בסופר.
- הפיכת מפת הסופר לקווים גאוגרפים .
- קביעת צמתים.
- קישור בין מוצר לצומת.
- חישוב מרחק בין צמת לצומת.
- הפיכת הנתונים לגרף ושליחה לדיוקסטרה.
- שינוי הקוד והתאמתו לפרויקט שלי.
- יצירת צד לקוח יפה ונוח.

14. קוד האלגוריתם האלגוריתם דייקסטרה:

להלן המחלקות הנדרשות, צומת, קשת, גרף:

Graph - גרף:

```
public class Graph
{
    public int v;
    public AdjList[] array;
    0 references | tahell, 3 days ago | 1 author, 1 change
    public Graph() { }
    // A utility function that creates a graph of V vertices
}
```

Route - קשת:

```
public class Route
{
    2 references | tahell, 18 days ago | 1 author, 1 change
    public Nodes source { get; set; }
    2 references | tahell, 18 days ago | 1 author, 1 change
    public Nodes destination { get; set; }
    2 references | tahell, 18 days ago | 1 author, 1 change
    public double value { get; set; }

    1 reference | tahell, 18 days ago | 1 author, 1 change
    public Route(Nodes source, Nodes destination, double value)
    {
        this.source = source;
        this.destination = destination;
        this.value = value;
    }
}
```

Node - צומת:

```
11 references | tahell, 25 days ago | 1 author, 1 change
public int Node_Kod { get; set; }
1 reference | tahell, 25 days ago | 1 author, 1 change
public string Name_Node { get; set; }
3 references | tahell, 18 days ago | 1 author, 3 changes
public Nullable<int> Value_X { get; set; }
3 references | tahell, 18 days ago | 1 author, 3 changes
public Nullable<int> Value_Y { get; set; }
```

ConvertProductToNodes - פונקציה זו ממירה מרשימת צמתים לרשימת מוצרים.

```
public static List_productNode ConvertProductToNodes(List<Products> lp)
{
    List<Products> productsList = new List<Products>();
    List<Nodes> nodesList = new List<Nodes>();
    List_productNode List_productNode = new List_productNode(); ;
    foreach (var p in productsList)
    {
        Nodes node;
        if (p.Columns.Start == 1) //1 - התחלה
        {
            node = p.Columns.Transition.Nodes;
        }
        else
        {
            node = p.Columns.Transition.Nodes1;
        }
        nodesList.Add(node);
        List_productNode.Add(p, node);
    }

    return List_productNode;
}
```

ComputeDijkstra - זוהי הפונקציה שממלאה את המטריצה בנתונים על פי רשימת צמתים ורשימת מרחקים שמקבלת, ומחזירה מטריצה מלאה בנתונים.

1 reference | tahell, 3 days ago | 1 author, 1 change

```
public static Cell[,] ComputeDijkstra(List<Nodes> nodes, List<Route> connections)
{
    int numNodes = nodes.Count() ;
    List_iCode list_ICode = new List_iCode(nodes);

    Graph graph = computeGraph(nodes, connections, list_ICode);
    //מטריצת המרחקים שתוחזר
    Cell[,] mat = new Cell[numNodes, numNodes];
    //משתנה עזר המקבל תוצאת דייקסטרה ומועתק למטריצה
    Cell[] arr = new Cell[numNodes];
    foreach (var item in nodes)
    {
        int i = list_ICode.getI(item.Node_Kod);
        arr = Dijkstra(graph,i);
        for (int j = 1; j < numNodes; j++)
            mat[i, j] = arr[j];
    }
    return mat;
}
```

הנתונים נשמרים במטריצה מסוג Cell. הנה התאור של מחלקה זו:

```
public class Cell
```

```
{
    1 reference | tahell, 3 days ago | 1 author, 1 change
    public int i { get; set; } /*מקור*/
    1 reference | tahell, 3 days ago | 1 author, 1 change
    public int j { get; set; } /*יעד*/
    12 references | tahell, 3 days ago | 1 author, 1 change
    public double distance { get; set; }
    2 references | tahell, 3 days ago | 1 author, 1 change
    public int Parent { get; set; } /*קור*/
}
```

CreateRoute - פונקציה זו מחשבת מרחק על פי נוסחת מרחק בין צומת לכל הצמתים השכנים לו (המידע נלקח מהdb). –

```
public static List<Route> CreateRoute(List<Nodes> ln)
{
    List<Route> rList = new List<Route>();
    using (var db = new EazyShopEntities())
    {
        ln = db.GetDbSet<Nodes>().ToList();
        foreach (Nodes node in ln)
        {
            foreach (var otherNode in node.Nodes1)
            {
                double x = Math.Sqrt(Math.Pow(Convert.ToDouble(node.Value_X - otherNode.Value_X), 2) +
                    Math.Pow(Convert.ToDouble(node.Value_Y - otherNode.Value_Y), 2));
                rList.Add(new Route(node, otherNode, x));
            }
        }
    }
    return rList;
}
```

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;
    0 references | tahell, 3 days ago | 1 author, 1 change
    public Graph() { }
    // A utility function that creates a graph of V vertices
    1 reference | tahell, 3 days ago | 1 author, 1 change
    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, double 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;
    }
}
```

```

public class AdjListNode
{
    public int dest;
    public double weight;
    public AdjListNode next;

    0 references | tahell, 3 days ago | 1 author, 1 change
    public AdjListNode()
    {
    }
    // A utility function to create a new adjacency list node
    2 references | tahell, 3 days ago | 1 author, 1 change
    public AdjListNode(int dest, double weight)
    {
        this.dest = dest;
        this.weight = weight;
        this.next = null;
    }
}

// A ure to represent an adjacency list
4 references | tahell, 3 days ago | 1 author, 1 change
public class AdjList
{
    // pointer to head node of list
    7 references | tahell, 3 days ago | 1 author, 1 change
    public AdjListNode head { get; set; }
    1 reference | tahell, 3 days ago | 1 author, 1 change
    public AdjList() { }
    0 references | tahell, 3 days ago | 1 author, 1 change
    public int SetHead() { head = null; return 1; }
}

```


פונקציית הדייקסטרה:

```
// The main function that calculates distances of shortest paths from src to all
// vertices. It is a  $O(E \log V)$  function
1 reference | tahell, 3 days ago | 1 author, 1 change
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 v
// 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;
}
}

// print the calculated shortest distances
//printArr(dist, vv);
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.

15. תיאור מסד הנתונים

פירוט הטבלאות ב- Data Base

טבלת עמודות – מטרתה היא למספר את העמודות בכל טור.

Kod_Column - זהו מספר אוטומטי שימספר את העמודות בכל הסופר.

Kod_Transition - הינו מפתח זר מטבלת טורים על מנת שנדע באיזה טור נמצאת כל עמודה.

Number_shelves - ממספר כמה מדפים יש בכל עמודה.

Start - זהו שדה שמכיל מספרים או 1 או 2 על פי מיקום העמודה. במידה והעמודה ממקוממת בתחילת התור שדה זה יקבל את הסיפורה 1. ובמידה והעמודה ממוקמת בסוף הטור שדה זה יקבל את הערך 2.

טיפוס	תיאור	שם שדה	מפתח
int	קוד עמודה	Kod_Column	Kod_Column
int	קוד טור	Kod_Transition	
int	מספר מדפים בעמודה	Number_shelves	
	התחלה או סוף	Start	

טבלת קטגוריות - בטבלה זו יוגדרו כל הקטגוריות שיש בסופר ולפיהם יחולקו המוצרים.

Class_Code - זהו מספר אוטומטי שימספר את הקטגוריות.

Class_Name - שדה זה מכיל את שם הקטגוריה.

טיפוס	תיאור	שם שדה	מפתח
int	קוד קטגוריה	Class_Code	Class_Code
nchar(10)	שם קטגוריה	Class_Name	

טבלת צמתים שכנים - טבלה זו מגדירה לי איזה צמתים שכנים לצומת המקור.

Source_Node - זהו צומת המקור שמהווה מפתח ראשי ומקושר לטבלת צמתים.

Destination_Node - שדה זה יציג לי מה הצומת הקרובה לצומת המקור.
מהווה גם כן מפתח ראשי ומקושר לטבלת צמתים.

טיפוס	תיאור	שם שדה	מפתח
int	קוד מקור	Source_Node	Source_Node
int	קוד יעד	Destination_Node	Destination_Node

טבלת צמתים-טבלה זאת ממספרת לי את הצמתים של הסופר.

Node_Kod - בשדה זה ישנו את קוד הצומת (מספור אוטומטי).

Name_Node - בשדה זה יש את שם הצומת. על מנת לכוון את המשתמש בצורה מילולית.

Value_X - לאחר חלוקה של מפת הסופר לערכים של x ו y כאן יוכנס ערך הא.

Value_Y - כאן יוכנס ערך הע.

טיפוס	תיאור	שם שדה	מפתח
int	קוד צומת	Node_Kod	Node_Kod
nchar(10)	שם צומת	Name_Node	
int	ערך X	Value_X	
int	ערך Y	Value_Y	

טבלת מוצרים-בטבלה זו מאוחסנים כל הפריטים בסופר.

Product Code - קוד מוצר (מספור אוטומטי).

product name – שם מוצר.

Kod Category – קוד קטגוריה. זהו מפתח זר מטבלת קטגוריות. לכל מוצר משויכת הקטגוריה.

Price- מחיר של כל מוצר.

Shelf number- מספר מדף. באיזה מדף נמצא המוצר.

טיפוס	תיאור	שם שדה	מפתח
int	קוד מוצר	Product_Code	Product_Code
nvarchar(50)	שם מוצר	product_name	
int	קוד קטגוריה	Kod_Category	
float	מחיר מוצר	Price	
int	קוד עמודה	Kod_Column	
int	מספר מדף	Shelf_number	

טבלת טורים- בטבלה זו יצויינו הטורים שהסופר מחולק אליהם.

Kod Transition- זהו מספר הטור (מספור אוטומטי).

Class Code- זהו מפתח זר מטבלת הקטגוריות.

Start Kod- קוד זה משויך לטבלת צמתים כאשר שדה זה מכיל את צומת ההתחלה באותו הטור. הכוונה כניסה לטור הנוכחי.

End Kod- קוד זה משויך גם כן לטבלת צמתים כאשר שדה זה מכיל את צומת הסיום באותו טור הכוונה יציאה מהטור הנוכחי.

טיפוס	תיאור	שם שדה	מפתח
Int	קוד טור	Kod_Transition	Kod_Transition
Int	קוד קטגוריה	Class_Code	
Int	קוד התחלה	Start_Kod	
int	קוד סיום	End_Kod	

טבלת משתמשים- בטבלה זו ישמרו נתוני המשתמשים במערכת.

User Code- זהו מספר אוטומטי על מנת למספר את המשתמשים.

User Name - זה שם שהמשתמש יבחר בו. כך הוא יוכל להיכנס למערכת בקלות.

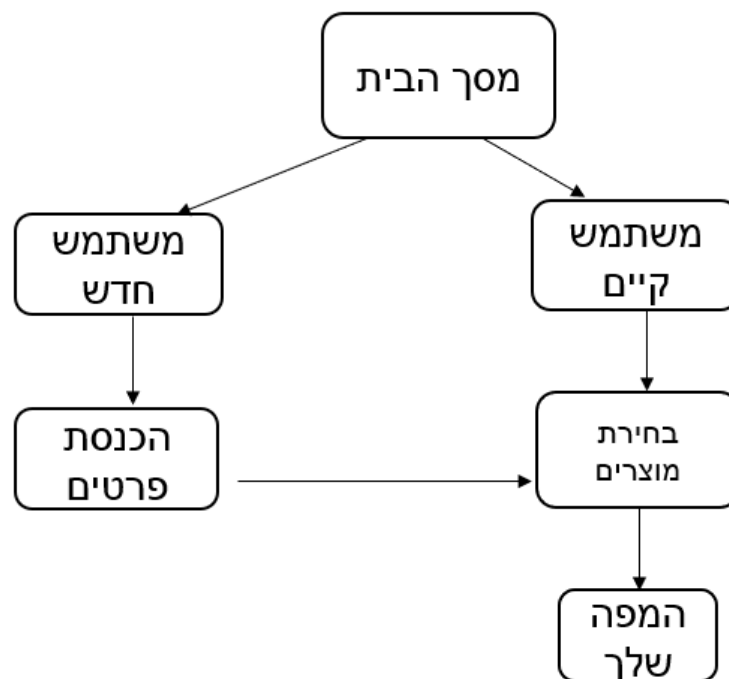
Password - סיסמא על מנת לאבטח את הכניסה.

טיפוס	תיאור	שם שדה	מפתח
int	קוד משתמש	User_Code	User_Code
varchar(50)	שם משתמש	User_Name	
varchar(50)	סיסמא	Password	

16. מדריך למשתמש

16.1. תיאור המסכים

תרשים זרימה של (כל) המסכים:



מדריך למשתמש

בכניסה לאתר יוצג למשתמש מסך הבית. במסך זה יהיה על המשתמש שתי אופציות הרשמה או התחברות.

הרשמה זהו מצב של כניסה ראשונה למערכת על המשתמש יש להכניס שם משתמש וסיסמא ולאמת את סיסמתו לאחר שלב זה המערכת תיצור עבורו קוד אישי וכך הוא ישר במערכת.

התחברות זהו מצב של משתמש שקיים במערכת ועליו רק לאמת את פרטיו על ידי הכנסת שם משתמש וסיסמא.

ההתחברות וההרשמה נועדו על מנת שהמערכת תוכל לאמוד בכל פעם את כמות המשתמשים שנצרכים למערכת זו. (במידה והסופר יחייב כל לקוח להשתמש באפליקציה זו הסופר יוכל לאמוד בכל שלב את כמות הלקוחות הנוכחית בסופר).

לאחר ההתחברות למערכת המשתמש יועבר למסך של בחירת מוצרים, בחירת המוצרים תעשה על פי קטגוריות. המסך שיוצג למשתמש יציג רק את הקטגוריות ולכל קטגוריה שיבחר תוצג לפניו רשימת המוצרים הקיימים באותה קטגוריה. בכל שלב של עריכת רשימה יוצג למשתמש אופציה לחיפוש של מוצר של כלשהו שזקוק לו ללא כניסה לקטגוריה ספציפית.

כל מוצר שהלקוח יבחר להוסיף לעגלת הקניות יהווה למערכת מקום שהלקוח צריך לעבור בו. בסיום הוספת כל הפריטים לרשימה הלקוח ילחץ על כפתור מעבר לעגלת הקניות וילחץ על אישור. לאחר לחיצה על אישור יוצג למשתמש מסך שיכיל את המפה בצורה וויזואלית וברורה באילו נקודות עליו לעבור על מנת להשלים את רשימת הקניות שלו.

במידה והמשתמש הינו מלקט פעולת ההרשמה וההתחברות תהיה זהה מלבד שעל המלקט לציין שהינו מלקט ולא לקוח רגיל. דבר זה יסייע לו להוסיף רשימה נוספת על הרשימה הקיימת. עבור המלקט בלחיצה על כתור האישור יקרו שתי פעולות. פעולה אחת שתאחד עבורו את הרשימות שהכניס ופעולה שנייה שתיצור עבורו את המפה שתסייע לו בעבודתו.

16.2. צילומי מסכים

מסך הבית:



Make your shop easier



הרשמה:

EazyShop Home Sign in Sign up About category-list Products ▼

Stacked Form

How to use CSS to create a stacked form:

First Name

Last Name

Country

Australia ▼

Submit

התחברות:

EazyShop Home Sign in Sign up About category-list Products ▼

שם משתמש

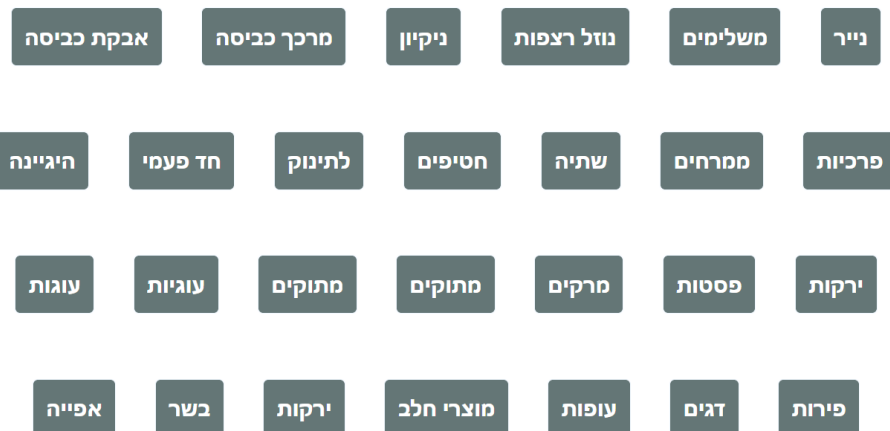
סיסמא

התחברות

קטגוריות:

Start buying easily. Visit our site to shop as quickly as possible

All you have to do is enter your shopping list and we will create the shortest route for you.

אודות:

EasyShop Home Sign in Sign up About category-list Products ▾



Welcome to EasyShop!

At our site you will be exposed to an innovative method that helps the consumer make his purchase easily and quickly.

If your work is a collector, this site will help you a lot in your work by creating for you the shortest route you need to go from your number of lists.

All you need is good luck!

17. בדיקות והערכה

לאחר הרצת האלגוריתם נבחנו כל האילוצים שדרושים כדי להביא למפה מדויקת, למסלול קצר ביותר ובזמן ריצה הנמוך ביותר. כאשר הופיעו טעויות האלגוריתם נבדק שוב ושוב עד שתוקנו כל הבעיות. כמו כן ניסתי לחשוב על כל מיקרי הקצה האפשריים בפרויקט שלי בתקווה שאלו היו רובם ככולם.

לאחר הרצת האלגוריתם מספר פעמים עם נתונים שונים הוא הגיע לקירוב האפשרי ביותר בכלים העומדים לרשותי ובפרק הזמן הנתון.

היו דברים נוספים שרציתי להוסיף בפרויקט אך עקב הזמן המצומצם לא הספקתי אני מקווה שאמשיך לפתח את פרויקט זה ולהגיע לתוצאה הרצויה במאת האחוזים.

18. ניתוח יעילות

בפרויקט ייחסתי חשיבות רבה ליעילות מתוך רצון למצוא אלגוריתם עם זמן ריצה נמוך ביותר. בעקבות כך פסלתי את אלגוריתם הסוכן הנוסע אך סיבכיות זמן הריצה שלו היא $O(n!)$ אין עצרת. ולכן עבדתי על דייקסטר תוך התאמה לפרויקט שלי כך שצימצמתי את זמן הריצה ל $O(n^2)(E+V)$ שזהו פער גדול מהאלגוריתם השני.

19. אבטחת מידע

על מנת שהמערכת תדע כמה רשומים קיימים במערכת, קיימת במערכת אבטחה בסיסית. על המשתמש להכניס שם משתמש וסיסמא על מנת להיות מחובר.

משתמש שרשום במערכת כשירצה להתחבר שנית יכניס את פרטיו והם ישלחו לפונקציה שתאמת אם פרטיו אכן נכונים במידה ואחד מהם שגוי המשתמש לא יוכל להיכנס למערכת ותוצג לו הודעת שגיאה.

20. מסקנות

המסקנות שלי מהפרויקט לרוב חיוביות אני חושבת שלמדתי דברים רבים שלא הגעתי קודם ולא הייתי מגיעה בשום הזדמנות אחרת. וגיליתי בעצמי כוחות וסבלנות שלא הייתי מודעת אליהם קודם.

לפרויקט זה תרומה אדיה ביחוד לגיל שלנו משום שזהו גיל שעל כל נערה ללמוד על עצמה אילו כוחות קיימים בה. בפרויקט יש המון רגעים של ייאוש ותחושה של כישלון. כשאת מנסה להבין בעיה פעם אחר פעם ולא מצליחה ולא תמיד יש מישהו שיעזור לך, או שאת מנסה להתקדם ולא יודעת לאן וכו'. עלולים ליפול לייאוש אך לבסוף מצאתי כוחות בעצמי שסייעו לי להמשיך הלאה. בנוסף הפרויקט סייע לי בהתקדמות מבחינה לימודית הן מבחינת אנגולר שרציתי להתנסות בה ולהכיר אותה טוב יותר והפרויקט עזר לי מאוד בכך והן מבחינת c# שגם בו הרגשתי קידום והבנה עמוקה יותר.

אז מלבד קידום אישי שסייע לי רבות היה גם קידום לימודי ולכן המסקנות שלי הם שהפרויקט הוא הצלחה גדולה עם הרבה אמונה עצמית ותקווה.

21. פיתוח עתידי


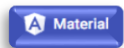





פיתוח עתידי זהו החלק המעניין ביותר משום שהיה עוד המון דברים שרציתי לפתח ולא יכולתי מפאת חוסר הזמן לדוגמא:

רציתי שללקוח תהיה אפשרות של צילום רשימת הקניות שלו ללא צורך בכתיבה ידנית וחיפוש בקטגוריה הנכונה כל מוצר שיצטרך. דבר זה יחסוך ללקוח עוד זמן יקר ויסייע לו רבות.

בנוסף רציתי שהפרויקט שלי יעבוד על כל מפה של סופר שנכניס לו ולא רק על סניף של סופר אחד. דבר זה יכול לסייע רבות לבעלי מספר סניפים כגון: "רמי לוי" או "אושר עד".

בנוסף רציתי לעשות בפרויקט סיוע מילולי למשתמש על פי מיקומו הנוכחי לצורך בעיה זו הייתי צריכה ללמוד את הנושא של ניווט במקום סגור. הכוונה היא שהמערכת תדע בכל רגע ורגע היכן הלקוח נמצא ועל פי זה תדריך אותו להמשיך דרכו. דבר זה גם לא יצא לפועל אבל בעזרת השם אולי בעתיד אגיע אל נושאים אלו ואממש אותם.

23. ביבלוגרפיה :

/https://stackoverflow.com	
/https://material.angular.io	
/https://angular.io	
/https://github.com	
/https://getbootstrap.com	
/https://www.w3schools.com	
/https://www.wix.com	
https://he.wikipedia.org/wiki	