
lg.waze

תוכן

lg.waze

הצעת פרויקט:

מבוא / תקציר:

רעיונות שהיו לי :
חיפשתי פרויקט עם דרכים ומסלולים, אך היו דברים שהם פחות מוצלחים כמו ניווט של מכוניות בכביש, ניווט בבית קברות, ולסיום חשבתי על מערכת שתעזור לאנשים לבחור בגדים על פי סגנון מסוים שהם אוהבים אבל זה היה פחות פרקטי, ולבסוף עלה לי רעיון של וויז בקניון רננים ברעננה.

איך בחרתי את הרעיון שלי :

חשבתי המון על משהו שיכול להיות יעיל באמת, ולתרום לסביבה, וניסיתי לחשוב ממש לעומק מה לא טוב בהתנהלות מסוימת בחיי היום יום ואיך מאידך ניתן לתקן היו לי המון רעיונות בעקבות המחשבה הנ"ל כמו :פינוי מתואם של זבל במקומות מגורים צפופים, למקם תחנות משטרה ברחובות נטושים ובקרבת פאבים ומועדונים, אבל לא ממש התחברתי לנושאים והרגשתי שרוב הדברים (הבעיות) לא הכי גרועים ודי יש שליטה על המצב, לא ברמה מקסימאלית אבל לגמרי בסדר, תוך כדי חשיבות ומציאת רעיון טוב הסתובבתי בקניונים ואחזתי שהבעיה ממש טובה ונוגעת להמון אנשים שבאמת צריך ניווט בקניון גדול ושמחתי שסוף סוף קיימת בעיה .

למה אני חושבת שהוא טוב בתור רעיון לפרויקט גמר?

אני חושבת שלחסוך בזמן במקסימום הנאה וסיפוק זה דבר שכל אחד רוצה, וזה מתבטא בכך שנותן מענה לאנשים שרוצים משהו מסוים או חנויות מסוימות או אחד שהזמן שלו יחסית יקר ואין לו כוח להסתובב, למרות שקניון זה מקום בילוי וזה פחות מתוכנן ומוגבל אף אחד לא מעוניין להסתבך ולחזור על עצמו אז פה הפרויקט שלי נכנס לסיטואציה ופותר את הבעיה–לא במאה אחוז כי אין מושלם אבל זה רוב אחוז ורוב פתרון :)

למה אני חושבת שיש בו תועלת?

התועלת היא מועילה מאוד אין בזמן חיפוש והן בהתמקדות במה לחפש, ובנוסף קיימת אפשרות לבחור את המסלול בשלושה אופציות :

1.מסלול לחנות מסוימת (יחידה).

2.מסלול לפי קטגוריות (רבות)/לפי חנויות (רבות).

3.מסלול חכם – משקלל לפי סטטוס האם המשתמש מעוניין בחנויות עם מבצעים בתחילה וכן מאפשר לשלב לפי הסטטוס חנויות מועדפות בתחילה .

למה מומלץ לפתח את הפרויקט שלי?

מומלץ מאוד לפתח את הרעיון שלי משום שזה מאוד נפוץ בכלל מרכזי הקנייה הגדולים ונותן אופציה למקסימום הספק ותועלת במינימום זמן דבר שכל אחד רוצה (במיוחד ישראלים שאין סבלנות).

מה קורה כיום – לפני הפיתוח?

כיום לפני הפיתוח קיים בזבוז מטורף של זמן וסיבובים מיותרים , ואנשים מאבדים את הסבלנות ואת עצמם בדרך ..

מה יהיה לאחר פיתוח הפרויקט?

לאחר הפיתוח הנ"ל יחסך המון זמן מיותר וחווית הקניה תהיה מהנה ולפי סטטוס- מיועד לכל משתמש שבוחר האם הוא מעונין ב חנויות מסוימות שהוא אוהב ,והאם יש עניין במבצעים בתחילת המסלול, וכך מצמצם האופן משמעותי את הבעיה ונותן מענה לכל אדם ונותן חוויית קניה מושלמת

איזה אלגוריתם עזר לי בפיתוח הרעיון?

האלגוריתם שעזר לי לפתח את הפרויקט הוא הדייאקסטרה חמדני .

עוד על האפליקציה :

השם של האפליקציה הוא- lg.waze .

לאחר עבודה ממושכת וחשיבה עמוקה ועבודה על האלגוריתם

והסתבכות במשקל הקשתות (רציתי שזה יצא הכי מדויק ולכן הייתה לי המון עבודה) – יצאה האפליקציה המדהימה הזאת שבאמת נותנת מענה ומהווה פתרון על ,ובקיצור זה היה שווה את זה כי זה ממש אתגר אותי וחידד את החשיבה –מבחירת הרעיון ועד לסיום פיתוח האפליקציה שתורמת לסביבה כולה.

תהליך המחקר :

הדבר הראשון שעשיתי חקרתי המון על הנושא – קראתי מלא על האלגוריתם פתרתי דוגמאות וראיתי קודים דומים בנושא ניווט, חיפשתי מפות של קניונים גדולים וכו' .

ניסיתי המון בכיוון של חיפוש מיידע וחומרים באינטרנט יותר מיידע תאורתי יחד עם קוד לדוג'. ולבסוף מצאתי מפה שתתאים מישום שבה הקניון ממש גדול ויהווה תחילת בעיה ממש כפי שרציתי, נעזרתי במנהלת שיווק של הקניון ששמה אורית היא נתנה לי כיווני חשיבה נוספים ,תוך כדי הבירורים מאיפה אני מתחילה התייעצתי עם המורות לפרויקט והם כיוונו אותי מאוד. בגלל המפה שאותה מצאתי – זאת מפה של קניון רננים ברעננה וכעיקרון הייתי צריכה ממש לעבוד ע"י ומשקלתי את כל המרחקים כי זה הבסיס . עבדתי המון על הdatabase שיהיה ונכון ,האלגוריתם כעיקרון עובד מצומת מקור ליעד, וכן מצומת מקור ליעד מתוך אחת ליותר משני צמתים, ועכשיו פה נפלתי לבאג ולמה ?! משום שבבסיס אני מקבלת רשימת חנויות –וממנה אני אמורה לחשב את המסלול הקצר ביותר, כי כביכול מהמון צמתים להמון צמתים ואת זה האלגוריתם לבדו אינו מבצעת ולכן על ידי חשיבה עמוקה בניסיון בלחפש פתרון נמצאה דרך מתוחכמת , שבאפשרותה לעבור בשני לולאות ובקוד דייאקסטרה (פירוט בהמשך ..).

ובחרתי דווקא דייאקסטרה חמדני משום שהוא הכי מתאים למציאת מסלול קצר.

משום שפתרון מושלם לבעיית הסוכן הנוסע זו סיבוכיות מטורפת !n.

וגם בלמן פורד לא מתאים בגלל שהוא כולל משקלות שלילים ובמרחקים אי אפשר משקל שלילי.

ולכן המסקנה היא דייאקסטרה הכי מתאים 😊

סקירה ספרותית :

- https://stackoverflow.com
- bootstrap, CSS
- Dijkstra
- angular SQL, ,C#

מטרות ויעדים :

מטרות :

על שימוש באפליקציה שלי חוסכים המון ריצות וסיבובים מיותרים, ובנוסף זה חוסך גם כסף כי אתה יותר ממוקד בקניה שלך בלי להתפזר , והנוחות היא מעל ומעבר שמאשפרת הגדרות ואופציות למסלול חכם (פורט מקודם).
וכן המטרה המרכזית היא לאפשר מענה לפתרון על של מינימום זמן במקסימום עשייה.

היעדים :

המשתמש בוחר באיזה אופציה להשתמש (אחד מהשלושה פורט מקודם)ועל פי בחירתו נשלחות החניות המבוקשות מהSERVIS לAPI ומשם ל- controller המתאים ולבסוף מתבצע האלגוריתם שמחזיר את המסלול הנוכחי בחזרה עד ל SERVIS ולבסוף מוצג על הקנבס באפליקציה ישר למשתמש . ובנוסף המערכת שומרת לכל משתמש האם מעונין בחניות עם מבצעים בתחילת המסלול וחניות מסוימות שיש להם עדיפות , ובכך מאפשרת גמישות לכל משתמש – על ידי הזנה של רשימת החניות שאותן מעדיף והאם הוא מעונין במצעים תחילה המיידע נשמר לכל משתמש בdb, ועוד המערכת מזהה מי לקוח רשום ומי חדש ונותנת מענה בהתאם.

אתגרים:

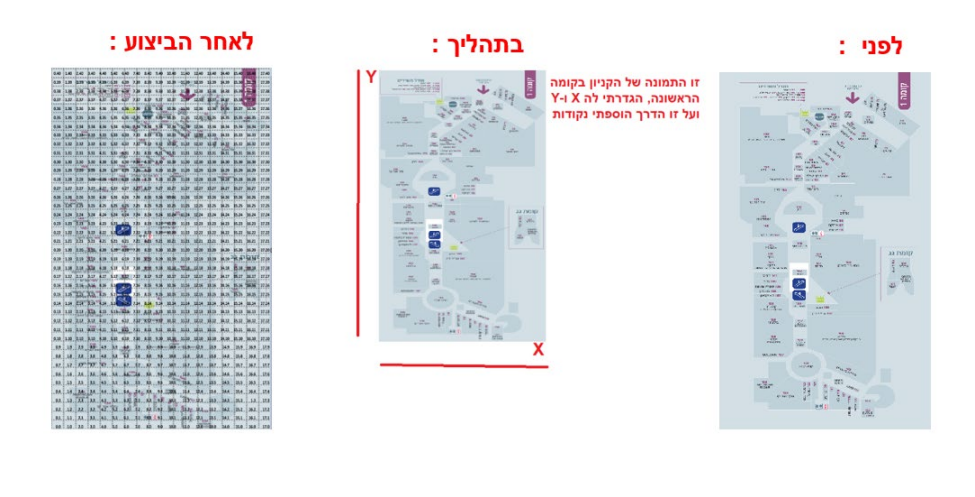
במהלך הפרויקט נתקלתי באתגרים רבים והם:

בעיה:

לא ידעתי מאיפה אני מתחילה למשקל את הקשתות –מה המרחק בין חנות לחנות ?ומה המיקום של כל חנות בקומה?

פתרון:

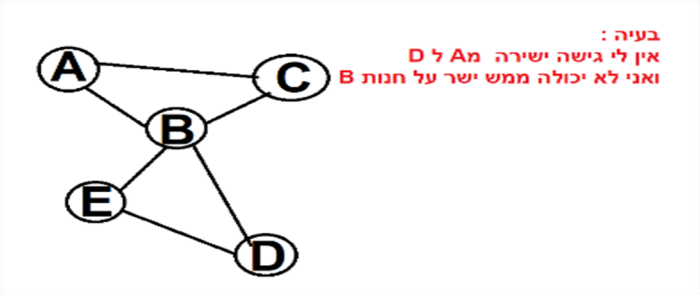
– בתחילה לקחתי את מפת הקניון והפכתי אותה לציר X ו– Y כפי שניתן לראות :



ולבסוף התאמתי לכל חנות במפה מיקום שמורכב וככה אפשר לדעת מה המיקום של כל חנות.

בעיה:

לא מכל חנות יש מרחק ישיר : (, עיין בשרטוט הנ"ל:



פתרון:

אבל אין מה לדאוג משום שכאן נכנס כל הקטע של האלגוריתם :

כי משקלתי לפי נוסחת הדיסטנס (מחשבת מרחק ביו נקודה לנקודה) ,נגדיר לכל מיקום שכנים (צמתים שאפשר להגיע אליהם ישירות) וכך ניתן לעבור מכל צומת לכל צומת ,ולפי הדוגמא : ניתן להגיע מ A ל – D דרך B והמסלול יהיה D,B,A .

בעיה:

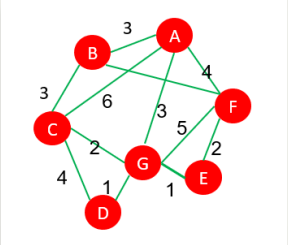
והרי האלגוריתם דייאקסטרה הינו פותר את הבעיה ומחזיר מסלול קצר ביותר בשני אופנים והם :

מצומת יחיד לצומת יחיד.

מצומת יחיד למלא צמתים.

x אבל לא יכול ממלא צמתים למלא צמתים, ונסביר ע״פ הדוגמא הנ״ל :

נתון המסלול של כל החניות בקומה 1 בקניון :

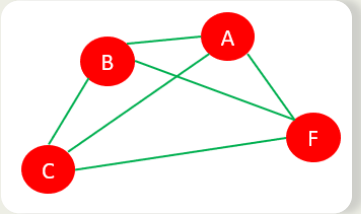


ואני מעוניינת לבקר בחניות : ABCF ,

כדי שדאייקסטרה ידע מה המסלול הקצר – הוא צריך לדעת הקשתות.

יש לי גרף חדש שנראה ככה:

ומשקל הקשתות אינו ידוע כרגע .



כדי להגיע לגרף שמדמה את החנויות שרוצים לעבור בהן עם מרחקים – גרף שיאפשר לדייאקסטרה למצוא את המסלול הקצר.

נצטרך לבנות את הגרף תוך כדי שימוש לדייאקסטרה.
איך נעשה את זה?

לכל חנות – נחשב את המרחק הקצר ביותר לכל חנות אחרת ברשימה . לדוג׳:

A

דייאקסטרה על A->B והמרחק שחוזר הוא המשקל של הקשת.

דייאקסטרה על A->C

דייאקסטרה על A->F

(ב-C# המשמעות היא: מעבר בלולאה ראשונה על החנויות הנבחרות,
לכל חנות מקור: מעבר בלולאה מקוננת על כל החנויות הנבחרות
לכל חנות יעד: מחשבים לדייאקסטרה . התוצאה שחוזרת היא המרחק הקצר ביותר בין המקור ליעד הוא המשקל של הקשת.

מרחק מחנות לחנות סמוכה הוא ידוע.
מרחק בין חנויות רחוקות לא ידוע. (אפשרויות רבות)

לכן:

נשתמש לדייאקסטרה פעמים רבות כדי לחשב מרחק קצר ביותר בין חנויות רחוקות.

ורק לאחר מכאן:

נתבסס על הקשתות שחישבנו כדי להפעיל לדייאקסטרה שמוצא את המסלול הקצר ביותר בין החנויות הרצויות.

אף על פי שהסיבוכיות הינה גדולה :**O(N^2)*(V+E)**

היא עדין עדיפה על אלגוריתם בעיית הסוכן הנוסע שהיא **N!**

מדדי הצלחה:

האפליקציה שלי הצליחה אם היא אכן מצליחה לתת את המסלול הקצר ביותר ומצליחה להציגו בקנבס של המשתמש

תיאור המצב הקיים:

כיום המצב הקיים שמשתמש מגיע לקניון-הוא לא הכי מוגדר במה הוא רוצה לקנות ואיפה ובסוף התוצאה הינה בזבוז זמן בחיפוש החנות או פריט כללי(אצלי זה מתבטא בקטגוריה),והוא הולך ומסתובב ממקום למקום הלוך ושוב .ובנוסף מבזבז יותר כסף כי הוא לא באמת ממוקד בקנייה .

רקע תאורטי:

האלגוריתם המרכזי בפרויקט הוא הקוד של דייאקסטרה:

המשתמש מזין את הקוד משתמש והסיסמא ,המערכת בודקת האם הוא משתמש קיים או חדש .אם הוא חדש יש לו אופציה להזין רק חנות בודדת. והמערכת מציע לו להתחבר כדי לקבל יותר אפשריות, כלומר יש עוד אופציות והן מציאת מסלול קצר מתוך רשימת חניות נבחרות + הגדרת סטטוס (הסטטוס כולל הכנסת חניות עם מצבעים על הדרך) לכל משמש, וכן קיימת אפשרות להזין רשימת קטגוריות (אוכל ,ביוטי, אופטיקה ....) והמערכת מחזירה מסלול קצר למשתמש.

הסבר על האלגוריתם :

האלגוריתם מורכב מכמה שלבים :

יש צמתים וקשתות מוכנים מנם האלגוריתם בונה גרף (כללי כמו אצלי ממיר את החניות והמרחקים בקומה 1 לגרף של צמתים וקשתות) ולאחר מכאן מקבל את רשימת החניות שבהם המשתמש אמור לעבור ויש מעבר בלולאה על הרשימה ,לכל חנות בודקים מה המרחק הכי קצר לחנות הסמוכה ברשימה ולבסוף לכל צמד שני צמתים בונה קשת חדשה (מפעיל דייאקסטרה) ולבסוף מחזיר את המסלול הקצר ביותר.

החלק הכי מורכב באלגוריתם הינו :

החלק הכי מורכב הינו בניית המסלול החדש והגדרת המרחקים החדשים כי הוא עובר בלולאה ולכל צומת מחשב מסלול קצר ולבסוף מחשב דייאקסטרה ומחזיר את המסלול חדש

הבעיה שאלגוריתם פותר :

האלגוריתם פותר את בעיית המסלול הקצר כי מחשב את המרחק הקצר ביותר (מכיל דייאקסטרה) אבל הבעיה הייתה גדולה יותר כי הוא מקבל רשימת של מיקומים ולא מקור ויעד או מקור ויעדים אלא מקורות ויעדים ועל ידי מעבר בלולאה מקוננת ובתוכה הפעלת דייאקסטרה ולבסוף פותר את הבעיה הזאת (של המקורות ויעדים) ומחזיר את המסלול

האלגוריתם שהכי מתאים :

אלגוריתם זה פותר את הביעה של מסלולים קצרים ביותר ממקור יחיד, עבור גרף מכוון ממשקל, עבור משקלות חיוביים בלבד.

הבעיה אצלי הייתה לחשב מסלול קצר מבין החניות שבקומה ,ולכן בחרתי להגדיר את הצמתים בתור החניות ואת הקשתות בתור המרחקים מחנות לחנות .

ניתוח חלופות מערכתי:

בעיית הסוכן הנוסע – הסוכן שלנו מקבל רשימה של ערים ועליו לעבור פעם אחת בדיוק דרך כל אחת מהן. מטרתו היא לעשות זאת במרחק הקטן ביותר האפשרי. זה אולי נשמע פשוט, אבל לא זה המצב: לא ידוע לנו כיום פתרון יעיל של הבעיה. מי שיצליח לפתור את הבעיה הזאת ביעילות יזכה לכבוד רב.

אלגוריתם בלמן פורד–הוא אלגוריתם הפועל על גרף מכוון וממשקל, ומשמש למציאת המסלול הקל ביותר מצומת אחד מסוים אל כל אחד משאר הצמתים בגרף. בכך, אלגוריתם זה משיג אותה תוצאה כמו אלגוריתם דייאקסטרה ,אך בניגוד לאלגוריתם דייאקסטרה הוא עובד גם כאשר הגרף מכיל קשתות בעלות משקל שלילי. יתר על כן, אם הגרף מכיל מעגל שסכום משקלי קשתותיו שלילי (מה שגורם לכך שאין תשובה מוגדרת לשאלת המסלולים הקצרים) הוא מסוגל לזהות זאת ולהתריע על כך, וזאת במחיר

זמן ריצה גדול יותר מזה של אלגוריתם דייאקסטרה. בגרף בעל צמתים ו -קשתות ,זמן

הריצה של האלגוריתם הוא .

אלגוריתם דייאקסטרה:

פותר את בעיית מציאת המסלול הקצר ביותר מנקודה בגרף ליעד. מכיוון שניתן למצוא באמצעות אלגוריתם זה, בזמן זהה, את המסלולים המהירים **לכל** הנקודות בגרף, בעיה זאת נקראת לעיתים **מציאת המסלולים הקצרים מנקודה**.

תוצאת האלגוריתם של דייאקסטרה זהה לתוצאת אלגוריתם בלמן-פורד

אך אלגוריתם בלמן-פורד פועל גם על גרפים הכוללים קשתות שמשקלן שלילי. לעומת זאת, זמן תוצאת האלגוריתם של דייאקסטרה זהה לתוצאת אלגוריתם בלמן-פורד

תיאור החלופה הנבחרת והנימוקים לבחירה:

מה בחרתי ?

בסופו של דבר בחרתי באלגוריתם הדייאקסטרה , משום שהוא הכי מתאים לי כי הסוכן הנוסע זו סיבוכיות מטורפת (N!) ,ובלמן פורד כולל משקולות שליליים ולי זה לא טוב כי אני עוסקת במרחקים והם חייבים להיות חיוביים, מכאן נובע שדאייקסטרה הכי מתאים לי !

פיתחתי מערכת שמחזירה מסלול קצר בתוך קניון, על יד שמקבלת רשימת חנויות נבחרות ומביאה את המסלול הקצר דרכם

לגבי ההצעות הקודמות הסוכן הנוסע ישלו סיבוכיות מטורפת וזמן תגובה מאוד איטי , וכן בלמן פורד לא ממש טוב הכי עובד עם משקלות שליליים, ולכן אינו מתאים כי מרחק רק חיובי ואין אופציה אחרת,

האלגוריתם עובד כך:

מוגדר רשימת קשתות ומילון צמתים,

יש פעולה שבונה מהרשימה והמילון גרף כלליי הקניון ,

ואחרי שהגרף בנוי המערכת מקבלת רשימה של חניות שבהם צריך לעבור את המסלול הקצר ביותר , והיא עובדת כך :

יש לולאה שעוברת על רשימת החנויות המבוקשות ,

שבתוכה קיים מעבר נוסף מחנות לחנות סמוכה ברשימה,

שבה יוצרים קשת בין החנות ברשימה לחנות אחרת ברשימה לפי סדר,

ואז מפעילים את שדאייקסטרה ואת המרחק הקצר ביותר מכניסים לקשת שיצרנו בשורה מעל שהוא בודק ממש ועובר על כל האפשריות בגרף הכללי

ואת הקשת מכניסים לרשימת הקשתות החדשות

ולבסוף בונה צמתים שהם הרשימה המבוקשת שקבלנו

ויש לי את רשימת הקשתות החדשות שנבנתה לי בפעולה

ומפעילים שוב את הפונקציה שבונה גרף מצמתים וקשתות ושולחים לה את שני הפרמטרים

רשימת הקשתות החדשות, הרשימה המבוקשת שקבלנו ,

ומה שחוזר לי מכל זה הוא מסלול הקצר ביותר :)

שאותו אני שולחת למשתמש בתצוגה .

אפיון המערכת:

סביבת פיתוח :

חומרה: מעבד

i7 RAM 32GB

עמדת פיתוח:

מחשב Lenovo

מערכת הפעלה: 10

Windows

שפות תוכנה: C# , תוך שימוש בטכנולוגי ת WebApi, אנגולר .

כלי תוכנה לפיתוח המערכת: Microsoft Visual Studio2019, vs code.

מסד

נתונים:. SQL

Server עמדת

משתמש

מינימאלית :

- חומרה: מעבדi5 RAM 4GB .
- מערכת ההפעלה: Windows 7ומעלה.
- חיבור לרש ת: נדרש .
- chrome: . תוכנות,

ניתוח דרישת 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.

מודול המערכת

- העלאת קובץ txt המכיל את רשימת חניות עם פרטים טכניים עליהם.
- קריאת הקובץ שהועלה והזנת הנתונים ב database, וכן הצגתם ללקוח.
- הזנת נתונים נוספים לתלמידים ע"י המשתמש.
- קריאה למערכת לביצוע התאמה.
- מציאת המסלול הקצר ביותר ככל שניתן ע"י האלגוריתם.
- הזנת פרטי השיבוץ בdatabase .
- הצגת פרטי השיבוץ ללקוח.
- שליחת מייל ללקוח עם השיבוץ הסופי

אפיון פונקציונאלי

 # (createMallRoutes(string path : פונקציה זו מקבלת ניתוב לקובץ שמכיל את המרחקים בין חנות לחנות , ועל ידי זה היא בונה רשימת קשתות כללית של כל הקומה ,

 # ()createMallNodes : הפונקציה בונה צמתים (חניות) על פי המיידע מה -DB-של כל החניות בקומה ,

#(createSelectedStoresGraph(List<DTOStor> stores : הפונקציה יוצרת רשימת צמתים של הגרף החדש – המסלול והוא בונה את הקשתות (המרחקים) על ידי מעבר על רשימת הצמתים שהפו' קיבלה ובתוכה עובר בקינון לכל אחד מהחניות ברשימה ומפעיל דייאקסטראה והערך שחוזר ממנו מציבים בקשת החדשה ומוסיפים מאיפה לאיפה הקשת נוצרת ,ואת הקשת הזאת מוסיפים לרשימת הקשתות של המסלול החדש.

 # createShortestPathForSelectedStores(): יוצר את המסלול הקצר ביותר ומפעיל את הפו' המתאימות,

 # CheckNode(List<Route> routes, Dictionary<string, Node> nodes, PrioQueue queue, List<Node> unvisited, Node destinationNode):

עובדת די דומה לאלגוריתם דייאקסטראה המקורי (ייהנו אותו רעיון ממש –רק שמותאם למה שאני עשיתי)

פו' זו הינה רקורסיבית ומקבלת רשימת קשתות ומילון צמתים ותור עדיפות מרחקים קצרים ורשימת צמתים שלא עברו בהם וצומת יעד הפונקציה מחזירה את המרחק הקצר ביותר .זאת אומרת ממש מחשבת את הדייאקסטרה –ממש מסלול בין כל החניות הנבחרות .(המסלול החדש)

<u>הביצועיים העיקריים :</u>

 # המשתמש (רק אם הוא מחובר למערכת) יכול להגדיר פרופיל , זאת אומרת להכניס:

האם הוא מעונין בחניות עם מבצעים בתחילה (כן / לא).

המשתמש מכניס שם משתמש וסיסמא

המערכת בודקת האם הוא מחובר או חדש

במידה והוא מחובר יש לו סטטוס (פרופיל קיים) ולכן הוא יכול לבחור באיזה סוג מסלול הוא מעוניין והן :מסלול לפי רשימת חניות לבחירה \ רשימת קטגוריות \ לחנות בודדת \ לקטגוריה בודדת

לאחר שהמשתמש בוחר מה הוא רוצה (איזה מסלול) ,ומזין את הפרטים בהתאם ,המערכת מחשבת והאלגוריתם רץ –ובסיומו מוגש למשמש על הקנבס את המסלול הקצר ביותר (המסלול הכי חכם שיכול להיות :)))

<u>אילוצים :</u>

המערכת מוגבלת כרגע לקניון אחד ספציפי (קניון רננים ברעננה),

המיידע אינו מתעדכן כל פעם כי זה בתאם למידע העכשווי,

המרחקים נם מחושבים באופן יחסי כך שהמסלול הוא מדויק בתכלית אבל זה לא אמור להיות משמעותי במקסימום זה פסיעה וחצי ,

<u>תיאור הארכיטקטורה :</u>

<u>הארכיטקטורה של הפתרון המוצע בפורמט של Design level Down-Top :</u>

צד השרת – server side פותח במודל 3 השכבות ומתחלק ל-4 פרויקטים

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.

API – שכבת ה Controller – חיבור בין צד השרת והלקוח.

BL – הלוגיקה של המערכת.

DAL – מכיל את הפונקציונאליות הנדרשת לכל התקשורת עם ה Data Base.

Models – 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 הModels ולהיפך, וכך מיוצגים הנתונים בכל הפרויקט.

<u>תיאור הרכיבים בפתרון :</u>

הפרויקט מחולק ל-2 חלקים:

- צד שרת – הנכתב בשפת #C ובטכנולוגיית WebApi.
- צד לקוח – נכתב בשפת Angular ובטכנולוגיית Html, TypeScript.

בחרתי לכתוב צד לקוח ב – אנגולר שהינה טכנולוגיה מתקדמת ועדכנית בעלת מאפייני Angular8 חדשניים ופונקציונאלית ביות ר.

אנגולר הינה סביבת עבודה שפותחה על ידי גוגל. מאפשרת לפתח אפליקציות Framework אינטרנט בקלות ומהירות. במקור היא באה לתת מענה לבניי ת Applications Page Single בצורה מושלמת ומהירה. מהיתרונות הבולטים והעיקריים של אנגולר אפשר למנות: חיסכון במשאבים, מהירות ביצוע, קוד קצר יותר, רוב העבודה מתבצעת בצד הלקוח ופחות בשרת ויכולת התמודדות טובה)סינון מהיר ופשוט לביצוע (של תוכן המתקבל מהשרת לפי מספר רב של פרמטרים.

צד שרת בחרתי לכתוב ב#C. #C היא שפת תכנות עילית מרובת-פרדיגמות, מונחית עצמים בעיקרה המשלבת רעיונות כמו טיפוסיות חזקה, אימפרטיביות, הצהרתיות, פונקציונאליות פרוצדוראליות וגנריות .

#C היא שפה מעניינת, נוחה ומלאה פונקציונאליות למתכנת. שימוש בשפה זו נפוץ כיום, וכתוצאה מכך, ניתן היה למצוא בה קודים שונים שנדרשו לפיתוח.

בנוסף, בחרתי להשתמש ב – Entity Framework טכנולוגית עבודה מתקדמת של מיקרוסופט.

ה Entity Framework מאפשר לטעון את הנתונים מה DB-ולעשות להם השמה בצורה ישירה ואוטומטית לתוך אובייקטים בקוד הממפים את מאגר הנתונים בצורה מידי ת.

הEntity Framework קורא נתונים מ הData Base שנכתב בשפת Sql Server .

למסד הנתונים של ה- SQL Server יש כלים נרחבים לגיבוי כל המידע של המערכת ,כולל מערכת ההפעלה, חשבונות המשתמשים והרשאותיהם, הגדרות ההתקנים, תוכניות וכן של שאר הרכיבים המסופקים עם השרת ואובייקטי המשתמש.

דוגמאות לזרימת מידע במערכת :

שליפת כל החניות המועדפות

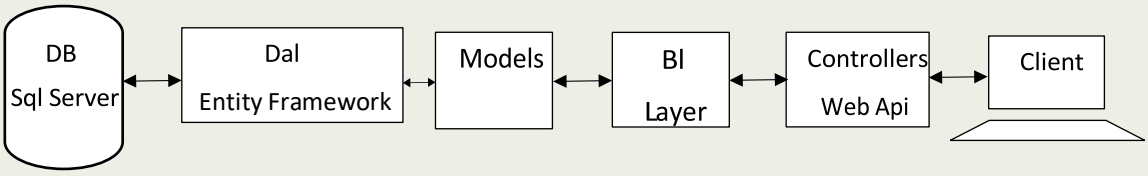
ברצוננו לקבל את כל החניות המועדפות של משתמש מסוים מ הDB ולכן יתבצעו השלבים הנ"ל:

- המשתמש יחפוץ לראות את כל החניות המועדפות שלו, הוא ילחץ על כפתור מסוים בתצוגה (html) ובקשתו תפנה לTypeScript.
- services אשר תפנה ל GetStorFaverite תתבצע קריאה לפונקציה script – Type לשרת URL תתבצע בקשת – services
- השרת מקבל את הבקשה ומנווט ל Controller שנמצא בAPI.
- הController יזמן את הפונקציה `GetAllStorFavoraite(DTOUsers` (`user`שנמצאת FavoriteStoresForTheUserController.
- BL מעוניין לקבל נתונים מה DB ולכן הוא פונה לDAL- דרך ה framework Entity
- ה-DAL שואב את הנתונים הרצויים ממסד הנתונים וכעת מתבצע שלב החזרה.

ה DAL מחזיר את רשימת החניות המועדפות לשכבת הBL בה מתבצעת פונקציית הסינון של הבאת החניות של משתמש מסוים.

- BL מהcontroller מחזירה את הנתונים ל GetStorFaverite הפונקציה
- service . הנתונים מוחזרים ל- controller מה
- מה- service חוזרת הרשימה לtypeScript.
- הרשימה מוצגת בHTML.

איור :



מסד הנתונים הבנוי מטבלאות וקשרי גומלין ביניהם.

שכבת הגישה לנתונים באמצעות Entity Framework.

שכבת הישויות.

שכבת ה – BL בה כתובים האלגוריתמים.

Web Api פרוטוקול התקשורת בי ן צד הלקוח וצד השרת.

angular, TypeScript צד לקוח.

תיאור פרוטוקולי התקשורת :

http – הוא פרוטוקול תקשורת שנועד להעברת אובייקטים ודפי html

שרת-לקוח :

צד השרת נכת ב בטכנולוגית WebApi ובשפת #c.

צד הלקוח נכתב בשפות Html, angular בטכנולוגית typescript, CSS.

10 .ניתוח ותרשים USE CASE של המערכת המוצעת :

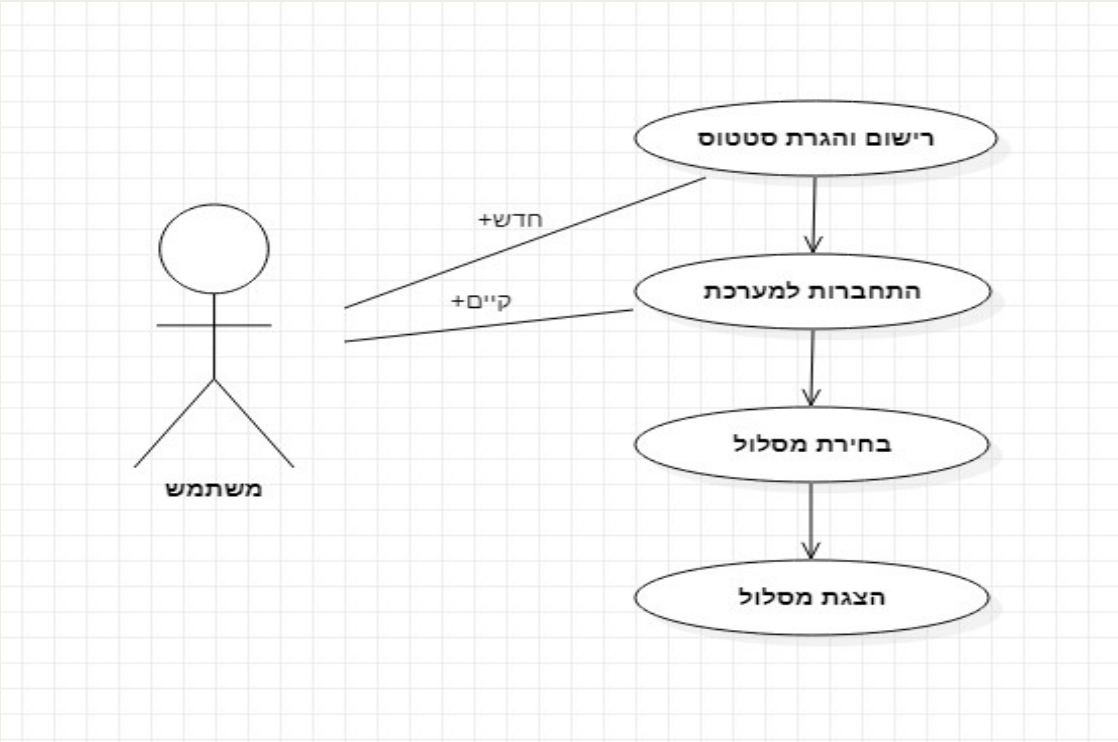
רשימת use case :

רשימת הפעולות המבוצעות על ידי המשתמש :

למשתמש חדש : מקליד שם משתמש וסיסמא מכניס סטטוס האם רוצה חניות עם מבצעים תחילה ,ורק אחרי שהוא מוכר במערכת (הרישום הצליח) המשתמש ראשי לבחור מסלול –על ידי בחירת חניות / קטגוריות ולאשר – לאחר מכאן מוצג המסלול הרצוי הקצר והטוב ביותר,

למשתמש וותיק : הקלדת שם משתמש וסיסמא , במידה והמערכת מזהה הוא רשאי לבחור מסלול ולבחור קטגוריות / חניות ולאשר – לאחר מכאן מוצג המסלול הרצוי הקצר והטוב ביותר,

1.10 תיאור ה- USE CASE :



1. פעולות המשתמש מתחלקות ל- שני סוגים : משתמש **חדש** ומשתמש ישן אם המשתמש חדש הוא נדרש להירשם, זאת אומרת להכניס שם משתמש וסיסמא , ולאחר מכאן הוא מחובר כלומר מזהה במערכת- אם המשתמש **ישן** הוא מתחבר – מכניס שם משתמש וסימא ולאחר שהוא מזהה במערכת הוא ראשי להיכנס.
2. הוא בוחר באיזה מסלול הוא רוצה וזה כולל לבחור את המסלול להיכנס או רשימת חנויות או חנות בודדת או רשימת קטגוריות לפי הנדרש
3. ולבסוף מוצג המסלול ממש בהתאם למה שהכניס ובחר (סוג המסלול, חנויות /קטגוריות).

מבני נתונים בהם משתמשים בפרויקט :

dictionary : השימוש במילון ניתן לידי ביוטי באלגוריתם בשמירת הצמתים של הגרף הכללי ושל הגרף שצירך להחזיר השתמשתי בו כי זה היה לי הרבה יותר נוח ורציתי קצת לגוון (לא כל הזמן רשימות):

```
        static Dictionary<string, Node> selectedStoresGraphNodes = new
Dictionary<string, Node>();//אתחול המילון הצמתים של הגרף החדש
```

```
        static Dictionary<string, Node> mallGraphNodes = new Dictionary<string,
Node>();//אתחול המילון שמורכב משם צומת וצומת
```

list : השימוש ברשימה ניתן לידי ביוטי באלגוריתם בשמירת הקשתות של הגרף הכללי ושל הגרף שצירך להחזיר השתמשתי בו כי זה היה לי הרבה יותר קל הזמן רשימות

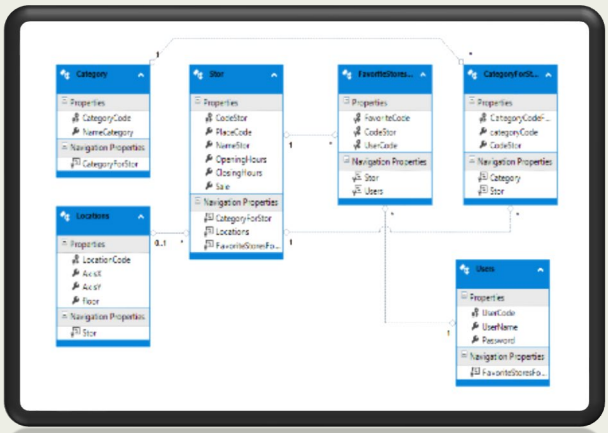
```
static List<Route> mallGraphRoutes = null;// אתחול רשימת הקשתות של הקומה הראשונה בקניון

static List<Route> selectedStoresGraphRoutes = new List<Route>();// אתחול רשימת הצמים בגרף החדש שניצור ייהנו המסלול
```

queue : השימוש בתור ניתן לידי ביוטי באלגוריתם בתור מבנה שעוזר לאלגוריתם בזמן ריצה כלומר ביעילות משום שמכניס כל פעם את הצומת שערכו הכי קצר שהוא נמצא בראש התור –ובסוף נמצא הצומת הכי גדול של הגרף הכללי ושל הגרף שצירך להחזיר השתמשתי בו כי זה היה לי הרבה יותר קל הזמן רשימות **(תור זה נותן יעילות מרובה כי הוא חוסך זמן מעבר על כל הקשתות הסמוכות – כי הוא פשוט משתמש בתור שמביא את כל הצמתים על ערך ה- value לפי סדר ממיון).** והוא מוגדר בתור מחלקה שמממשת אותו .

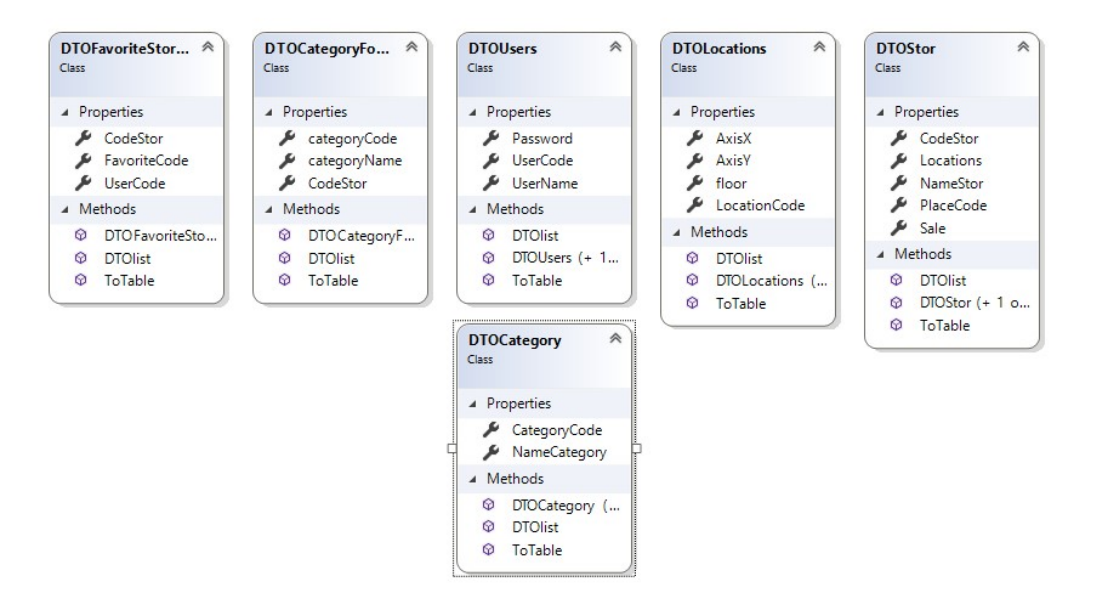
```
. PrioQueue queue = new PrioQueue();
```

תרשים מחלקות :

של ה – DAL

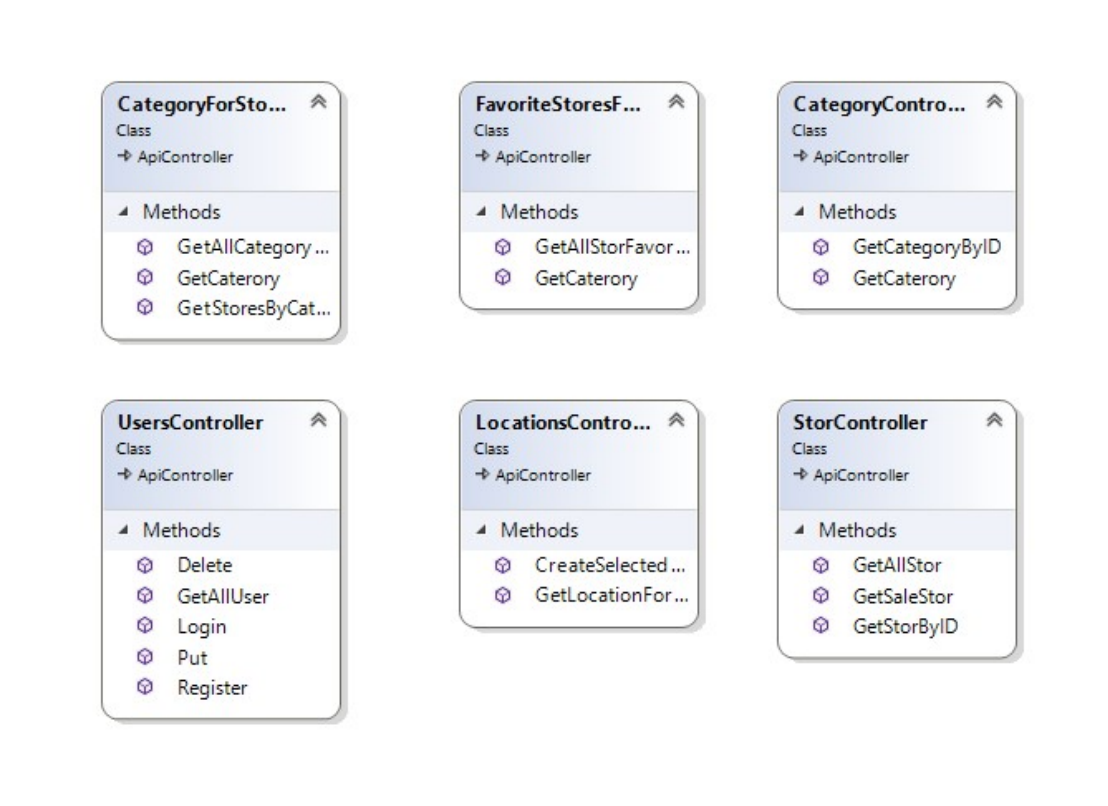


סדס מחלקת :



מחלקת :

ה-CONTROLER



controlerUser :

פונקציה ששולפת את כל הקטגוריות בקניון//

```
[Route("api/Category/GetCaterory")]
[HttpGet]

public List<DTOCategory> GetCaterory()
{
```

```
            List<DTOCategory> u = BL.ManagerCaterory.GetCategories();
            return u;
        }
```

מחזיר את הקטגוריה לפי המספר ID :

```
        [Route("api/Category/GetCategoryByID/{id}")]
        [HttpGet]

        public DTOCategory GetCategoryByID(long id)
        {
            DTOCategory dc =
BL.ManagerCaterory.GetCategories().FirstOrDefault(a=>a.CategoryCode==id);
            return dc;
        }
```

CategoryForStorController:

ממחזיר את כל הקטגוריות לחנות:

```
        [Route("api/CategoryForStor/GetCaterory")]
        [HttpGet]
        public List<DTOCategoryForStor> GetCaterory()
        {
            List<DTOCategoryForStor> u =
BL.ManagerCategoryForStor.GetCategoryForStor();
            return u;
        }
```

מחזיר רשימה של חניות לפי קטגוריה מסוימת:

```
        [Route("api/Stor/GetStoresByCategory/{category}")]
        [HttpGet]
        public List<DTOStor> GetStoresByCategory(string category)
        {
            return BL.ManagerCategoryForStor.GetAllStorOfXContaining(category);
        }
```

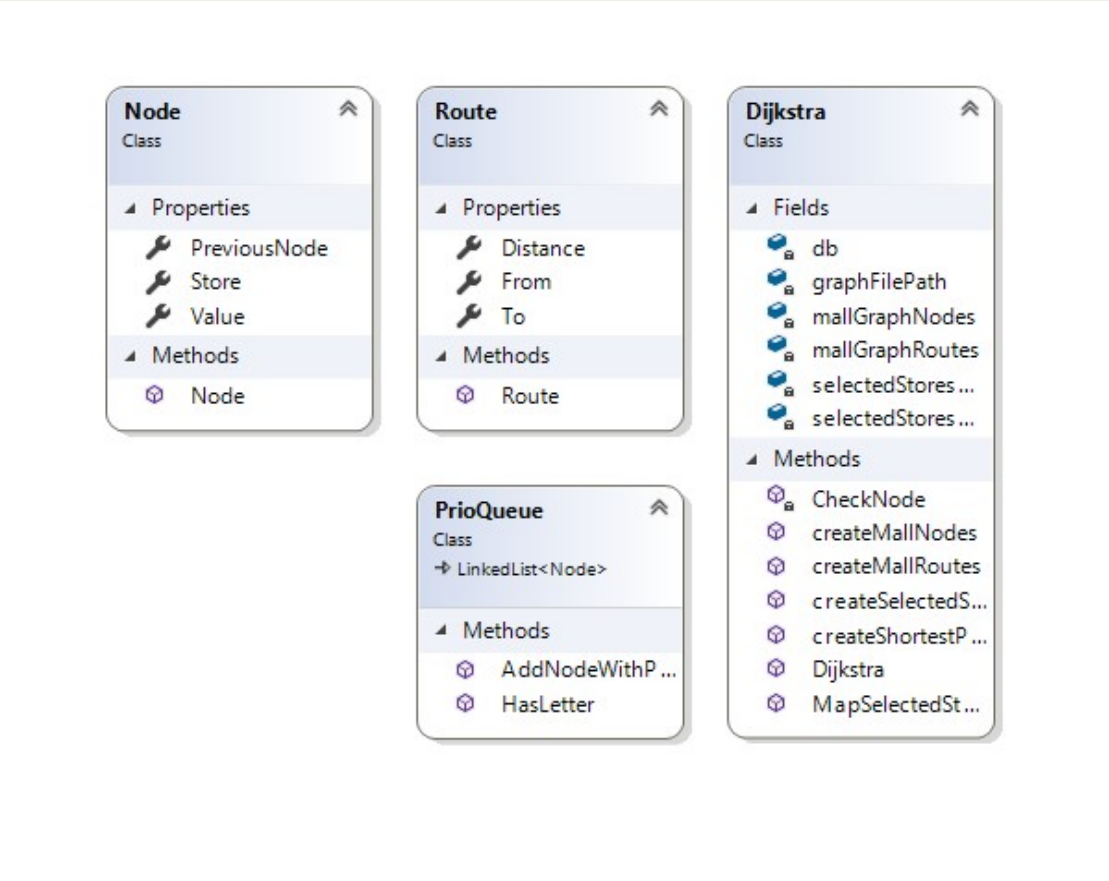
מחזיר רשימת של קטגוריות לחנות

```
        [Route("api/Stor/GetAllCategorysForStor/{stor}")]
        [HttpGet]
        public List<DTOCategory> GetAllCategorysForStor(string stor)
        {
            return BL.ManagerCategoryForStor.GetAllCtegoryForStor(stor);
```

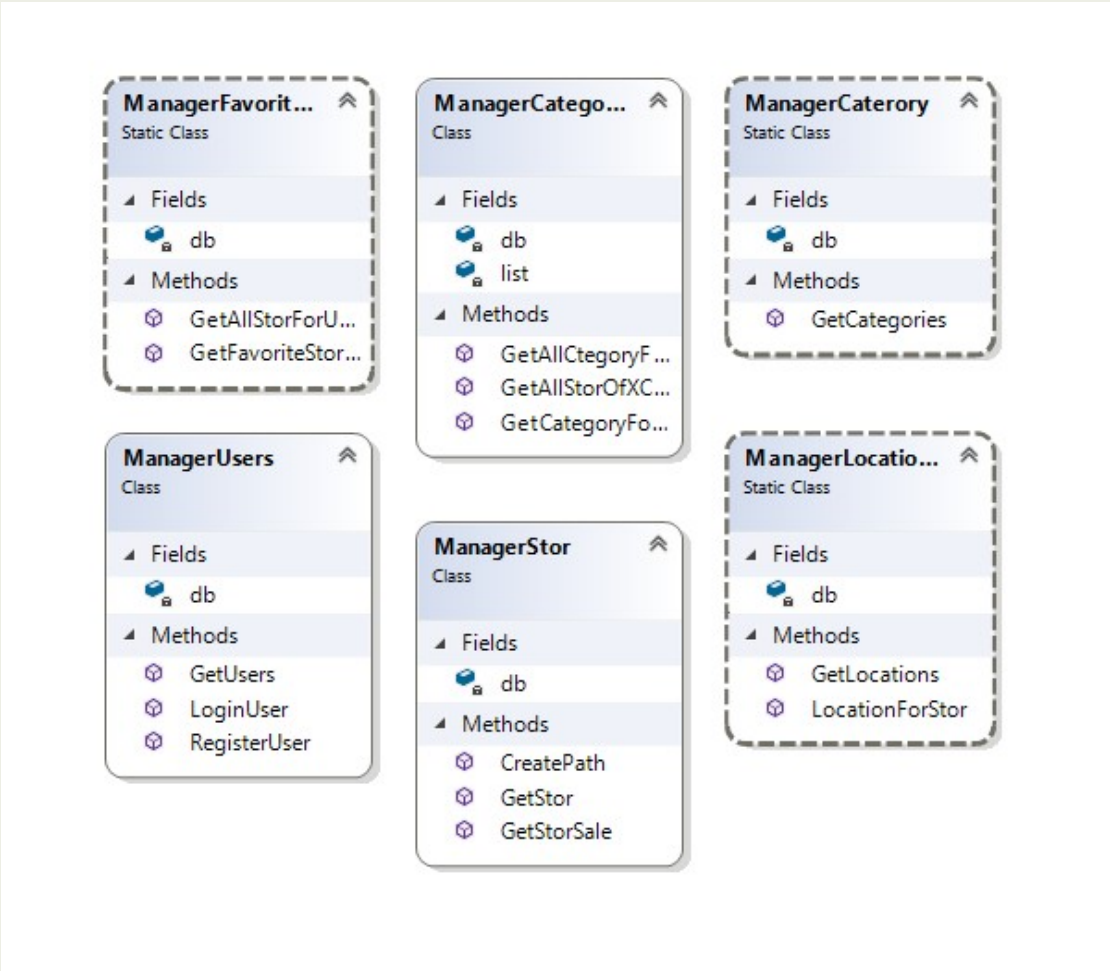

```
}
```

FavoriteStoresForTheUserController :

מחלקת BL :



ה- MENEGER :



<u>לכל שכבה תיאור המחלקות שלה:</u>

<u>שכבת ה-DAL</u> :

<u>תיאור המחלקות :</u>

<u>מחלקות בתוך ה DTO-dal :</u>

הסבר :

המחלקות האלו יש בהם את התכנות של

הטבלאות המקוריות מלבד הקשרים.

ונחלק את זה :

מחלקה תצוגה – DTO

מחלקה מקורית -כולל הקשרים

-DTOCategory

מחלקת קטגוריה .

קיימות בתוכה הפונקציות הללו:

`public DTOCategory()` - פעולה בונה ריקה

#

-DTOCategoryForStor

מחלקת קטגוריה לחנות (טבלת קשר ביו קטגוריות לבין חניות)

-DTOFavoriteStoresForTheUser

מחלקת חניות מועדפות למשתמש שומרת לכל משתמש איזה חניות מועדפות לו

-DTOLocations

מחלקת מיקומים של פתחי חנות

-DTOStor

מחלקת חניות

-DTOUsers

מחלקת משתמשים

שכבת ה- BL :

תיקייה של האלגוריתם בה :

מחלקת Dijkstra :זה האלגוריתם הכתוב עצמו

מחלקת Node : זה המחלקה של הצמתים בגרף כדי ליצור מסלול (החניות הם הצמתים)

מחלקת PrioQueue : זה מחלקת שמממשת תור שעחר לצמצם באופן משמעותי את זמן הריצה

מחלקת Route: זה מחלקת קשתות שבניות וערך-מרחק צומת מקור וצומת יעד

תיקיית מודול :

מחלקת UserInformation : כמין מחלקה שמתארת אובייקט של משתמש כאשר הינו מעוניין להירשם (מחלקה זו מכילה רק שם משתמש וסיסמא משום שבמחלקת משתמשים יש קוד משתמש אשר ניתן באופן אוטומטי ולא קשור למה שהמשתמש מזין)

ManagerCategoryForStor– מחלקה אשר מכילה בתוכה את הפונקציות הקשורות קטגוריות לחנויות.

ManagerCaterory–מחלקה אשר מכילה בתוכה את הפונקציות הקשורות לקטגוריות.

ManagerFavoriteStoresForTheUser– מחלקה אשר מכילה בתוכה את הפונקציות הקשורות לחניות אהובות למשתמש.

ManagerLocations– מחלקה אשר מכילה בתוכה את הפונקציות הקשורות למיקומים.

ManagerStor– מחלקה אשר מכילה בתוכה את הפונקציות הקשורות לחנויות.

ManagerUsers– מחלקה אשר מכילה בתוכה את הפונקציות הקשורות למשתמשים.

שכבת WEB- API :

הסבר :

CategoryController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל- ManagerCaterory ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

CategoryForStorController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה לחנות ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל–ManagerCategoryForStor ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

FavoriteStoresForTheUserController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה לחנות ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל–ManagerFavoriteStoresForTheUser ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

LocationsController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה לחנות ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל- ManagerLocations ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

StorController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה לחנות ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל- ManagerStor ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

UsersController: מחלקה אשר מקבלת בקשות בנוגע לקטגוריה לחנות ומחזיר את המיידע המבוקש היא פונה לפי שם הבקשה ומנווטת לפונקציה הנכונה ב- BL ופונה ל- ManagerUsers ולאחר מכאן מחזירה את המיידע המבוקש. מבצעת את הפעולה שנתבקשה .

תיאור התוכנה

- סביבת עבודה:

Visual Studioו Visual Studio Code

- שפות תכנות:

צד השרת נכתב בטכנולוגית WebApi ובשפת c#.

צד הלקוח נכתב Html, CSS, typescript- בטכנולוגית angular .
בשפות

אלגוריתמים מרכזיים :

אני קבלתי רשימת צמתים – חנויות –לעבור בהם ומהם ליצור את המסלול הכי קצר שיש.

שלב ראשון :

רשימת קשתות של כל הקניון :

אני בונה את הרשימה של הקשתות אשר כל קשת מורכבת מחנות מקור מחנות יעד וממרחק מהיעד למקור, ואת כל המידע הזה אני לוקחת מקובץ טקסט אדר מאחל את הרשימה –ובסוף ברשימה הזו יש את כל הקשתות בקניון ,

מילון צמתים של כל הקניון :

מילון זה בנוי משם חנות וחנות (מפתח, ערך– בהתאמה), ואני בונה אותו מdb– ולבסוף המילון קיימות כל בחניות בקניון ,

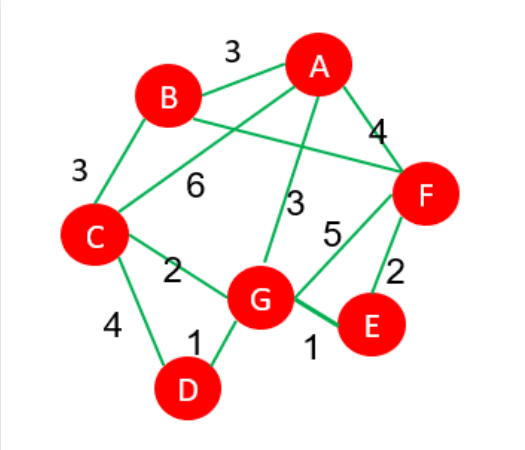
יצירת הגרף הכללי (כל הקניון) :

ואז אני מפעילה פונקציה שהיא מחברת בין המילון והרשימה ובונה ככה את הגרף,

הפעלת הדייאקסטרה על הגרף הכללי של הקניון :

הרי **הדייאקסטרה יכול לפעול רק ב–2 אופציות :**

היא לעבור ממקום ליעד (מרחק הכי קצר בתוך גרף מנקודת התחלה ונקודת סיום –2 נקודות)



מרחק קצר בין כל הנקודות –נגיד לפי הציור הזה :

אני עוברת על כל הצתים ומחשבת מסלול קצר בין

כל הצמתים (מציאת המסלול הכי קצר בכל הגרף)

על הגרף הכללי (כל הקניון) ,

אבל זה לא מה שאני צריכה :

כי אני אמורה לעבור על רשימה חלקית מהקניון

ולבנות גרף חדש שמורכב מהחנויות האלו , ואת הקשתות אני צריכה לחשב לבד (כי יש אין ספור דרכים להגיע) מכל צומת לצומת ברשימה ובנוסף אני צריכה את סדר החניות לפי מה שיותר קצר מבין הרשימה (מסלול קצר –חישוב על כל החנויות ברשימה)

בקיצור מה שאני צריכה – זה לחשב על הגרף של כל הקניון חישוב של דייאקסטרה מצות מקור ליעד (לכל צומת מקור –חישוב הכי קצר של הצומת השכן שלו) בלולאה על כל הגרף מה שיוצא זה חישוב של כל זוג צמתים שכנים הכי קצר (ע״י דייאקסטרה) השתמשתי כאן במה שהאלגוריתם דייאקסטרה יכול לעשות מציאת מסלול קצר בין 2 צמתים, ובסוף יצא לי גרף של הקניון שהקשתות ביין כל צומת הם הקצרות ביותר – אני מוצאת את המסלול הקצר ביותר בין כל זוג צמתים בגרף –אז הייתה פה הפעלה של דייאקסטרה מסוג של חישוב לשני צמתים והשתמשנו פה בפונקציה –CheckNode

 # אני יוצרת את הגרף החדש (הסלול שצריך לחזור – מה שחישבתי) :

אתחול הצמתים : את הצמתים אני מאתחלת לפי הרשימה שקבלתי (הרשימה שבה המשתמש צריך לעבור),

את הקשתות : אני מפעילה שוב את CheckNode הפעם על הגרף החדש וילך על פי הקשתות שחישבנו מקודם (מרחק קצר בין כל הנקודות), והפעם ההפעלה תהיה שונה מכוון שתחשב מסלול קצר בין כל הנקודות בגרף החדש,

ואני הגעתי לשלב שאני יודעת כמה כל קשת במסלול החדש שווה :

אני יודעת מה הקשת בין F –c וכן הלאה ,

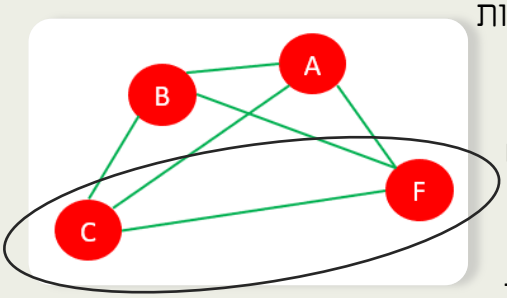
אבל עדיין לא שיממנו אמנם הגעתי לקשתות

בגרף זה יפה מאוד אבל זה לא מספיק

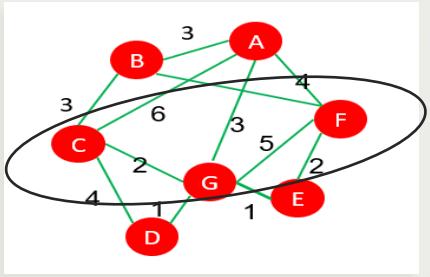
מדויק משום שלא תמיד קיימת קשת ממש

ישירה כלומר הקשת שמצאנו בין C–לF

היא לא קיימת ב=גרף של הקניון כולו ולכן



בדרך חזור אני שומרת את המסלול שמצאתי בין Cל F לפי הדוגמא ואני מחזירה אותו לפי הגרף הכללי בתיאום כלומר אם יצא לי שהקשת הכי קצרה היא נוצרת על ידי המסלול – CFG



אז אני צריכה לסמן את זה במסלול אני לא יכולה לעשות CFוזהו אני צריכה להתחשב במה שקיים ולבסוף אני מחזירה את המסלול הטוב ביותר למעבר הקצר ביותר בהתאם לגרף הכללי,

קוד האלגוריתם

כל המשתנים הגלובליים :

```
//הדאטה ביס של כל המערכת
static DBConection db = new DBConection();
//הניתוב של הגרף
static string graphFilePath = @"C:\Users\student\Desktop\liel\צד שרת\routes.txt";// @"..\
//אתחול רשימת הקשתות של הקומהה הראשונה בקניון
static List<Route> mallGraphRoutes = null;
//אתחול המילון שמורכב משם צומת וצומת
static Dictionary<string, Node> mallGraphNodes = new Dictionary<string, Node>();
// אתחול רשימת הצמים בגרף החדש שניצור יהנו המסלול
static List<Route> selectedStoresGraphRoutes = new List<Route>();
//אתחול המילון הצמתים של הגרף החדש
static Dictionary<string, Node> selectedStoresGraphNodes = new Dictionary<string, Node>();
//מקבלת רשימה של חניות למעבר ומחזירה את הרשימה הטובה =היעילה ביותר
1 reference | lielGlazer, 4 days ago | 1 author, 1 change
```

הפונקציה הראשית :

```
public static List<DTOStor> MapSelectedStores(List<DTOStor> stores)
{
    List<DTOStor> GETlist=new List<DTOStor>();

    //1 - יצירת גרף של כל הקניון - קריאת הקשתות מקובץ
    mallGraphRoutes = createMallRoutes(graphFilePath);
    //2 - יצירת הצמתים שנמצאים בכל הקניון
    mallGraphNodes = createMallNodes();
    //3 - בין החנויות הנבחרות - הפעלת הדיאקסטרה לכל חנות ברשימה ( שאר החנויות כיעד
    //יצית הגרף החדש - אתחול רשימה חדשה של קשתות וצמתים
    createSelectedStoresGraph(stores);
    //4 - חישוב מסלול בגרף החדש - דיאקסטרה שני
    Node end = createShortestPathForSelectedStores();
    // 5 - מציאת הצמתים במסלול ע''י חזרה אחורה
    List<Node> finalNodes = FindNodesOfShortestPath(end);
    //המרת הרשימה ללקוח על ידי המרה מNODEלDTOStor
    foreach (var g in finalNodes)
    {
        GETlist.Add(g.Store);
    }
    return GETlist;
}
```

יצירת כל הקשתות בקניון :

```
//המרחקים הקשתות יוצרת גרף של כל המרחקים בקניון
1 reference | lielGlazer, 4 days ago | 1 author, 1 change
public static List<Route> createMallRoutes(string path)//
{
    List<Route> routes = new List<Route>();//
    using (var reader = new StreamReader(graphFilePath, Encoding.Default))
    {
        for (int i = 0, countWord = 0; !reader.EndOfStream; i++, countWord = 0)
        {
            var line = reader.ReadLine();
            var values = line.Split(',');
            Stor s1 = db.GetDbSet<Stor>().FirstOrDefault(s => s.Locations.LocationCode == long.Parse(values[2]));
            Stor s2 = db.GetDbSet<Stor>().FirstOrDefault(s => s.Locations.LocationCode == long.Parse(values[1]));

            routes.Add(new Route(new Node(new DTOStor(s1)), new Node(new DTOStor(s2)), double.Parse(values[0])));
        }
    }
    return routes;
}
```

יצירת כל הצמתים בקניון (החניות) :

```
//הצמתים יוצרת גרף של כל החנויות בקניון
1 reference | lielGlazer, 4 days ago | 1 author, 1 change
public static Dictionary<string, Node> createMallNodes()
{
    Dictionary<string, Node> nodes = new Dictionary<string, Node>();
    List<Stor> stores = db.GetDbSet<Stor>();
    foreach (var s in stores)
    {
        DTOStor dtos = new DTOStor(s);
```

פונקציה שמקבלת רשימת צמתים (חניות) שצריך לעבור בה , ויוצרת גרף חדש (כולל שלב האתחול שלו) קיים שימוש ב CheckNode שימוש ראשון :

```
public static void createSelectedStoresGraph(List<DTOStor> stores)//יצירת הגרף החדש
{
    Stor entrance = db.GetDbSet<Stor>().FirstOrDefault(s => s.NameStor.Equals("כניסה"));
    stores.Add(new DTOStor(entrance));
    //לכל חנות ברשימה הנבחרת
    for (int i = 0; i < stores.Count(); i++)
    {   //מאיפה להגיע מאיזה מקור
        DTOStor from = stores[i];
        //מוסיפה אותו לרשימת הצמתים בגרף החדש
        selectedStoresGraphNodes.Add(from.NameStor, new Node(from));
        //ניצור קשתות לשאר החנויות ברשימה
        for (int j = 0; j < stores.Count(); j++)
        { //כדי למנוע כפילויות - נדלג על החנות הנוכחית
            if (i == j)
                continue;
            //לאיפה אני מגיעה
            DTOStor to = stores[j];
            //נגדיר חנות נוכחית בתור התחלה
            Node start = new Node(from);
            //נגדיר חנות אחרת ברשימה בתור יעד
            Node end = new Node(to);
            //ניצור קשת - אבל - נאתחל את המרחק ב-0 כי עדיין לא ברור מה המרחק הכי קצר - צריך חישוב
            Route newRoute = new Route(start, end, 0);
            //חישוב המרחק הקצר ע''י דייקסטרה
            //השמת ערך 0 על הנקודה שבה אנחנו מתחילים
            mallGraphNodes[start.Store.NameStor].Value = 0;
            //בונה תור עדיפות למרחקים קצרים
            PrioQueue queue = new PrioQueue();
            //מוסיפה לתור את הצומת התחלה
            queue.AddNodeWithPriority(mallGraphNodes[start.Store.NameStor]);
            //מגדירה רשימת צמתים שלא בקרתי בהם
            List<Node> unvisited = new List<Node>();
            //מעבר בלולאה כל הצמתים בקומה
            foreach (var n in mallGraphNodes.Values)
            {//מוסיפה לרשימה את כל החניות בקומה
                unvisited.Add(n);
            }
            Node lastNode = null;
            //מקבלת  את המרחק הקצר שחוזר  - פר מתחילה הריקורסיה של הדיאקסטרה
            CheckNode(mallGraphRoutes, mallGraphNodes, queue, unvisited, end, ref lastNode);
            double shortestDistance = lastNode.Value;
            //אתחול משקל הקשת למרחק הקצר שחזר
            newRoute.Distance = shortestDistance;
            //הוספה לרשימת הקשתות של הגרף החדש  את הקשת שחשבנו
            selectedStoresGraphRoutes.Add(newRoute);
        }
    }
}
```

פונקציית עזר שמחזירה את הצומת האחרונה לצורך שחזור המסלול קיים שימוש ב CheckNodeשימוש שני :

```
// המציאת המשלול הקצר והחזרת הצומת האחרונה במסלול - לצורך שחזור המסלול
1 reference | liel, 3 hours ago | 2 authors, 3 changes
public static Node createShortestPathForSelectedStores()
{
    Node lastNode = null;
    //נגדיר את הכניסה כנקודת המוצא
    Node start = selectedStoresGraphNodes["כניסה"];
    //אין יעד - הדאקסטרה צריך ליצור מסלול בין כל הצמתים
    Node end = null;
    //חישוב המרחק הקצר ע''י דייקסטרה
    //השמת ערך 0 על הכניסה
    selectedStoresGraphNodes[start.Store.NameStor].Value = 0;
    //בונה תור עדיפות למרחקים קצרים
    PrioQueue queue = new PrioQueue();
    //מוסיפה לתור את הצומת התחלה
    queue.AddNodeWithPriority(selectedStoresGraphNodes[start.Store.NameStor]);
    //מגדירה רשימת צמתים שלא בקרתי בהם
    List<Node> unvisited = new List<Node>();
    //מעבר בלולאה כל הצמתים בקומה
    foreach (var n in selectedStoresGraphNodes.Values)
    {//מוסיפה לרשימה את כל החניות בקומה
        unvisited.Add(n);
    }
    CheckNode(selectedStoresGraphRoutes, selectedStoresGraphNodes, queue, unvisited, end, ref lastNode);

    return lastNode;
}
```

פונקציית השחזור :

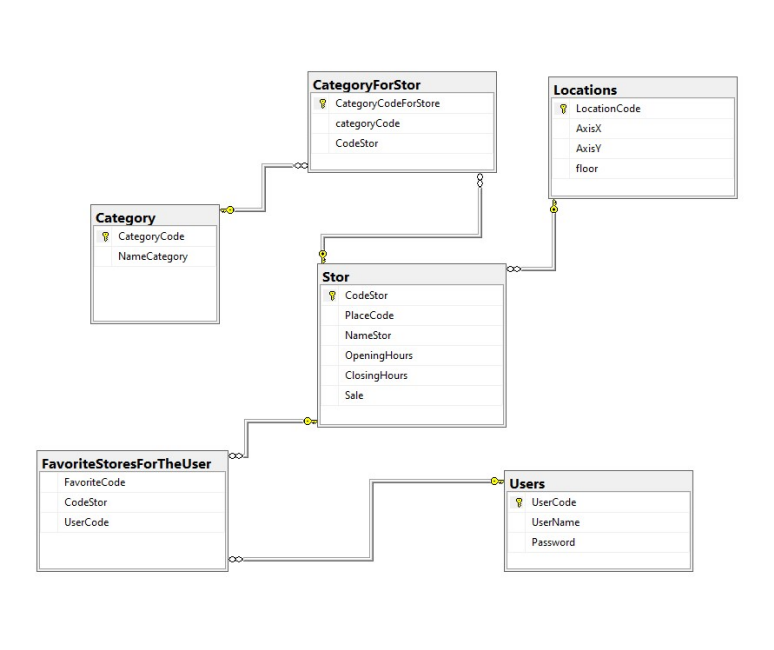
```
//שחזור בעצמו
1 reference | 0 changes | 0 authors, 0 changes
public static List<Node> FindNodesOfShortestPath(Node end)
{
    List<Node> nodes = new List<Node>();
    Node current = end;
    Node prev = end.PreviousNode;
    while (end.PreviousNode != null)
    {
        Node mallCurrent = end;
        while (true)
        {
            if (mallCurrent == prev)
                break;
            nodes.Add(mallCurrent);
            mallCurrent = mallCurrent.PreviousNode;

        }
        current = prev;
        prev = current.PreviousNode;
    }
    return nodes;
}
```

:CheckNode

```
        //מחזירה את המרחק הקצר ביותר//
        3 references | 0 changes | 0 authors, 0 changes
        private static void CheckNode(List<Route> routes, Dictionary<string, Node> nodes, PrioQueue queue, List<Node> unvisited, Node destinationNode, ref Node lastNode)
        {
            //תנאי עצירה בחיפוש מסלול קצר בין כל הנקודות בגרף//
            if (queue.Count == 0)
            {
                return;
            }
            //תנאי עצירה בחיפוש מסלול קצר בין מקור ליעד//
            if (queue.First.Value == destinationNode)
            {
                lastNode = queue.First.Value;
                return;
            }
            //רשימת קשתות של צומת
            List<Route> neighborRoutes = routes.Where(s => s.From == queue.First.Value).ToList();
            foreach (var r in neighborRoutes)
            {
                //אם קוד היעד לא נמצא ברשימה שצריך לבקר בה סימון שביקרו בו - נדלג עליו//
                if (!unvisited.Contains(r.To))
                {
                    continue;
                }
                //מעדכנת מה המרחק עד לצומת הנוכחית - מה היה המרחק עד הנעה לצומת הנוכחית//
                double travelDistance = nodes[queue.First.Value.Store.NameStor].Value + r.Distance;
                //בדיקה האם מה שחישבתי עכשיו יותר קצר ממה שקיים בצומת היעד עד עכשיו//
                if (travelDistance < nodes[r.To.Store.NameStor].Value)
                {//אם כן -תעדכן
                    nodes[r.To.Store.NameStor].Value = travelDistance;
                    //הקודם//
                    nodes[r.To.Store.NameStor].PreviousNode = r.From;
                }

                if (!queue.HasLetter(r.To))
                {
                    queue.AddNodeWithPriority(r.To);
                }
            }
            unvisited.Remove(queue.First.Value);
            lastNode = queue.First.Value;
            queue.RemoveFirst();
            CheckNode(routes, nodes, queue, unvisited, destinationNode, ref lastNode);
            return;
        }
    }
}
```

תיאור מסד הנתונים

תצלמי את הדיאגרמה מה-SQL



טבלאות :)

טבלת קטגוריות :)

משמשת לתיאור הקטגוריות

מפתח	שם שדה	תיאור	טיפוס	שדה שאינו חובה
Pk ראשי	CategoryCode	קוד קטגוריה	long	
	Name Category	שם קטגוריה	String	

טבלה זו מתארת את הקטגוריות במערכת ,

טבלת קטגוריות לחנות :)

טיפוס	תיאור	שם שדה	מפתח
long	קוד קטגוריה לחנות	CategoryCodeForStore	Pk ראשי
long	קוד קטגוריה	category Code	
long	קוד חנות	CodeStor	

טבלה זו היא טבלת הקשר בין קטגוריות לחניות ,

טבלת חניות מועדפות :)

טיפוס	תיאור	שם שדה	מפתח
long	קוד מועדף	FavoriteCode	Pk ראשי
long	קוד חנות	CodeStor	
long	קוד משתמש	UserCode	

טבלת זו מארת לכל משתמש איזה חניות אוהבות

טבלת מיקומים :)

טיפוס	תיאור	שם שדה	מפתח
long	קוד מיקום	LocationCode	Pk ראשי
double	ערך X	AxisX	
double	ערך Y	AxisY	
int	מספר קומה	floor	

טבלה זו מתארת את המיקומים של כל החניות במערכת

טבלת חנות :)

טיפוס	תיאור	שם שדה	מפתח
long	קוד חנות	CodeStor	Pk ראשי
long	קוד מיקום	PlaceCode	
string	שם חנות	NameStor	
bool	סטטוס מבצעים	Sale	

טבלה זו מתארת את החניות במערכת -קיים בה קום מיקום אשר מפט את מיקומה PlaceCode

טבלת משתמשים :)

טיפוס	תיאור	שם שדה	מפתח

long	קוד משתמש	UserCode	Pk ראשי
string	שם משתמש	UserName	
string	סיסמא	Password	

טבלה זו מתארת את המשתמשים במערכת ,

מדריך למשתמש :

התחילה קיים מסך הבית , ומלמעלה קיים ניווט ובו שלל אופציות :

#רישום לאפליקציה על ידי הזנת שם משתמש וסיסמא.

#חיבור על ידי אימות שם משתמש וסיסמא .

#צפייה בכל החניות בקניון.

בחירת מסלול :

מסלול לחנות בודדת קיימת גם למשתמש חדש (שלא מחובר).

מסלול לפי קטגוריות המשתמש מזין רשימת קטגוריות שבהם הוא מעונין .(רק למחובר)

מסלול לפי חנויות המשתמש מזין רשימת חניות שבהם הוא מעונין.(רק למחובר)

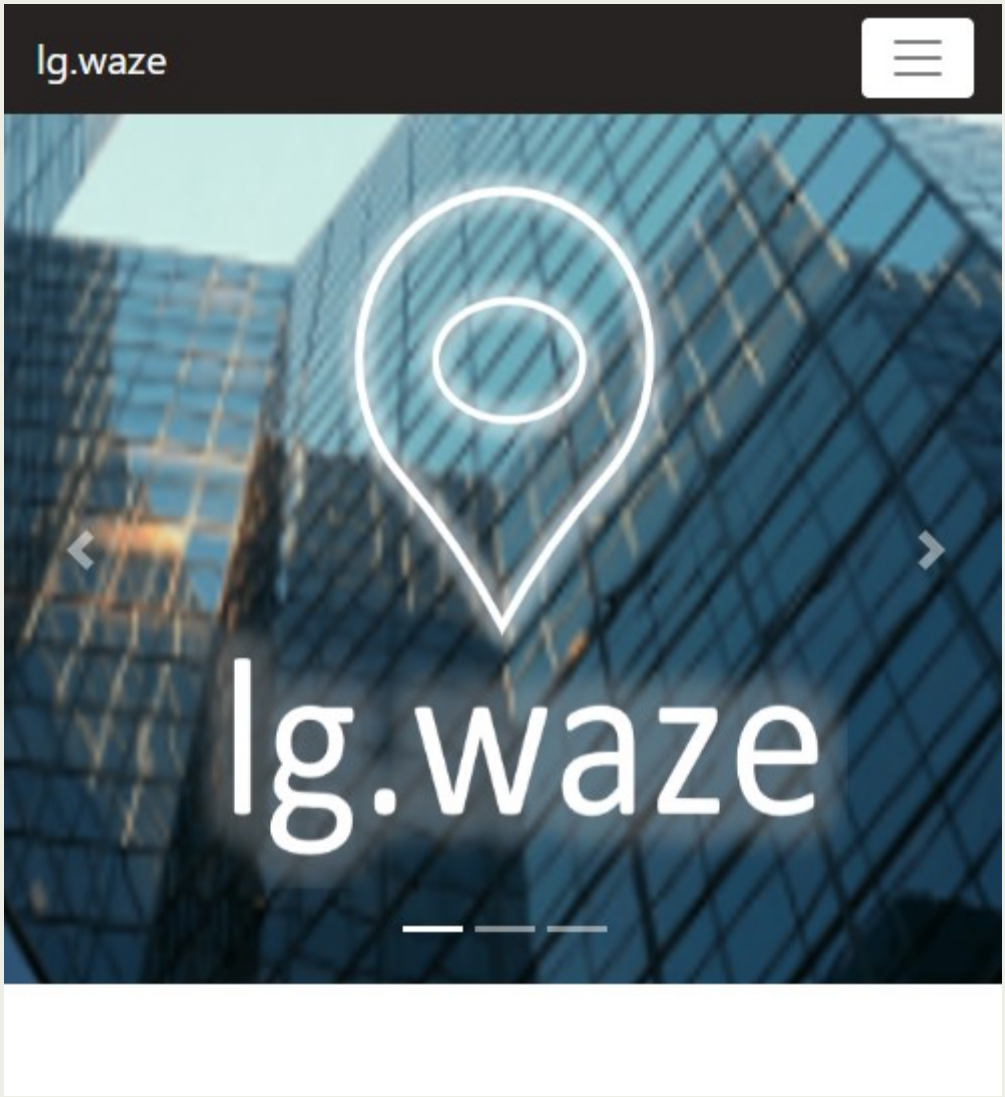
לוחצים אישור ומוצג המסלול לפי הבחירה

תיאור מסכים :

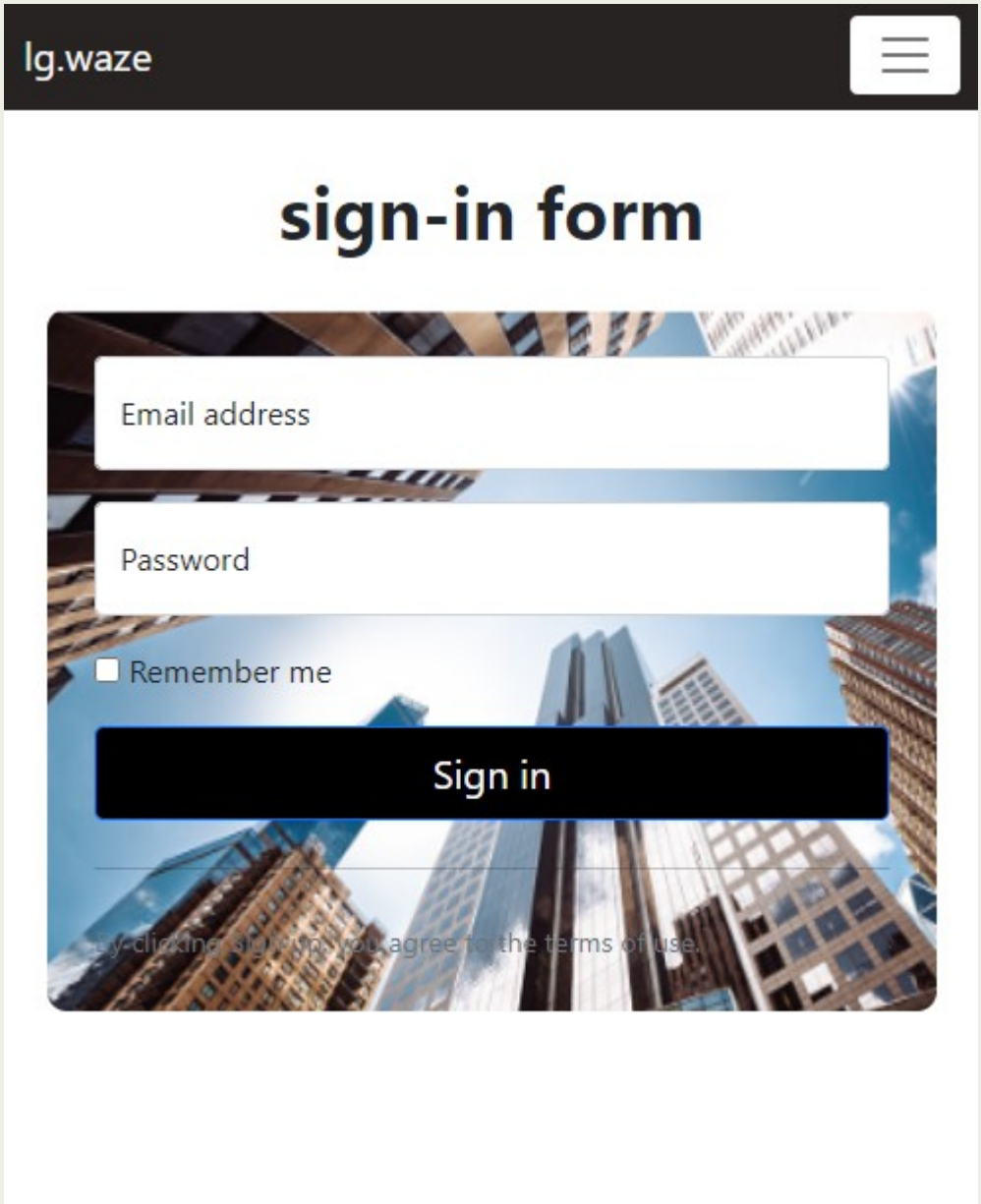
תרשים זרימה של כל המסכים :

<u>צילומי מסכים :</u>

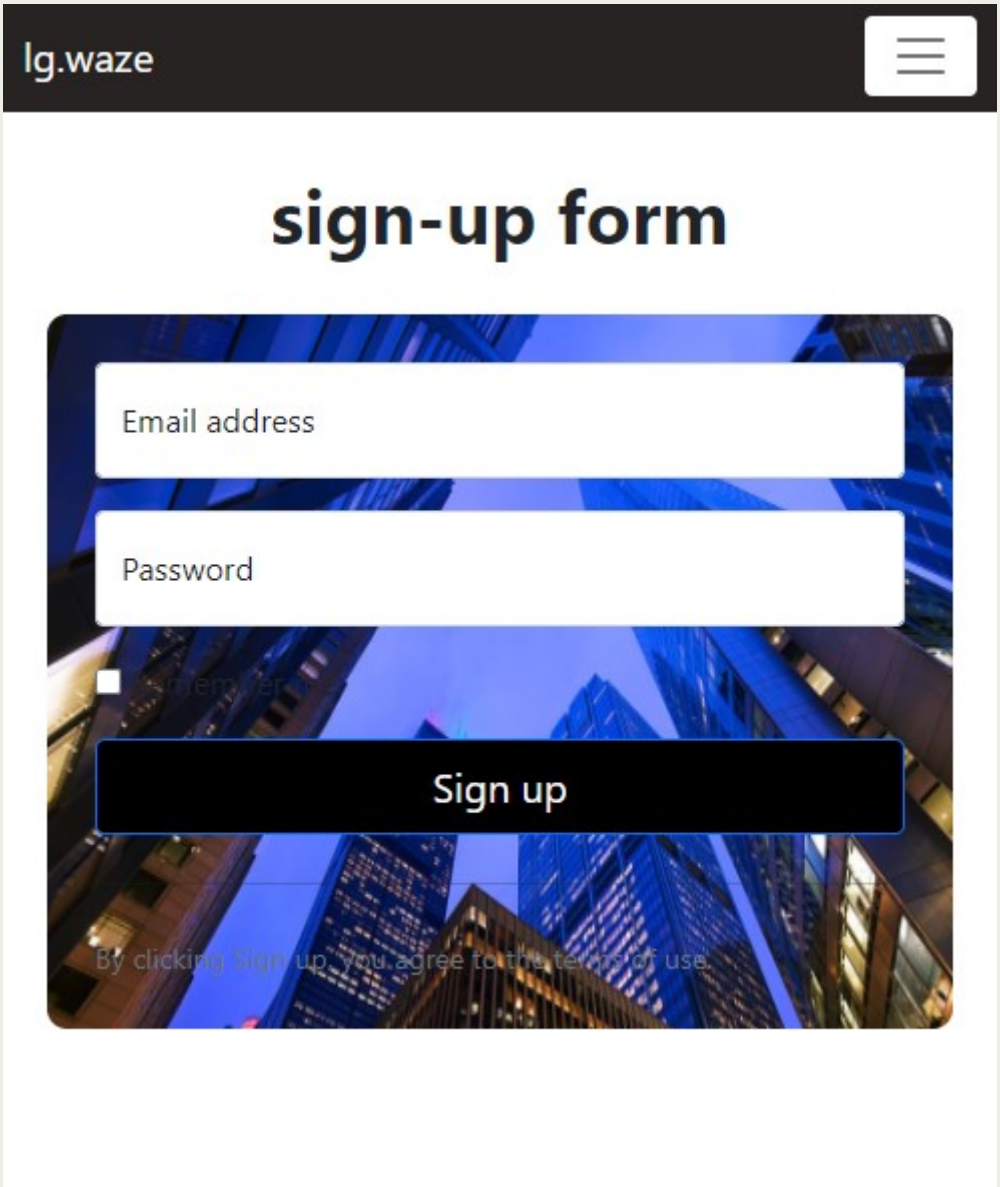
מסך הבית :

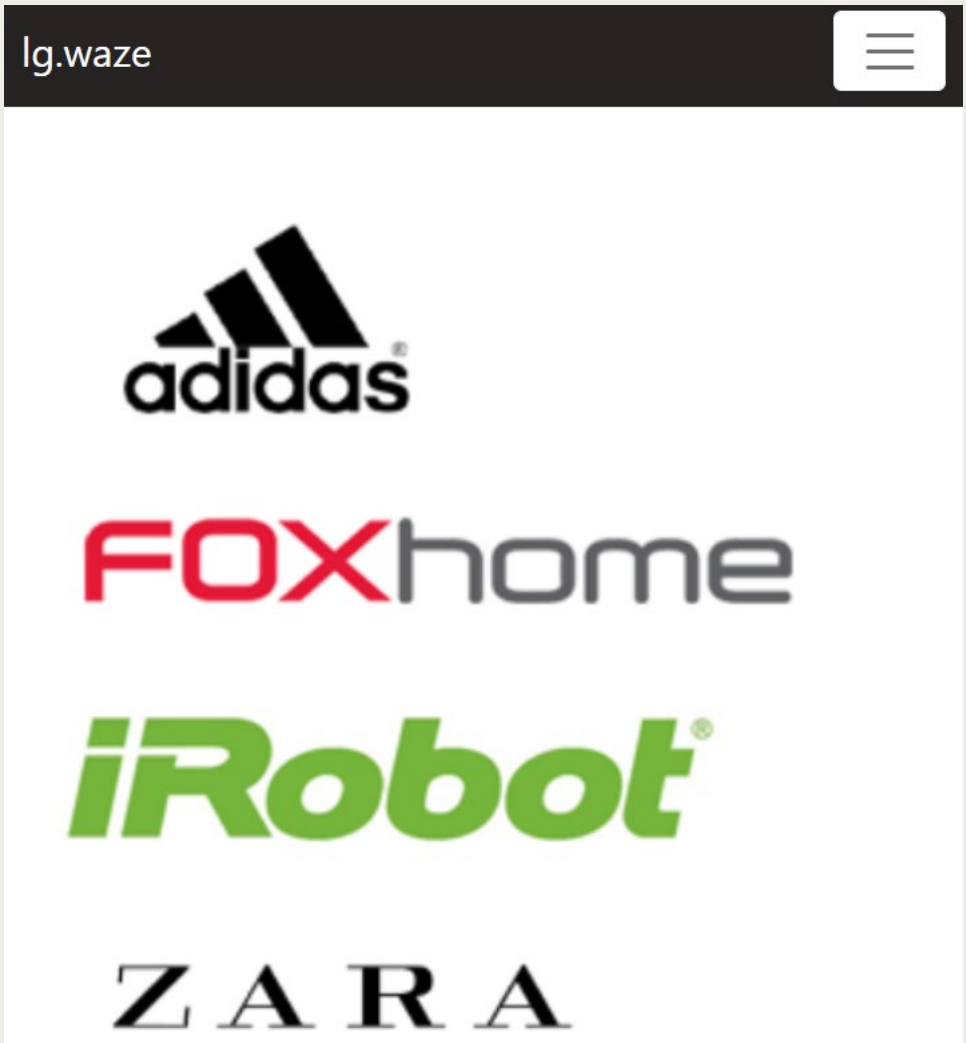


מסך התחברות :

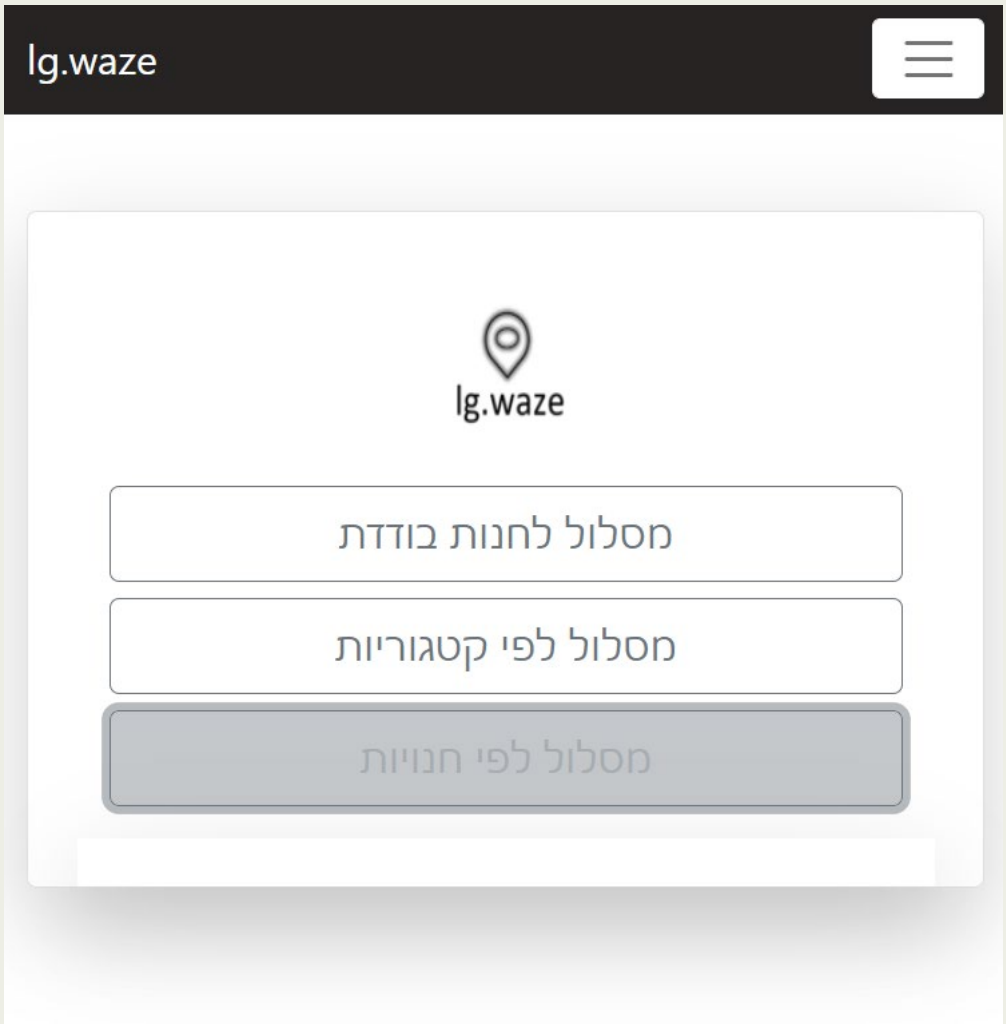

lg.waze
sign-in form
Email address
Password
Remember me
Sign in

מסך רישום :

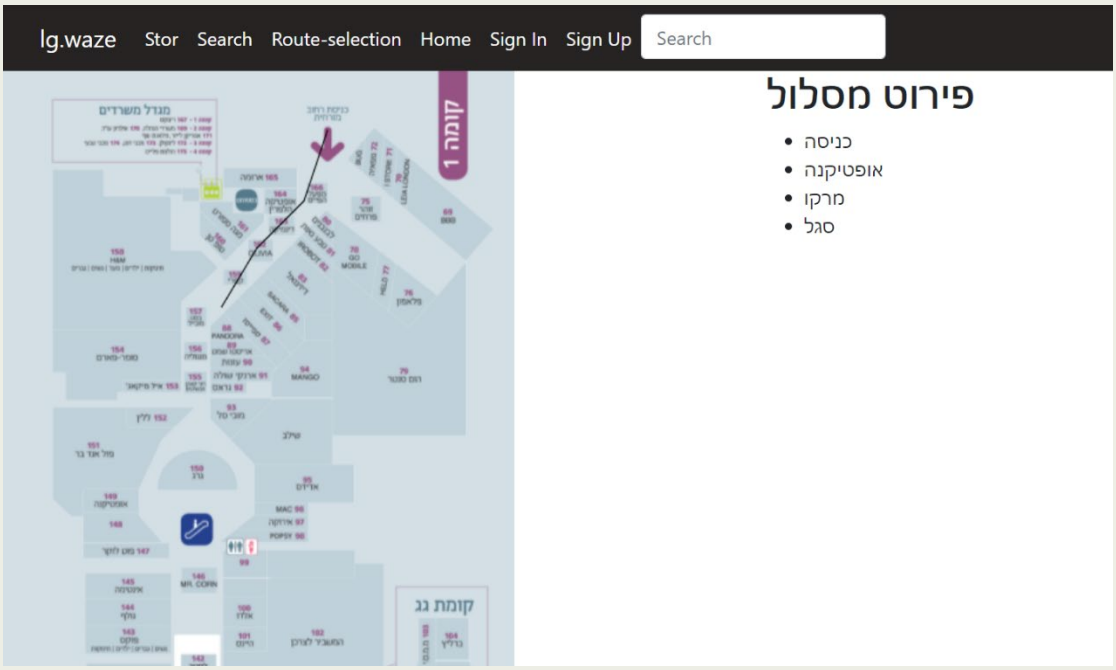

lg.waze
sign-up form
Email address
Password
Remember me
Sign up
By clicking Sign up, you agree to the terms of use.


lg.waze
adidas
FOXhome
iRobot
ZARA

מסך הבחירה של המסלול :



אחרי שמזין רשימת חניות הוא מקבל את המסלול :



בדיקות והערכה :

לאחר הרצת האלגוריתם נבחנו כל האילוצים שדרושים כדי להביא למציאת המסלול הקצר ביותר שקיים –וכן האילוצים הם כרגע רק בקומה הראשונה של הקניון יהיה המסלול ,ובהנחה שהמשתמש מפעיל אותה (את האפליקציה) בכניסה לקומה ,ובנוסף לאחר הרצת האלגוריתם ופתירת בעיותיו האלגוריתם הגיע למצב הקרוב ביותר בו יוכל לתת את המסלול הטוב ביותר

ניתוח יעילות :

יעילות – פירושה זמן הריצה כמה זמן ייקח מאז שהמשתמש שולח בקשה ועד התגובה ,

באלגוריתם שלי הסיבוכיות היא גבהה–אך היא יותר טובה מסיבוכיות של הסוכן הנוסע שהיא n! ויותר מזה קיים תור (שממשתי אותו במחלקה בפני עצמה) שמשמש את האלגוריתם במעבר על הצמתים כך שהתור מסודר שהצומת הראשון בתור הערך ה–VALUE שלו הכי קצר והאחרון התור הוא הכי רחוק – ולכן התור הזה חוסך לי מעבר מיותר כי פשוט אפשר לשלוף ,

ולכן במקרה הגרוע ביותר זמן הריצה יהיה :O(n^2)*O(v+e)

אבטחת מידע :

בתחילת האפליקציה קיימת אפשרות לכניסה באמצעות סיסמא ושם משתמש גם למשתמש חדש וגם לחבר. וברגע שהמערכת מזהה שהמשתמש אינו רשום היא אינה מאפשרת לו להיכנס

מסקנות :

המסקנה העיקרית שלי היא–שלכל בעיה קיים פתרון רק צריך להגיע אליו. כמו החישוב עצמו של האלגוריתם שלי כי הדייאקסטרה נתן לי שני אופציות ושניהם לא היו טובות ומה שאני צריכה אבל שילוב של שניהם נתן לי לבסוף את הפתרון המושלם

פיתוח עתידי :

אם היה לי יותר זמן . . .

הייתי מוסיפה מיקום נוכחי(מזהה איפה הבנאדם ברגע זה ומפה מתחיל מסלול ומתעדכן כל פעם) ,ובנוסף הייתי מגדילה את מספר הקומות ל-3 , והייתי מגדילה לא רק לקניון רעננה ברננים ספציפי אלא כל הקניונים בישראל .וגם אופציה לתחזק את האפליקציה והתקנת גרסאות חדשות כולל טיימר כמה זמן לוקח לך להגיע עד ליעד\ים הרצוי (לסוף המסלול).

ביבליוגרפיה :

אתרים

GitHub- https://github.com

StackOverflow–/https://stackoverflow.com

Bootstraps–/https://getbootstrap.com/docs/5.0/getting-started/introduction

הסברים

#

הסבר על דייאקסטרה:
https://www.hamichlol.org.il/%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9D_%D7%93%D7%99%D7%99%D7%A7%D7%A1%D7%98%D7%A8%D7%94

#
https://www.hamichlol.org.il/%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9D_%D7%93%D7%99%D7%99%D7%A7%D7%A1%D7%98%D7%A8%D7%94

חומר עזר בפיתוח אפליקציה :

https://globalbit.co.il/%D7%90%D7%99%D7%9A-%D7%9C%D7%A4%D7%AA%D7%97-%D7%90%D7%A4%D7%9C%D7%99%D7%A7%D7%A6%D7%99%D7%94-/%D7%9E%D7%A6%D7%9C%D7%99%D7%97%D7%94

הסברים נוספים וסרטונים בנושא :

https://www.bekaloot.co.il/%D7%9E%D7%93%D7%A8%D7%99%D7%9A-1203-%D7%90%D7%99%D7%9A%20%D7%9E%D7%91%D7%A6%D7%A2%D7%99%D7%9D%20%D7%90%D7%AA%20%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9D%20%D7%93%D7%99%D7%99%D7%A7%D7%A1%D7% 98%D7%A8%D7%90%20(Dijkstra).aspx

https://www.youtube.com/watch?v=VY3ZpH3aYxY

https://www.youtube.com/watch?v=_rfqURrpg9o

https://www.youtube.com/watch?v=eJC3Ql2e3CU

https://www.ai-blog.co.il/2019/04/04/%d7%9c%d7%9e%d7%99%d7%93%d7%94-%d7%a2%d7%9e% d7%95%d7%a7%d7%94-%d7%a2%d7%9c-%d7%92%d7%a8%d7%a4%d7%99%d7%9d/