

# פרויקט גמר להנדסאי תוכנה

שם הפרויקט: Taxi Booking  
שם התלמידות: שרית נויגרשל ורחל סטביצקי  
בהנחיית: הגב' מירי ויכלדר  
התשפ"ה / 2025

## תוכן העניינים

### 1. הצעת פרויקט – עמוד 3

### 2. הגדרת דרישות ותיאור כללי

- תיאור ורקע כללי – עמוד 13
- תיאור תוכנת המערכת -
- תיאור חומרת המערכת
- תיאור פונקציית המערכת
- זרימת המידע במערכת

### 3. ממשק המשתמש

- כללי
- תיאור המסכים

### 4. מבנה הנתונים

- ארגון קבצים
- מבנה נתונים

### 5. תכנון

- כללי
- עקרונות התכנות
- פונקציות
- בקרת תוכנה
- אבטחת מידע

### 6. מה הקנה הפרויקט

### 7. ביבליוגרפיה

תאריך: 21.2.2025

לכבוד  
יחידת הפרויקטים  
מה"ט

**הצעה לפרויקט גמר**

**פרטי הסטודנט**

שם הסטודנט	ת.ז. 9 ספרות	כתובת	טלפון נייד	תאריך סיום הלימודים
שרית נויגרשל	214606501	מירון 5 בני ברק	0548404313	
רחלי סטביצקי	325978674	אבן גבירול 11 אלעד	0533168759	

- שם המכללה: שצ'רנסקי ת"א
- סמל המכללה:
- מסלול ההכשרה: הנדסאים
- מגמת לימוד: תכנות
- ביצוע הפרויקט: בסמינר ובבית

**1. פרטי המנחה האישי**

שם המנחה	כתובת	טלפון נייד	תואר	מקום עבודה/תפקיד
הגב מרים ויכלדר	מסליאנסקי 38 רמות	052-7158720	B.ED	



חתימת המנחה

חתימת הסטודנט ב'

חתימת הסטודנט א'

האישי

חתימת הגורם המקצועי מטעם מה"ט

## **2. שם פרויקט הגמר**

Taxi Booking

## **3. רקע**

### **• תיאור ורקע כללי**

בשנים האחרונות, שירותי הזמנת מוניות הפכו לנפוצים בעיקר דרך אפליקציות מובייל, מה שמגביל את הגישה לשירות עבור אוכלוסיות שאין להן פלאפונים חכמים או מתקשות להשתמש באפליקציות. נוסעים רבים, במיוחד מבוגרים ואנשים ללא גישה קבועה לטלפון חכם, מתקשים להזמין מונית באופן נוח ויעיל.

בנוסף, נהגי המוניות מתמודדים עם אתגרים כגון קבלת נסיעות לא רלוונטיות, זמני המתנה ארוכים בין נסיעות, וחוסר סנכרון בין הביקוש להיצע בזמן אמת.

לאור זאת, פיתחנו מערכת חדשנית להזמנת מוניות, המספקת פתרון נגיש הן לנוסעים והן לנהגים. המערכת מאפשרת לנוסעים להזמין מונית בצורה פשוטה, ללא תלות באפליקציות מובייל בלבד, ולנהגים לקבל נסיעות מסוננות בהתאם למיקומם ולזמינותם. המערכת משפרת את חוויית המשתמש, מייעלת את תהליך ההזמנה, ומאפשרת לכלל האוכלוסייה ליהנות משירות תחבורה נוח, מהיר ושוויוני יותר.

### **• מטרות המערכת**

**א. חיסכון במערכות נפרדות** – במקום להפעיל שתי אפליקציות נפרדות לנוסעים ולנהגים, המערכת שלנו מצליחה לשלב את שני השימושים בממשק אחד, ומקטינה את הצורך בהורדות רבות.

**ב. הנגשת שירותי הזמנת מוניות לכלל האוכלוסייה** – לאפשר לנוסעים להזמין מונית בצורה נוחה, גם ללא צורך בטלפון חכם או באפליקציות ייעודיות.

**ג. שיפור חוויית המשתמש לנוסעים ולנהגים** – לספק ממשק ידידותי ואינטואיטיבי שמאפשר הזמנה פשוטה ומהירה לנוסעים, תוך מתן מידע ברור לנהגים על הנסיעות הרלוונטיות עבורם.

**ד. הגדלת היצע הנסיעות עבור נהגים ושיפור ההכנסה** – לאפשר לנהגים לנצל את זמנם ביעילות, לקבל נסיעות קרובות יותר ולהפחית זמן נסיעה ללא נוסעים.

## **4. סקירת מצב קיים בשוק, אילו בעיות קיימות:**

- **תלות באפליקציות מובייל בלבד** – רוב שירותי הזמנת המוניות כיום מוגבלים לאפליקציות טלפון חכם, מה שמגביל את הגישה לשירות עבור אנשים שאין להם סמארטפון או מתקשים להשתמש בטכנולוגיה.

- **הצורך בהורדה והתקנה של אפליקציות מרובות** – רוב האפליקציות לשירותי מוניות דורשות הורדה של אפליקציות נפרדות לכל צורך, מה שיכול להיות מסורבל למשתמשים.

- **תקשורת בזמן אמת באמצעות צ'אט** – אחד הקשיים בשירותי מוניות הוא חוסר היכולת לתקשר בקלות בזמן אמת בין הנהג לנוסע, דבר שעלול לגרום לאי-הבנות או עיכובים.

## 5. מה הפרויקט אמור לחדש או לשפר:

- המערכת מאפשרת להזמין מונית גם דרך אתר אינטרנט, כך שגם אנשים ללא טלפון חכם או כאלו שלא רגילים לאפליקציות יכולים להזמין מונית בקלות.
- האתר מציג פתרון אחד שבו הנוסע והנהג יכולים לגשת למערכת בקלות, בלי הצורך בהורדת אפליקציות נוספות.
- המערכת שלנו מספקת **צ'אט בזמן אמת** בין הנהג לנוסע, כך שניתן יהיה לתאם שינויים מיידיים בנסיעה, כמו שינוי מיקום או עדכון אחר, בקלות ובמהירות.

## 6. דרישות מערכת ופונקציונאליות:

### 1. דרישות מערכת, סביבת הטמעה ושימוש:

דרישות מערכת: מחשב PC עם מערכת win10 ומעלה, חיבור לאינטרנט.

### שרידות, ביצועים / התמודדות עם עומסים:

הנתונים ישמרו ב SQL DB שזהו מסד נתונים המתמודד עם עומסים כבדים.

### 2. דרישות פונקציונאליות:

צד לקוח (Frontend):

- הזמנת נסיעות.
- צאט בזמן אמת
- הצגת היסטוריית נסיעות
- סינון לפי מיקום וזמינות
- הודעה למשתמש על סטטוס הזמנה
- ניהול פרטי משתמש

צד שרת (Backend):

- הזמנת נסיעה והקצאת נהג
- תיאום נסיעות לנהג
- ניהול צאט
- ביטול נסיעות ועדכון סטטוס
- שליחת הודעות לנוסעים ונהגים
- יצירת פרופילי נהגים ונוסעים

## 7. אופציות להרחבה

1. התחברות ל Google API Services – google maps, google pay

2. Live chat - לתקשורת בין הנהג לנוסע

3. הוספת זכאות לנסיעת חינוך – עבור נוסעים ותיקים

4. הצגת המסלולים על המפה ותנועה תוך כדי נסיעה

### 8. בעיות צפויות במהלך הפיתוח ופתרונות:

- בעיה: ההודעות בצאט וכן ההודעה לנוסע שבקשתו בוצעה לא התעדכנו בזמן אמת  
פתרון: שימוש ב socket.io
- בעיה: להציג את המונית בהתקדמות תוך כדי נסיעה  
פתרון: שימוש נכון בפונקציות של @react-google-maps/api

### 9. פתרון טכנולוגי נבחר:

#### 1. טופולוגית הפתרון:

ארכיטקטורת Client-Server עם API מבוסס REST.

#### 2. טכנולוגיות בשימוש:

- **Front:** React.js, HTML, CSS
- **Back:** Node.js, Express
- **Database:** MySQL
- **Extras:** Socket.IO, Google Maps API, axios

### 3. תיאור הארכיטקטורה הנבחרת:

המערכת פועלת על פי ארכיטקטורת Client-Server, כאשר צד הלקוח נבנה ב-React.js ומתקשר עם צד השרת שנבנה על ידי Node.js באמצעות API RESTful. המידע נשמר ב-MYSQL.

### 4. חלוקה לתוכניות ומודלים:

- Frontend: React.js
- Backend: Node.js

### 5. סביבת שרת:

סביבת שרת מקומית המסופקת ע"י VS.

אם האתר יירכש ע"י לקוח, נעלה אותו לשרת אירוח מתאים, כגון **Microsoft Azure, AWS**, או נשתמש ב **Docker**-לניהול והפצת היישום בסביבה מבודדת וניתנת להרחבה.

### 6. ממשק המשתמש/ לקוח GUI :

שכתב ה GUI מורכבת מדפי HTML שמוצגים למשתמש דרך הדפדפן.

## 7. ממשקים למערכות אחרות / API:

התממשקנו לשרותים שונים של גוגל כמו Google Maps, Google Pay

## 8. שימוש בחבילות תוכנה:

Entity framework, CSS כמו כן במהלך הפיתוח נשתמש ב NPM להתקנת חבילות תוכנה נוספות באם תידרשנה.

## 10. ממוש במבני נתונים וארגון קבצים:

### 1. נתונים:

#### Drivers

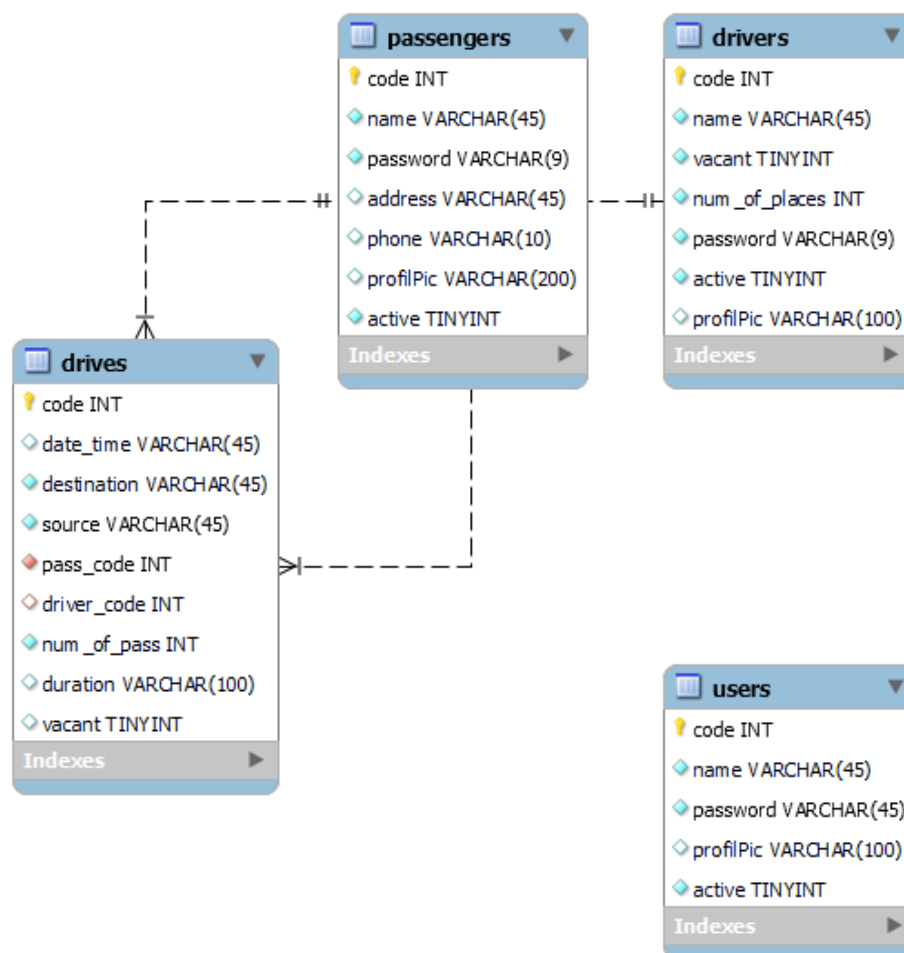
- מזהה ייחודי לנהג - code.
- שם - name.
- האם הנהג פנוי (בוליאני) - Vacant.
- מספר המקומות - num\_of\_places.
- סיסמה - password.
- שדה המאפשר מחיקה ללא איבוד נתונים - active.
- תמונת פרופיל - profilePic.

#### Drives

- מזהה ייחודי לנסיעה - code.
- תאריך ושעת נסיעה - dateTime.
- יעד הנסיעה - destination.
- נקודת היציאה - source.
- קוד נוסע - Pass\_code.
- קוד נהג - Driver\_code.
- מספר נוסעים - Num\_of\_pass.
- משך הנסיעה - Duration.
- האם הנסיעה נתפסה או שעדיין ממתינה לנהג - Vacant.

#### Passengers

- מזהה ייחודי לנוסע - code.
- שם - name.
- סיסמה - password.
- כתובת - adress.
- טלפון - phone.
- תמונת פרופיל - profilePic.
- שדה המאפשר מחיקה ללא איבוד נתונים - active.



## 2. שיטת אחסון

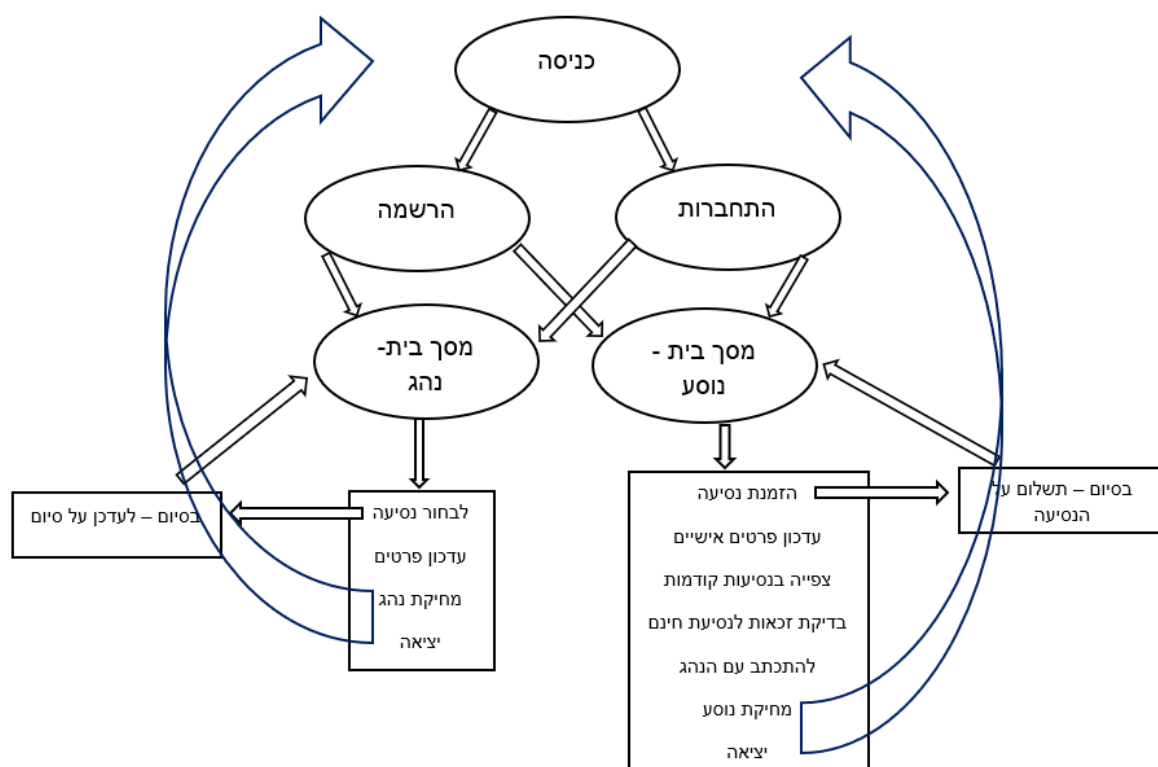
שיטת אחסון: הנתונים יאוכסנו במסד נתונים מסוג SQL Server.

## תמיכה/ נפילה

SQL Server - בנוי לתמיכה במס' גדול של קריאות בו זמניות ואין חשש לקריסה ברמה של הפרויקט



## זרימת המידע במערכת



## תיאור המרכיב האלגוריתמי – חישובי :

האלגוריתם אחראי על התאמת נסיעות לנהגים ולנוסעים בצורה אופטימלית, תוך התחשבות בפרמטרים כמו מיקום, זמינות והעדפות משתמשים.

### **שלבי התהליך:**

1. **קלט מהמשתמש** – הנוסע מזין את כתובת היעד, מיקום האיסוף והעדפות נוספות (גודל רכב, נגישות וכו').
  2. **שליפת נתונים** – המערכת לא מחפשת נתונים מראש, אלא מחכה שהנהג יבחר נסיעה מתוך הרשימה הזמינה.
  3. **התאמת נסיעה** – כשהנהג בוחר את הנסיעה המתאימה לו, המערכת מקצה את הנסיעה אליו ומעדכנת את הנוסע שהמונית בדרכו.
  4. **עדכון סטטוס** – ברגע שהנסיעה מאושרת, היא נשמרת בשני לוחות המשתמשים – לוח הנהג ולוח הנוסע, וסטטוס הנסיעה משתנה בהתאם.
- האלגוריתם גם מנהל **סינון חכם** לנהגים, כך שכל נהג מקבל מספר הצעות נסיעה בהתאם למיקומו וזמינותו, ומעדכן את הנוסע ברגע שהמונית בדרכו אליו.

### 11. תיאור/התייחסות לנושאי אבטחת מידע:

- ריבוי משתמשים וריבוי קריאות ואבטחה:  
כדי לדאוג לריבוי משתמשים וריבוי קריאות ואבטחה, בחרנו להשתמש ב-SQL Server המטפל בדרישות אלו.
- מקרים ותגובות לאבטחה:
  - הרשמה למערכת: במידה שמשתמש חדש מנסה להיכנס כמשתמש רשום, המערכת תפנה אותו למערכת הרשמה לאתר.
  - סיסמאות: במקרה שבעת כניסת משתמש הסיסמה אינה תואמת לשם המשתמש שהוקש, המערכת תציג לו הודעת שגיאה ולא תאפשר כניסה.
  - הסיסמה תהיה מוסתרת במערכת.
  - המפתח שמאפשר את הגישה לגוגל MAPS לא הועלה לגיט.
- חלוקה לממשקים שונים:  
ישנה חלוקה ברורה בין הממשק של בעל נוסע לבין הממשק של הנהג, כדי למנוע שימוש בפונקציות שאינן מורשות למשתמש.

### 12. תכנית עבודה ושלבים למימוש הפרויקט

1. אפיון ותכנון:
  - אפיון הדרישות והפונקציות המרכזיות.
  - תכנון ארכיטקטורת המערכת.
2. כתיבת צד שרת וצד לקוח:
  - פיתוח צד השרת (Node.js ו-Express).
  - פיתוח צד הלקוח (React).

- 3. בדיקות תוכנה:  
בדיקות של המערכת ווידוא תקינות כל הפונקציות.
- 4. כתיבת ספר פרויקט:

### 13. משאבים הנדרשים לפרויקט

**מספר שעות המוקדש לפרויקט:** 700 שעות .  
**ידע חדש שנדרש ללמוד לצורך ביצוע הפרויקט:**  
 העמקת ידע והיכרות עם הטכנולוגיות בשימוש בפרויקט  
 (React, Node.js, SQL Server, socket.io, google API).

#### **ספרות ומקורות מידע:**

- [W3Schools](https://www.w3schools.com/)
- [StackOverflow](https://stackoverflow.com/)
- [Codepen](https://codepen.io/)
- [MDN](https://developer.mozilla.org/)
- [Google Developer](https://developers.google.com/)

### 14. בקרת הפרויקט

תהליך בקרת הפרויקט מתבצע במתודולוגיה מסודרת תוך שימוש ב-Git לניהול קוד ושיתוף פעולה בצוות זוגי. להלן שלבי הבקרה המרכזיים:

1. ניהול קוד באמצעות **Git**
  - חלוקת העבודה בין הצוות, כאשר כל אחת אחראית על חלקים מוגדרים בקוד.
  - ביצוע משיכות (Pull Requests) ובדיקות קוד (Code Reviews) לפני מיזוג (Merge) לענף הראשי.
2. פגישות בקרה
  - מתקיימות פעמיים בשבוע לצורך מעקב אחר התקדמות העבודה.
  - כל משתתף מציג את ההתקדמות שלו, האתגרים בהם נתקל, ודרכי פתרון מוצעות.
  - נמדדים ההספקים ונקבעים היעדים להמשך הפיתוח.
3. מעקב ותיעוד
  - שימוש ב-Git לתיעוד היסטוריית השינויים והתקדמות הקוד.
  - ניהול משימות ותיעוד הערות באמצעות כלים תומכים
  - עדכון התכנון בהתאם לממצאים ולשינויים בפרויקט.

15. הערות ראש המגמה במכללה

---

---

---

---

16. אישור ראש המגמה

שם: \_\_\_\_\_ חתימה: \_\_\_\_\_ תאריך: \_\_\_\_\_

---

17. הערות הגורם המקצועי מטעם מה"ט

---

---

---

---

18. אישור הגורם המקצועי מטעם מה"ט

שם: \_\_\_\_\_ חתימה: \_\_\_\_\_ תאריך: \_\_\_\_\_

## הגדרת דרישות ותיאור כללי

### תיאור כללי

המערכת עוסקת בניהול והזמנת מוניות בזמן אמת, ומאפשרת למשתמשים נוסעים ונהגים – לבצע מגוון פעולות בהתאם לתפקידם. כל משתמש נכנס עם סיסמה אישית בהתאם להרשאותיו, ורואה את המידע והפעולות הרלוונטיות עבורו.

### משתמש נוסע

כל נוסע יכול להיכנס למערכת, לבחור יעד וגודל רכב, ולשלוח בקשת נסיעה. בנוסף, הוא יכול לצפות בפרטיו האישיים, עריכתם, מעקב אחרי נסיעה בזמן אמת, והטבות שהפלטפורמה מציעה.

### משתמש נהג

הנהג מקבל בקשות נסיעה מסוננות לפי קרבה גיאוגרפית למקור הנסיעה, בוחר אם לקבל את הבקשה, וצופה בפרטי הנוסע וביעד. כמו כן, הנהג רואה את כל היסטוריית הנסיעות שלו ויכול לעדכן את הפרופיל האישי שלו.

### היקף הפרויקט

כ- 700 שעות עבודה.

### משימות המערכת

- משימות המערכת מכילות אפשרויות שונות:
- הרשמה וזיהוי משתמשים (נוסעים ונהגים).
- שליחת בקשת נסיעה הכוללת פרטי רכב יעד ומקור
- קבלת רשימת נסיעות מסוננת על פי מיקום לנהגים
- הצגת מיקומים בזמן אמת על גבי Google Maps
- צפייה בהיסטוריית נסיעות עבור כל משתמש
- התכתבות לנוסע עם הנהג
- חישוב זכאות לנסיעה חינם עבור נוסע ותיק

## תיאור חומרת המערכת

### כללי

המערכת פותחה בטכנולוגיות:

- Front: React, html, css
- Back: Node.js, Express
- DB: MySQL

### מרכיבי המערכת

- מחשב קצה עם דפדפן אינטרנט .
- עכבר ומקלדת
- שרת מותאם לביצועים גבוהים

## תיאור תוכנת המערכת

### כללי

הפרויקט מתמקד בפלטפורמה חכמה להזמנת ושיבוץ מוניות,

המיועדת לשני סוגי משתמשים עיקריים: נוסעים ונהגים. המערכת מאפשרת יצירת וביצוע נסיעות

בצורה נוחה, יעילה ומבוססת מיקום, תוך שימוש בטכנולוגיות מתקדמות כמו

Google Maps, WebSockets ועוד.

מהלך המערכת:

בעת הכניסה למערכת מוצגת המפה והאופציה להזמין מונית. על מנת לבצע בפועל את ההזמנה וכן

לצפות בנתונים אישיים על הלקוח להתחבר, לאחר ההתחברות ההזמנה מתבצעת והבקשה עוברת

לרשימת הנסיעות לנהגים.

במקביל, יכול הנהג להכנס לאפליקציה, להתחבר ולצפות ברשימת נסיעות שמתדעכנת בזמן אמת ו

ומסוננת לו לפי קרבתו למיקום מקור הנסיעה, ברגע שנהג בחר נסיעה הוא מוגדר כתפוס ואינו יכול

לקבל נסיעות נוספות עד לסיום הנסיעה, ובנוסף הנוסע מקבל עדכון כי הנהג שלו בדרך.

במהלך הנסיעה הנוסע יכול לצפות במונית המתקדת על גבי המסלול עד להגעתה ליעד,

כשנגמרת הנסיעה הנהד מתפנה ויכול לקבל נסיעות נוספות.

### **כלי פיתוח**

- Frontend: React, HTML, CSS
- Backend: Node.js
- Database: MySQL
- IDE: Visual Studio Code

### **תיאור פונקציות המערכת**

#### **ניהול משתמשים**

- הרשמה וזיהוי משתמשים (נוסעים ונהגים)
- עריכת פרטים אישיים .

#### **ניהול מערכת הזמנות**

- שמירת פרטי נסיעה
- הצגת הנסיעות המסוננות לנהג
- ציוות נהג פנוי לנסיעה
- עדכונים לנוסע ולנהג בזמן אמת

### **תפעול Google Maps**

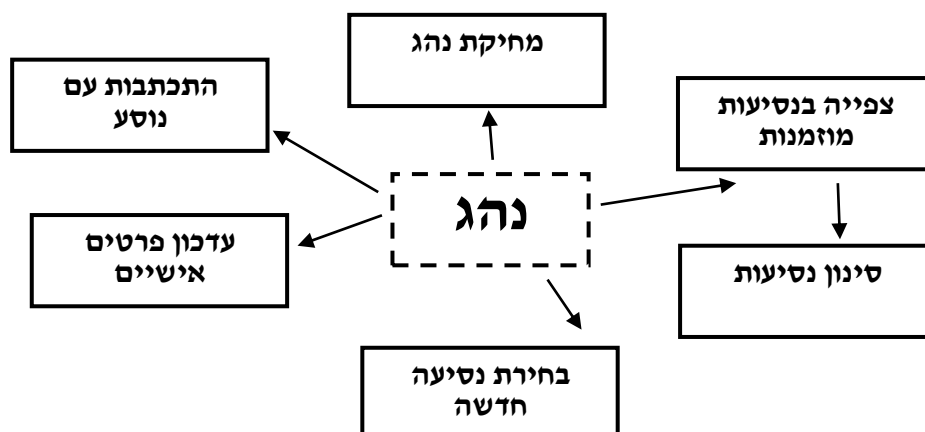
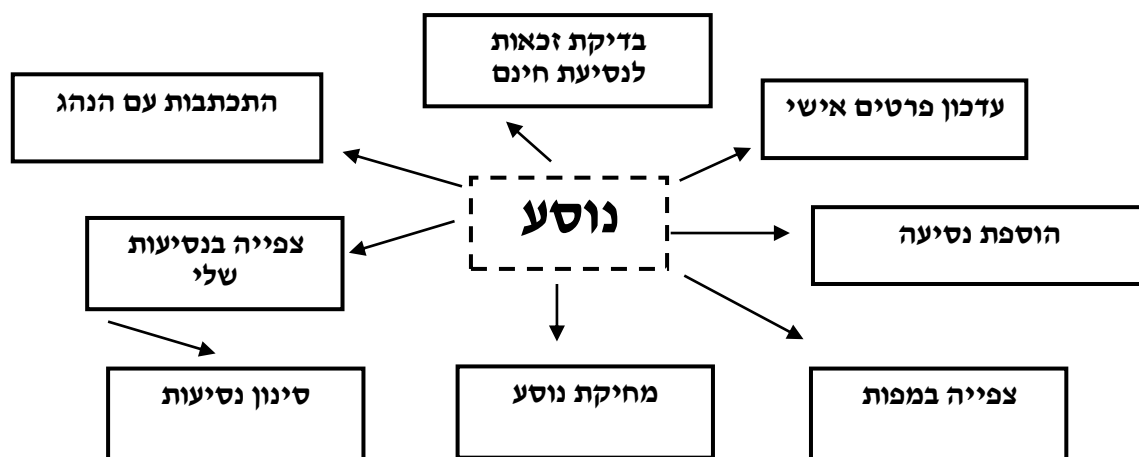
- הטמעת מפות
- הצגת מסלולים
- הצגת התקדמות הנסיעה
- בחירת מקור ויעד נסיעה מתוך המפה

### **כללי**

- תקשורת בין הנוסע לנהג על ידי ווטסאפ
- פונקציית חישוב זכאות לנסיעת חינם

## זרימת המידע הכללי במערכת

- נוסע מזמין נסיעה, צופה בהסטורית נסיעות, צופה בהתקדמות נסיעה פעילה, עורך פרטים אישיים.
- נהג עורך פרטים אישיים, מקבל נסיעה, צופה בהסטוריית נסיעות מתכתב עם הנוסע





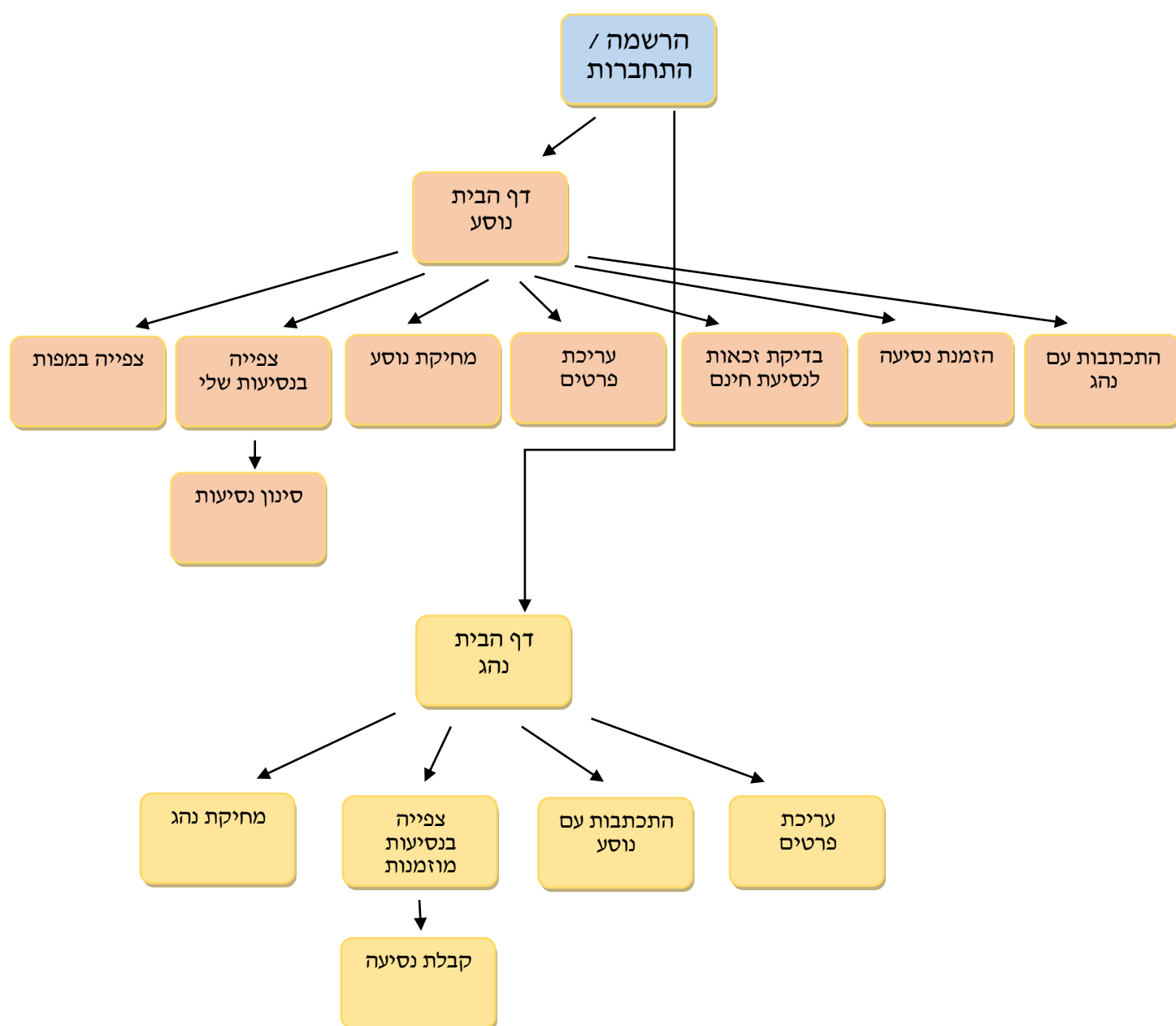
## ממשק משתמש

### כללי

הושם דגש רב על יעילות המערכת מגוון האפשרויות למשתמשים המעוניינים להזמין נסיעות ולנהגים

המקבלים אותן, וכן הושם דגש על עיצוב נעים ונקי לעין, ממשק ידידותי ונח כך שהשימוש באתר הוא

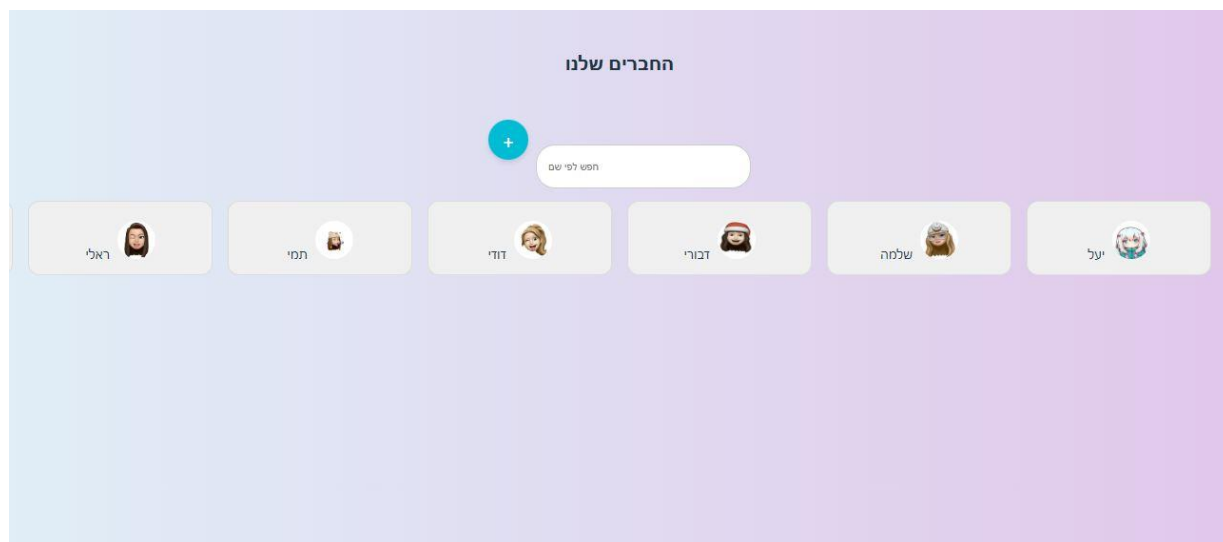
חווייה נעימה



## תיאור מסכים

### דף כניסה

מסך התחברות בו המשתמשים נכנסים ויכולים להתחבר או להרשם

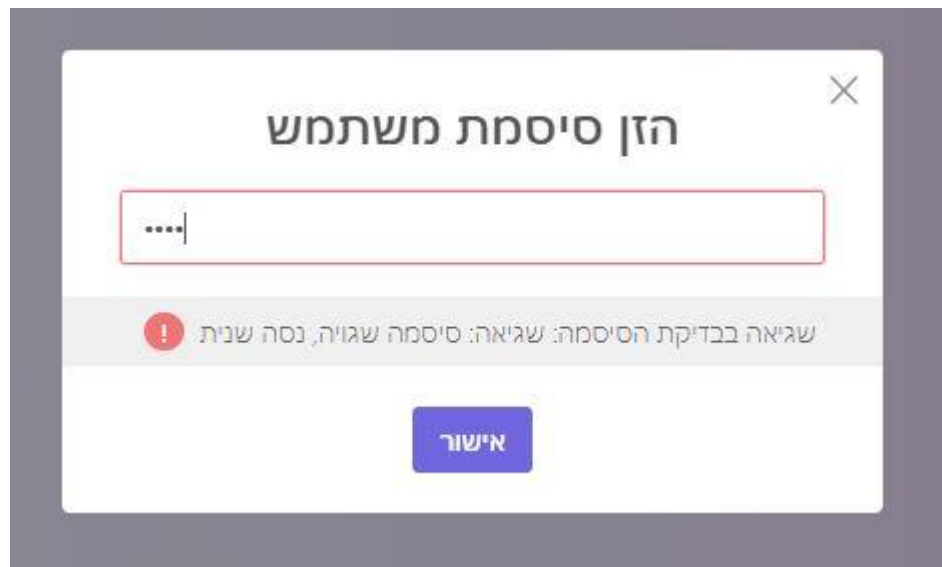


### דף התחברות

במידה והמשתמש כבר רשום ומעוניין להכנס עליו להזין סיסמה



עבור סיסמה שגויה מופיע ההודעה הבאה



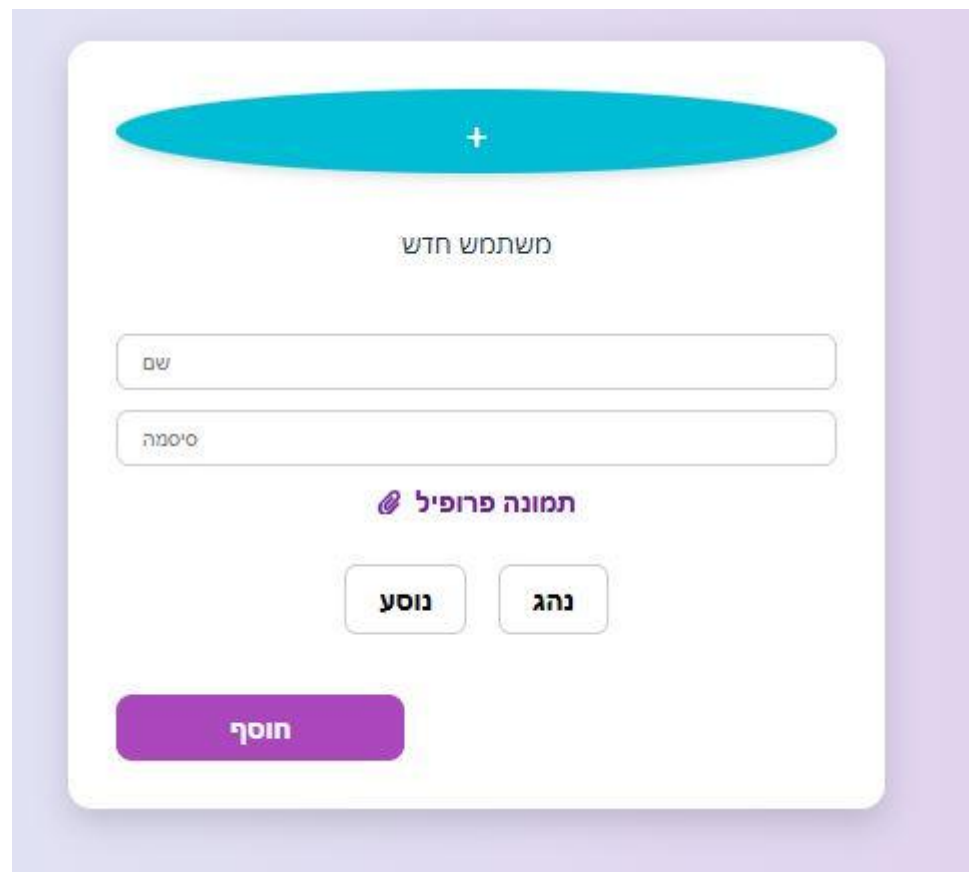
הזן סיסמת משתמש

שגיאה בבדיקת הסיסמה: שגיאה: סיסמה שגויה, נסה שנית.

אישור

## דף הרשמה

משתמש חדש נרשם בדף זה



משתמש חדש

שם

סיסמה

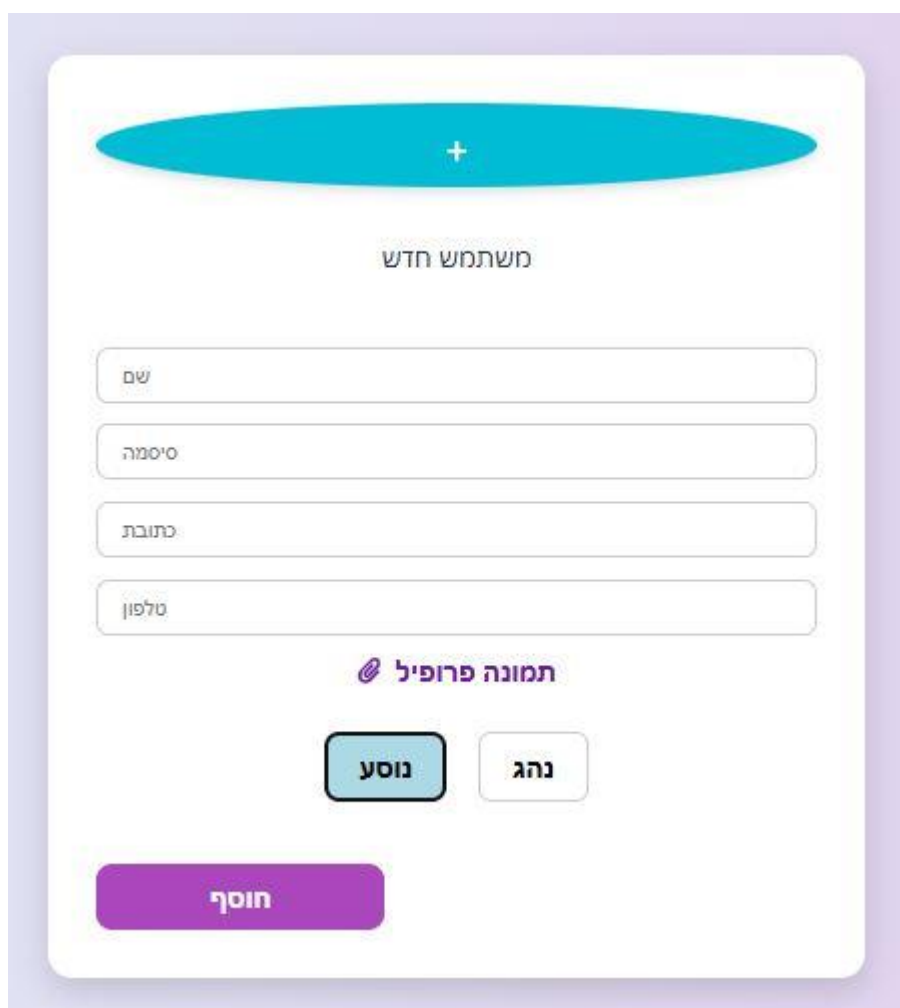
תמונה פרופיל

נוסע נהג

חוסף

בתחילה ממלא פרטים שרלוונטיים לכל סוגי הנוסעים ולאחר מכן עליו לבחור סוג משתמש ולמלא פרטים בהתאם לבחירתו.

## הרשמת נוסע



The registration form is titled "משתמש חדש" (New User) and is set against a light purple background. At the top is a blue oval with a white plus sign. Below the title are four input fields: "שם" (Name), "סיסמה" (Password), "כתובת" (Address), and "טלפון" (Phone). Below these fields is a purple icon of a person and the text "תמונה פרופיל" (Profile Picture). At the bottom are three buttons: a blue "נוסע" (Passenger) button, a white "נהג" (Driver) button, and a large purple "חוסף" (Host) button.

משתמש חדש

שם

סיסמה

כתובת

טלפון

תמונה פרופיל

נוסע

נהג

חוסף

## הרשמת נהג

+

חשתמש חדש

שם

סיסמה

מספר נוקמנות ברכב

תאריך לידה

תמונה פרופיל

נוסע

נהג

חוסף

לאחר מכן מוצגים דפי המשתמשים בהתאם לבחירה

## דף הבית נוסע

הזמנת נסיעה

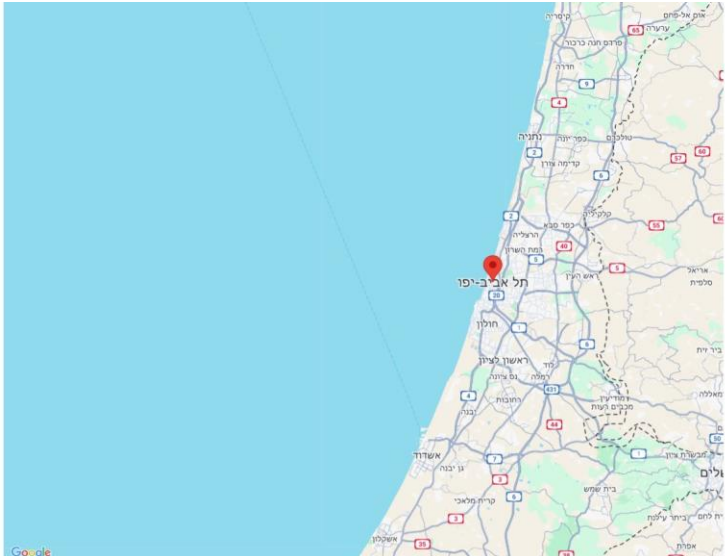
נקודת התחלה


נקודת יעד

גודל המונות

✓

הזמן





יעל

עדכון הפרטים

הנסיעות שלי

נסיעת חנים

מחיקת נוסע

logout

בדף זה מופיעה מפה, רובריקת הוספת נסיעה ושאר האופציות שמתאפשרות לנוסע  
רשימת האופציות

## הוספת נסיעה

### הזמנת נסיעה

נקודת התחלה

נקודת יעד

▼

גודל המונית

הזמן

לאחר ביצוע הזמנה, ההזמנה מצטרפת לרשימת הנסיעות וממתינה לנהג שיקבל אותה.  
אופציות נוספות לנוסע:

## עריכת פרטים

X

עריכת פרטים אישיים

raeli

שם

aa

כתובות

12345

טלפון

ערוך

## זכאות לנסיעת חינם

X

הסבר קצר:

נסיעת חינם ניתנת לכל לקוח שנסע מעל ל-30 שעות עם נהגים מהחברה שלנו

האם את/ה זכאים לנסיעת חינם? לחצו על הכפתור למטה ובדקו זאת!!

בדיקת זכאות

X

### הנסיעות שלי

מאיפה: בני ברק

לאן: אלעד

תאריך: 16-05-2024

מאיפה: רמת גן

לאן: אבן גבירול 11

תאריך: 16-05-2024

מאיפה: אבן גבירול


לאן: יצחק אלחנן 4

תאריך: 16-05-2024

מאיפה: קרית שמונה

לאן: תל אביב

תאריך: 16-05-2024



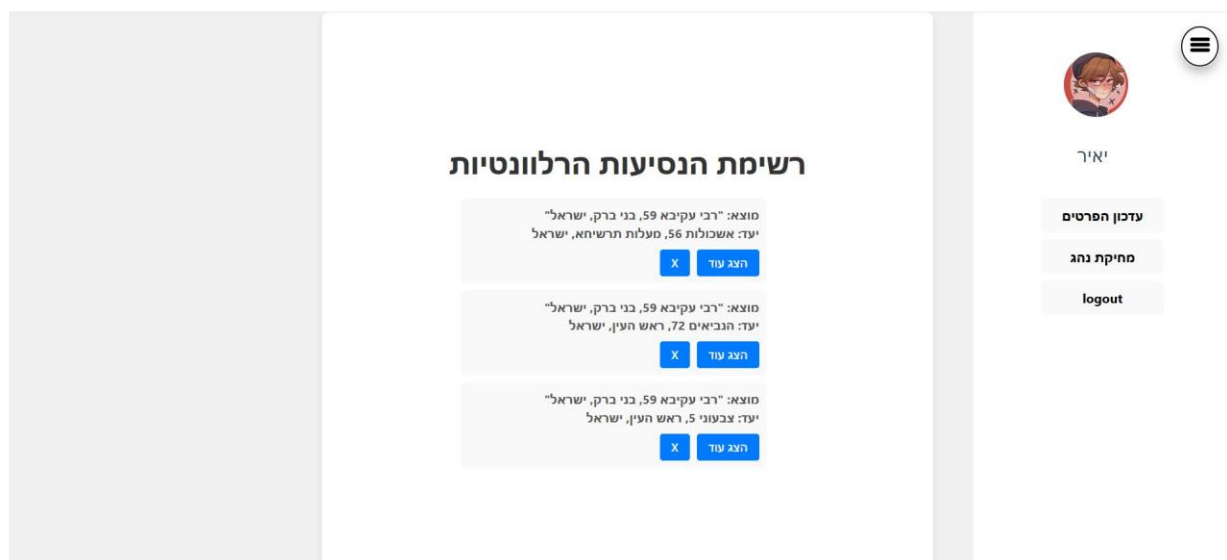


## מחיקת נוסע

קבלת אישור סופי למחיקה



## דף הבית נהג



## פירוט דף נהג

לנהג מוצגות רשימת נסיעות הלוונטיות לו ביותר – מסוננות על פי מרחק ממנו

### רשימת הנסיעות הלוונטיות

מוצא: "הזית 5, לוד, ישראל"  
יעד: אבן גבירול אלעד

X

הצג עוד

מוצא: "רבי עקיבא 59, בני ברק, ישראל"  
יעד: אשכולות 56, מעלות תרשיחא, ישראל

X

הצג עוד

מוצא: "רבי עקיבא 59, בני ברק, ישראל"  
יעד: הנביאים 72, ראש העין, ישראל

X

הצג עוד

בלחיצה על הצג עוד מופיעה הנסיעה עם כל הפרטים ועם אופציה לבחור אותה

מוצא: "הזית 5, לוד, ישראל"  
יעד: אבן גבירול אלעד  
מספר מקומות ברכב: 5  
סטטוס: פנוי

X

בחר נסיעה זו

לאחר שנבחרה נסיעה היא מוצגת לנהג ויש לו אופציה ללחוץ על סיום בסוף הנסיעה

### פרטי הנסיעה שנבחרה

מוצא: "הזית 5, לוד, ישראל"

יעד: אבן גבירול אלעד

תאריך: 1.1.1970

מספר מקומות ברכב: 5

סיום נסיעה

### עדכון פרטים אישיים

X

### עריכת פרטים אישיים

שם

איר

מספר  
מקומות

5

ערוך

## מחיקת נהג

אישור סופי לפני מחיקת נהג

### אישור מחיקת נהג

האם למחוק את הנהג יאיר?

אישור

ביטול

## מבנה נתונים

### ארגון קבצים

#### ארגון כללי

המערכת מתבססת על שרת. השואב את הנתונים והגדרות מבסיס נתונים של SQL Server, ניתן לגלוש לאתר מכמה מוקדים בו זמנית. בצד השרת מתבצעות העיבוד, השליפות של הנתונים והלוגיקה ובצד לקוח מתבצעת התצוגה וחלק מן הלוגיקה.

### ארגון קבצים

להלן תמונת ארגון הקבצים של הפרויקט.  
כאן חיברנו את מסד הנתונים לפרויקט.

```
1 import mysql from "mysql2"
2 const pool = mysql.createPool({
3   host: "127.0.0.1",
4   password: "root@123456",
5   user: 'root',
6   database: 'server_gettaxi'
7 }).promise();
8
9 export default pool;
```

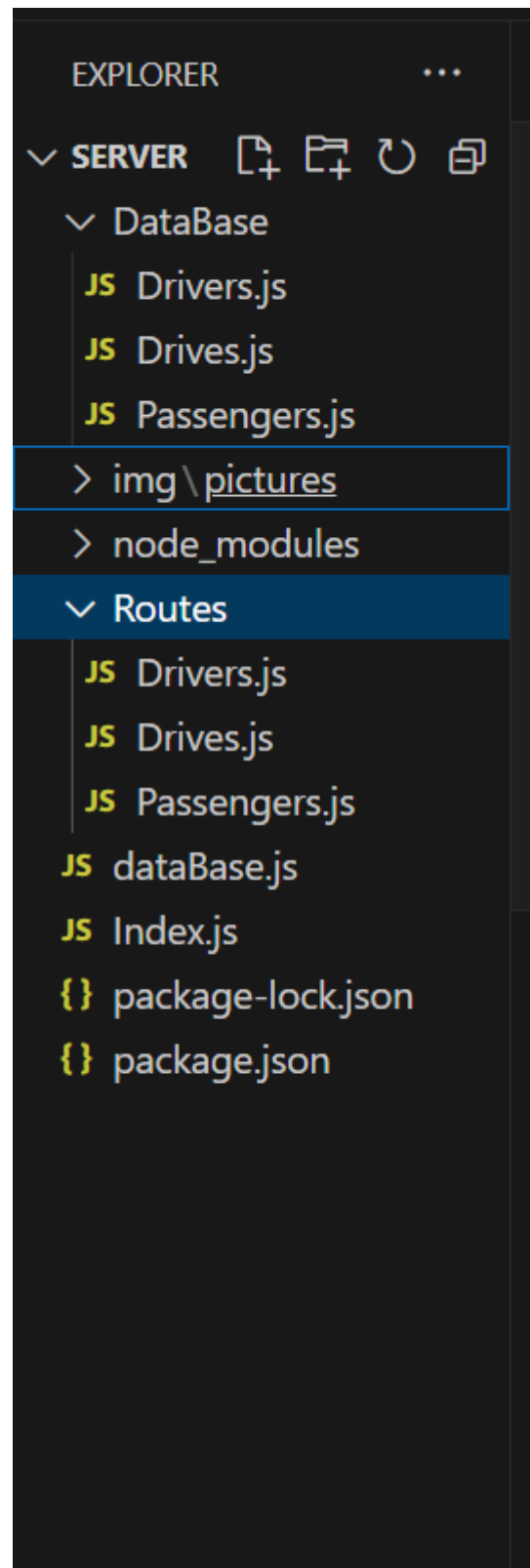
תיקיות השרת מסודרות:

בתוך תיקיית ה DATABASE נמצאות הפונקציות שמתעסקות עם הנתונים ב db עצמו.

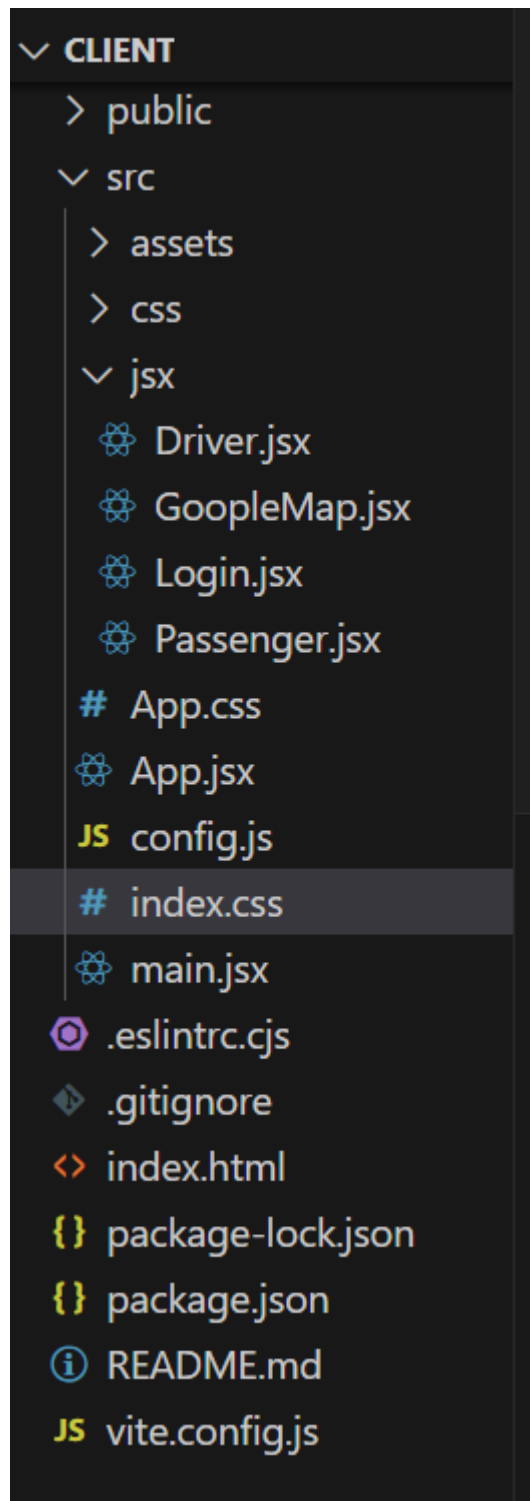
בתוך תיקיית ה INDEX נמצאות הפונקציות שנשלחות מהלקוח- פונקציות השרת. (חיבור מסד הנתונים עם הלקוח)

בתוך תיקיית ה Routes נמצאות פונקציות של חיבור השרת עם הלקוח.

בתוך תיקיית ה img נמצאות התמונות שהועלו על ידי המשתמש.



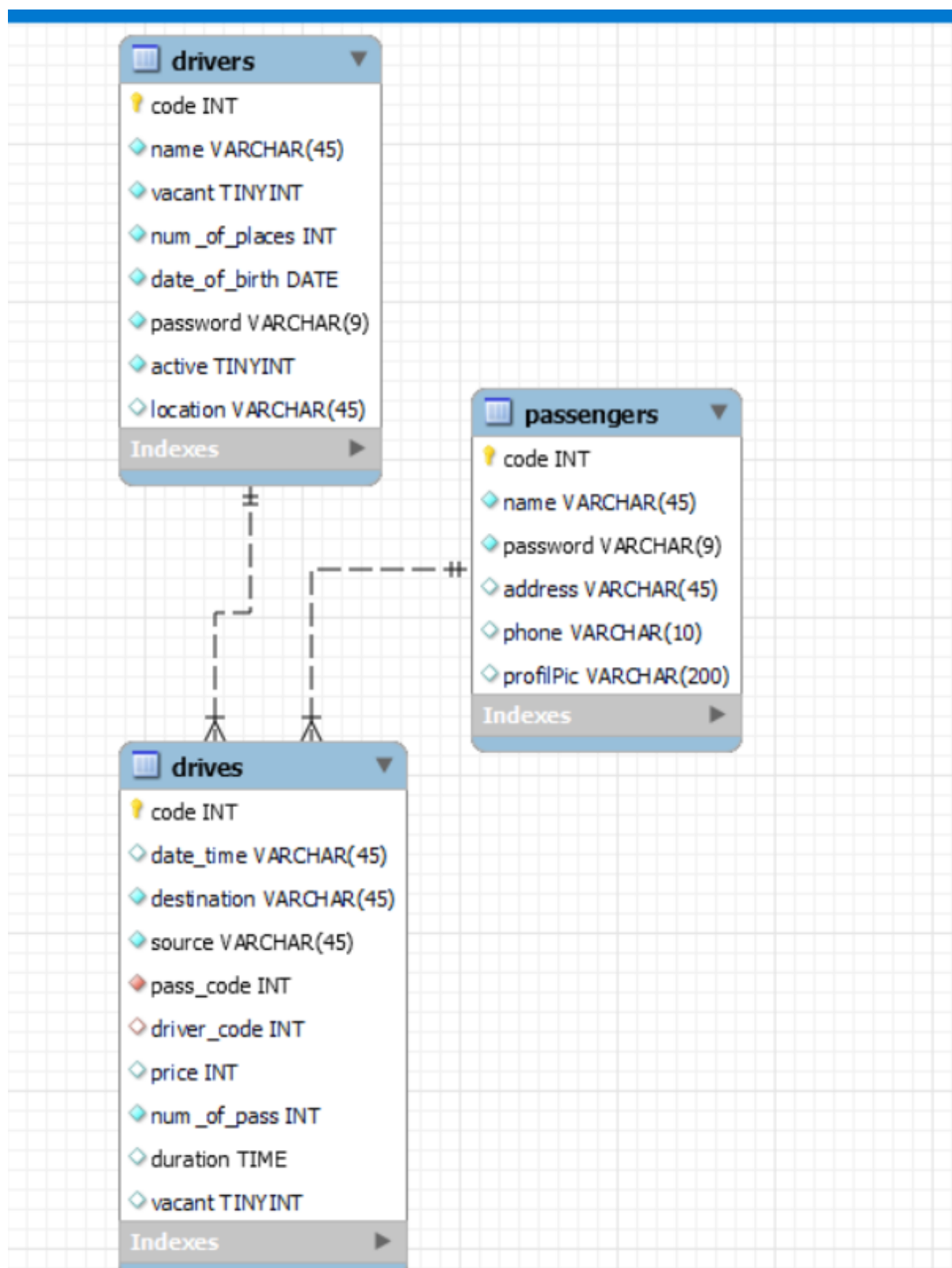
ארגון הפרויקט בצד הלקוח- REACT  
בתוך תיקיית - jsx נמצאות הקומפוננטות.



## מבנה נתונים

המערכת מבוססת על מסד נתונים המורכב מטבלאות המכילות מידע אודות פרטי המשתמשים, הנהגים, והנסיעות.

האתר מקושר למסד נתונים באמצעות Framework Entity, כל ישות הינה טבלה המתורגמת למחלקה בצד השרת. שמות הטבלאות וקשריהן כמתואר בדיאגרמה.





## **תכנון**

### **כללי**

מטרתנו בכתיבת הפרויקט הינה : אפשרות להנעים את חווית המשתמש שאינו בעל רכב משל עצמו ויש לו צורך להשתמש וכן להנעים את עבודתם של הנהגים תוך שמירה על נתוני הנסיעות ופרטי המשתמשים והנהגים כך שתיווצר החוויה הטובה ביותר עבור שני הצדדים..

לצורך כך חילקנו את הפרויקט לשני חלקים : נהג ומשתמש. שניהם מקושרים למסד נתונים וכל אחד יכול לראות את הפרטים המותאמים לתפקידו. דאגנו לעיצוב נעים לעין, הוראות תפעול קלות וברורות ולהפשטה של התהליכים המתבצעים, הקפדנו על קלות השימוש ומגוון אפשרויות ועוד נקודות רבות שהפכו את השימוש באתר ומועיל ומהנה במיוחד.

## **עקרונות התכנות**

### **כללי**

צד שרת : sql server,node.js  
צד לקוח : react, html, css

### **עקרונות התכנות**

הפרויקט נכתב בהתייחסות רבה לזמן ריצה קצר . קוד קצר וחסכוני, קריא ומתועד. הושם דגש על יעילות הקוד הימנעות מכפילויות.

### **פונקציות**

```

9   const Login = () => {
10     const [allUsers, setAllUsers] = useState([]);
11     const [hoveredUser, setHoveredUser] = useState(null);
12     const [currentUser, setCurrentUser] = useState(null);
13     const [file, setFile] = useState(null);
14     const [isAddUser, setIsAddUser] = useState(false);
15     const [isAddingPassenger, setIsAddingPassenger] = useState(false);
16     const [isAddingDriver, setIsAddingDriver] = useState(false);
17     const [userDetails, setUserDetails] = useState({
18       name: '',
19       password: '',
20       address: '',
21       phone: '',
22       vacant: '',
23       numOfPlaces: '',
24       dateOfBirth: '',
25       profilPic: ''
26     });
27     const [originalUsers, setOriginalUsers] = useState([]);
28     const navigate = useNavigate();
29
30     // שליפת כל המשתמשים
31     useEffect(() => {
32       axios.get('http://localhost:8080/passengers')
33         .then(response => {
34           const passengers = Array.isArray(response.data) ? response.data : [];
35           return axios.get('http://localhost:8080/drivers')
36             .then(response => {
37               const drivers = Array.isArray(response.data) ? response.data : [];
38             });
39         });
40
41     // שליפת כל המשתמשים
42     ✓ useEffect(() => {
43       ✓ axios.get('http://localhost:8080/passengers')
44       ✓ .then(response => {
45         ✓ const passengers = Array.isArray(response.data) ? response.data : [];
46         ✓ return axios.get('http://localhost:8080/drivers')
47           ✓ .then(response => {
48             ✓ const drivers = Array.isArray(response.data) ? response.data : [];
49             ✓ const allUsers = [
50               ✓ ...passengers.map(user => ({ ...user, type: 'passenger' })),
51               ✓ ...drivers.map(user => ({ ...user, type: 'driver' })));
52             ✓ ];
53             ✓ setAllUsers(allUsers);
54             ✓ setOriginalUsers(allUsers); // Save original users
55           ✓ });
56       ✓ });
57       ✓ .catch(error => {
58         ✓ if (error.response) {
59           ✓ alert(`Server failed: ${error.response.data}`);
60         } else if (error.request) {
61           ✓ alert('No response received from server');
62         } else {
63           ✓ alert(`Error: ${error.message}`);
64         }
65       ✓ });
66     }, []);

```

```
// פונקציית התחברות לחשבון
const handleSelect = (user) => {
  setCurrentUser(user);

  Swal.fire({
    title: 'הזן סיסמת משתמש',
    input: 'password',
    inputAttributes: {
      autocapitalize: 'off'
    },
    showCloseButton: true,
    closeButtonHtml: '<span style="font-size: 24px; color: #888;">&#10005;</span>',
    confirmButtonText: 'אישור',
    showLoaderOnConfirm: true,
    preConfirm: (pass) => {
      return axios.post('http://localhost:8080/passengers/checkPassword', {
        name: user.name,
        password: pass
      })
        .then(response => {
          if (response.data.isValid) {
            const newUser = { ...user };
            localStorage.setItem('currentUser', JSON.stringify(newUser));
            setCurrentUser(newUser);
            return "ברוך הבא";
          } else {
            throw new Error("סיסמה שגויה, נסה שנית");
          }
        })
        .catch(error => {
          Swal.showValidationMessage(
            `שגיאה בבדיקת הסיסמה: ${error}`
          );
        });
    },
    allowOutsideClick: () => !Swal.isLoading()
  }).then((result) => {
    if (result.isConfirmed) {
      navigate(`/${user.type}/${user.name}`);
    }
  });
};

// תפוסת הפרטים המוכנסים
const handleChange = (event) => {
  const { name, value } = event.target;
  setUserDetails((prevDetails) => ({
    ...prevDetails,
    [name]: value
  }));
};
```

## פונקציה להוספת משתמש חדש בתוספת בדיקות תקינות:

```
// הוספת משתמש חדש
const addUser = () => {
  const phoneRegex = /^d{10}$/;
  if (userDetails.name.length === 0) {
    alert("יש להזין שם");
  } else if (userDetails.password.length === 0) {
    alert("יש להזין סיסמה");
  } else if (!phoneRegex.test(userDetails.phone) && userDetails.phone.length !== 0) {
    alert('מספר טלפון לא תקין');
  } else {
    const data = new FormData();
    Object.keys(userDetails).forEach(x => {
      data.append(x, userDetails[x])
    })
    // הוספת התמונה
    data.append('profilPic', file)
    const url = isAddingPassenger ? 'http://localhost:8080/passengers' : 'http://localhost:8080/drivers';
    axios.post(url, userDetails).then(newUser => {
      localStorage.setItem('currentUser', JSON.stringify(newUser));
      setCurrentUser(newUser.data);
    }).catch(() => { alert('ERROR') });
    setIsAddUser(false);
  }
};
```

## חיפוש לפי שם משתמש:

```
// חיפוש לפי שם משתמש
const handleSearch = (event) => {
  const searchValue = event.target.value.toLowerCase();
  if (searchValue.trim() === '') {
    setAllUsers(originalUsers); // Restore original users
  } else {
    setAllUsers(prevUsers => prevUsers.filter(user => user.name.toLowerCase().includes(searchValue)));
  }
};
```

## הזמנת נסיעה:

```
// הזמנת נסיעה
const orderDrive = () => {
  if (!newDrive.driveSource || !newDrive.driveDest || !newDrive.num_of_pass) {
    alert('חסרים פרטים להזמנת נסיעה');
  } else {
    axios.post('http://localhost:8080/drives', newDrive).then((res) => {
      alert("הזמנת נוספה למערכת");
      setNewDrive(prevD => ({
        ...prevD,
        driveDest: '',
        pass_code: passenger.code,
        num_of_pass: '',
        duration: ''
      }));
      setDestination(null)
      window.location.reload();
    }).catch(error => console.error("Error fetching comments:", error));
  }
};
```

## שליפת כל הנסיעות שלי + שמירת הנסיעה הנבחרת

```
//שליפת כל הנסיעות שלי
useEffect(() => {
  axios.get(`http://localhost:8080/drives?code=${passenger.code}`)
    .then(res => {
      setMyDrives(res.data);
    })
    .catch(err => { alert(err) });
}, [passenger.code]);

//שמירת נסיעה נבחרת
const handleDriveClick = (drive) => {
  setSelectedDrive(drive);
};

const closeMyDrives = () => {
  setShowMyDrives(false);
  setSelectedDrive(null);
};

const logOut = () => {
  localStorage.removeItem('currentPassenger');
  navigate('/login');
};
```

פונקציה לבדיקת זכאות לנסיעה חינם:

```
const checkFreeDrive = () => {
  let totalMinutes = 0;
  myDrives.forEach(drive => {
    if (drive.duration !== null) {
      const durationParts = drive.duration.match(/\d+\/g);
      const hours = parseInt(durationParts[0]);
      const minutes = parseInt(durationParts[1]);
      totalMinutes += (hours * 60) + minutes;
    }
  });
  const totalHours = Math.floor(totalMinutes / 60);
  const remainingMinutes = totalMinutes % 60;
  setTotalDuration(` ${totalHours} שעות ו-${remainingMinutes} דקות`);

  setIsChecking(true);
  setEligibilityText('מחשב...');
  setTimeout(() => {
    setIsChecking(false);
    setEligibilityText('');
    if (totalHours > 30) {
      setEntitlement(true);
      return totalHours, remainingMinutes;
    }
    else {
      setNoEntitlement(true);
    }
  }, 1000);
}
```

## הצגת המפה ברקע:

```
// המפה ברקע
useEffect(() => {
  setKey("AIzaSyBfgzVdk3QnZZBbyu1tguleiguMLT1SQck")
  Geolocation.getCurrentPosition((pos) => {
    const crd = pos.coords;
    setPosition({
      lat: crd.latitude,
      lng: crd.longitude,
    });
    geocode(RequestType.LATLNG, `${crd.latitude},${crd.longitude}`)
      .then(({ results }) => {
        const address = results[0].formatted_address;
        setNewDrive(prevDrive =>
          ({
            ...prevDrive,
            driveSource: address
          })
        )
      })
      .catch(console.error);
  })
}, []);

const mapContainerStyle = {
  width: "100%",
  height: "100%",
}
```

## ציור המסלול על המפה:

```
const [arrivalTime, setArrivalTime] = useState(null)
const directionsCallback = (response) => {
  if (response !== null && response.status === 'OK') {
    setDirections(response);
    setDuration(response.routes[0].legs[0].duration.text.match(/\d+/g));
    setNewDrive((prev) => ({ ...prev, duration: duration }))
  } else {
    console.error('שגיאה בקבלת הנתונים', response);
  }
};
```

שינוי יעד ע"י לחיצה על המפה:

```
// ...  
const changeDest = (event) => {  
  const { latLng } = event;  
  const latitude = latLng.lat();  
  const longitude = latLng.lng();  
  const newDestination = { lat: latitude, lng: longitude };  
  setDestination(newDestination);  
  geocode(RequestType.LATLNG, `${latitude},${longitude}`)  
    .then(({ results }) => {  
      const address = results[0].formatted_address;  
      setNewDrive(prevDrive => ({  
        ...prevDrive,  
        driveDest: address  
      }));  
    })  
    .catch(console.error);  
};
```

שליפת נסיעות רלוונטיות לנהג:

```
// שליפת נסיעות רלוונטיות  
useEffect(() => {  
  fetchRelevantDrives(); // קריאה ראשונית לפונקציה  
  const intervalId = setInterval(fetchRelevantDrives, 100); // הגדרת אינטרוול ל-20 שניות  
  return () => clearInterval(intervalId); // ניקוי האינטרוול כשנטענת הקומפוננטה  
}, [driverDetails.code]);  
  
// Function to fetch relevant drives  
const fetchRelevantDrives = () => {  
  axios.get(`http://localhost:8080/drives/relevant-drives/${driverDetails.code}/${location.lat}/${location.lng}`)  
    .then(response => {  
      const relevantDrives = response.data;  
      setRelevantDrives(relevantDrives);  
    })  
    .catch(error => console.error('Error fetching relevant drives:', error));  
};  
  
useEffect(() => {  
  fetchRelevantDrives(); // קריאה ראשונית לפונקציה  
}, [driverDetails.code]); // תלות בקוד הנהג והמיקום
```



```

useEffect(() => {
  // Get current location using Geolocation API
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition((position) => {
      const { latitude, longitude } = position.coords;
      setLocation({
        lat: latitude,
        lng: longitude,
      });
    }, (error) => {
      console.error('Error getting current position:', error);
    });
  } else {
    console.error('Geolocation is not supported');
  }

  if (isLoading && selectedDrive && !isvacant) {
    displaySelectedDriveOnMap(selectedDrive);
  }
}, []);

```

## בחירת נסיעה ע"י הנהג:

```

const toggleVacantStatus = () => {
  if (selectedDrive) {
    selectedDrive.vacant = true;
    selectedDrive.driver_code = driverDetails.code
    selectedDrive.date_time = new Date();
    setIsVacant(false);
    // Update drive's vacant status on the server
    axios.put(`http://localhost:8080/drives/${selectedDrive.code}`, selectedDrive)
      .then(() => {
        localStorage.setItem('currentDrive', JSON.stringify(selectedDrive));
      })
      .catch(error => console.error("Error updating drive vacant status:", error));
    driverDetails.vacant = false;
    axios.put(`http://localhost:8080/drivers/${driverDetails.code}`, driverDetails)
      .then(() => {
        setDriverDetails(prevDriver => ({
          ...prevDriver,
          vacant: false
        }));
        // Save updated driverDetails to localStorage
        localStorage.setItem('currentDriver', JSON.stringify(driverDetails));
        setShowTripList(false)
      })
      .catch(error => console.error("Error updating driver vacant status:", error));
  }
};

```

## הצגת הנסיעה במפה:

```
const displaySelectedDriveOnMap = (drive) => {  
  const geocoder = new window.google.maps.Geocoder();  
  
  const geocodeAddress = (address) => {  
    return new Promise((resolve, reject) => {  
      geocoder.geocode({ address: address }, (results, status) => {  
        if (status === window.google.maps.GeocoderStatus.OK) {  
          const location = results[0].geometry.location;  
          resolve(location);  
        } else {  
          reject(`Geocode was not successful for the following reason: ${status}`);  
        }  
      });  
    });  
  };  
};
```

## סיום נסיעה של הנהג:

```
// הנהג סיים נסיעה  
const finishDrive = () => {  
  if (selectedDrive) {  
    driverDetails.vacant = true;  
    axios.put(`http://localhost:8080/drivers/${driverDetails.code}`, driverDetails)  
      .then(() => {  
        setDriverDetails(prevDriver => ({  
          ...prevDriver,  
          vacant: true  
        }));  
        localStorage.setItem('currentDriver', JSON.stringify(driverDetails));  
        fetchRelevantDrives();  
        setIsVacant(true);  
        setSelectedDrive(null);  
        setShowTripList(true);  
      })  
      .catch(error => console.error("Error updating driver vacant status:", error));  
  }  
};  
  
if (loadError) return 'Error loading maps';  
if (!isLoading) return 'Loading maps';
```

```
import { useState, useEffect } from 'react'
import { BrowserRouter as Router, Navigate, Route, Routes } from 'react-router-dom'; // Import BrowserRouter as Router

import Passenger from './jsx/Passenger'
import Login from './jsx/Login';
import Driver from './jsx/Driver'
import './App.css'

function App() {

  return (

    <Router>
      <Routes>
        <Route path="/" element={<Navigate to="/passenger/" />} /> />
        <Route path="/login" element={<Login />} />
        <Route path="/passenger/:passName?" element={<Passenger />} />
        <Route path="/driver/:driverName" element={<Driver />} />
      </Routes>
    </Router>
  )
}

export default App
```

הפונקציות בשרת:

דף ה-[index.js](#)

```
const app = express();
const PORT = process.env.PORT || 8080;

const server = http.createServer(app);
const io = new Server(server, {
  cors: {
    origin: "http://localhost:5173",
    methods: ["GET", "POST"]
  }
});
const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'img/pictures');
  },
  filename: (req, file, cb) => {
    const extension = file.mimetype.split('/')[1];
    const fileName = v4() + '.' + extension;
    cb(null, fileName);
  },
});
```

```

const upload = multer({ storage });

app.use(express.json());
app.use(cors());
app.use('/img', express.static(path.join(__dirname, 'img')));

app.post('/picture', upload.single('profilPic'), (req, res) => {
  res.send({ imagePath: `/pictures/${req.file.filename}` });
});
app.post('/upload', upload.single('profilPic'), (req, res) => {
  const imagePath = `${req.file.filename}`;
  res.send({ imagePath });
});

app.use(express.static('public'));
app.use('/passengers', passengers);
app.use('/drivers', drivers);
app.use('/drives', drives);
io.on('connection', (socket) => {
  socket.on("addDrive", async (drive) => {
    const newD = await postDrive(drive);
    io.emit("newDrive", newD);
  });
});

```

## דף הנוסעים:

```

import express from "express"
import { postPassenger, getPassengers, getPassenger, putPassenger, deletePassenger, isDriveAccept, getPassengerforPas
import { getDriverforPassWord } from "../DataBase/Drivers.js"
const app = express.Router();

import { v4 } from "uuid";
import multer from "multer"; const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'img/pictures');
  },
  filename: (req, file, cb) => {
    const extension = file.mimetype.split('/')[1];
    const fileName = v4() + '.' + extension;
    cb(null, fileName);
  },
});

```

החזרת נוסעים:

```
//חזרת כל הנוסעים
app.get('/', async (req, res) => {
  const { code } = req.query;
  const data = await getPassengers(code);
  res.send(data);
});

app.get('/passenger', async (req, res) => {
  const { name, password } = req.query;
  if (!name || !password) {
    return res.status(400);
  }
  try {
    const data = await getPassenger(name, password);
    res.send(data);
  } catch (err) {
    res.status(400).json({ error: 'Database error' });
  }
});
```

הוספת נוסע:

```
//הוספת נוסע
app.post('/', upload.single('profilPic'), async (req, res) => {
  console.log(req)
  const fileName = req.file.filename;
  const { name, password, address, phone } = req.body;
  try {
    const newPassenger = await postPassenger(name, password, address, phone, fileName);
    res.status(201).send(newPassenger);
  }
  catch (error) {
    console.error('Error adding passenger:', error);
    res.status(500).send('Error adding passenger. Please try again.');
```

## מחיקת ועדכון נוסע:

```
// עידכון נוסע
app.put('/', async (req, res) => {
  const { code } = req.query;
  const { name, address, phone } = req.body; try {
    const updatedPost = await putPassenger(code, name, address, phone);
    res.send(updatedPost);
  } catch (error) {
    res.status(404).send(error.message);
  }
});

// מחיקה נוסע
app.delete('/', async (req, res) => {
  const { code } = req.query;
  try {
    await deletePassenger(code);
    res.send({ code });
  } catch (error) {
    res.status(404).send(error.message);
  }
});
```

## בדיקה האם נסיעה נבחרה:

```
// פונקציה לבדיקה האם הנסיעה נבחרה
app.get('/isDriveAccept', async (req, res) => {
  const { code } = req.query;
  const data = await isDriveAccept(code);
  res.send(data);
});
```

## פונקציה לבדיקת הסיסמא האם היא קיימת:

```
//פונקציה לבדיקת סיסמה קיימת
app.post('/checkPassword', async (req, res) => {
  const { name, password } = req.body;
  try {
    const passengers = await getPassengerforPassWord(name);
    const drivers = await getDriverforPassWord(name);
    if (passengers.length === 0 && drivers.length === 0) {
      return res.json({ isValid: false, error: 'משתמש לא נמצא' });
    }
    let isValid, currentpass;
    if (passengers.length !== 0) {
      isValid = passengers.some(p => p.password === password);
      currentpass = await getPassenger(name, password)
    }
    else {
      isValid = drivers.some(p => p.password === password);
    }
    if (isValid) {
      res.json({ isValid: true, currentpass: currentpass });
    } else {
      res.json({ isValid: false, error: 'סיסמה שגויה' });
    }
  } catch (error) {}
  console.error('שגיאה בבדיקת הסיסמה:', error);
  res.status(500).json({ isValid: false, error: 'שגיאה בבדיקת הסיסמה' });
});
```

## דף הנוסע-שאלות לSQL: החזרת נוסע והחזרת כל הנוסעים:

```
export async function getPassengers(code) {
  if (!code) {
    const [allPassengers] = await pool.query("SELECT code, name, address, phone, profilPic, active FROM passenger");
    return allPassengers;
  }
  return getPassenger(code);
}

export async function getPassenger(name, password) {
  if (name !== null && password == null) {}
  const [[passenger]] = await pool.query(`SELECT code, name, address, phone, profilPic, active FROM passengers`);
  return passenger;
}

const [[passenger]] = await pool.query(`SELECT code, name, address, phone, profilPic, active FROM passengers WHERE`);
return passenger;
}
```



## הוספת ועדכון נוסע:

```
export async function postPassenger(name, password, address, phone, profilPic) {
  const passengerIsExists = await getPassenger(name, password);
  if (passengerIsExists)
    throw new Error(`passenger ${name} already exist`);
  const [{ insertCode }] = await pool.query(`INSERT INTO passengers ( name, password, address, phone, profilPic) VALUES (${name}, ${password}, ${address}, ${phone}, ${profilPic})`);
  return await getPassenger(insertCode);
}

export async function putPassenger(code, name, address, phone, profilPic) {
  const passengerExists = await getPassengers(code);
  if (!passengerExists) {
    throw new Error(`passenger ${name} does not exist`);
  }
  const [result] = await pool.query(`UPDATE passengers SET name=?, address=?, phone=?,profilPic=? WHERE code=?`, [name, address, phone, profilPic, code]);
  if (result.affectedRows === 0) {
    throw new Error(`passenger ${name} does not found`);
  }
  return await getPassengers(code);
}
```

## מחיקת נוסע:

```
export async function deletePassenger(code) {
  try {
    // להיות active עדכון השדה false
    const [result] = await pool.query(`UPDATE passengers SET active=? WHERE code=?`, [false, code]);

    // בדיקת מספר הרשומות המעודכנות
    if (result.affectedRows === 0) {
      throw new Error(`passenger ${code} does not found`);
    }

    return { code };
  } catch (error) {
    console.error("Error updating driver:", error);
    throw error;
  }
}
```

בדיקה האם נבחרה נסיעה והאם סיסמא קיימת :

```
export async function isDriveAccept(code) {  
  const [result] = await pool.query(`SELECT *  
    FROM drives d1  
    JOIN drivers d2 ON d1.driver_code = d2.code  
    WHERE d1.pass_code = ? AND d1.vacant = 1 AND d2.vacant = 0 AND d2.active = 1`, [code]);  
  if (result.affectedRows !== 0) {  
    return result;  
  }  
  return false;;  
}  
  
export async function getPassengerforPassWord(name) {  
  try {  
    const [rows] = await pool.query('SELECT * FROM passengers WHERE name = ?', [name]);  
    if (rows.length === 0) {  
      return [];  
    }  
    return rows; // החזרת כל הרשומות שנמצאו  
  } catch (error) {  
    console.error('Error executing query:', error);  
    throw error;  
  }  
}
```

כנ"ל בכל הטבלאות. כמובן בהתאמה לדרישות של כל טבלה.

## בקרת תוכנה

### בטבלאות

בעת מילוי או עדכון פריט בטבלאות המערכת בודקת אם כל המאפיינים תקינים. על כל שדה קלט יש בדיקה האם הערך שמוזן עונה על הדרישות וכן שדות חובה- אי אפשר לבצע שמירה / הוספה / עדכון ללא מילוי שדות החובה. המשתמש מקבל הודעה כי לא מילא את שדות החובה. בכניסה יש בדיקות תקינות על השם וסיסמת משתמש שהערכים המוזנים חוקיים. אם הערכים חוקיים המערכת בודקת את השם + הסיסמה ששייכים למשתמש אם כן מתאפשרת כניסה כמשתמש קיים. אם לא, המשתמש מקבל הודעה שהפרטים שגויים.

### אבטחת מידע

- כדי לדאוג לריבוי משתמשים וריבוי קריאות והבטחה בחרנו להשתמש ב SQL Server המטפל בדרישות אלו.
- דוגמאות למקרים ותגובות להם ניתן מענה אבטחתי
  - הסיסמה תהיה מוסתרת
  - הסיסמה לא נשמרת במערכת בצורה שתהיה אפשרות לגשת אליה
  - לא ניתן לעבור בין משתמשים בכתובת URL
  - הנתונים ישמרו ב DB SQL שזהו מסד נתונים המתמודד עם עומסים
  - בכל מצב של חוסר תקשורת עם השרת תופיע הודעה על תקלת תקשורת למשתמש.
  - קריאות השרת נעשות בצורה אסינכרונית ומאפשרת פעילות באתר תוך כדי.

### מה הקנה הפרויקט

- לימוד ושליטה בשפות React ו Node.js
- רכישת ניסיון בפיתוח הידע ב SQL
- חשיבה לוגית מפותחת
- ידע בהתמקצעות עיצוב אתרים על אופניו השונים
- פתרון בעיות
- ניסיון בניתוח מערכת
- ניסיון בתכנון נתונים ועיבוד יעיל
- ניסיון בבדיקות מקיפות
- פיתוח יכולת הלמידה העצמית

## ביבליוגרפיה

- [w3schools.com](https://www.w3schools.com)
- [stackoverflow.com](https://stackoverflow.com)
- [Codepen.io](https://codepen.io)
- npm

ועוד אתרים שונים...