פרויקט גמר להנדסאים

הנדסת תוכנה

2023

שם | מיכל דינר

ת.ז. | 214213639

שם המנחה | חנה סירוקה

תוכן העניינים

1. הצעת פרויקט

הצעת נושא לפרויקט במגמת הנדסת תכנה

סמל מוסד: 590265

שם המכללה: סמינר שערי דעת

שם הסטודנט: מיכל דינר

הנושא: ניהול יעיל של חברת שליחויות בארץ

שם הפרויקט: Dolivelivery

הסבר כללי

הפרויקט מיועד עבור חברת שליחויות המעסיקה שליחים בכל רחבי הארץ.

המערכת תעביר כל חבילה על ידי שליח אחד או יותר, באופן החסכוני ביותר מבחינת זמן נסיעה של העובדים, ותוך התחשבות באילוצי הזמן של המשלוח.

הגדרת הבעיה האלגוריתמית:

על האלגוריתם להכריע מי הם השליחים שישתתפו בהעברת החבילה אל היעד, ומה יהיה מסלולם בהתאם לכל החבילות אותן הם צריכים להעביר.

באלה נקודות איסוף להניח את החבילות, ואלו לקחת את החבילות הלאה.

רקע תאורטי בתחום הפרויקט:

הפרויקט מתעסק בהעברת חבילות ע''י שליחים המועסקים ע"י חברת שליחויות.

מרבית חברות השליחויות מעבירות כל משלוח על ידי שליח יחיד. מה שיוצר מצב של מרחקים רבים אותם צריכים השליחים לעבור. מטרת הפרויקט היא לאפשר לכל שליח להישאר ברדיוס מסוים ולנוע רק בו, תוך העברת המשלוחים מעובד לעובד בנקודות ההשקה בין האזורים.

המערכת תאפשר את התהליכים הבאים:

1. קבלת מידע על השליחים.
2. קבלת הזמנות מאנשים שמעוניינים לשלוח חבילות.
3. החלטה אלגוריתמית איזה שליחים יעבירו את החבילה בכל שלב עד להגעת החבילה ליעדה.

מטרת הפרויקט

ניווט וניתוב חכם של שליחים, תוך חיסכון בזמן העבודה והנסיעה של השליחים, ושמירה על אילוצי הזמן של החבילות.

ראשי פרקים:

1. קבלת מידע על השליחים
2. קבלת הזמנות מאנשים המעוניינים לשלוח חבילה ע''י חברת השליחויות.
3. בדיקת אילו שליחים מתאימים להעברת החבילה.
4. בחינת מיקום ומרחק השליחים ממקום היעד של החבילה
5. הכרעה והצגה הודעה לשליחים שדרכם המערכת החליטה להעביר את החבילה.

תיאור הטכנולוגיה:

ממשק המשתמש ייבנה בעזרת טכנולוגית react-

המערכת תיעזר בטכנולוגית המיקומים והמרחקים הניתנת ע"י google maps

בשביל לקבוע את המרחקים בין השליחים ליעדי החבילות.

צד שרת:

צד השרת יחולק לשכבות:

שכבת ה-DAL

שכבת ה-BLL

שכבת ה-WEB API

צד השרת ייכתב בשפת #C וייעזר בטכנולוגית WEB API

צד לקוח:

צד הלקוח, ה-UI , יבנה כממשק נח ויעיל למשתמש וישים דגש על:

1. חווית משתמש גבוהה.
2. אבטחת מידע.
3. הימנעות מקריאות שרת מיותרות ובניה נכונה של האתר עצמו.

צד הלקוח יפותח ב- React.

מסד נתונים:

מסד הנתונים יבנה בשפת SQL בסביבת ה-SQL SERVER.

מפרט טכני:

עמדת פיתוח: מחשב Lenovo

RAM 8 GB

Core i5

מערכת הפעלה:Windows 10

שפת תכנות: c#

עמדת משתמש מינימאלית:

חומרה: Core i5 or higher

מערכת הפעלה: Windows 10

תוכנות: visual studio

חיבור לאינטרנט: נדרש

דפדפן: google chrome

ספריות:

- התחברות ל- google maps

לוח זמנים לביצוע הפרויקט לפי שלבים :

חקר מצב קים- ספטמבר

הגדרת הדרישות- ספטמבר

אפיון המערכת-אוקטובר

אפיון בסיס הנתונים- נובמבר

עיצוב המערכת- דצמבר

בנית התכנה- ינואר

בדיקות- פברואר

הכנת תיק פרויקט- מרץ

הטמעת המערכת- אפריל

הגשת פרויקט סופי- מאי

חתימת הסטודנט : מ.דינר

אישור חתימת המנחה : _____

חתימת רכז המגמה: _____

אישור הועדה : _____

2. מבוא

2.1 הרקע לפרויקט:

הפרויקט מנהל ניהול שיבוץ וניווט חבילות לעובדים בחברת משלוחים, השיבוץ והניווט בהתאמה לדרישות החברה ובהתאם למיקומי הלקוחות.

החברה ברצוני לנהל הינה חברה המעסיקה עובדים. כאשר לקוח מעוניין בשירות החברה ולשלוח חבילה דרך החברה, הוא שולח בקשה לחברה, החברה צריכה להעביר את החבילה ליעדה במהירות האפשרית, בכל יום החברה מעבירה או מקדמת מספר רב של חבילות ככל האפשר.

החברה לא מעבירה את החבילות בצורה ישירה מבית הלקוח אל היעד, אלא לחברה יש נקודות איסוף הפזורות ברחבי הארץ, החבילות עוברות דרך מספר שליחים ונקודות איסוף לפי הצורך, בצורה שכל שליח יעבוד באזור מסוים, הנקבע דינאמי.

הפעלת האלגוריתם היא על ידי מנהל המערכת בשעות הערב, הלקוחות שנבחרו שאליהם יבוא השליח יקבלו הודעה על זמן ההגעה המשוער על ידי גורם חיצוני.

המערכת משתמשת בשרות של מפות גוגל על מנת לדעת את הזמן נסיעה בין השליחים, בתי הלקוח, נקודות האיסוף והיעדים, וכן כדי לחשב את המיקום של כתובות השליחים, נקודות האיסוף וכדומה, נתונים אלו משתקללים באלגוריתם.

2.2 תהליך המחקר:

פעמים רבות נתקלתי בצורך להשתמש בחברת שליחויות, וכשחיפשתי רעיון לפרויקט חשבתי שאפשר לשפר את ביצועי חברות שליחויות, במקום שכל עובד ייסע ממקום למקום ללא סדר ויבזבז זמן רב על נסיעות מיותרות, יותר הגיוני שכל עובד יעביר חבילות רק באזור שלו, ולא יבזבז זמן על נסיעות למקומות רחוקים, כך חבילה אחת תעבוד דרך כמה שליחים עד שתגיע ליעדה, וכך כל שליח יעסוק יותר בהעברת החבילות מאשר בנסיעות.

כך אחרי שראיתי שאפשר לשפר את ההתנהלות של חברות שליחויות יצאתי לדרך ליצור אלגוריתם שינהל חברת שליחויות בצורה יעילה יותר.

2.3 סקירת ספריות:

חלק מהאתרים שהשתמשתי לצורך הפרויקט:

https://en.wikipedia.org/
https://www.geeksforgeeks.org

2.4 אתגרים מרכזיים:

2.4.1 הבעיה איתה התמודדתי:

הבעיה העיקרית איתה התמודדתי בכתיבת הפרויקט הייתה למצוא אלגוריתם יעיל אשר ימצא את הדרך שתקדם כמה שיותר חבילות.

כמו כן, לאחר שמצאתי את האלגוריתם המתאים, נדרשתי לבנות אופציה אפשרית לפתרון, האופציה הייתה צריכה להיות הגיונית, כדי שלא ייגרם מצב שעובד ייקח או לא ייקח חבילה שהוא יכול לקחת, רק בגלל שהאלגוריתם לא ידע עדיין שחבילה מסוימת נמצאת במקום מסוים.

דבר נוסף, רציתי לשמור את הנקודות איסוף כגרף, כאשר המשקלים בגרף -הם המרחק והזמן בין הנקודות, הייתי צריכה לבנות גרף נוח וקל לשימוש.

2.4.2 הסיבות לבחירת הנושא:

הסיבה המרכזית שבגללה בחרתי בנושא זה הייתה ההיחשפות שלי לחברת שליחויות, ובזמן הגדול שלוקח לחבילות להגיע לידען, חשבתי על הזמן המבוזבז ששליחים מעבירים בדרכים, וחשבתי איך אפשר לצמצם את הזמן של העובדים בדרכים, ואיך לחלק את העבודה בצוות בצורה שהעובדים יעברו בדרכים בצורה שתעביר את מירב החבילות.

2.4.3 על איזה צורך הפרויקט עונה:

הפרויקט עונה על הצורך הקיים בחיי היום יום לחברות שליחויות שרוצות לשפר את הביצועים והמהירות של הגעת החבילות ליעדן.

2.4.4 דרכי פתרון שנבדקו:

כדי לפתח מערכת זו חשבתי רבות כיצד לממש את האלגוריתם אשר בסופו של יום יקדם כמה שיותר חבילות ליעדן. ראשית ללא ספק ידעתי שנדרש שימוש במבנה נתונים של גרף כיוון שיש לי נקודות איסוף וקשתות ביניהם, המסמלות את המרחק בזמן וק"מ בין הנקודות.

חשבתי רבות כיצד לממש את האלגוריתם בצורה שתקדם כמה שיותר חבילות בזמן הקצר ביותר.

בתחילה חשבתי לעשות אזורים סטטיים לכל עובד, שנקבעים מראש, אך ראיתי שקשה לקבוע אזורים בצורה שתהיה טובה ויעילה, קרובה לעובדים, ומביאה תוצאות טובות לאלגוריתם.

לכן החלטתי לקבוע אזורים דינאמיים, כלומר כל עובד האזור שלו הוא האזור שנמצא ברדיוס של מספר קילומטרים ממנו.

דרך נוספת שנפסלה, היה האלגוריתם דאקסטרה, בתחילה חשבתי לבדוק לכל חבילה מה המסלול האופטימלי בשבילה ע"י הרצת האלגוריתם דאקסטרה, ולהשתמש במסלול זה בלבד, אבל נוכחתי לראות שכיוון שהשליח עובר בנקודות באזור שלו בצורה אקראית, יכול להיווצר מצב שעדיף לחבילה לסטות ממסלול דייקסטרה שלה למסלול אחר, כיוון שהוא יתבצע מידי ע"י עובדים, ואילו מסלול

הדאקסטרה יהיו טוב רק במקרה שיבצע מייד כולו, ואילו באלגוריתם יכול להיות שחבילה תחכה בנקודות איסוף זמן מה עד ששליח יעביר אותה הלאה, כלומר המרחקים בגרף לא רלוונטיים במקרה שחבילה צריכה לחכות בנקודות איסוף.

לכן החלטתי שחבילה תילקח ע"י שליח במקרה שהשליח מתכנן לבוא באותו יום לנקודה שתקדם את החבילה ליעדה.

3. יעדים ומטרות

מטרות:

- לשבץ חבילות לעובדים כך שבסך הכל, כל יום יועברו או יקודמו כמה שיותר חבילות ליעדן.
- לקבוע חלוקת אזורים לשליח באופן דינאמי כדי להתקרב לפתרון האופטימאלי.

יעדים:

- רישום והזמנה למערכת
- בנית מפה בצורה יעילה (לפי האזורים הדינאמיים)
- להעביר חבילה ליעדה במהירות האפשרית

4. אתגרים

אתגרים רבים היו לי במהלך כתיבת הפרויקט, כמו התמצאות באתרים שונים, יעילות בחיפוש מידע, וכן בכתיבת קוד ואלגוריתם בצורה יעילה.
אתגר עיקרי היה למצוא אלגוריתם שיפתור את הבעיה ותיתן מענה לבעיה בצורה יעילה, וכן לבנות את הגרף בצורה יעילה כדי לדעת באיזה מסלול תעבור החבילה.

5. מדדי הצלחה

• אופטימליות- עד כמה האלגוריתם ייתן את סדר יום העבודה היעיל ביותר, כלומר אלגוריתם שייתן מסלול יומי לכל עובד, שייתן את תפוקת העבודה הגבוה ביותר.

• מהירות- איזה סדר גודל של זמני ריצה וזמן תגובה ראשוני למשתמש.

6. רקע תיאורטי

תעשיית השליחויות היא חלק מהותי מהכלכלה העולמית, ומספקת שירות חיוני לעסקים ולפרטיים כאחד. עם צמיחת המסחר האלקטרוני והקניות באינטרנט, הביקוש לשירותי שליחויות גדל משמעותית בשנים האחרונות.

בתגובה לדרישה זו, חברות שליחויות נאלצו להסתגל ולחדש כדי להישאר תחרותיות .אחד האתגרים העומדים בפני חברות השליחויות הוא אופטימיזציה של תהליך המשלוח כדי להבטיח אספקת חבילות במהירות וביעילות.

בפרויקט זה אנו שואפים לפתח מערכת שתשפר את תהליך המשלוח לחברת שליחויות עם נקודות איסוף בכל הארץ .המערכת תתוכנן כך שלכל עובד יהיה רדיוס משלוחים משלו והוא מספק רק

חבילות באזור המיועד לו. עבור חבילות שצריכות לנסוע למרחקים ארוכים, הן יועברו בין נקודות איסוף על ידי מספר עובדים.

המטרה היא למזער את זמן האספקה ולהבטיח משלוח חבילות ליעד הנכון .כדי להשיג מטרה זו, נשתמש באלגוריתם אקראי המגריל באופן אקראי אפשרויות אפשריות עבור כל עובד באזורו. עבור כל נקודת איסוף שעובר יגיע אליה, האלגוריתם יקבע אילו חבילות לקחת ואילו להוריד בכל נקודת איסוף. האלגוריתם יפיק אפשרויות רבות ויעניק ניקוד לכל אפשרות, תוך בחירת האפשרות הטובה ביותר כדי להבטיח את זמן האספקה המהיר ביותר .

על ידי שימוש באלגוריתם זה, אנו שואפים לשפר את היעילות והדיוק של תהליך המשלוח עבור חברת השליחויות.

לסיכום, המערכת המוצעת תיתן מענה לאתגרים העומדים בפני חברות השליחויות בייעול תהליך המשלוח. האלגוריתם האקראי יאפשר לחברה לספק חבילות במהירות וביעילות, להבטיח את שביעות רצון הלקוחות ושמירה על יתרון תחרותי בענף השליחויות.

7. תיאור מצב קיים

תעשיית השליחויות היא תחרותית ביותר, עם חברות מבוססות רבות המציעות מגוון שירותים ללקוחות. אמנם ישנן חברות רבות המציעות שירותי שליחויות, אך רובם לא פועלות בשיטה של החברה שלי.

עם זאת, יש כמה חברות בעיקר בינלאומיות שיש להן מודלים דומים לתפעול. לדוגמה, ל DHL-Express יש רשת של נקודות שירות ברחבי העולם שבהן לקוחות יכולים להוריד ולאסוף את החבילות שלהם. דוגמה נוספת היא UPS שיש לה רשת של נקודות גישה שבהן לקוחות יכולים להוריד ולאסוף את החבילות שלהם.

אך הסיבה שלחברות אלו יש נקודות שירות ברחבי העולם שונה משלי, החברות הללו הן בינלאומיות ואין דרך טכנית הגיונית ששליח אחד יעשה את הדרך, וגם הם פועלים בצורה שעובדים בכל ארץ מביאים את החבילות לנקודה מסוימת באותה ארץ ומשם ישנם אחראיים להעביר את החבילות בין הארצות, וכן להיפך, אך החברות הללו לא מעבירים חבילה מסוימת על ידי מספר שליחים באותה ארץ.

בסך הכל, בעוד שיש כמה חברות שפועלות באופן דומה לחברה המוצעת שלי, עדיין יש מקום לחדשנות ושיפור בתעשיית השליחויות. המערכת שלי מהווה גישה חדשה שיש לה פוטנציאל לשפר את היעילות והדיוק של תהליך האספקה, ויכולה לספק יתרון תחרותי בשוק.

8. ניתוח חלופות מערכתי

את גרף האופציות של מסלולים עבור חבילה בפרויקט ניתן היה לייצג ע"י שתי אפשרויות:

1. Adjacency matrix – מטריצת סמיכויות.
2. Adjacency list – רשימת סמיכויות – האפשרות שבה בחרתי.

פירוט והסבר:

Adjacency matrix – מטריצת סמיכויות:

בשיטה זו הגרף מיוצג כמטריצה בגודל |V| * |V| כאשר כל צומת מיוצג ע"י האינדקס של שורה ושל עמודה. הקשתות מיוצגות באמצעות התאים: תא (i,j) במטריצה, מכיל את הספרה '1' אם יש בגרף קשת מצומת i לצומת j, אחרת '0'.

במטריצת סמיכויות כל זוג צמתים מיוצג ע"י סיבית בודדת במטריצה, בין אם קיימת קשת ביניהם ובין אם הקשת אינה קיימת, לכן, המקום שנדרש לייצוג כזה הוא כריבוע של מספר הצמתים, כלומר O(V^2) סיביות.

החיסרון בייצוג גרף בצורה כזו – ע"י מטריצה, היא שסיבוכיות הזמן של האלגוריתמים הפועלים על גרף המיוצג כך גבוהים יותר מייצוגים אחרים, לכן לא בחרתי לייצג את הגרף ע"י מטריצת סמיכויות.

גם בבחירת האלגוריתם היו אפשרויות רבות כגון:

1. האלגוריתם הראשון שעולה בראש הוא לחשב את כל האפשרויות האפשריות עבור כל עובד, מסלול וחבילה, ולאחר מכן לבחור את האפשרות הטובה ביותר. עם זאת, לגישה זו תהיה מורכבות חישובית גבוהה במיוחד, מה שהופך אותה לבלתי מעשית ליישום בעולם האמיתי. עם מספר רב של עובדים, נקודות איסוף וחבילות, מספר האפשרויות האפשריות הופך במהירות לגדול מדי עבור כל מחשב לטיפול בפרק זמן סביר. לכן, גישה זו אינה מעשית לבעיה הספציפית שלי וסביר להניח שתוביל לעיכובים וחוסר יעילות במשלוח החבילות.
2. אלגוריתם חמדן:
אלגוריתם זה כולל בחירת החבילה הקרובה ביותר עבור כל עובד למסירה, מבלי להתחשב במסלול הכולל. אמנם גישה זו יכולה להיות מהירה ופשוטה, אך היא עלולה להוביל למסלולים לא אופטימליים וזמני אספקה ארוכים יותר .

בסופו של דבר, בחרתי באלגוריתם האקראי כי זה הפתרון היעיל והטוב ביותר לבעיה הספציפית שלי. האלגוריתם האקראי מייצר אפשרויות אפשריות רבות ומקצה ניקוד לכל אחד, בוחר את האפשרות הטובה ביותר עבור כל עובד. גישה זו מאפשרת משלוח חבילות יעיל ברחבי הארץ באמצעות רשת של נקודות איסוף ועובדים עם אזורי מסירה ספציפיים. סיבה נוספת לכך שהאלגוריתם האקראי הוא בחירה טובה יותר לבעיה הספציפית שלך היא כי יש לו מורכבות זמן ריצה נמוכה יותר מאלגוריתמים אחרים כמו אלגוריתמים המחשב את כל האפשרויות האפשריות. אשר יכול להוות חיסרון גדול כאשר מתמודדים עם מערכי נתונים גדולים או משלוחים רגישים לזמן. לעומת זאת, האלגוריתם האקראי הוא הרבה יותר מהיר ומעשי ליישום בעולם האמיתי. בנוסף, סביר להניח שהאלגוריתם האקראי יתקע באופטימיות מקומית פחות מאשר אלגוריתם החמדני, מה שהופך אותו לפתרון אמין יותר. בסך הכל, האלגוריתם האקראי היה הבחירה הטובה ביותר לבעיה הספציפית שלך מכיוון שהוא מאזן בין יעילות, אפקטיביות ומעשיות.

9. תיאור החלופה הנבחרת

ואכן, לאחר ראיית דרכי הפתרון האפשריים, בחרתי בסופו של דבר להשתמש לייצוג הגרף ברשימת סמיכויות – adjacency list, אך מימשתי זאת ע"י רשימה ולא ע"י מערך כדי שיהיה דינאמי, כאשר כל תא ברשימה מייצג צומת בגרף-נקודת איסוף.

קשתות הגרף מיוצגות באמצעות רשימה היוצאת מכל תא ברשימה של הקודקודים, ומייצגת את כל הקשתות היוצאות מקודקוד זה.

כל צומת ברשימת הקשתות מיצג קשת בגרף ומכיל מצביע לקודקוד היעד של קשת זו. הסיבה לשימוש במצביע לקודקוד היעד של קשת זו, ולא בשימוש מזהה של קודקוד היעד - כיוון שהקודקודים בגרף לא מאוחסנים בסדר שרירותי ברשימה, ורציתי למנוע סיבוכיות גבוהה כאשר עלי לגשת מקשת לקודקוד היעד של הקשת, ולכן השתמשתי במצביעים.

ייצוג הגרף בצורה זו של רשימת סמיכויות היא היעילה ביותר מבחינת סיבוכיות הזמן, ולכן בחרתי באפשרות זו.

בנוסף, מבין כל האלגוריתמים לפתרון, החלטתי להשתמש באלגוריתם האקראי, שהוא ייתן לי תוצאה שוודאי תהיה אפשרית, הסיבוכיות לא תהיה גבוה מידי, כיוון שהאלגוריתם של בניית האופציה אפשרית מורץ מספר מוגבל של פעמים, וכן כיוון שהאלגוריתם מגריל אפשרויות רבות, הפתרון הסופי יהווה קרוב טוב לפתרון הסופי.

10. אפיון המערכת

10.1 ניתוח דרישות המערכת

סביבת פיתוח:

- חומרה: מעבד i5 8GB RAM.
- עמדת פיתוח: מחשב Lenovo.
- מערכת הפעלה: Windows 10.
- שפות תכנה: C# תוך שימוש בטכנולוגית Web Api, React.
- כלי תכנות לפיתוח המערכת: Visual Studio 2022.
- מסד נתונים: SQL Server.

עמדת משתמש מינימלית:

- חומרה: מעבד i7 8GB RAM.
- מערכת הפעלה: Windows 7 ומעלה.
- חיבור לרשת: נדרש.
- תוכנות: Internet Explorer 7 and up, or chrome.

10.2 מודול המערכת

התחומים בהם המערכת עוסקת:

- הוספת לקוחות חדשים למערכת
- הוספת הזמנות של משלוחים
- שיבוץ וניווט חבילות לעובדים
- הצגת רשימת היעדים היומית לשליח

התחומים בהם המערכת לא עוסקת:

- שיבוץ החבילות ברכב ההובלה
- חלוקת משכורות לעובדים.

10.3 אפיון פונקציונלי:

המערכת בונה את לוח זמני השליחים ליום הקרוב.

10.4 ביצועים עיקריים:

האלגוריתם העיקרי של הפרויקט הוא אלגוריתם אקראי, בו מגרילים אופציות אפשריות למסלולים לעובדים, ובודקים מה האופציה בעלת הניקוד הגבוה יותר.

ביצועי האלגוריתם:

<u>אתחול ראשוני</u>:

בניית גרף שבו כל הקודקודים בגרף הם נקודות איסוף, ובתי המקור והיעד של החבילות, מחברים את הקודקודים בצורה שבה כל הנקודות שבאזור של אותו עובד (בתוך רדיוס שנקבע מראש) מחוברים ביניהם מכל קודקוד לכל קודקוד.

המשקלים של הקשתות הם המרחק בין 2 נקודות האיסוף בק"מ על פי נוסחת Haversine שמחשבת את המרחק בין 2 נקודות בכדור הארץ, לפי המיקומים שלהם.

נוסחת Haversine היא נוסחה מתמטית המחשבת את מרחק המעגל הגדול בין שתי נקודות על פני השטח של כדור, כגון כדור הארץ.

וכן זמן מרחק נסיעה בין 2 נקודות האיסוף, על פי נתונים ממפות גוגל.

נצטרך שני נתונים אלה עבור המשך האלגוריתם.

לאחר בניית הגרף, נוסיף לרשימת נקודות האיסוף את כל בתי המקור והיעד של החבילות שהוזמנו.

כמו כן, נחשב לכל עובד אלו נקודות נמצאות ברדיוס שלו, ורק בנקודות איסוף אלו, העובד יכול לבקר במהלך יום העבודה.

כעת הגרף מוכן, ועלינו לקבוע עבור כל עובד את סדר העבודה שלו באותו יום כך שמקסימום החבילות יקודמו במקסימום הקידום האפשרי.

<u>מהלך האלגוריתם האקראי</u>:

האלגוריתם מגריל אופציות אפשריות רבות, מחשב ניקוד לכל אפשרות, ובוחר את הטוב ביותר.

איך מגרילים אופציה אפשרית?

הגרלת הנקודות בהן יעבור כל עובד אינה מספיקה כיון שלא נוכל לדעת אלו חבילות ישהו בנקודת האיסוף באותה נקודת זמן שהעובד יגיע לשם, לכן לא מספיק להגריל לכל עובד את הנקודות שהוא יעבור בהם, אלא צריך לחשב אלו חבילות יהיו בנקודות האיסוף בכל זמן נתון על פי המסלול שהגרלנו עבור כל שאר העובדים.

כדי לפתור זאת, בתחילה הגרלנו לכל עובד נקודות איסוף מתוך הנקודות שבאזורו, שבהם הוא יבקר במהלך היום, כל נקודה לפני שהתווספה למסלול היומי של העובד, נבדקה אם נשאר מספיק זמן עד סוף יום העבודה של העובד, אחרת הוא לא יכול להגיע לנקודה זו.

לאחר שהוגרלו לכל העובדים הנקודות שיהוו את המסלול היומי שלהם, כעת צריך לקבוע לכל עובד, לכל נקודה ממסלולו, אלו חבילות להוריד בנקודה זו, ואלו חבילות לאסוף מנקודה זו.

חישוב זה חייב להתבצע בסדר של ציר הזמן, ולכן כדי לחשב זאת ''נזיז'' כביכול את העובדים ואת החבילות בגרף לפי סדר התרחשותם, באופן זה נדאג שכל עובד שמגיעה לנקודה, ידע אלו חבילות נמצאות בנקודה.

לכן, בודקים בכל פעם מי העובד שהזמן שלו להגיעה לנקודה הבאה במסלולו הוא הקרוב ביותר, ואותו נבחר להזיז ממקומו, אחרי שנזיז אותו למקומו החדש, נבדוק את החבילות שנמצאות בנקודת איסוף, העובד יאסוף מנקודת האיסוף את כל החבילות שהוא יוכל לקדם אותם באותו יום לנקודת איסוף אחרת שקרובה יותר ליעד הסופי של החבילות. המונח "קרובה יותר" בהקשר זה ייקבע לפי מרחק אוירי ולא לפי מרחק של google maps כיון שלא נוכל לדעת באיזה מסלול מתעתדת החבילה להמשיך, וכל מטרתנו היא לקרב אותה ליעדה.

כמו כן, נבדוק את כל החבילות שנמצאות אצל העובד, האם הגיעה חבילה שהגיעה זמנה לרדת- כלומר שהגיעה כעת לנקודת איסוף שהכי תקרב אותה היום ליעדה.

וכמובן נוסיף לשעון של העובד את זמן הנסיעה לנקודה זו.

לאחר מכן, שוב נחפש את העובד הבא שהזמן נסיעה אל הנקודה הבאה במסלול שלו היא המוקדמת ביותר, וחוזר חלילה.

בסופו של דבר, לכל עובד תישמרנה הנקודות שבמסלול שלו, ולכל נקודה במסלולו אלו חבילות להוריד ואלו חבילות לאסוף- זו אופציה אפשרית בודדת.

כעת נחשב ניקוד לכל אופציה אפשרית, הניקוד הוא סכום של היחס של המרחק שנשאר לחבילה לעבור לעומת המרחק הכללי בין נקודת המקור של החבילה ליעדה, עבור כל חבילה.

כך מוגרלות עשרות ואלפי אופציות אפשריות, כאשר האלגוריתם בוחר בסופו של תהליך את האופציה בעלת הניקוד הגבוה ביותר.

10.5 אילוצים:

- המערכת צריכה להיות מחוברת לאינטרנט עבור חישוב האלגוריתם, בגלל השימוש בנתוני google maps.
- המערכת חייבת לספק ללקוחות שרות תוך מספר ימים מינימלי

11. תיאור הארכיטקטורה

11.1 הארכיטקטורה של הפתרון המוצע בפורמט של Top – Down level design:

צד שרת מחולק כמקובל לשכבות:

- שכבת DAL.
- שכבת BLL.
- שכבת Web Api.

החלוקה לשכבות נועדה להפריד באופן מוחלט בין הלוגיקה של הפרויקט, לבין הנתונים עצמם. הפרדה זו מאפשרת לבצע שינויים בכל אחת מהשכבות, בלי תלות וזעזועים בשכבות האחרות ונוחה לאיתור באגים.

פירוט:

DAL: שכבת זו היא השכבה דרכה ניגשים לנתונים היושבים בDB, שכבה זו מחולקת לשני פרויקטים נפרדים:

1. הפרויקט DBContext- בפרויקט זה קיימת מחלקה שנבנתה ע"י טכנולוגית .EntityFrameWork והיא אחראית על בנית מסד הנתונים (בשיטת code first), ותקשורת עם מסד הנתונים.

2. הפרויקט Repositories- בפרויקט זה קיים מודל שנבנה ע"י טכנולוגיית ,EntityFrameWork ובו מחלקה מקבילה לכל טבלה במסד הנתונים.

בנוסף יש בו מחלקת Repository לכל ישות, הממשת את הממשק IRepository בו מוגדרות פעולות הוספה/עדכון/מחיקה/שליפה של ישות- פעולות אלו ישמשו את השכבה שמעל.

BLL: שכבה זו מכילה את הלוגיקה והאלגוריתמים. כמו כן, משמשת כמתווך בין שכבת ה DAL לשכבת ה WebApi-בשליפת ובעדכון הנתונים.

הספריות שבפרויקט זה:

Algorithm- כאן יהיו מחלקות שונות המטפלות בחישוב האלגוריתם ובשאר חלקי הלוגיקה.

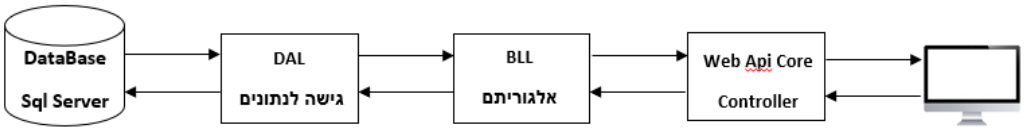
Services- מחלקות עזר המסייעות לתקשורת בין השכבות

הישויות בהן משתמשת שכבה זו הן ה DTOs המוגדרות בפרויקט נפרד ובלתי תלוי- **Common**- פרויקט זה מכיל מחלקות מקבילות למבנה הנתונים כדי לקשר בין שכבת ה- DAL וה- WebApi.

Web Api: שכבה זו מתקשרת עם צד הלקוח, באמצעות Controllers – בקרים, המקבלים פניות מהלקוח ומאחזרים מידע בהתאם.

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

11.2 תיאור הרכיבים בפתרון:



- **DB –** מסד הנתונים, מורכב מטבלאות המערכת.
- **DAL –** שכבת הגישה לנתונים.
- **BLL –** הלוגיקה של המערכת בו כתוב האלגוריתם.
- **Web Api –** פרוטוקול התקשורת בין צד השרת לצד הלקוח.
- **צד הלקוח, React.**

11.3 ארכיטקטורת רשת:

חיבור ל google maps

11.4 תיאור פרוטוקולי התקשורת:

פרוטוקול התקשורת הינו HTPP.

11.5 שרת – לקוח:

צד השרת נכתב בטכנולוגית #C ובשפת Web Api.

צד הלקוח נכתב בשפות CSS, html, ו- TypeScript בטכנולוגית React.

11.6 תיאור הצפנות:

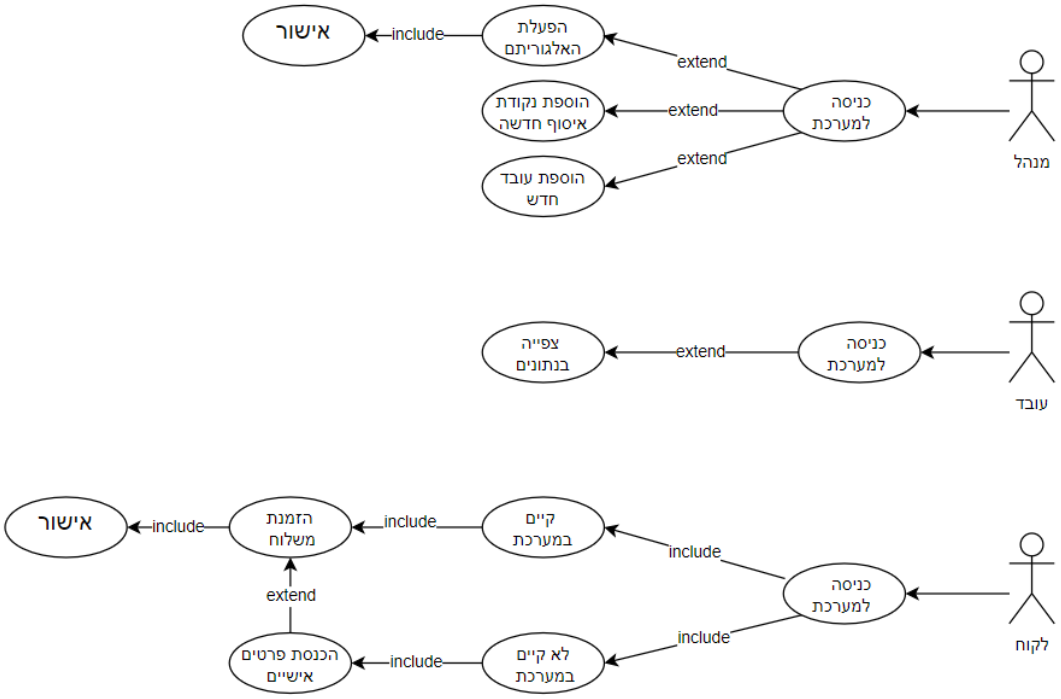
לא רלוונטי.

12. ניתוח ותרשים של UML/Use cases של המערכת המוצעת

12.1 רשימת ה- Use cases:

- הוספת לקוח חדש
- הוספת הזמנת משלוח חדש
- הצגת היעדים אליהם השליח עתיד להגיע והחבילות שבהן הוא אמור לטפל
- הפעלת האלגוריתם

12.2 תרשים ה- Use cases:



12.3 תיאור ה- Use cases העיקריים:

שם	הוספת לקוח חדש
תאור קצר	רישום הלקוח לרשימת הלקוחות באתר.
שחקנים	לקוח.
תנאי קדם	-
הזנק	קבלת נתונים.
תיאור מהלך התרחיש	1. הלקוח מזין את פרטיו. 2. המערכת שומרת את הלקוח החדש. 3. המערכת מפנה את ה לקוח לביצוע הזמנת משלוח.
תרחישים אלטרנטיביים	הלקוח כבר רשום במערכת.
תנאי סופי	שמירת נתונים.

שם	הוספת הזמנה למערכת
תאור קצר	משתמש מזמין משלוח.
שחקנים	משתמש.
תנאי קדם	המשתמש רשום במערכת.
הזנק	צפייה בנתונים וקליטת נתונים
תיאור מהלך התרחיש	1. המערכת מבקשת פרטים על המשלוח. 2. המשתמש מכניס פרטים על המשלוח. 3. המשתמש מאשר הזמנה. 4. המערכת שומרת נתונים.
תרחישים אלטרנטיביים	המשתמש לא קיים המערכת – על המשתמש להירשם למערכת..
תנאי סופי	המשתמש אישר את ההזמנה.

שם	הפעלת האלגוריתם
תאור קצר	הפעלת האלגוריתם שיבוץ וניווט החבילות לעובדים.
שחקנים	המנהל.
תנאי קדם	קיים מאגר שליחים
הזנק	קליטת נתונים
תיאור מהלך התרחיש	1. המערכת מציגה נתונים למנהל. 2. המנהל מפעיל את האלגוריתם. 3. המערכת שולחת את הנתונים להמשך עיבוד.
תרחישים	אין עובדים במאגר.

אלטרנטיביים

תנאי סופי	יום עבודה חדש נפתח.

שם	צפייה בנתוני עובד
תאור קצר	העובד צופה ביעדים אליהם הוא עתיד להגיע, ובחבילות אותן הוא צריך להעביר.
שחקנים	עובד.
תנאי קדם	יום עסקים חדש אכן נפתח
הזנק	צפייה בנתונים.
תיאור מהלך התרחיש	1. המערכת מבקשת פרטים של העובד. 2. המשתמש מקיש פרטים. 3. המערכת מחפשת את העובד לפי הנתונים. 4. המערכת מציגה את הנתונים שנמצאו לעובד.
תרחישים	שגיאה בהקלדת הנתונים של העובד.

אלטרנטיביים

תנאי סופי	העובד מסיים לצפות ביעדי היום הבא שלו.

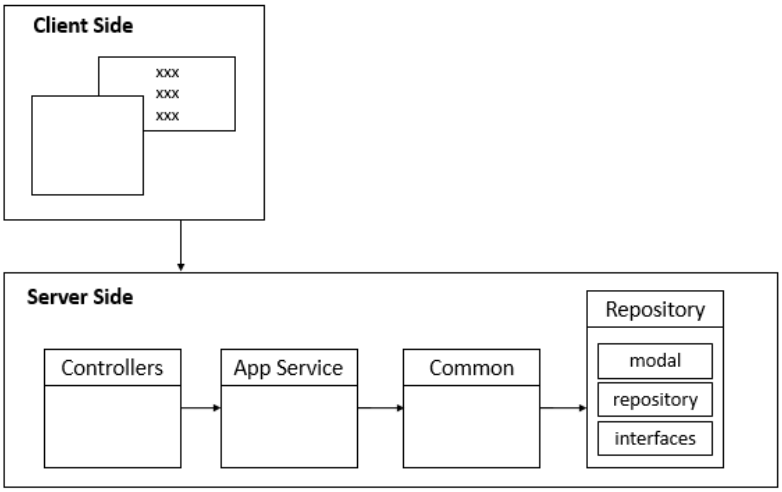
12.4 מבנה הנתונים:

מבני הנתונים אשר השתמשתי בהם:

רשימה- מבנה נתונים נוח לשליפה מהמסד נתונים.

גרף- בשביל לשמור את הנקודות איסוף בצורה שהגישה אליהם תהיה נוחה ויעילה הייתי צריכה ליצור מבנה נתונים אשר יכיל קודקודים וקשתות הגרף הוא המבנה הנוח ביותר, הגרף מיוצג ע"י רשימת סמיכויות, כלומר רשימת קודקודי הגרף, כל תא ברשימה זו מייצג רשימת קשתות היוצאות מקודקוד זה.

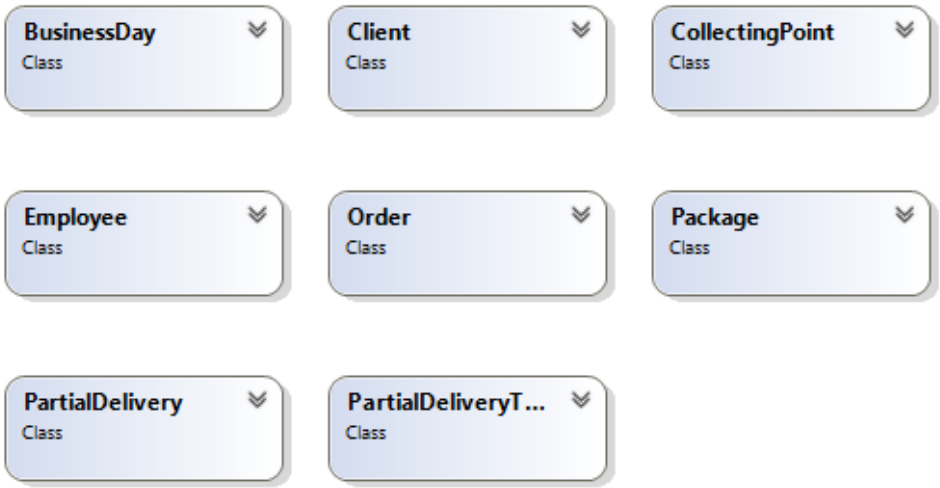
12.5 עץ מודולים:



12.6 תרשים מחלקות הפרויקט:

- **שכבת DAL:**

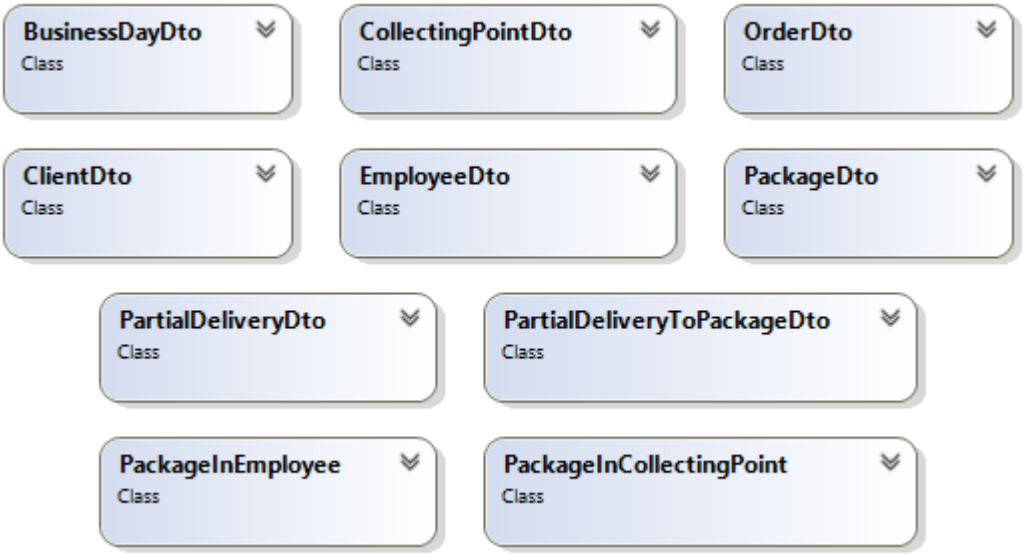
Entities



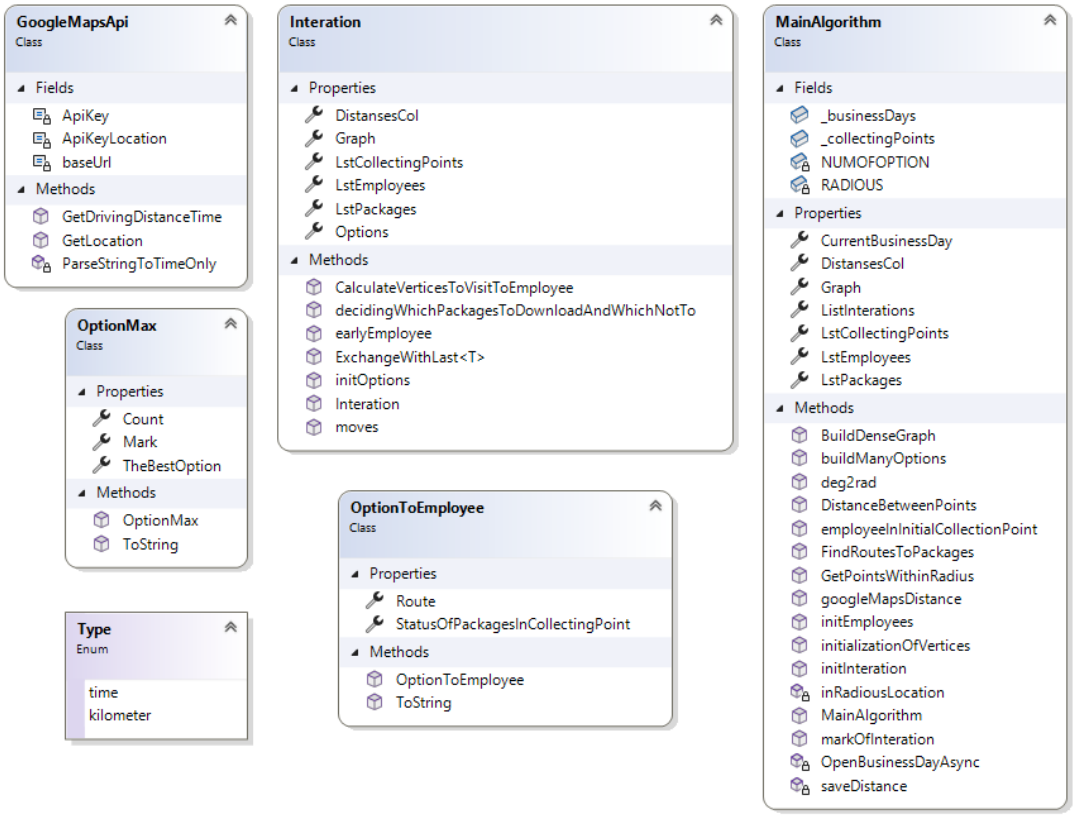
Repositories

Irepository

- **שכבת DTO:**

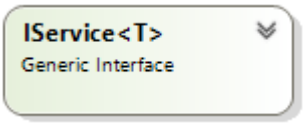


- **שכבת BLL:**
  - מחלקות האלגוריתם



- Services

- Iservices



- **שכבת Web Api:**

12.7 תיאור המחלקות המוצעות:

כעת נכנס ונעמיק ברכיבים ובשכבות.

- **שכבת DB:**

שכבה זו היא בסיס הנתונים בה נשמרים הנתונים עבור הפרויקט. שכבה זו בנויה כקובץ של SQL DataBase ומכילה את כל הטבלאות הנחוצות עבור נתוני הפרויקט.

- **שכבת DAL:**

דרך שכבה זו ניגשים לנתונים היושבים בDB – המחלקות בשכבה זו נוצרות אוטומטית אחרי שייבאנו את המודל. לכל מחלקה בשכבה זו נוצרים מאפיינים זהים למאפיינים שהגדרנו בבסיס הנתונים, הן מבחינת השמות והן מבחינת הטיפוסים, ולכן לאף אחת מהם אין קלטים ופלטים.

- **שכבת BLL :**

שכבת הלוגיקה העסקית (BLL) אחראית על יישום הכללים העסקיים והלוגיקה של האפליקציה. שכבה זו פועלת כמתווך בין שכבת המצגת (ממשק משתמש) לשכבת הגישה לנתונים (בסיס הנתונים). המחלקות בשכבה זו מכילים את ההיגיון העסקי ותוכננו לשימוש חוזר ובלתי תלוי בכל ממשק משתמש ספציפי. שכבת BLL היא המקום שבו מיושמת האלגוריתם. שכבת BLL גם מבטיחה שהנתונים המועברים בין שכבת המצגת לשכבת הגישה לנתונים תקפים ועקביים.

- **שכבת Web Api:**

BusinessDay,ClientController, CollectingPointController, EmployeeController, OrderController, PackageController, PartialDeliveryController, PartialDeliveryToPackageController

תפקיד המחלקה: מחלקות אלו נוצרות כדי ליחצן פונקציות למשתמש, לכן הן נקראות controller שמתחבר למשתמש, שכל בקשה שלו מתחברת ל- controller המתאים.

ולכן:

קלט: הקלט של כל אחת ממחלקות אלו הוא המשתמש. בכל מחלקה כזו יש פעולות: Get, Post, Put, Delete וכל פעולה כזאת פונה למחלקה המתאימה בDB ומטפלת שם במה שהיא צריכה, אם זה לעדכן רשומה/ות או להחזיר רשומה/ות.

פלט: הפלט הוא לפי הפעולה שהתבצעה.

- **שכבת DTO:**
  BusinessDayDto,ClientDto, CollectingPointDto, EmployeeDto, OrderDto, PackageDto, PartialDeliveryDto, PartialDeliveryToPackageDto,
  תפקיד המחלקה: העצמים מסוג DTO הם עצמים שחוזרים למשתמש, ולכן בכל אחת מהמחלקות יש את המאפיינים שנוצרו אוטומטית במחלקה התואמת בשכבת ה- DAL, וכן יש בכל מחלקה כזו פונקציית המרה מהעצם המקביל ב-DAL לעצם מסוג ה-DTO, ופונקציית המרה מסוג DTO לעצם המקביל ב-DAL.
  קלט: הקלט הוא עצם מסוג שרוצים להמיר ממנו.
  פלט: הפלט הוא העצם המומר לסוג הרצוי.

בנית הפרויקט עפ"י מודל אומנם דורש תיכנון רב אך כך אנו מספקים אתר עם קוד נקי, קל להבנה, ונוח לשינויים ושדרוגים.

13. רכיבי ממשק

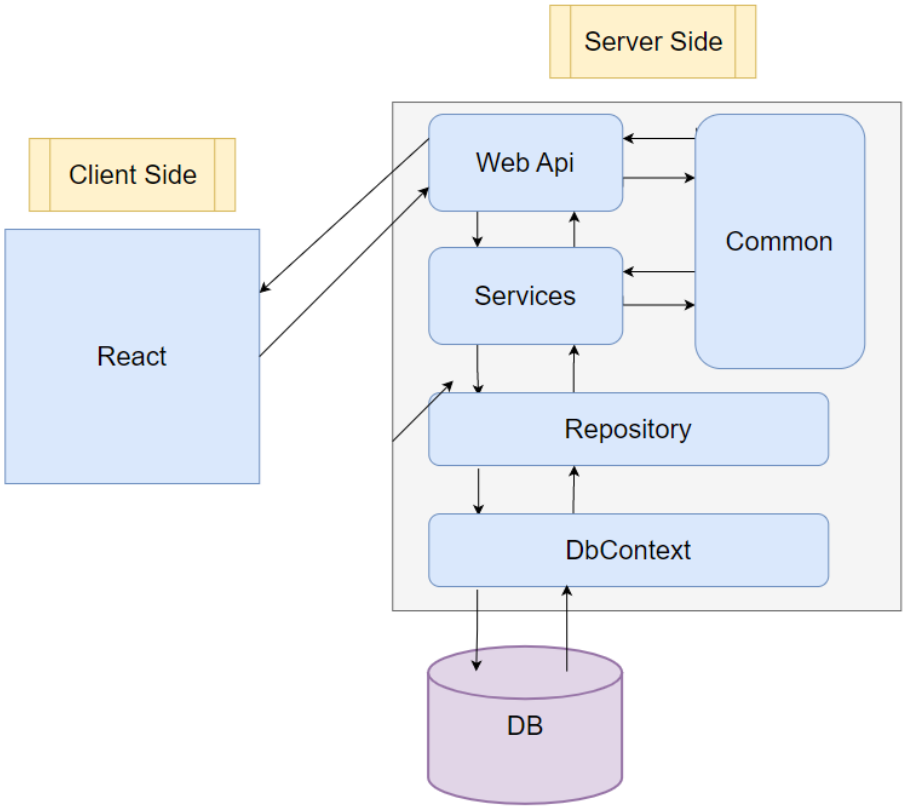
המערכת מורכבת מ-2 חלקים:

1. תוכנית הלקוח.
2. תוכנית השרת.

14. תיכון המערכת

המערכת מוצאת את המסלולים הטובים ביותר על מנת להעביר את החבילות ליעדן ע"י מעבר בנקודות איסוף הפזורות ברחבי הארץ, כך שהחבילה תגיע ליעדה במהירות הגבוהה ביותר האפשרית.

14.1 ארכיטקטורת המערכת:



14.2 תיכון מפורט:

השתמשתי בטכנולוגית API שהוא ערכה של ספריות קוד ,פקודות ,פונקציות ופרוצדורות מן המוכן, בהן יכולים המתכנתים לעשות שימוש פשוט, בלי להידרש לכתוב אותן בעצמם כדי שיוכלו להשתמש במידע של היישום שממנו הם רוצים להשתמש לטובת היישום שלהם.

14.3 חלופות לתיכון המערכת:

יכולתי להשתמש בטכנולוגית MVC אבל יש לה כמה חסרונות מול API:

1. Asp.Net MVC משמש ליצירת יישומי אינטרנט שמחזירים תצוגות ונתונים, אך Asp.Net Web Api משמש ליצירת שירותי HTTP מלאים בצורה קלה ופשוטה המחזירה נתונים בלבד ולא תצוגה.
2. Web Api עוזר לבנות שירותי Rest מלאים על פני Net FramWork. והוא תומך גם במשא ומתן על תוכן) זה החלטה על נתוני פורמט התגובה הטובים ביותר שיכולים להיות מקובלים על ידי הלקוח. זה יכול להיות Atom ,XML ,Json או נתונים מעוצבים אחרים, (אירוח עצמי שאינו בMVC).
3. Web Api דואג גם להחזיר נתונים בפורמט מסוים כמו Json, XML או כל דבר אחר המבוסס על כותרת קבל בבקשה ואתה לא דואג לזה. MVC מחזיר נתונים רק בפורמט Jsonבאמצעות JsonResult.
4. ב- Web Api הבקשה ממופה לפעולות המבוססות על פעלים ב- HTTP, אך בMVC היא ממופה לשם הפעולות.

5. Asp.Net Web Api הוא מסגרת חדשה וחלק ממסגרת הליבה של Asp.Net. כריכת המודל, המסננים, הניתוב ותכונות אחרות של MVC קיימות ב- Web Api שונות מMVC וקיימות במכלול System.Web.Http החדש. בMVC, תכונות אלו קיימות בתוך System.Web.MVC. מכאן שניתן להשתמש ב-Web Api גם עם Asp.Net וכיכבת שירות עצמאית.
6. אתה יכול לערבב Web Api ובקר MVC בפרויקט אחד לטיפול בקשות AJAX מתקדמות שעשויות להחזיר נתונים בתבנית XML, Json או בכל אחרים ולבנות שירות HTTP מלא. בדרך כלל זה ייקרא אירוח עצמי של Web Api.
7. יתר על כן, Web Api הוא ארכיטקטורה קלילה, פרט ליישום האינטרנט, ניתן להשתמש בו גם עם אפליקציות לסמארטפונים.

15. תיאור התוכנה

15.1 סביבת עבודה:

Visual Studio גרסת 2022 עם גרסת Net Framework 4.7.

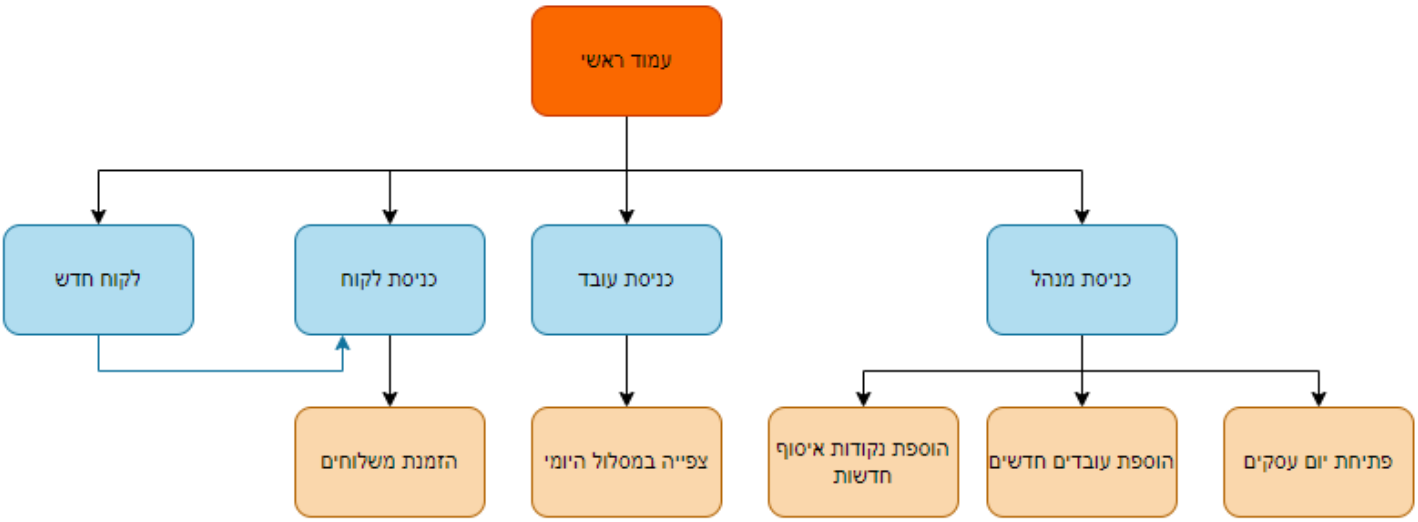
15.2 שפות תכנות:

- C#
- SQL
- HTML, CSS, JS בטכנולוגית React.

16. תיאור מסכים

- כניסה לאתר.
- כניסת מנהל לביצוע האלגוריתם
- כניסת עובד לצפייה במסלול יומי
- כניסת לקוח רשום להזמנת משלוח
- רישום לקוח חדש
- הוספת עובדים חדשים
- הוספת נקודות איסוף חדשות

17. תרשים המסכים

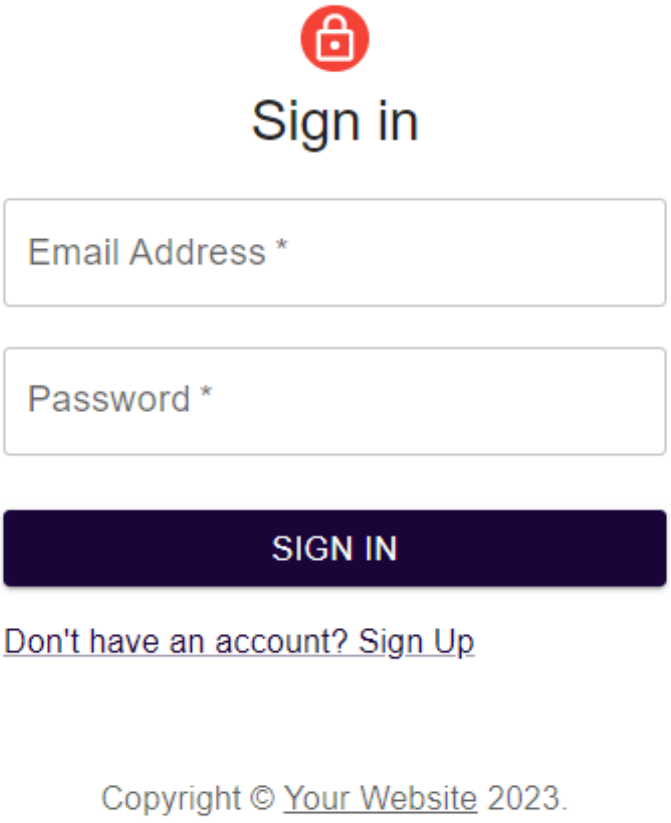


18. פרוט מסכים

מסך הכניסה לאתר:

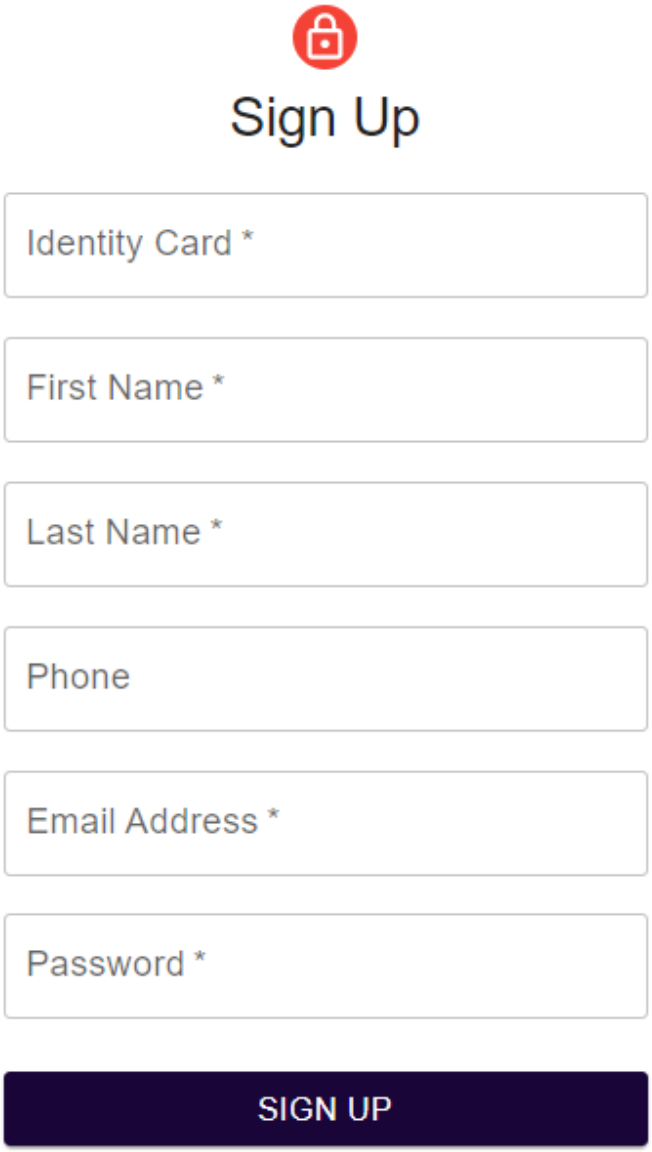
כניסת משתמש רשום לחשבון

המשתמש - המנהל/עובד/לקוח יכניס מייל וסיסמא ויועבר לאזור האישי שלו:

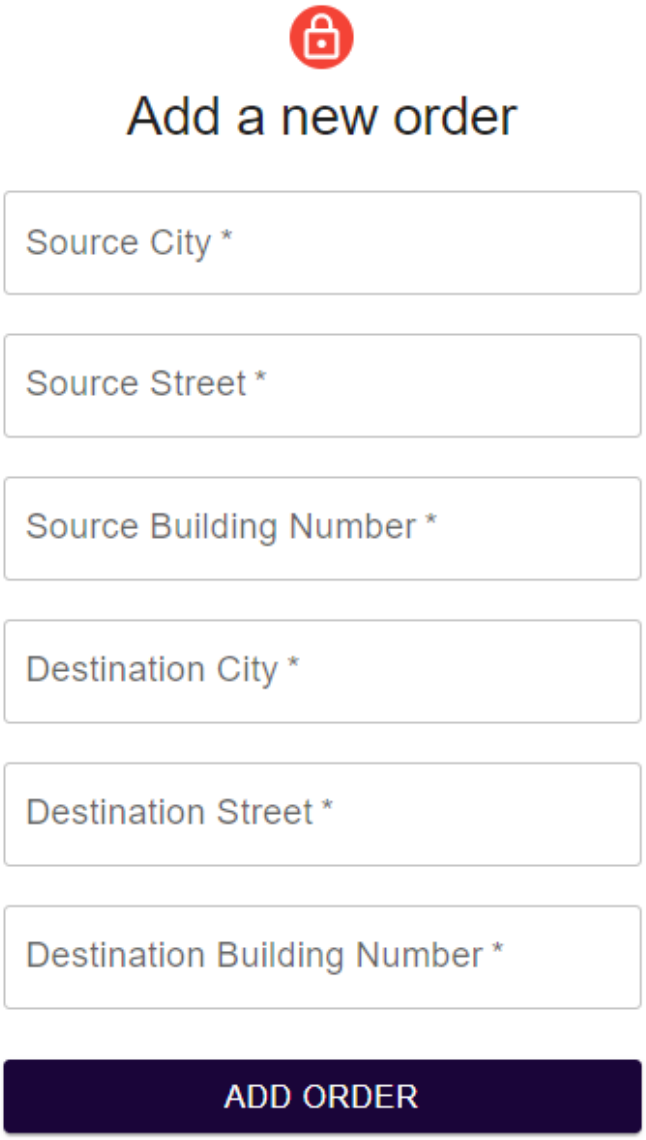


הוספת משתמש חדש למערכת

הלקוח יכניס את פרטיו. ולאחר שמירת נתוניו הוא יעבור לדף הזמנת משלוח



הוספת הזמנה חדשה למערכת- הלקוח מזמין משלוח חדש



כניסת עובד- עובד נכנס לאזור האישי שלו ע"י מייל וסיסמא
העובד יכול לצפות במסלול היומי שלו, כלומר בנקודות האיסוף בהם הוא צריך לבקר היום, וכן בחבילות שהעובד צריך להוריד ולאסוף מכל נקודת איסוף.

Hello to Israel

#	Collection Point Number	Address
1	38	Rothovski 8 St, Petach Tikva
2	43	Rabi Akiva 100 St, Bney Brak
4	13	Ben Guryon 8, Tel Aviv

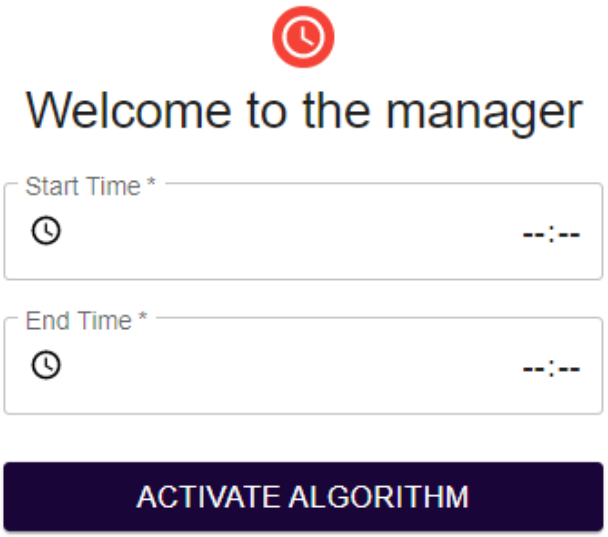
Hello to Israel

#	Collection Point Number	Address
1	38	Rothovski 8 St, Petach Tikva

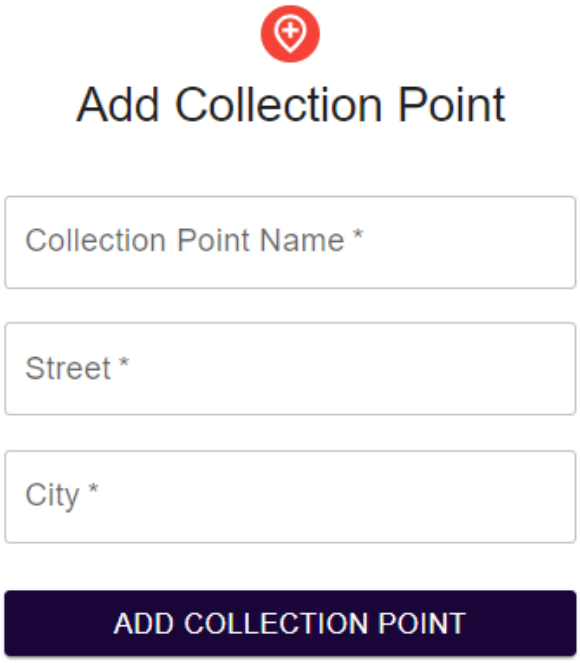
Package ID	Status
27	upload
14	download

2	43	Rabi Akiva 100 St, Bney Brak
4	13	Ben Guryon 8, Tel Aviv

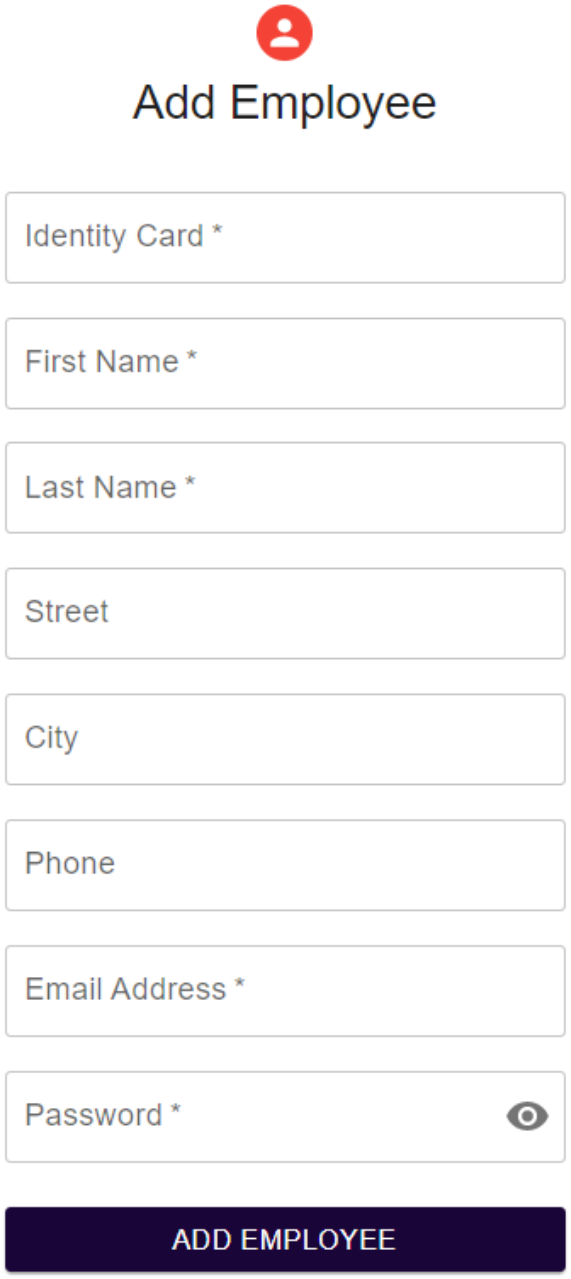
כניסת מנהל- המנהל נכנס לאזור האישי שלו ע"י מייל וסיסמא, המנהל בוחר שעות של יום עבודה ומפעיל את האלגוריתם



הוספת נקודת איסוף חדשה למערכת, מתבצעת רק ע"י המנהל



הוספת עובד חדש למערכת, מתבצעת רק ע''י המנהל



19. תיעוד המחלקות והפונקציות של האלגוריתם

המחלקות הקשורות לאלגוריתם של בניית הגרף:

מחלקה ליצירת גרף ממושקל-WeightedGraph

תפקיד המחלקה:

המחלקה WeightedGraph היא מבנה נתונים המשמש לייצוג גרף עם קצוות ממושקלים.

הפונקציות העיקריות של מחלקה זו כוללים:

1. Nodes: מאפיין המייצג רשימה של GraphNode. כל GraphNode מייצג צומת בגרף.
2. AddNode(CollectingPointDto value): שיטה שמוסיפה צומת חדש לגרף עם הערך שצוין.
1. AddEdge(GraphNode<CollectingPointDto>node1, GraphNode <CollectingPointDto> node2, DistanceType weight1, DistanceType weight2):

שיטה המוסיפה קשתות משוקלל חדשות בין שני צמתים בגרף. הפרמטר weight1 מייצג את משקל הקשת מnode1 לnode2, בעוד שweight2 מייצג את משקל הקשת מnode2 לnode1.

מחלקה ליצירת צומת בגרף-GraphNode:

תפקיד המחלקה:

המחלקה GraphNode מייצגת צומת בגרף.

הפונקציות והמאפיינים העיקריים של מחלקה זו כוללים:

1. Value: מאפיין המייצג את הערך של הצומת- נקודת איסוף.
2. Neighbors: מאפיין המייצג מילון של צמתים שכנים והמרחקים בק"מ ומשך הזמן נסיעה ביניהם. המפתח הוא אובייקט GraphNode המייצג את הצומת השכן, והערך הוא אובייקט DistanceType המייצג את המרחק או את משך הזמן בין שני הצמתים.
3. GraphNode(CollectingPointDto value): בנאי שיוצר אובייקט GraphNode חדש עם הערך שצוין.

4. AddNeighbor(GraphNode<CollectingPointDto> neighbor, DistanceType weight): פונקציה שמוסיפה צומת שכן לצומת הנוכחי עם המרחק ומשך הזמן שצוינו. הפרמטר neighbor מייצג את הצומת השכן, בעוד הפרמטר weight מייצג את המרחק ואת משך הזמן בין שני הצמתים.

מחלקת GoogleMapApi:

תפקיד המחלקה: פונקציות המשתמשות ב- Google map api עבור שירותים שונים של Google Maps.

פרוט הפונקציות:

פונקציית GetDrivingDistanceTime מחזירה את משך הזמן הדרוש לנסיעה בין שתי נקודות נתונות, בהתאם למסלול המומלץ של Google Maps.

פונקציית GetLocation מחזירה את המיקום הגיאוגרפי של כתובת מסוימת, באמצעות פרטי המיקום שמוחזרים משירות ה-API של Google Maps.

המחלקות שקשורות לאלגוריתם האקראי:

המחלקה הראשית של האלגוריתם- mainAlgoritm

מתודות של המחלקה:

1. RADIOUS: זהו קבוע המייצג את המרחק בק"מ של רדיוס.
2. NUMOFOPTION: זהו קבוע המייצג את מספר האפשרויות האפשריות שמגרילים עבור האלגוריתם האקראי.

3. Graph: זהו מאפיין המייצג את הגרף המשוקלל המשמש למודל של פעולות המחלקה. זהו מופע של המחלקה WeightedGraph, שהיא יישום מותאם אישית של מבנה נתוני גרף המאפשר קשתות עם משקלים. הוא מכיל את כל נקודות האיסוף כקודקודים, ואת המרחקים והזמנים ביניהם כקשתות משוקללות.
4. LstCollectingPoints: זהו מאפיין המייצג את רשימת כל נקודות האיסוף במחלקה. ומכילה מידע על כל נקודת איסוף, כגון מזהה, כתובת ומיקומה.
5. LstEmployees: זהו מאפיין המייצג את רשימת כל העובדים במחלקה. ומכילה מידע על כל עובד, כגון תעודת הזהות, השם והמיקום שלו.
6. LstPackages: זהו מאפיין המייצג את רשימת כל החבילות במחלקה. ומכילה מידע על כל חבילה, כגון מזהה, מקור, יעד ומיקומה הנוכחי.
7. ListInteractions: זהו מאפיין המייצג רשימה של כל האופציות האפשריות שהוגרלו עבור פתרון אפשרי של האלגוריתם.
8. DistancesCol: זהו מאפיין המייצג מטריצת מרחק בין כל זוגות נקודות האיסוף במחלקה. זהו מערך דו מימדי, כאשר DistancesCol[i,j] מייצג את המרחק בין נקודות האיסוף i ל-j.

<u>פונקציות של המחלקה:</u>

1. MainAlgorithm(List<CollectingPointDto> lstCollectingPoints, List<EmployeeDto> lstEmployees, List<PackageDto> lstPackages, IService<CollectingPointDto> collectingPoints): זהו הבנאי של המחלקה MainAlgorithm. הוא מאתחל את המאפיינים של המחלקה, כגון הגרף, רשימת נקודות האיסוף, רשימת העובדים ורשימת החבילות. וגם קורא לפונקציות אחרות כדי לאתחל את הקודקודים, העובדים והמרחקים בין נקודות, לבנות את הגרף, ולחשב לכל עובד את הנקודות שברדיוס שלו.

2. saveDistance(): פונקציה זו מחשבת ושומרת את המרחקים בין כל זוגות נקודות האיסוף באמצעות נוסחת Haversine. הוא מאתחל את מערך DistancesCol, שהוא מערך דו מימדי המאחסן את המרחקים בין כל זוגות נקודות האיסוף.

3. googleMapsDistance(CollectingPointDto c1, CollectingPointDto c2): פונקציה זו מחשבת את המרחק וזמן הנסיעה בין שתי נקודות איסוף באמצעות Google Maps API. הוא מחזיר את המרחק והזמן כאובייקט DistanceType.

4. buildManyOptions(TimeOnly open, TimeOnly close): פונקציה זו בונה מספר אפשרויות של יום עבודה אפשריות ומחזירה את הטוב ביותר. הוא מאתחל אובייקט Interaction חדש ומחשב את הניקוד של האינטראקציה על סמך פונקצית חישוב המפורטת בהמשך. לאחר מכן הוא יוצר מספר רב של אינטראקציות חדשות ומחשב את הציון של כל אחת מהן. הפונקציה מחזיר את האפשרות עם הציון הגבוה ביותר.

5. initInteraction(TimeOnly open): פונקציה זו מאתחלת אינטראקציה חדשה על ידי איפוס כל הנתונים מאינטראקציות קודמות. הפונקציה מגדיר את כל החבילות לנקודת האיסוף שלהם- בתי המקור, את כל העובדים לנקודת האיסוף הראשונה במסלול שלהם, ואתחול נקודות האיסוף -ללא חבילות.

6. markOfInteraction(): פונקציה זו מחשבת את הניקוד של אופציה בהתבסס על מיקומי החבילות לאחר יום העבודה של אופציה מסוימת, הפונקציה מחשבת את המרחק הכולל שלוקח להעביר את החבילה מבית הלקוח אל היעד, וכן את המרחק שנותר לחבילה כדי להגיע ליעדה הסופי

מנקודת האיסוף שהיא נמצאת שם בסוף יום העבודה של האופציה הנוכחית. לאחר מכן הפונקציה מחשבת את ניקוד כסכום של אחוזי המרחק שנותר לכל אחת מהחבילות.

7. initEmployees(): פונקציה זו מאתחלת את רשימת העובדים על ידי הגדרת רשימת החבילות שלהם, איסוף נקודות, קודקודים לביקור ושעון לערכים ההתחלתיים שלהם.

8. initializationOfVertices(): פונקציה זו מאתחלת את נקודות האיסוף ואת חבילות המקור-יעד. הפונקציה עוברת בלולאה ולכל חבילה יוצרת נקודת איסוף חדשה עבור המקור והיעד של החבילה. הפונקציה מוסיפה את נקודות האיסוף החדשות הללו למסד הנתונים, ולאחר מכן היא מעדכנת בחזרה את רשימת נקודות האיסוף ממסד הנתונים. לאחר מכן, עוברים בלולאה דרך כל נקודת איסוף ומוסיפים אותה כקודקוד חדש בגרף. וכן מעדכנים כל חבילה היכן היא נמצאת בגרף. וגם מוסיפים את החבילה לרשימת החבילות בנקודת האיסוף של המקור של החבילה.

9. BuildDenseGraph(): פונקציה זו בונה את הגרף הראשוני על ידי חיבור כל הנקודות באותו אזור עם קשתות ביניהן. הפונקציה עוברת בלולאה על כל העובדים ועוברת על כל הקודקודים בגרף. במקרה שנמצאו שני קודקודים אשר ברדיוס של עובד מסוים, הן מתחברות ביניהם על ידי הוספת קשת כפולה בין שני הקודקודים בגרף וקביעת משקל הקשת- כפונקציה של זמן ומרחק ביניהם באמצעות הקריאה לפונקצית googleMapsDistance(CollectingPointDto c1, CollectingPointDto c2) המחשבת את המרחק והזמן בין שני נקודות איסוף.

10. inRadiousLocation(): פונקציה זו בודקת אם נקודת איסוף נמצאת ברדיוס מסוים ממיקום העובד על הגלובוס. הוא מחשב את המרחק בין העובד לנקודת האיסוף באמצעות פונקציית עזר DistanceBetweenPoints() ובודקת אם הוא קטן או שווה לקבוע RADIOUS- מספר הקילומטר הנחשבים לאותו רדיוס. אם כן, הפונקציה מחזירה true, אחרת היא מחזירה false.

11. GetPointsWithinRadius(): פונקציה זו מוצאת את כל נקודות האיסוף שנמצאות ברדיוס לפי מיקומו של כל עובד על הגלובוס. הפונקציה עוברת בלולאה דרך כל עובד וכל נקודת איסוף בגרף. ועבור כל נקודת איסוף, היא מחשבת את המרחק בין העובד לנקודת האיסוף באמצעות פונקציית עזר DistanceBetweenPoints(). אם המרחק קטן או שווה לקבוע RADIOUS, נקודת האיסוף מתווספת לרשימת נקודות האיסוף המשויכות לעובד.

12. DistanceBetweenPoints(): פונקציה זו מחשבת את המרחק בקילומטרים בין שתי נקודות על הגלובוס באמצעות קואורדינטות קווי הרוחב והאורך שלהן. הוא משתמש בנוסחת Haversine כדי לחשב את המרחק.

13. deg2rad(): פונקציה זו ממירה מעלות לרדיאנים. הוא משמשת כפונקציית עזר לפונקציה DistanceBetweenPoints().

<u>המחלקה ליצירת אופציה אפשרית ליום עבודה- Interation()</u>

המחלקה Interaction מכילה פונקציות לאתחול אופציה אפשרית לעובדים וחישוב הקודקודים בהם יבקר כל עובד במהלך יום העבודה.

<u>המתודות של המחלקה:</u>

1.מתודות לשמירת הרשימות מהמסד נתונים וכן את הגרף כמו במחלקה העיקרית של האלגוריתם.

2.Options: זהו מאפיין המייצג אופציה אפשרית של יום עובדה, זהו מילון שבעבור כל עובדים ממופה האובייקט OptionToEmployee אשר מכיל מידע על נקודות האיסוף והחבילות שיטופלו על ידי העובד ביום זה באופציה זו.

<u>הפונקציות של המחלקה:</u>

1. Interaction: המחלקה בונה של המחלקה, הפונקציה מזמנת פונקציות לאתחול אופציה אפשרית לעובדים והפונקציות החישוביות של בניית האופציה האפשרית.

2. InitOptions(): פונקציה המאתחלת אפשרות אפשרית עבור כל העובדים.

3. CalculateVerticesToVisitToEmployee(): פונקציה המחשבת לכל עובד את הנקודות שיעבור במהלך יום העבודה. תחילה הוא מגדיר את השעה הנוכחית של העובד לשעת ההתחלה ומוסיף את המיקום הנוכחי של העובד לרשימת הקודקודים לביקור. אם המיקום הנוכחי הוא מקור, הוא מסיר אותו מרשימת נקודות איסוף העובד. לאחר מכן, הפונקציה בוחרת נקודת איסוף אקראית מרשימת נקודות האיסוף הנותרת ומחשבת את זמן הנסיעה כדי להגיע אליה. אם זמן ההגעה לנקודת האיסוף הוא לפני שעת הסיום, נקודת האיסוף מתווספת לרשימת הקודקודים לביקור, ומתעדכנים השעה והמיקום הנוכחיים של העובד. הפונקציה ממשיכה לבחור באקראי נקודות איסוף ולהוסיף אותן לרשימת הקודקודים לביקור עד שמגיעים לשעת הסיום או שכל נקודות האיסוף עברו ביקור.

4. ExchangeWithLast(): פונקציית עוזר המחליפה את האלמנט באינדקס נתון עם האלמנט האחרון ברשימה ולאחר מכן מסירה את האלמנט האחרון מהרשימה.

5. MovingEmployeesAndPackagesInGraph (): פונקציה המוסיפה לאופציה האפשרית את הקודקודים שנבחרו למסלול של העובד, לאחר מכן הפונקציה עוברת בלולאה עד שכל העובדים ביקרו בכל הקודקודים שלהם. בכל איטרציה של הלולאה, הפונקציה מוצאת את העובד שהזמן הנוכחי שלו בתוספת הזמן להגיע לקודקוד הבא הוא הקרוב ביותר לזמן הנוכחי. העובד המבוקש מעודכן לשעה הנוכחית שלו, הקודקוד שביקר מרשימת הקודקודים שלו לביקור מוסר מהרשימה. אם העובד ביקר בכל הקודקודים שלו, הוא מוסר מרשימת העובדים. לאחר מכן הפונקציה מעדכנת את מצב החבילות בנקודת האיסוף החדשה ומחליטה אילו חבילות לאסוף או להוריד.

6. decidingWhichPackagesToDownloadAndWhichNotTo(): פונקציה שמחליטה אילו חבילות העובד צריך לאסוף או להוריד בנקודת איסוף בהתבסס על המצב הנוכחי של החבילות ויכולת העובד לקדם את החבילה למקום טוב יותר. כלומר העובד אוסף את כל החבילות שהוא יכול לקרב אותם לידען לפי המסלול המתוכן לו ליום עבודה זה. והעובד מוריד את כל החבילות שהגיעה זמנן לרדת, כיוון שהגיעו לנקודת איסוף שתקדם אותם במקסימום ביום עבודה זה.

7. getNextEmployeeToMove (): פונקציה שמוצאת את העובד שהזמן הנוכחי שלו בתוספת הזמן להגיע לקודקוד הבא הוא הקרוב ביותר לשעה הנוכחית.

20. קוד התוכנית + תיעוד

מחלקות הגרף:

מחלקה גרף ממושקל:

```
5 references
public class WeightedGraph<CollectingPointDto>
{
    11 references
    public List<GraphNode<CollectingPointDto>> Nodes { get; set; }
    1 reference
    public WeightedGraph()
    {
        Nodes = new List<GraphNode<CollectingPointDto>>();
    }
    1 reference
    public void AddNode(CollectingPointDto value)
    {
        Nodes.Add(new GraphNode<CollectingPointDto>(value));
    }
    1 reference
    public void AddEdge(GraphNode<CollectingPointDto> node1, GraphNode<CollectingPointDto> node2, DistanceType weight1,DistanceType weight2)
    {
        node1.AddNeighbor(node2, weight1);
        node2.AddNeighbor(node1, weight2);
    }
}
```

מחלקה צומת בגרף:

```
37 references
public class GraphNode<CollectingPointDto>
{
    30 references
    public CollectingPointDto Value { get; set; }
    5 references
    public Dictionary<GraphNode<CollectingPointDto>, DistanceType> Neighbors { get; set; }

    1 reference
    public GraphNode(CollectingPointDto value)
    {
        Value = value;
        Neighbors = new Dictionary<GraphNode<CollectingPointDto>, DistanceType>();
    }
    2 references
    public void AddNeighbor(GraphNode<CollectingPointDto> neighbor, DistanceType weight)
    {
        Neighbors[neighbor] = weight;
    }
}
```

הרצת האלגוריתם:

<u>המחלקה הראשית את האלגוריתם:</u>

פעולה בונה:

```
public MainAlgorithm(List<CollectingPointDto> lstCollectingPoints, List<EmployeeDto> lstEmployees
    , List<PackageDto> lstPackages, IService<CollectingPointDto> collectingPoints)//פעולה בונה של האלגוריתם
{


    OpenBusinessDayAsync();

    initializationOfVertices();// אתחול הקודקודים בגרף-נקודות איסוף
                               // , ועדכון הנקודת איסוף שם היא נמצאת, נקודות של חבילות-יעד ומקור וכן אתחול כל חבילה היכן היא יושבת

    initEmployees();

    saveDistance(); //שמירת המרחקים בק''מ מכל נקודה לכל נקודה

    BuildDenseGraph();//בנית גרף- החיבור הקשתות

    GetPointsWithinRadius();//פונקציה שקובעת לכל עובד מי הנקודות שלו לפי מיקומים

}
```

שמירת מרחקים בין נקודות איסוף:

```
//שמירה במערך את כל המרחקים בין נקודות
1 reference
private void saveDistance()//שומרת מרחק -ע''י נוסחה חשבונית
{
    for (int i=0;i<LstCollectingPoints.Count;i++)
    {
        LstCollectingPoints[i].indexDistance = i;
    }
    int max = LstCollectingPoints.Count;
    DistansesCol = new double[max, max];
    for (int i = 0; i < LstCollectingPoints.Count; i++)
    {
        for (int j = 0; j < LstCollectingPoints.Count; j++)
        {
            if (i != j)
                DistansesCol[LstCollectingPoints[i].indexDistance, LstCollectingPoints[j].indexDistance] =
                    DistanceBetweenPoints((double)LstCollectingPoints[i].LocationX, (double)LstCollectingPoints[i].LocationY,
                    (double)LstCollectingPoints[j].LocationX, (double)LstCollectingPoints[j].LocationY);
        }
    }
}
```

פונקציית לאתחול עובדים ונקודות איסוף:

```
//לקבוע לכל עובד באיזה נקודה הוא יושב בתחילת האינטרציה- אם זה באמצע יום עבודה-אז איפה שישב בסוף האינטרציה הקודמת
2 references
public void employeeInInitialCollectionPoint()
{

    Random random = new Random();
    int i = 0;
    List<GraphNode<CollectingPointDto>> lstCollectingPointWithPackages;
    foreach (var employee in LstEmployees)
    //לכל עובד מרנדמים בתוך הנקודות שבאזורו שיש שם חבילות נקודה כל שהיא
    {
        //תביא לי נקודה שיש שם חבילה
        lstCollectingPointWithPackages = employee.colectingPointsOfEmployees.Where(c => c.Value.Packages.Count > 0).ToList();
        if (lstCollectingPointWithPackages.Count > 0)
        {
            i = random.Next(lstCollectingPointWithPackages.Count - 1);
            employee.CurrentLocation = lstCollectingPointWithPackages[i];
        }
        else
        {
            i = random.Next(employee.colectingPointsOfEmployees.Count-1);
            employee.CurrentLocation = employee.colectingPointsOfEmployees[i];
        }
    }
}
```

פונקציה המחשבת מרחק וזמן בין נקודות לפי מפות גוגל.

```
//מחשב מרחק בין נקודות ע''י זמן של מפות גוגל ומרחק ע''י המטריצה ששמרה מרחק
2 references
public static DistanceType googleMapsDistance(CollectingPointDto c1, CollectingPointDto c2)
{
    if (c1 == c2)
        return new DistanceType() { kilometer = 0, time = new TimeSpan() };
    return new DistanceType() { kilometer = DistansesCol[c1.indexDistance, c2.indexDistance]
        , time =GoogleMapsApi.GetDrivingDistanceTime(c1.Address,c2.Address)};

}
```

הפונקציה הראשית שמגרילה אופציות אפשריות ליום עבודה ובוחרת את הטוב ביותר

```
//פונקציה שבונה הרבה אופציות אפשריות ומחזירה את הטובה מביניהם
1 reference
public OptionMax buildManyOptions(TimeOnly open, TimeOnly close)
{
    int count= 1;
    employeeInInitialCollectionPoint();//קביעה מחדש איפה העובדים יושבים בתחילת היום
    Interation interation =new Interation(open, close,LstEmployees,Graph,DistansesCol);
    double max = markOfInteration();
    Dictionary<EmployeeDto, OptionToEmployee> optionMax=interation.Options;
    double mark;
    ListInterations = new List<Dictionary<EmployeeDto, OptionToEmployee>>();
    for (int i = 0; i < NUMOFOPTION; i++)
    {
        initInteration(open);//איתחול כללי של אופציה חדשה (איפוס נתונים מאופציות קודמות)
        employeeInInitialCollectionPoint();//קביעה מחדש איפה העובדים יושבים בתחילת היום
        interation = new Interation(open, close,LstEmployees,Graph,DistansesCol);
        ListInterations.Add(interation.Options);
        mark = markOfInteration();
        if (mark == max)
            count++;
        if(mark>max)
        {
            max=mark;
            optionMax = interation.Options;
            count = 1;
        }
    }
    return new OptionMax(optionMax, max,count);
}
```

פונקציה לחישוב ניקוד לאופציה אפשרית:

```
2 references
public double markOfInteration(/*Dictionary<EmployeeDto, OptionToEmployee> option*/)
{
    //ניקוד לאופציה אפשרית
    //אם קודם עושה את כל האופציות ורק אז מחשב ציון אז ברשימת החבילות מעודכן רק האופציה האחרונה//
    double mark = 0;
    foreach (var package in LstPackages)
    {
        double totalTime = DistansesCol[package.Source.Value.indexDistance, package.Destination.Value.indexDistance];
        double timeRemainWay;
        if (package.Source.Value == package.CurrentLocation.Value)
            timeRemainWay = totalTime;
        else
            timeRemainWay = DistansesCol[package.CurrentLocation.Value.indexDistance, package.Destination.Value.indexDistance];

        mark += (totalTime - timeRemainWay) / totalTime;

    }
    return mark;
}
```

בניית הגרף-חיבור הקשתות:

```
1 reference
public  void BuildDenseGraph()
{
    foreach (var employee in LstEmployees)
    {
        for (int i = 0; i < this.Graph.Nodes.Count; i++)
        {
            if (inRadiousLocation(employee, Graph.Nodes[i].Value) == true)
            {
                for (int j = i + 1; j < this.Graph.Nodes.Count; j++)
                {
                    if (inRadiousLocation(employee, Graph.Nodes[j].Value) == true)
                    {
                        //חיבור הגרף -קשת כפולה-קבועת זמן ומרחק//
                        Graph.AddEdge(Graph.Nodes[i], Graph.Nodes[j], googleMapsDistance(LstCollectingPoints[i], LstCollectingPoints[j])
                            , googleMapsDistance(LstCollectingPoints[j], LstCollectingPoints[i]));

                    }
                }
            }
        }
    }
}
```

פונקציה הבודקת האם העובד ונקודת איסוף נמצאים באותו רדיוס

```
//בודק רדיוס לפי מיקום בכדור הארץ
2 references
private bool inRadiousLocation(EmployeeDto employee, CollectingPointDto collectingPoint)
{
    //או להשתמש במערך ששומרים
    double distance = DistanceBetweenPoints((double)employee.LocationX, (double)employee.LocationY
        , (double)collectingPoint.LocationX, (double)collectingPoint.LocationY);

    // אם הנקודה נמצאת ברדיוס של מספר מסוים של ק"מ מהעובד, הוסף אותה לרשימת הנקודות ברדיוס של העובד
    if (distance <= RADIOUS)
        return true;
    return false;
}
```

חישוב לכל עובד אלו נקודות ברדיוס שלו

```
//פונקציה שמחשבת לכל עובד אלו נקודות נמצאות ברדיוס שלו-לפי מיקומים על כדור הארץ
1 reference
public  void GetPointsWithinRadius()
{

    // בדוק אילו נקודות נמצאות ברדיוס של מספר ק"מ מכל עובד
    foreach (EmployeeDto employee in LstEmployees)
    {
        // קבל את המיקום של העובד
        Tuple<double, double> workerLocation = new Tuple<double, double>((double)employee.LocationX, (double)employee.LocationY);

        foreach (GraphNode<CollectingPointDto> point in Graph.Nodes)
        {
            // קבל את המיקום של הנקודה
            Tuple<double, double> pointLocation = new Tuple<double, double>((double)point.Value.LocationX, (double)point.Value.LocationY);

            // חשב את המרחק בין העובד לנקודה בק"מ
            double distance = DistanceBetweenPoints(workerLocation.Item1, workerLocation.Item2, pointLocation.Item1, pointLocation.Item2);

            // אם הנקודה נמצאת ברדיוס של 3 ק"מ מהעובד, הוסף אותה לרשימת הנקודות ברדיוס של העובד
            if (distance <= RADIOUS)
            {
                employee.colectingPointsOfEmployees.Add(point);

            }
        }
    }

}
```

פונקציות עזר לחישוב מרחק בין שני מיקומים בכדור הארץ

```
// פונקציית עזר לחישוב המרחק בין שתי נקודות בק"מ
3 references
public static double DistanceBetweenPoints(double lat1, double lng1, double lat2, double lng2)
{
    double R = 6371; // רדיוס הארץ בק"מ
    double dLat = deg2rad(lat2 - lat1);
    double dLng = deg2rad(lng2 - lng1);
    double a =
        Math.Sin(dLat / 2) * Math.Sin(dLat / 2) +
        Math.Cos(deg2rad(lat1)) * Math.Cos(deg2rad(lat2)) *
        Math.Sin(dLng / 2) * Math.Sin(dLng / 2);
    double c = 2 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));
    double d = R * c; // המרחק בק"מ
    return d;
}

// פונקציית עזר להמרת מעלות לרדיאנים
4 references
public static double deg2rad(double deg)
{
    return deg * (Math.PI / 180);
}
```

<u>המחלקה לבניית אופציה אפשרית ליום עבודה</u>

פעולה בונה

```
2 references
public Interation( TimeOnly open,TimeOnly close,List<EmployeeDto> lste, WeightedGraph<CollectingPointDto> graph, double[,] distancesCol)
{
    DistansesCol = distancesCol;
    Graph = graph;
    LstEmployees = lste;
    Options = new Dictionary<EmployeeDto, OptionToEmployee>();
    initOptions(open);
    CalculateVerticesToVisitToEmployee(open, close);
    moves();
}
```

פונקציה שמחשבת לכל עובד את המסלול היומי שלו:

```
//לחגריל לכל עובד את הנקודות שהוא יעבור בהם
//צריך לדאוג שלמקור לא יוכל לבוא יותר מפעם אחת-כי לא מתווספות שם חבילות שלא היו קודם
1 reference
public void CalculateVerticesToVisitToEmployee(TimeOnly beginTime, TimeOnly endTime)
{
    EmployeeDto employee;
    for (int i=0;i<LstEmployees.Count;i++) //עובר על כל העובדים
    {
       employee= LstEmployees[i];
        employee.clock = beginTime;//אולי צריך///////////////////////////
        TimeOnly currentTime = (TimeOnly)(employee.clock == null ? new TimeOnly(0,0):employee.clock);//חזמן הנוכחי של העובד
        var currentLocation = employee.CurrentLocation;//המיקום הנוכחי של העובד
        //הנקודה הראשונה שהעובד צריך לבקר בה זה הנקודה שהוא נמצא בה כרגע
        //אולי הוספתי במקום אחר-לבדוק
        int countColOfEmployee = employee.colectingPointsOfEmployees.Count;//מספר הנקודות איסוף
        employee.VerticesToVisit.Add(employee.CurrentLocation);//הוספת הקודקוד לרשימת הקודקודים שהעובד יבקר בהם
        var collectingPoints = employee.colectingPointsOfEmployees;//הרישמה של הקודקודים ששיכים לעובד
        if (employee.CurrentLocation.Value.ColPointType==CollctingPointType.source)//(כאילו) אם אתה במקור תוציא אותו מהרשימה
        {
            int indx = collectingPoints.IndexOf(currentLocation);
            ExchangeWithLast<GraphNode<CollectingPointDto>>(collectingPoints, indx);
            countColOfEmployee--;
        }

 //כל עוד ישנם נקודות איסוף שלא נבדקו וגם חזמן של העובד עדיין לא הגיע לזמן סיום

 while (currentTime< endTime && countColOfEmployee > 0)
 {
     var randomIndex = new Random().Next(countColOfEmployee);//מרנדמים נקודה איסוף
     var collectingPoint = collectingPoints[randomIndex];
     if (collectingPoint == currentLocation)
     {
         if (countColOfEmployee == 1)
             break;
         continue;
     }

     var travelTime = currentLocation.Neighbors[collectingPoint].time;//חזמן שצריך כדי ללכת לנקודה הבאה
     //מוסיפים לזמן העכשוי של העובד את הזמן החדש כדי להגיע לנקודה המרונדמת
     var arrivalTime = currentTime.Add(travelTime);

     //אם אפשר להוסיף ועדיין לא יעבוד זמן הסיום- אז מוסיפים את הנקודה לרשימת הנקודות שבהם העובד צריך לבקר
     //וגם מעדכנים את הזמן של העובד, מעדכנים את הנקודה הנוכחית שבה העובד נמצא, ומסירים את נקודה מהרשימה
     if (arrivalTime.CompareTo(endTime) < 0)
     {
         //אם החלטת ללכת למקור, או שלקחת חבילה או שלא -אין מה לנסות פעם נוספת כי לא יגיעו עוד חבילות
         //צריך לעשות את זה גם בנקודה הראשונה אם זה מקור
         if(collectingPoint.Value.ColPointType==CollctingPointType.source)
         {
             ExchangeWithLast<GraphNode<CollectingPointDto>>(collectingPoints, randomIndex);
             countColOfEmployee--;
         }
         //מוסיפה לעובד נקודהה שבא יבקר
         employee.VerticesToVisit.Add(collectingPoint);

                 //מעדכנת את השעון של העובד
                 //צריך שהשעון של העובד יהיה timeonly
                 //והזמן שמחזיר מהמפות גוגל יהיה time span
                 currentTime = arrivalTime;
                 //employee.weights.Add(new TimeOnly(0, travelTime));//שומרת את המרחקים כל פעם מהנקודה הנוכחית להבאה
                 currentLocation = collectingPoint;//צריך להשתנות באמת----------------------------------------------

             }
             else
             {
                 ExchangeWithLast<GraphNode<CollectingPointDto>>(collectingPoints, randomIndex);
                 countColOfEmployee--;
             }
         }
         //מעדכנים בחזרה שהעובד יושב עכשיו בנקודה הראשונה
         //והזמן הוא הזמן ההתחלתי
         employee.CurrentLocation = employee.VerticesToVisit.First();
         employee.clock = beginTime;
     }
 }
```

פונקציה שמזיזה את העובדים בגרף

```
//בלולאה כל פעם לקחת עובד שיגיע לנקודה הבאה שלו בזמן הכי קרוב
1 reference
public void moves()
{
    EmployeeDto employee;
    foreach (var emp in LstEmployees)
    {
        Options[emp].Route.Add(emp.VerticesToVisit.First());

        Options[emp].StatusOfPackagesInCollectingPoint.Add(Options[emp].Route.Count - 1, new PackageState());

        //קובעת לכל חבילה האם העובד ייקח או יוריד אותה
        decidingWhichPackagesToDownloadAndWhichNotTo(emp);
    }
    int countEmployees=LstEmployees.Count;
    //כל עובד ישנם עובדים שלא סימו עדיין לבקר בכל הנקודות שהם צריכים לבקר בהם
    while(countEmployees>0)//שוב-אסור באמת למחוק
    {
        //מחזירים מי העובד שחזמן הנוכחי שלו+ חזמן לנקודה הבאה=לזמן הכי קרוב לזמן העכשוי
        int indx= earlyEmployee();
        if (indx == LstEmployees.Count)
            break;
        employee = LstEmployees[indx];

        TimeOnly clock = (TimeOnly)employee.clock;
        //להוסיף לזמן את הזמן של הנקודה הבאה
        employee.clock = clock.Add(employee.VerticesToVisit.First().Neighbors[employee.VerticesToVisit[1]].time);

        //מסירים את הנקודה הנ''ל מהרשימה של הנקודות שהעובד צריך לבקר בהם
        employee.VerticesToVisit.Remove(employee.VerticesToVisit.First());

        //לקבוע את הנקודה הנוכחית לנקודה הראשונה מבין הנקודות שהעובד צריך לבקר בהם
        employee.CurrentLocation = employee.VerticesToVisit.First();

        Options[employee].Route.Add(employee.VerticesToVisit.First());

        Options[employee].StatusOfPackagesInCollectingPoint.Add(Options[employee].Route.Count-1, new PackageState());

        //אם עכשיו מחקת את הנקודה האחרונה שהעובד הזה צריך לבקר- אז הסר אותו

        if(employee.VerticesToVisit.Count==1)
        {
            ExchangeWithLast<EmployeeDto>(LstEmployees, indx);
            countEmployees--;
        }

        //קובעת לכל חבילה האם העובד ייקח או יוריד אותה
        decidingWhichPackagesToDownloadAndWhichNotTo(employee);
    }
}
```

פונקציה עזר שמחזירה את העובד הבא שצריך 'להזיז' בגרף

```
//מוצאת ומחזירה מי העובד הבא שעכשיו צריך להתקדם פיזי
1 reference
public int earlyEmployee()
{
    TimeOnly min = TimeOnly.MaxValue;
    int employeeMin = LstEmployees.Count;// LstEmployees.First();

    EmployeeDto employee;
    for (int i = 0; i < LstEmployees.Count; i++)
    {
        employee = LstEmployees[i];
        if (employee.VerticesToVisit.Count <= 1)//בגלל שלא מוחקים באמת זה מסתבך-צריך לחשוב על פיתרון יותר טוב
            continue;
        //בדיקה האם הזמן עכשיו ועוד הזמן הגעה עד הנקודה הבאה מוקדם יותר מהמינימלי
        TimeOnly clock = (TimeOnly)employee.clock;
        TimeOnly time = clock.Add(employee.VerticesToVisit.First().Neighbors[employee.VerticesToVisit[1]].time);
        if (time < min)
        {
            employeeMin = i;
            min = time;
        }
    }
    return employeeMin;
}
```

פונקציה שקובעת לכל עובד בנקודת איסוף איך לטפל בחבילות

```
//להעביר את העובד לנקודה הבאה-פיזי
//להחליט איזה חבילות להוריד ואיזה חבילות לקחת
2 references
public void decidingWhichPackagesToDownloadAndWhichNotTo(EmployeeDto employee)
{
    //בדיקה עבור חבילות שנמצאות אצל העובד-מה להוריד
    foreach (var packageInEmployee in employee.Packages)
    {
        if (packageInEmployee.IsExist == false)
            continue;
        //לשמור לכל חבילה שהלקוח לוקח איתו-מה הנקודה מבין הנקודות שהוא עתיד להגיע אליהן
        //תקרב את החבילה במקסימום לכיוון היעד-במרחק-וכך הבדיקה הבאה תתבצע בקלות
        if(packageInEmployee.package.whereToGetOff==employee.CurrentLocation)
        {
            //מוסיפה לנקודה בגרף את החבילה הזאת- החבילה ירדה
            employee.CurrentLocation.Value.Packages.Add(new PackageInCollectingPoint(packageInEmployee.package,true));
            //מעדכנת את החבילה -שכעת היא יושבת בנקודה זו
            packageInEmployee.package.CurrentLocation = employee.CurrentLocation;
            //מסירה את החבילה מהעובד
            packageInEmployee.IsExist= false;
            //מאפסת -איפה החבילה צריכה לרדת-כי היא  ירדה עכשיו
            packageInEmployee.package.whereToGetOff = null;
            //מעדכן את האופציה שעובד זה בנקודה זו מוריד את החבילה הזאת
            Options[employee].StatusOfPackagesInCollectingPoint
                [Options[employee].Route.Count-1].PackagesDownloaded.Add(packageInEmployee.package);
        }
    }
    double distance;
//בדיקה עבור חבילות שנמצאות בנקודת איסוף שלשם הגיעה עכשיו העובד-מה לקחת
foreach (var packageInColPoint in employee.CurrentLocation.Value.Packages)
{
    if (packageInColPoint.IsExist == false)
        continue;
    //חיפוש האם יש נקודה שהמרחק שלה יהיה קטן יותר ליעד מאשר
    //המרחק בין איפה שעכשיו החבילה יושבת ליעד

    double min = DistansesCol[employee.CurrentLocation.Value.indexDistance, packageInColPoint.package.Destination.Value.indexDistance];
    GraphNode<CollectingPointDto> minColPoint = employee.CurrentLocation;
    //עובד על כל הנקודות ומחפש את הנקודה שתביא  למרחק המינימלי בין החבילה ליעד
    foreach (GraphNode<CollectingPointDto> colPoint in employee.VerticesToVisit)
    {
        distance = DistansesCol[colPoint.Value.indexDistance, packageInColPoint.package.Destination.Value.indexDistance];
        //רק אם המרחק יותר קטן וגם רק אם זה נקודת איסוף- אחרת אי אפשר להוריד לשם חבילות
        if (distance<min && (
            colPoint.Value.ColPointType==CollctingPointType.collectingPoint)||
             packageInColPoint.package.Destination==colPoint)
        {
            minColPoint = colPoint;
            min = distance;
        }
    }
        //אם באמת יש חבילה כזאת-שתקדם יותר את החבילה- אז נשלח את החבילה עם העובד-העובד יעלה את החבילה
        if(minColPoint!=employee.CurrentLocation )
        {
            //מעלה-מוסיפה לעובד את החבילה
            employee.Packages.Add(new PackageInEmployee( packageInColPoint.package,true));
            //כעת אין לחבילה מיקום-היא זזה אם העובד
            packageInColPoint.package.CurrentLocation = null;
            //מסירה מרשימת החבילות שנמאות בנקודה-את החבילה הזאת
            packageInColPoint.IsExist= false;
            //מעדכנת את החבילה איפה היא תצטרך לרדת(לפי הנקודה שחכי מקרבת כפי החיפוש לעיל
            packageInColPoint.package.whereToGetOff = minColPoint;
            //מעדכנת את האופציה- שלעובד כאשר הוא בנקודה זו אז- החבילות שהוא לוקח זה גם חבילה זו
            Options[employee].StatusOfPackagesInCollectingPoint[Options[employee].Route.Count - 1]
                .PackagesTaken.Add(packageInColPoint.package);
        }
    }
}
```

פונקציה לשמירת התוצאה ב-Date Base:

```
private async void saveResult(OptionMax o)//פונקצית שמירת הפיתרון של האלגוריתם במסד נתונים
{
    PartialDeliveryDto partialDelivery;
    foreach (var employee in o.TheBestOption.Keys)//עובר על כל העובדים
    {
        for (int i = 0; i < o.TheBestOption[employee].Route.Count; i++)//עובר על כל הנקודות שבמסלול של העובד
        {
            var col = o.TheBestOption[employee].Route[i];

            partialDelivery = new PartialDeliveryDto();
            partialDelivery.EmployeeId = employee.EmployeeId;
            partialDelivery.CollectingPointId = col.Value.CollectingPointId;
            partialDelivery.IndexOfDelivery = i;
            int partialDeliveryID = (await _partialDeliveris.AddAsync(partialDelivery)).Last().PartialDeliveryId;
            //עבור כל החבילות שהוא צריך להוריד
            foreach (PackageDto package in o.TheBestOption[employee].StatusOfPackagesInCollectingPoint[i].PackagesTaken)
            {
                PartialDeliveryToPackageDto PartialDeliveryToPackageDto = new PartialDeliveryToPackageDto();
                PartialDeliveryToPackageDto.PackageId = package.PackageId;
                PartialDeliveryToPackageDto.IsTakenOrDownloaded = (int)TakenOrDownloaded.taken;
                PartialDeliveryToPackageDto.PartialDeliveryId = partialDeliveryID;// partialDelivery.PartialDeliveryId;
                PartialDeliveryToPackageDto.IsTakenOrDownloaded = (int)TakenOrDownloaded.taken;
                await _partialDeliveryToPackageDto.AddAsync(PartialDeliveryToPackageDto);
            }
            foreach (PackageDto package in o.TheBestOption[employee].StatusOfPackagesInCollectingPoint[i].PackagesDownloaded)
            {
                PartialDeliveryToPackageDto PartialDeliveryToPackageDto = new PartialDeliveryToPackageDto();
                PartialDeliveryToPackageDto.PackageId = package.PackageId;
                PartialDeliveryToPackageDto.IsTakenOrDownloaded = (int)TakenOrDownloaded.downloaded;
                PartialDeliveryToPackageDto.PartialDeliveryId = partialDeliveryID;// partialDelivery.PartialDeliveryId;
                PartialDeliveryToPackageDto.IsTakenOrDownloaded = (int)TakenOrDownloaded.downloaded;
                await _partialDeliveryToPackageDto.AddAsync(PartialDeliveryToPackageDto);
            }
        }

        //עוברת על כל החבילות שנשמרו באופציה כי עברו שינויים
        //מעדכנת ברשימה ולאחר מכן מעדכנת בחזרה במסד נתונים

        var tPackages=  _packages.GetAllAsync();
        Task.WaitAll(tPackages);
        List<PackageDto> packages = tPackages.Result;

        var tCol=  _collectingPoints.GetAllAsync();
        Task.WaitAll(tCol);
        List<CollectingPointDto> collectingPoints = tCol.Result;


        foreach (var packageId in optionMax.StatePackageAfterDay.Keys)
        {
            var packageStatus = optionMax.StatePackageAfterDay[packageId];
            var packageToUpdate = packages.Where(p => p.PackageId == packageId).FirstOrDefault();
            if (packageToUpdate != null)
            {
                packageToUpdate.State = (int?)optionMax.StatePackageAfterDay[packageId].statusPackage;
                if (packageToUpdate.State==(int)PackageStatusEnum.InTransit)//אם החבילה באמצע הדרך
                {
                    //החבילה יושבת בנקודת איסוף ששמרו באלגוריתם
                    packageToUpdate.CollectingPointId = optionMax.StatePackageAfterDay[packageId].collectingPointId;
                    CollectingPointDto collectingPointCurrentLocation = collectingPoints
                        .Where(c => c.CollectingPointId == packageToUpdate.CollectingPointId).FirstOrDefault(); ;
                                                //מחוק את הנקודת איסוף המקור של החבילה שכעת באמצע הדרך//
                    CollectingPointDto collectingPointSource = collectingPoints
                        .Where(c => c.PackageId == packageToUpdate.PackageId
                        && c.ColPointType == CollctingPointType.source).FirstOrDefault();
                    collectingPointSource.State = 0;
                    tasks.Add( _collectingPoints.UpdateAsync(collectingPointSource));//מוחקת את נקודת המקור

                }
```

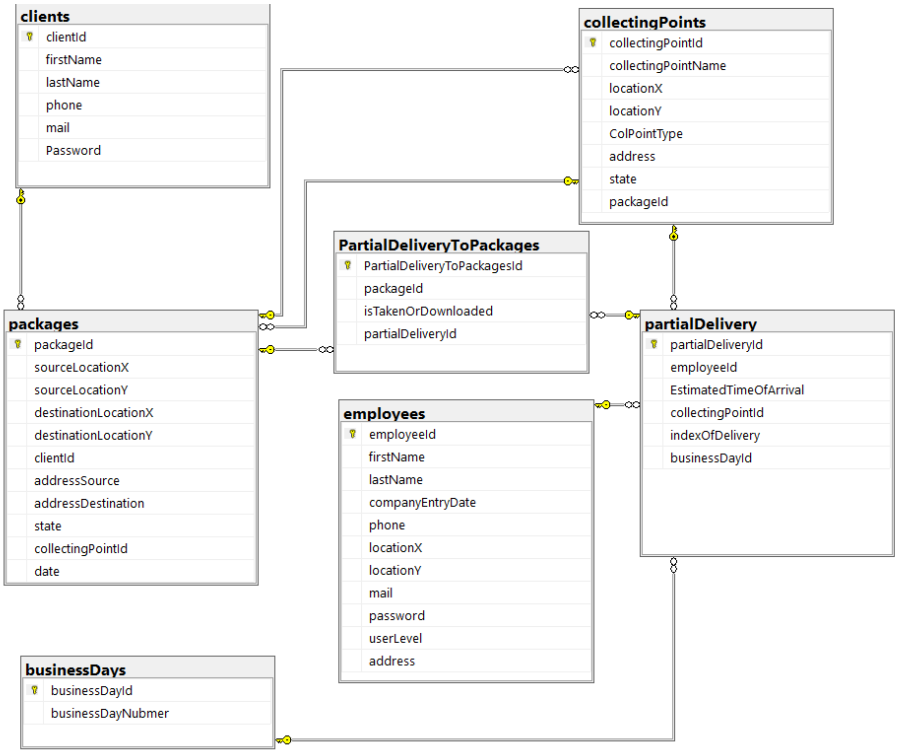
```
            if (packageToUpdate.State==(int?)PackageStatusEnum.Delivered ||//כאשר החבילה הסתימה
                packageToUpdate.State == (int?)PackageStatusEnum.Awaiting)//גם כאשר לא התחילה כי מחר יכניס שוב
            {
                //אם חבילה הסתימה צריך למחוק את נקודות המקור והיעד שנוצרה בגלל חבילה זו
                CollectingPointDto source=collectingPoints
                    .Where(c=>c.PackageId == packageToUpdate.PackageId
                    && c.ColPointType==CollctingPointType.source).FirstOrDefault();

                CollectingPointDto destination = collectingPoints
                    .Where(c=>c.PackageId == packageToUpdate.PackageId
                    && c.ColPointType==CollctingPointType.destination).FirstOrDefault();
                source.State = 0;
                destination.State=0;
                tasks.Add( _collectingPoints.UpdateAsync(source));//מוחקת כביכול את נקודות המקור והיעד
                tasks.Add( _collectingPoints.UpdateAsync(destination));
            }
             tasks1.Add( _packages.UpdateAsync(packageToUpdate));
        }

    }
    Task.WaitAll(tasks.ToArray());
    Task.WaitAll(tasks1.ToArray());
```

21. תיאור מסד הנתונים

21.1 תרשים טבלאות + קשרי גומלין:



21.2 פירוט:

- **טבלת יום עסקים BusinessDay:**

בשביל לספור כמה ימי עסקים עברו מיום הקמת החברה, אי אפשר לדעת רק עפ"י התאריך כי יתכן והיו ימי חופש כגון שבת וחגים שאינם נחשבים ליום עבודה. טבלה זו נועדה לספור את ימי העסקים, רק יום עסקים ממוספר אצלה.

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
BusinessDayID	int	מזהה יום עסקים	PK	V
BusinessDayNumber	int	מספר יום העסקים החל מהיום הראשון בו החברה החלה לעבוד		V

- **טבלת לקוח Client :**

הטבלה שומרת את פרטי הלקוח

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
ClientId	int	מזהה לקוח	PK	V
FirstName	string	שם פרטי		V
LastName	string	שם משפחה		V
Phone	string	טלפון		
Mail	string	כתובת מייל		V
Password	string	סיסמה		V

- **טבלת עובד Employee:**

הטבלה שומרת את פרטי העובד

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
EmployeeID	int	מזהה עובד	PK	V
FirstName	string	שם פרטי		V
LastName	string	שם משפחה		V
CompanyEntryDate	DateTime	היום בו נכנס לעבוד בחברה - ותק		
Phone	string	טלפון		
LocationX	decimal	ייצוג כתובת על ציר ה x	FK	
LocationY	decimal	ייצוג כתובת על ציר ה y		
UserLever	int	רמת משתמש		
Mail	string	כתובת מייל		V
Password	string	סיסמה		V

Address	string	כתובת		V

- **טבלת נקודת איסוף CollectingPoints:**

הטבלה שומרת את פרטי נקודת האיסוף של החבילות שפזורות ברחבי הארץ

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
CollectingPointId	int	מזהה נקודת איסוף	PK	V
CollectingPointName	string	שם נקודת איסוף		V
LocationX	Decimal	ייצוג כתובת על ציר ה-X		
LocationY	decimal	ייצוג כתובת על ציר ה-Y		
Address	string	כתובת		V
CollectingPointType	int	סוג נקודת איסוף		
State	int	מצב נקודת איסוף-קיים/ לא קיים		
PackageId	int	אם זה נקודת מקור/ יעד- לאיזה חבילה שייכת הנקודה	FK	

- **טבלת חבילות Packages:**

הטבלה שומרת את הנתונים של החבילות

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
PackageID	int	מזהה חבילה	PK	V
AddressSource	string	כתובת מקור		V
SourceLocationX	decimal	ייצוג כתובת על ציר ה-X		
SourceLocationY	decimal	ייצוג כתובת על ציר ה-X		
AddressDedtination	string	ייצוג כתובת על ציר ה-X		V
DedtinationLocationX	decimal	ייצוג כתובת על ציר ה-X		
DedtinationLocationY	decimal	ייצוג כתובת על ציר ה-X		
ClientID	int	מזהה לקוח	FK	V
Date	DateTime	תאריך שבו הוזמנה החבילה		
State	int	מצב החבילה-ממתינה למשלוח, הועברה חלקית, הועברה ליעדה		
CollectingPointId	int	אם במצב שהחבילה הועברה חלקית-באיזה נקודת איסוף היא ממתינה ליום המחר	FK	

- **טבלת מסלול חלקי PartialDelivery:**

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה

PartialDeliveryId	Int	מזהה מסלול חלקי	PK	V
EmployeeId	int	מזהה עובד	FK	V
CollectingPointId	int	מזהה נקודת איסוף	FK	V
IndexOfDelivery	Int	מספר מסלול חלקי מהמסלול השלם		
EstimetedTimeOfArrival	dateTime	השעה שבה העובד צריך להגיעה לנקודת האיסוף		
BusinessDayId	int	מזהה יום עסקים	FK	

- **טבלת מסלול חלקי לחבילה PartialDeliveryToPackages:**

שם שדה	סוג שדה	תיאור	מפתחות	שדה חובה
PartialDeliveryToPackageId	Int	מזהה מסלול חלקי לחבילה	PK	V
PartialDeliveryId	Int	מזהה מסלול חלקי	FK	V
PackageId	int	מזהה חבילה	FK	V
IsTakenOrDownloaded	Int	האם צריך לקחת את החבילה או להוריד		

22. מדריך למשתמש

ברוכים הבאים למערכת ניהול המשלוחים שלנו! מדריך למשתמש זה ינחה אותך בתהליך השימוש באתר שלנו. המערכת שלנו מנהלת חברת שליחויות ייחודית המאפשרת לנהל את פעולות המשלוח שלהם ביעילות.

מתחילים

כדי להשתמש באתר שלנו, עליך ליצור חשבון. אם אתה לקוח חדש, תוכל ללחוץ על כפתור "SignIn" בדף הכניסה ליצירת חשבון חדש. אם אתה לקוח קיים, אתה יכול להזין את האימייל והסיסמה שלך כדי להיכנס.

משתמשי המערכת:

למערכת שלנו שלושה סוגי משתמשים: מנהל, עובד ולקוח. לכל משתמש יש גישה לדפים ויכולות שונות.

מנהל נכבד

כמנהל, יש לך גישה לאפשרויות הבאות:

1. פתיחת יום עסקים חדש: ניתן לפתוח יום עסקים חדש בלחיצה על כפתור "פתח יום עסקים". זה ייצור יום חדש במערכת לפי שעות הפתיחה והסגירה שתכניס למערכת

2. הוספת נקודות איסוף: ניתן להוסיף נקודות איסוף חדשות על ידי לחיצה על כפתור "הוסף נקודת איסוף". זה יפתח טופס שבו תוכל להזין את הפרטים של נקודת האיסוף החדשה, כמו הכתובת.

3. הוספת עובדים חדשים: ניתן להוסיף עובדים חדשים על ידי לחיצה על כפתור "הוסף עובד". זה יפתח טופס שבו תוכל להזין את הפרטים של העובד החדש, כגון שמו ופרטי ההתקשרות שלו.

עובד מסור

כעובד, יש לך גישה לאפשרות הבאה:

הצג את המסלול היומי שלך: תוכל לצפות במסלול היומי שלך על ידי לחיצה על כפתור "מסלול יומי". זה יראה לך רשימה של נקודות איסוף שאתה צריך לבקר, יחד עם החבילות שאתה צריך לאסוף או להוריד בכל נקודה.

לקוח יקר

כלקוח, יש לך גישה לאפשרות הבאה:

הזמנת משלוח חדש: ניתן להזמין משלוח חדש בלחיצה על כפתור "הזמנת משלוח". זה יפתח טופס שבו תוכל להזין את פרטי המשלוח, כגון מקומות האיסוף והמשלוח וגודל החבילה.

סיכום

מערכת ניהול המשלוחים שלנו נועדה להפוך את פעולות המשלוח שלך ליעילה יותר. על ידי ביצוע מדריך למשתמש זה, תוכל לנווט בקלות באתר האינטרנט ולגשת לפונקציות הדרושות לך. אם יש לך שאלות או בעיות, אל תהסס לפנות לצוות תמיכת הלקוחות שלנו לקבלת סיוע.

23. בדיקות והערכה

הבדיקות:

- בדיקה האם המערכת מסדרת מסלול יומי לעובדים בצורה הגיונית ויעילה.
- בדיקה האם הגרף בנוי כמו שצריך.
- בדיקה האם הנתונים השונים מתוספים למערכת כמו שצריך.

נוהל הבדיקה:

- הכנסת נתונים למסד נתונים.
- הפעלת האלגוריתם לבניית הגרף.
- הפעלת האלגוריתם האקראי.

תוצאת הבדיקה:

בתהליך הבניה של המערכת הופיעו טעויות ובאגים בביצוע, האלגוריתמים נבדקו שוב ושוב עד שתוקנו כל הבעיות.

לאחר בדיקות ותיקונים אלו והרצת האלגוריתם מספר פעמים על נתונים שונים נוכחתי לראות שאכן האלגוריתם עובד מצוין והכל מתבצע כנדרש.

24. ניתוח יעילות

24.1 ניתוח ויעילות אלגוריתם בניית הגרף וחישוב הנקודות איסוף השייכות לכל עובד:

כל הנוסחאות של חישוב זמן ריצה הבאות מתייחסות למשתנים הבאים:

n – עובדים

m – נקודות איסוף

p – חבילות

NUMOFOPTION- מספר הפעמים שמוגרלת אופציה אפשרית ליום עבודה

כדי לחשב את הנקודות איסוף הנמצאות באזור של כל עובד, ובשביל האלגוריתם לבנית האופציות האפשריות, החלטתי לשמור במטריצת עזר את המרחק בק"מ בין כל שתי נקודות איסוף, חישוב המרחק בין הנקודות הוא על ידי פונקציית עזר המחשבת את המרחק על ידי נוסחת Haversine, זמן הריצה של פונקציה זו הוא O(1), והפונקציה לחישוב המרחקים במטריצה רצה $O(m^2)$.

בניית הגרף, וחיבור הקשתות בגרף = O(n*m)

הפונקציה שמחשבת לכל עובד את הנקודות איסוף שברדיוס שלו= O(n*m) כיוון הפונקציה בודקת בשביל כל עובד, אלו מבין כל נקודות האיסוף נמצאות באזורו.

24.2 ניתוח ויעילות אלגוריתם לבניית אופציה אפשרית ליום עבודה:

הפונקציה המגרילה לכל עובד נקודות איסוף למסלול יומי תגריל מספר נקודות איסוף ביחס לזמן יום העבודה, מספר זה הוא קבוע די קטן ולכן פונקציה זו רצה בזמן O(n)

הפונקציה ה'מזיזה' כביכול את העובדים בגרף ומחשבת את הטיפול בחבילות בנקודה= O(n*n) אמנם הפונקציה שמחשבת את הטיפול בחבילות עוברת בלולאה על החבילות שנמצאות אצל העובד, ובנקודת האיסוף, כיוון שמספר החבילות שיכול להימצא שם הוא קטן וזניח לעומת מספר החבילות הקימות במערכת.

24.3 ניתוח ויעילות אלגוריתם של חישוב ובחירת האופציה הטובה מיותר מבין כל האופציות

הפונקציה שמגרילה את האופציות האפשריות ובוחרת את האופציה הטובה ביותר = O(n *NUMOFOPTION) , חישוב הניקוד של כל אופציה= O(1), מספר האפשרויות המוגרלות הם- NUMOFOPTION, ולכן זהו זמן הריצה של האלגוריתם.

24.4 לסיכום

בנית הגרף ושאר הפעולות לפני הרצת האלגוריתם האקראי רץ O(n*m)

הגרלת כל האופציות האפשריות רץ O(n*m*NUMOFOPTION)

ולכן זמן הריצה של האלגוריתם הוא O(n*m*NUMOFOPTION)

25. אבטחת מידע

עובד ולקוח חייבים להיכנס למערכת ע"י כתובת מייל וסיסמא השמורים במערכת, כאשר

נרשמים בפעם הראשונה. סיסמת המנהל שמורה במסד נתונים.

26. מסקנות

כעת, בסיום כתיבת הספר, בו חשפתי עד כמה שאפשר את ההשקעה, המחשבה והלמידה שעומדים מאחורי פרויקט זה הגעתי לכמה מסקנות מגוונות.

ראשית, לאחר השקעה רבה מאד בבניית מסד נתונים בצורה נכונה וברורה, וכן בניתוח האלגוריתם על מנת שיהיה חכם ויעיל, חלוקה לוגית לשכבות ולפונקציות, וחשיבה מראש על הבעיות ומקרי הקצה שיכולים להיווצר, כאשר ניגשתי לשלב הביצוע, הדרך הייתה לי ברורה, מה שמראה של אף התיכנון הארוך והמייגע ההשקעה לא הייתה לריק וכל דקה של תכנון הייתה שווה, ועל כן הגעתי למסקנה כי פרויקט שאינו בנוי בצורה נכונה ושהבסיס שלו אינו יעיל מועד יותר לכישלון.

בנוסף, נתקלתי בתהליך בנית הפרויקט בתחומי ידע לא מוכרים או מורכבים. ההיתקלויות האלו גרמו להתנסות ולהשתפשף ולקבל פרקטיקה במציאת פתרונות שונים בבוחות עצמי, וכן לרכוש יכולת גבוהה של למידה עצמאית.

בתחילה, הקף הפרויקט שבניתי היה נראה גדול ומאיים, אך ראיתי שבסופו של דבר עם תיכנון טוב והצבת יעדים, הכל אפשרי. ראיתי שעם הרבה מוטיבציה, רצון וארגון נכון ניתן להשלים כל משימה.

את הפרויקט כתבתי עצמאית מהחל ועד כלה, כולל הכל! זו הייתה משימה שגרמה לי לעבוד קשה ולהתאמץ הרבה מאד אך יחד עם זאת מדהימה!! הכלים שרכשתי הם נכס ענק שנשאר איתי ושווה את הכל.

הפרויקט הזה קידם אותי באופן ניכר מבחינה מקצועית ואישית, והיווה אבן דרך משמעותית בהתמקצעותי בתחום הנדסת תוכנה.

27. פיתוחים עתידיים

המערכת שנכתבה ממוקסמת לזמן הנתון, וניתן בהמשך להוסיף לה פיתוחים שונים כדלהלן:

- התייחסות לנפח ומשקל החבילה שנכנסת למערכת, וכן לכלי התחבורה של העובדים.
- שיבוץ מקסימום חבילות ברכב של העובד.
- שדרוג ממשק משתמש

28. ביבליוגרפיה

https://en.wikipedia.org/
https://www.geeksforgeeks.org/
https://www.drawio.com/
https://developers.google.com/maps
https://mui.com/
https://www.youtube.com/