**מגישות:**

**מגמה:** הנדסת תוכנה

**שם המנחה:**

**מגישות:**

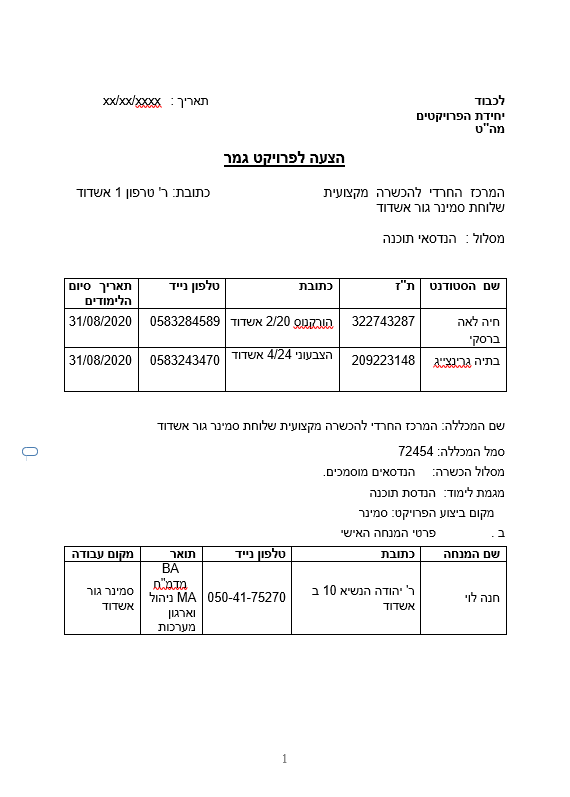
**מגמה:** הנדסת תוכנה

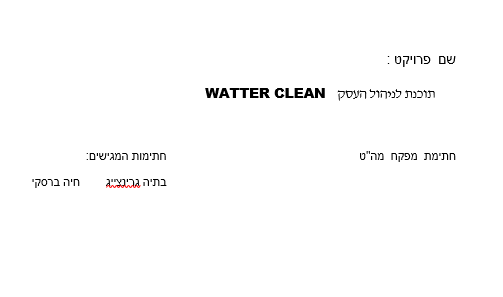
**שם המנחה:**

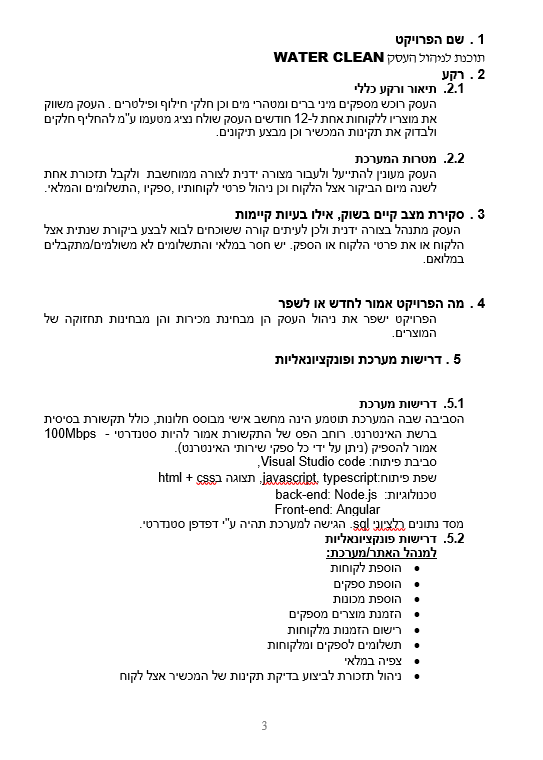
חתימה:\_\_\_\_\_\_\_\_\_\_

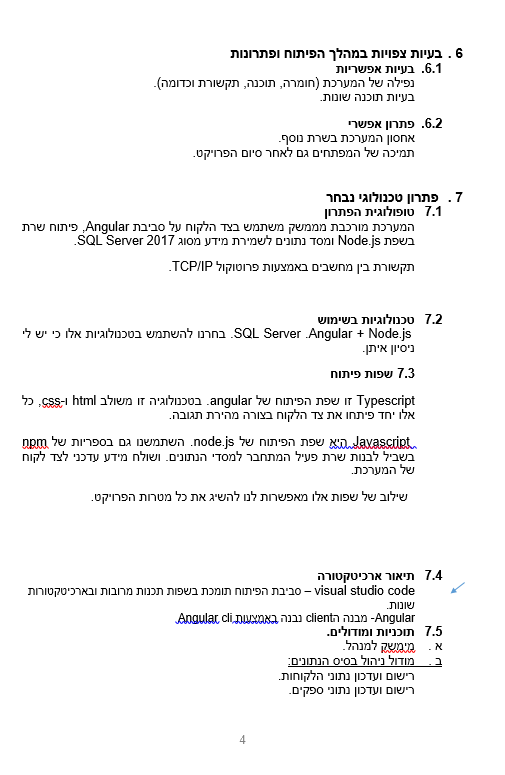
**שם מרכז המגמה: \_\_\_\_\_\_\_\_\_\_**

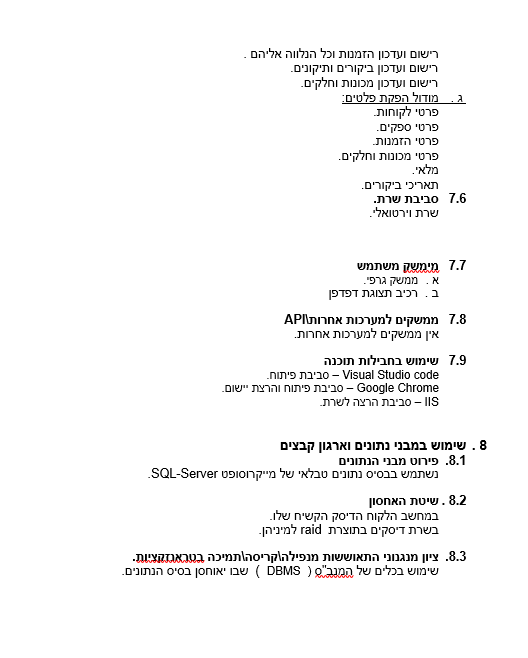
חתימה:\_\_\_\_\_\_\_\_\_\_

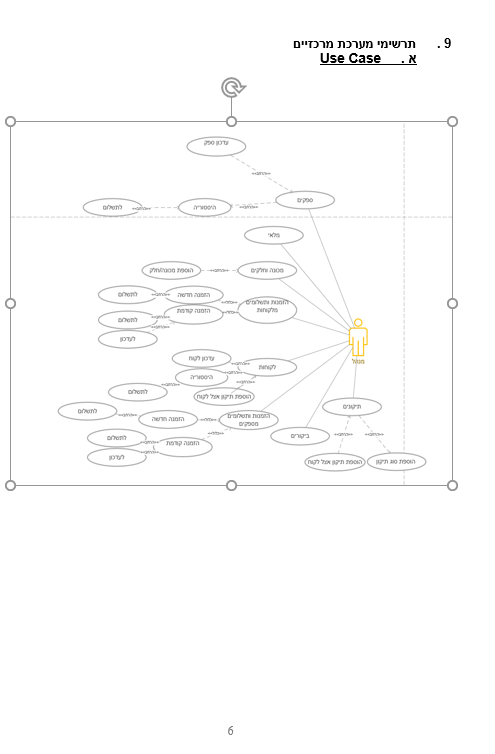


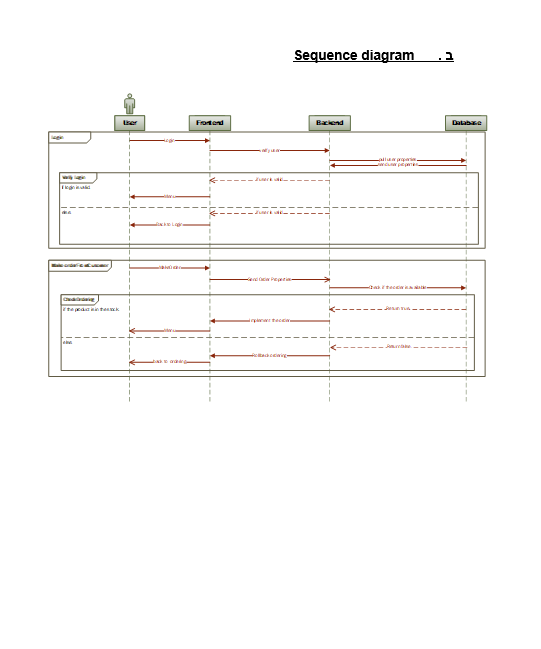


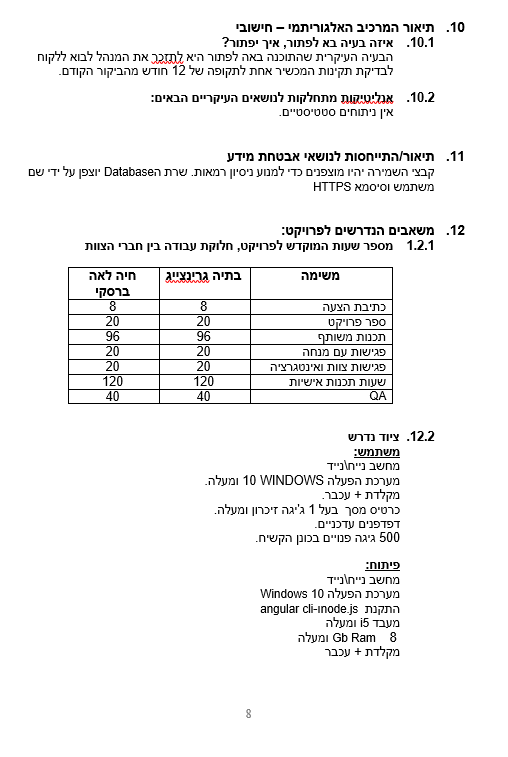


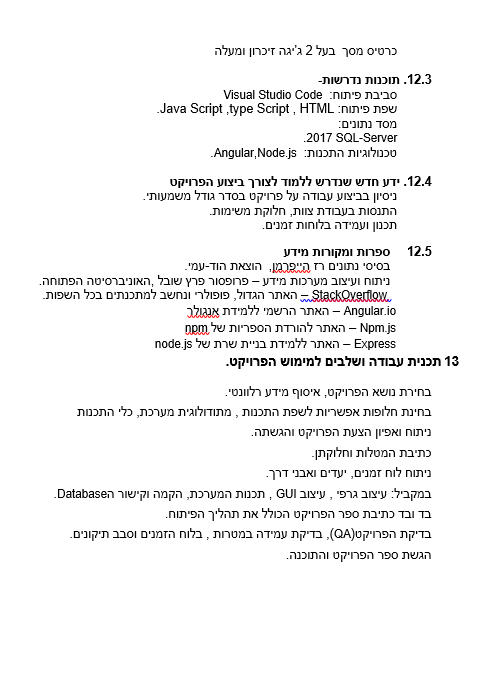


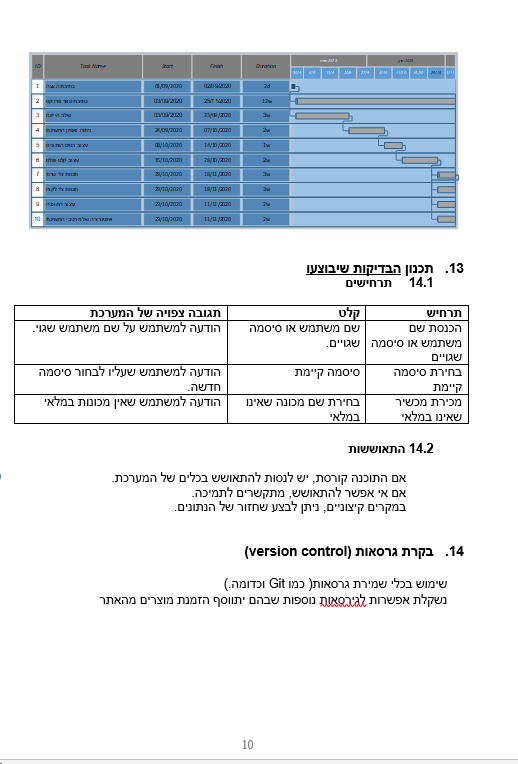












# אישור הצעת הפרויקט ממה"ט

# תודות

רצינו להביע את תודתנו לכל מי שסייע לנו במהלך העבודה על הפרויקט.  
לגב'\_\_\_\_\_\_, מנחת הפרויקט שהדריכה אותנו במהלך העבודה ולימדה אותנו את עקרונות  
השפה והטכנולוגיה שבה עבדנו.  
לגב' \_\_\_\_\_\_\_\_, רכזת מגמת מחשבים אשר דאגה שנקבל את כל התנאים שהיינו צריכות בשביל  
הפרויקט ונתנה לנו עצות בעת הצורך.  
להורים שלנו שעמדו לצדינו, תמכו ועודדו אותנו לאורך הפרויקט.  
וכמובן, אין אנחנו מספיקין להודות... לריבונו של עולם שראינו הרבה סיעתא דשמיא במהלך  
העבודה.

# הצהרה

 **המכון הממשלתי להכשרה בטכנולוגיה ובמדע**

**יחידת הפרויקטים**

חוזר מנהל מה"ט 11-4-52 – נספח מס' 3

\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**הצהרת סטודנט**

שם הסטודנט: \_\_\_\_\_\_\_\_\_ ת.ז. :\_\_\_\_

שם הסטודנט: \_\_\_\_\_\_\_ ת.ז.: \_\_\_\_\_\_\_\_\_

שם המכללה בה לומד הסטודנט: \_\_\_\_\_\_\_

אני הח"מ, מצהיר בזאת כי פרויקט הגמר וספר הפרויקט המצ"ב נעשו על ידי בלבד.

פרויקט הגמר נעשה על סמך הנושאים שלמדתי במכללה ובאופן עצמאי.

פרויקט הגמר וספר הפרויקט נעשו על בסיס הנחייתו של המנחה האישי.

מקורות המידע בהם השתמשתי לביצוע פרויקט הגמר מצוינים ברשימת המקורות המצוינים בספר הפרויקט.

אני מודע לאחריות שהנני מקבל על עצמי על ידי חתימתי על הצהרה זו שכל הנאמר בה אמת ורק אמת.

חתימת הסטודנט: תאריך:

חתימת הסטודנט: תאריך:

**אישור המנחה האישי**

הריני מאשר שהפרויקט בוצע בהנחייתי, קראתי את ספר הפרויקט ומצאתי כי הוא מוכן לצורך הגשת

הסטודנט להגנה על פרויקט גמר.

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

**אישור ראש המגמה**

הריני מאשר שספר הפרויקט מוכן לצורך הגשת הסטודנט להגנה על פרויקט הגמר.

שם ראש המגמה: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ חתימה \_\_\_\_\_\_\_\_\_\_\_\_\_\_ תאריך: \_\_\_\_\_\_\_\_\_\_\_

# **בית ליאו גולדברג, דרך מנחם בגין 86, תל אביב, ת.ד. 36049, מיקוד 67138, טלפון 03-7347400**

יחידת הפרויקטים - טלפון: 7347521 -03 פקס': 03-7347644

# תוכן העניינים

שער [1](#_Toc73312263)

הצעת פרויקט [iii](#_Toc73312264)

[תודות 1](#_Toc73312337)

[הצהרה 2](#_Toc73312338)

[תוכן העניינים 4](#_Toc73312341)

[מבוא 5](#_Toc73312342)

[מדריך למתכנת: 6](#_Toc73312343)

[2.1 אסטרטגיות טכנולוגיות: 6](#_Toc73312344)

[2.2 תיאור מבנה הפרויקט: 10](#_Toc73312345)

[2.3 עקרונות התכנון : 11](#_Toc73312346)

[2.4 תרשימים: 13](#_Toc73312347)

[2.5 מבנה נתונים מאוכסנים: 14](#_Toc73312348)

[3.1 מדריך למשתמש: 22](#_Toc73312349)

[הוראות כלליות לשימוש באתר: 22](#_Toc73312350)

[מסכים: 23](#_Toc73312351)

[סיכום ומסקנות: 26](#_Toc73312352)

[נספחים: 27](#_Toc73312353)

[ביבליוגרפיה: 29](#_Toc73312355)

## מבוא

פרויקט זה נכתב ללקוח המנהל מערכות מים חשמליות לבתים ומוסדות

המערכת אמורה לסדר לו את כל נושא הרכישות מספקים ואת המכירות ללקוחות וביקורי הבית שמבצעים למכשירים בבתי הלקוחות.

מודל הפרויקט מחולק ל2 חלקים:

1. בנית classes מתאימים ויצירת השדות והפונקציות הנדרשות בתוכם.

יצירת controllers המנהלים את קריאות הclient ומתחברים לSql כדי לשמור את הנתונים

1. קוד Typescript הנכתב בפלטפורמת angular ובו שימוש בספריות רבות

## מדריך למתכנת:

### 2.1 אסטרטגיות טכנולוגיות:

צד השרת פותח ב . nodejs

. visual code בסביבת Angular 7 צד הלקוח פותח ב

**nodejs**- בהגדרה, NodeJS היא פלטפורמה שנבנתה מעל מנוע הג’אווה-סקריפט של גוגל, V8, ומטרתה להוות כלי לפיתוח אפליקציות רשת שעוסקות בטיפול אינטנסיבי במידע, בזמן אמיתי, ושממעטות בחישובים. במילים אחרות – הרבה I/O ומעט CPU. היא מבוססת על מודל של אירועים (events) ועושה שימוש בספריות I/O אסינכרוניות.

אם צריך לשים אצבע על המאפיין הכי חשוב של NodeJS זה בהחלט יהיה העיסוק האינטנסיבי ב-I/O אסינכרוני.

**לולאת אירועים (Event Loop)**

אפליקציות שרת יודעות לבצע המון פעולות במקביל ובכך להצליח לתת שירות להרבה משתמשי קצה. קיימות כמה שיטות למיקבול תהליכים אך לצורך העניין נתמקד בזו הרלוונטית אלינו – לולאת אירועים.

ביסודה, לולאת אירועים היא תור של פונקציות. בכל זמן נתון מבוצעת פונקציה אחת בלבד עד לסיומה ואז נשלפת הפונקציה הבאה. מאחר ומדובר ב-thread אחד, חשוב לשים לב שהפונקציות יהיו קצרות ותכליתיות על מנת שלא “יתקעו” את הלולאה. בפורמט הזה אנו מעבירים את ההתעסקות עם I/O ו-threads ל-NodeJS ומספקים פונקציה שתיקרא בתום התהליך ובאופן הזה חוסכים לעצמנו כאבי ראש כמו גישה למידע משותף במקביל (בעיה מוכרת מעולם ה threads).

העניין הוא ש NodeJS אינה הפלטפורמה הראשונה שמציעה מימוש לרעיון הזה וחוץ מזה, אם מדובר בטריק כל כך טוב שאפשר לצמצמם אותו עד כדי thread אחד, איך זה שלא כל העולם רץ להשתמש בו?! הנה שתי סיבות לדוגמא:

חינוך – הכתיבה הסנכרונית מוטמעת בתוכניות לימוד ובספרים. קחו דוגמא קלאסית לקוד שיופיע בשלבים המוקדמים בלימוד קלט/פלט:

מבקשים מהמשתמש להזין את שמו, מחכים עד שהוא יסיים, ואז מדפיסים את ההודעה.

הסבירות שתתקלו בקוד הבא היא אפסית:

בדומה לקטע הקוד שראינו בדוגמא הניגשת לבסיס נתונים, גם פה התהליך לא ימתין לחזרת הערך מהמשתמש אלא ימשיך לעיסוקים אחרים.

תשתיות – אפליקציות שנכתבות בעולם האמיתי, בדרך כלל, נעזרות בספריות צד-שלישי (לוג, דרייברים, וכו’). מספיק שאחת מאותן ספריות אינה אסינכרונית כדי לפגוע קשה בביצועי האפליקציה. מאחר ומדובר ב-thread אחד, האפשרות שהוא ימתין עד לחזרת תשובת קלט כלשהי אינה באה בחשבון.

**Angular**- הוא framework של JavaScript שמשמש ליצירת אפליקציות מבוססות.

דף יחיד SPA (Singel Page Apllication). אפליקציות SPA מתנהגות כמו אפליקציות שרצות על מכשירים ניידים כי ניתן לעבור בין הדפים בללי צורך לטעון את כל הדף. כתוצאה מכך, אתר האינטרנט מציג התנהגות חלקה וחוויית משתמש מעולה.

HTML :**Html**

זוהי השפה הטבעית ליצירת דפי אינטרנט ברשת. זו שפה פשוטה , אוניברסלית , המאפשרת לעורכי אתרים ליצור דפים מורכבים שמכילים טקסט ותמונות , שיכולים להראות בידי כל המשתמשים ברשת האינטרנט ללא תלות בסוג המחשב או בסוג הדפדפן.

**CSS**: CSS היא שפת עיצוב שמגדירה את תבנית העיצוב של מסמכיHTML. למשל, CSS מטפלת בגופנים ,צבעים, גבולות, שורות, גובה, רוחב, תמונות רקע, מיקום מתקדם ודברים רבים נוספים .

את העקרונות העיקריים שעל פיהם פעלנו לאורך כתיבת הקוד.

**SQL**

היא מערכת לניהול בסיס נתונים במודל היחסי (RDBMS) של חברת מיקרוסופט., שפת הפיתוח שבאמצעותה מועברות הפקודות למערכת (על ידי כתיבת פקודות או שימוש בממשק גרפי ששולח פקודות באופן סמוי) היא Transact-SQL ‏(TSQL) שהיא מימוש תקן ANSI של שפת SQL משמשת לתשאול וטיפול בנתונים (DML), יצירת טבלאות והיחסים ביניהן (DDL) ותחזוקת המערכת תוך שימוש בתוכניות שירות שונות.

SQL הוא מסד נתונים בו נשמרים הנתונים בטבלאות בעלות קשרי גומלין, וניתן לשלוף ממנו מידע ע"י שאלתות SQL.

יש לו יכולות רבות נוספות כמו view, פרוצדורות, פונקציות, טרנזקציות ועוד.

זהו מסד הנתונים הנפוץ ביותר.

**Web API**

Iמה זה API?

API הוא קיצור של .Interface Program Application במונחים פשוטים, זהו הממשק של פיסת

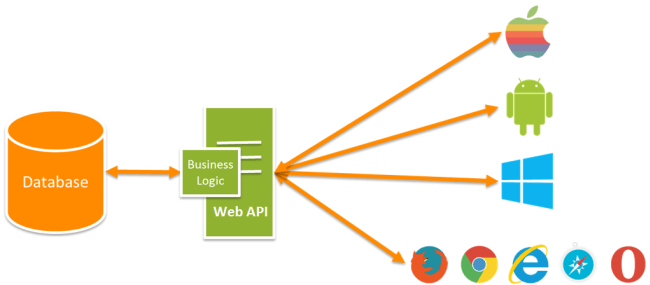
תוכנה עם העולם החיצוני, הקובע איך ישתמשו בה. אלה החוקים שנקבעו עבור האינטראקציה

שלה עם העולם הרחב, אלה יקבעו אילו חלקים של התוכנה יכולות לדבר עם תוכנות אחרות, ואיך

היא תגיב.

מה זה Web API ?

Web API הוא מסגרת המקלה על בניית שירותי HTTP המגיעים למגוון רחב של לקוחות, כולל דפדפנים



מה זה REST?

Rest( ראשי תיבות Transfer State Representational )היא סגנון ארכיטקטוני לכתיבת צד שרת.

התפיסה הארכיטקטונית ב- Rest היא תפיסת שרת-לקוח. תפיסה זו מחייבת קיום לקוח ושרת.

לקוחות יוזמים פניות המכילות בקשות לשרתים. השרת מעבד את הפנייה, ומחזיר תגובות

מתאימות.

בכל מצב נתון הלקוח יכול להיות בתהליך של שינוי מצב או במצב מנוחה (rest .(במצב של מנוחה

הלקוח יכול להיות באינטראקציה עם המשתמש, אבל אינו תופס משאבים בשרת. הלקוח שולח פניות

כאשר הוא מוכן לעבור למצב חדש. כאשר קיימת פנייה אחת או יותר שטרם הסתיים הטיפול בהן,

הלקוח נמצא במצב של מעבר ממצב למצב.

REST הוא לא פרוטוקול )כמו SOAP )אלא יותר קונבנציה שבה אנחנו משתמשים. Rest מבוסס ברב

המקרים על פרוטוקול HTTP ,למרות זאת REST היא ארכיטקטורה כללית הניתנת למימוש גם

בסביבות אחרות ולא רק תחת HTTP.

עבודה ב REST-מחייבת התחשבות באילוצים ובמגבלות של ארכיטקטורה זו, נפרט את חלקם:

• שרת-לקוח: עובד רק בתפיסת שרת לקוח, כאשר האחד אינו מושפע ממה שמתרחש באחר,

למעט המסרים העוברים ביניהם.

• Stateless :ההקשר (context (של הלקוח בפניה לשרת, אינו נשמר בשרת. מגבלה זו נועדה

לשפר את מדרגיות (Scalability (השרת.

כשאנו יוצרים שירות שממלא אחר הקונבנציות של REST אנחנו יוצרים שירות RESTful.

:Rest API - לסיכום

ממשק API של REST מגדיר קבוצה של פונקציות אשר מפתחים יכולים לבצע בקשות ולקבל תגובות

באמצעות פרוטוקול HTTP כגון GET ו- POST.

Typescript

היא שפת תכנות חינמית ומבוססת קוד פתוח המפותחת ומתוחזקת על ידי מיקרוסופט .היא מכילה

את קבוצת כל פקודות ותחביר JavaScript הפופולרית, כלומר כל קוד

JavaScriptהוא גם קוד Typescript תקין, ומוסיפה עליה טיפוסים סטטיים ותכנות מונחה עצמים

מבוסס מחלקות. בין התכונות שהיא מוסיפה: typing static ,תמיכה במחלקות ותמיכה במודולים

ובדקורטורים.

Typescript היא סופר-סט (superset) של JavaScript כלומר, הקוד מבוסס על JavaScript וחייב

לעבור קומפילציה (תרגום) ל JavaScript-כדי שהדפדפנים יבינו אותו מפני שדפדפנים

מבינים JavaScript בלבד. בסופו של דבר קוד Typescript מעובד לכדי קוד JavaScript ולכן ניתן

לומר "באחריות" שהשפה נתמכת בכל דפדפן שתומך ב- JavaScript.

קבצי Typescript יישמרו בסיומת ts.

השתמשנו בסביבות העבודה הבאות:

**:Visual Studio code**

סביבת הפיתוח המרכזית בעולם המייקרוסופטי נקראתVisual Studio. סביבה זאת מכילה מאות אפשרויות ותומכת במספר שפות תכנות. בנוסף, יש לציין כי רוב שפות התכנות והטכנולוגיות הנתמכות על ידי סביבת פיתוח זו.. תשתית זו מכילה מספר עצום של ספריות קוד וטכנולוגיות המקלות עלינו בתהליך יצירת התוכנה.

**SQL Server:** Microsoft SQL Server היא פלטפורמת שרת מידע ומבנה נתונים המיועדת למפתחים, ארגונים קטנים וגדולים כאחד אשר מציעה חבילה טכנולוגית שלמה של כלים ארגוניים המאפשרים להשיג את הערך המירבי מתוך המידע שלנו במחיר ובעלות הכוללת הנמוכים ביותר. תוכנת ניהול השרת מיועדת למשתמשים הזקוקים לפתרונות ניהול מידע באמצעות בסיסי נתונים בצורה יעילה ומאובטחת. בנוסף ניתן לפתח ולנהל תוכנות למחשבים שולחניים ותוכנות מבוססות רשת העושות שימוש בבסיס נתונים המנוהל תחת שרת SQL Server במהלך העבודה עימן.

### 2.2 תיאור מבנה הפרויקט:

ה- Solution שלנו כולל 3 פרויקטים:

DAL •

פרויקט זה מורכב ממקור נתונים –מסד הנתונים שלנו, וממערכת תוכנה אשר תפקידה לקרוא את המידע הנדרש למערכת, לשמור את העדכונים

ולהוסיף מידע חדש או למחוק פרטי מידע קיימים.

פרויקט זה מתקשר למסד הנתונים הנדרש לפרויקט בשיטת DB First.

ולכן ה DB נבנה ראשון ועל סמך זה נבנו המחלקות והמאפיינים.

BL •

הפרויקט שאחראי על הלוגיקה של המערכת, עוסקת בעיבוד המידע, בחישובים שונים ושליחתו לשכבת התצוגה.

בפרויקט זה נממש את הפונקציונאליות של המערכת.

מסד הנתונים והממשק משתמש מתקשרים דרך השכבה הזו.

DTO • מכיל את המודלים המקבילים לישויות מסד הנתונים ומשמש מעטפת.

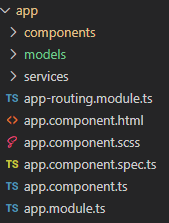
GUI •

הפרויקט שמכיל את ממשק המשתמש

תיאור ה-Client Side

הפרויקט נכתב ב-Angular8

1. Component - כל חלק בתצוגה כולל החלק הניהולי- הפונקציות והמשתנים שלו.
2. Services - מכילים את הפונקציות האחראיות על הגישה לפרויקט ה – server.
3. Models - מבנה המחלקות.



Models

Services

Component

### 2.3 עקרונות התכנון :

2.3.1 עקרונות תיאורטיים:

**הפרדת שכבות**

ישנם אתרים המבוססים על ארכיטקטורת שכבות הנקראת Tree Tire Architecture .

בארכיטקטורה זו קיימת הפרדה בין השכבות במבנה של DAL-BL-GUI. זוהי תבנית עיצוב בסיסית שמגדירה את הפרדת האפליקציה לשכבת נתונים, שכת לוגיקה ושכבת ממשק משתמש.

לתבנית עיצוב זו יתרונות רבים:

* תחזוקה:

ניתן להחליף או לתקן מימוש פנימי של שכבה אחת בארכיטקטורה בלי לשנות שכבה אחרת.

* נוחות פיתוח:

אדם אחד עובד על רכיב בתוכנה, אדם אחר עובד על רכיב אחר. כל עוד החתימות זהות ניתן לשלב כוחות ולייעל את מהירות הפיתוח.

* בדיקות:

כאשר נוצרת תקלה היא מבודדת בשכבה אליה היא שייכת. לדוגמא אם לא קיבלנו רשימת נתונים לתצוגה , נבדוק קודם בשכבת הנתונים (DAL) האם הנתונים שם תקינים, במידה והם אכן תקינים נעבור לבדוק את שכבת הביניים, שכבת הלוגיקה (BL) ואם גם שם הנתונים תקינים נדע בוודאות שהתקלה בשכבת התצוגה (GUI) ונפתור אותה שם.

* שימוש חוזר:
* במידה ששכבות ה-DAL וה-BL עובדות היטיב וברצוננו להחליף את פלטפורמת ה-GUI, ניתן לעשות זאת במינימום מאמץ.

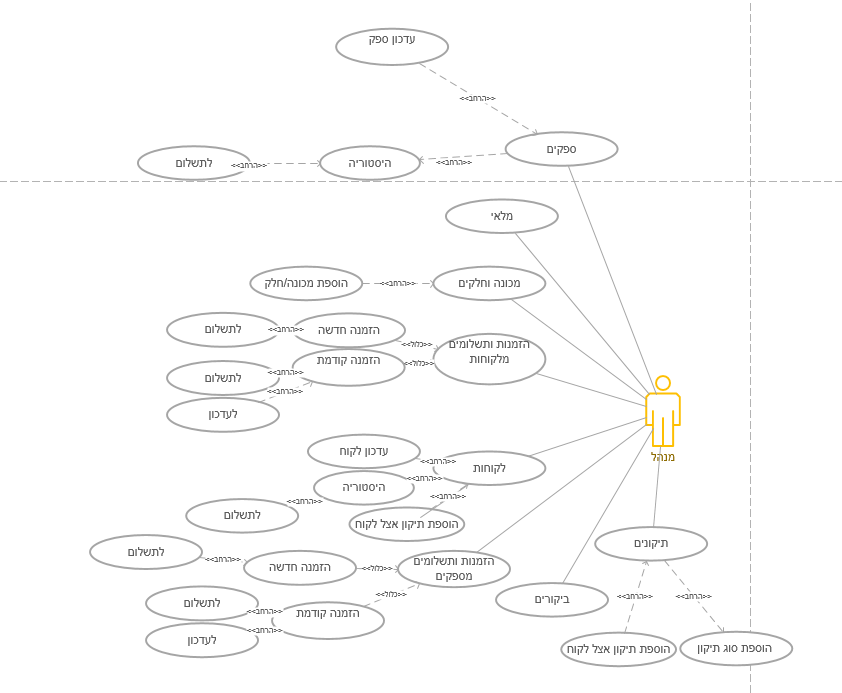
כיום הגישה לגבי אפליקציות WEB הוא כמה שיותר כתיבה בצד ה-client מכיוון שכל פניה לשרת מכבידה על המערכת וגורמת לה לחכות עוד זמן, מה שלא כדאי שיקרה היות ולמשתמשים אין סבלנות לחכות עד לתוצאות הרצויות, בעקבות זאת השתדלנו שרוב הנתונים שלא צריכים עיבוד בשרת ימומשו בצד ה-client וכן השתמשנו בספריית Angular8 כדי לחסוך פניות לשרת.

הפרויקט נכתב בשפת nodejs ומחולק לשלוש שכבות:GUI,BL,DAL

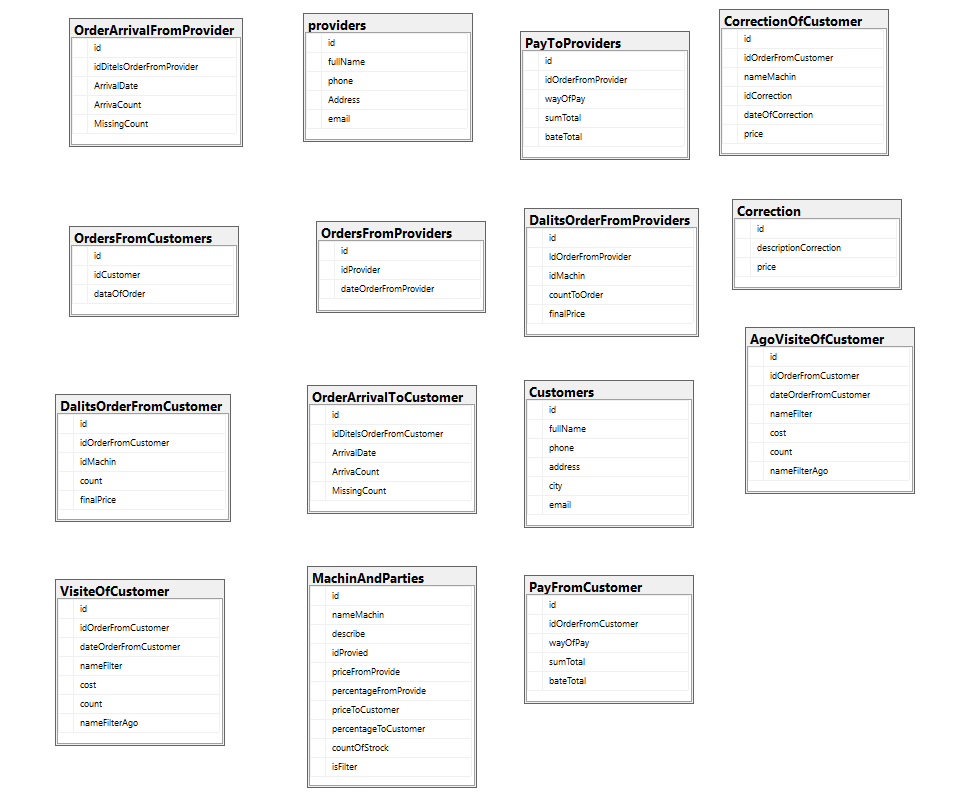
* DAL – שכבת מסד הנתונים.
  + - * + BL – שכבת הלוגיקה.
        + GUI – שכבת התצוגה למשתמש.

### תרשימים:

#### תרשים Uml:



#### תרשים מראה המחלקות:



### מבנה נתונים מאוכסנים:

2.5.1 תרשים SQL

Customer **-** טבלה זו מכילה את פרטי הלקוחות.

- Providers טבלה זו מכילה את כל הסוכנים הקיימים.

OrderFromCustomer - טבלה זו מכילה את כל פרטי הזמנות הלקוחות.

- PayFromCustomerטבלה זו מכילה את כל פרטי התשלומים של הלקוחות.

- MachineAndParties טבלה זו מכילה את כל פרטי המכשירים והפלטרים.

- VisitOfCustomerטבלה זו מכילה את פרטי ההזמנות שהגיעו ללקוחות.

- OrderArrivalToCustomer טבלה זו מכילה את פרטי לקוחות המועדון.

Corection - טבלה זו מכילה את פרטי התיקונים וסוגי הביקורים השונים.

OrderFromProvider - טבלה זו מכילה את כל פרטי הזמנות מהסוכנים.

- PayFromProviderטבלה זו מכילה את כל פרטי התשלומים של הסוכנים.

- OrderArrivalFromProvider טבלה זו מכילה את פרטי ההזמנות מהסוכנים שהגיעו.

#### מבנה קבצים ותיקיות:

##### תיקיית תמונות לעיצוב האתר-

#### 2.6 תוכן הפרויקט:

#### 2.6.1 תיאור המחלקות:

**מחלקות ב- Dal:**

* מחלקת הבאת נתונים מהדטה בייס:
* let sql = require('mssql')
* var config = {
* port: 1433,
* user: 'chaya',
* password: '207322868',
* server: 'DESKTOP-B5S1T71',
* database: 'waterClean2'
* }
* module.exports = function MyObject(col) {
* let table = col
* this.findAll = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select \* from ${table} order by id`)
* return result.recordset;
* }
* this.findNameProviderById = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select DISTINCT r.fullName
* from (select p.fullName,m.nameMachin,m.idProvied
* from Providers p join  ${table} m
* on p.id=m.idProvied) r
* where r.idProvied=${query}`)
* console.log(result.recordset[0]);
* return result.recordset;
* }
* this.showAllProvidersIsHaveOrder = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select DISTINCT p.fullName,p.id
* from ${table} o join [dbo].[providers] p
* on p.id=o.idProvider`)
* return result.recordset;
* }
* this.showAllCustomerIsHaveOrder = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select DISTINCT c.fullName,c.id
* from ${table} o join [dbo].Customers c
* on c.id=o.idCustomer`)
* return result.recordset;
* }
* this.showAlldateIsHaveOrderfromProvider = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select a.fullName,a.dateOrderFromProvider as [date],a.id from
* (select o.dateOrderFromProvider,p.fullName,o.id
* from ${table} o join providers p
* on o.idProvider=p.id)a
* where a.fullName=${query}`)
* return result.recordset;
* }
* this.showAlldateIsHaveOrderfromCustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select a.fullName,a.dataOfOrder as [date],a.id from
* (select o.dataOfOrder,c.fullName,o.id
* from ${table} o join Customers c
* on o.idCustomer=c.id)a
* where a.fullName=${query}`)
* console.log(result.recordset);
* return result.recordset;
* }
* this.showMachinByIdProvider = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select \*
* from ${table}
* where idProvied=${query}`)
* return result.recordset;
* }
* this.showIdMachinByNameMachin = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select id
* from ${table}
* where nameMachin='${query}'`)
* console.log(result.recordset[0].id)
* return result.recordset[0].id;
* }
* this.showHistroyProvider = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select d.id,o.dateOrderFromProvider as dateOrder,m.nameMachin as nameMachin,d.countToOrder as countToOrder,m.priceFromProvide as priceOfSingel,m.percentageFromProvide as percentageOfSingel,d.IdOrderFromProvider as IdOrder,d.[finalPrice]
* ,o.idProvider as idMan
* from OrdersFromProviders o join ${table} d
* on o.id=d.IdOrderFromProvider join MachinAndParties m
* on d.idMachin=m.id
* where ${query}
* order by  d.IdOrderFromProvider`)
* console.log(result.recordset);
* return result.recordset;
* }
* this.showHistroyCustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select d.id,o.dataOfOrder as dateOrder,m.nameMachin as nameMachin,d.[count] as countToOrder,m.priceToCustomer as priceOfSingel,m.percentageToCustomer as percentageOfSingel,d.idOrderFromCustomer as IdOrder,m.isFilter,
* o.idCustomer as idMan,d.[finalPrice]
* from OrdersFromCustomers o join ${table} d
* on o.id=d.idOrderFromCustomer join MachinAndParties m
* on d.idMachin=m.id
* where ${query}`)
* return result.recordset;
* }
* this.showCountArrivalToCustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select  d.id,sum(a.ArrivaCount) as countArrival
* from OrdersFromCustomers o join ${table} d
* on o.id=d.idOrderFromCustomer join MachinAndParties m
* on d.idMachin=m.id left join OrderArrivalToCustomer a
* on a.idDitelsOrderFromCustomer=d.id
* where ${query}
* group by d.id `)
* return result.recordset;
* }
* this.showCountArrivalToProvider = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select  d.id,sum(a.ArrivaCount) as countArrival
* from OrdersFromProviders o join ${table} d
* on o.id=d.IdOrderFromProvider join MachinAndParties m
* on d.idMachin=m.id left join OrderArrivalFromProvider a
* on a.idDitelsOrderFromProvider=d.id
* where ${query}
* group by d.id `)
* return result.recordset;
* }
* this.showIdByName = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(` select id from providers
* where fullName=${query}`)
* return result.recordset;
* }
* this.getBateTotal =
* async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(` select TOP 1 p.bateTotal from
* ${table} p
* ${query}
* ORDER BY p.bateTotal `)
* return result.recordset;
* }
* this.getVisitors = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select c.\*,v.dateOrderFromCustomer,m.\*,v.count, v.cost,o.id as idOrderFromCustomer
* from Customers c join OrdersFromCustomers o
* on o.idCustomer=c.id join ${table} v
* on v.idOrderFromCustomer=o.id  join MachinAndParties m
* on m.nameMachin=v.[nameFilter]
* where m.isFilter='true'
* and cast (v.dateOrderFromCustomer as date)<= dateadd(day,30,cast (cast(month(GETDATE())as nvarchar(2)) +'/'+CAST( day(GETDATE())as nvarchar(2)) +'/'+cast(year(GETDATE())-1as nvarchar(4)) as nvarchar))
* and cast (v.dateOrderFromCustomer as date)> =dateadd(day,-30,cast (cast(month(GETDATE())as nvarchar(2)) +'/'+CAST( day(GETDATE())as nvarchar(2)) +'/'+cast(year(GETDATE())-1as nvarchar(4)) as nvarchar))
* ORDER BY CONVERT(DATE, v.dateOrderFromCustomer) ASC
* `)
* return result.recordset;
* }
* this.getfilterim = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select distinct \*
* from ${table}
* where isFilter='true' and countOfStrock>0  `)
* return result.recordset;
* }
* this.getOldVisitors = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select c.\*,v.dateOrderFromCustomer,m.\*,v.count, v.cost,o.id as idOrderFromCustomer
* from Customers c join OrdersFromCustomers o
* on o.idCustomer=c.id join ${table} v
* on v.idOrderFromCustomer=o.id  join MachinAndParties m
* on m.nameMachin=v.[nameFilter]
* where m.isFilter='true'
* and cast (v.dateOrderFromCustomer as date)< =dateadd(day,-31,cast (cast(month(GETDATE())as nvarchar(2)) +'/'+CAST( day(GETDATE())as nvarchar(2)) +'/'+cast(year(GETDATE())-1as nvarchar(4)) as nvarchar))
* ORDER BY CONVERT(DATE, v.dateOrderFromCustomer) ASC`)
* return result.recordset;
* }
* this.getAllVisitThisYear = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select c.\*,v.dateOrderFromCustomer,m.\*,v.count, v.cost,o.id as idOrderFromCustomer
* from Customers c join OrdersFromCustomers o
* on o.idCustomer=c.id join ${table} v
* on v.idOrderFromCustomer=o.id  join MachinAndParties m
* on m.nameMachin=v.[nameFilter]
* where m.isFilter='true'
* and cast (v.dateOrderFromCustomer as date)>= dateadd(day,-31,cast (cast(month(GETDATE())as nvarchar(2)) +'/'+CAST( day(GETDATE())as nvarchar(2)) +'/'+cast(year(GETDATE())-1as nvarchar(4)) as nvarchar))
* ORDER BY CONVERT(DATE, v.dateOrderFromCustomer) ASC
* `)
* return result.recordset;
* }
* this.showVisitercustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select a.id as IdOrder ,a.dateOrderFromCustomer as dateOrder ,a.nameFilter as nameMachin,m.priceToCustomer as priceOfSingel,m.percentageToCustomer as percentageOfSingel,  a.cost as finalPrice ,a.[count] as countToOrder,a.nameFilterAgo from AgoVisiteOfCustomer a
* join ${table} o
* on a.idOrderFromCustomer=o.id join MachinAndParties m
* on m.nameMachin=a.nameFilter
* where o.idCustomer=${query}
* union
* select v.id as IdOrder ,v.dateOrderFromCustomer as dateOrder ,v.nameFilter as nameMachin,m.priceToCustomer as priceOfSingel,m.percentageToCustomer as percentageOfSingel,  v.cost as finalPrice ,v.[count] as countToOrder,v.nameFilterAgo from VisiteOfCustomer v
* join OrdersFromCustomers o
* on v.idOrderFromCustomer=o.id join MachinAndParties m
* on m.nameMachin=v.nameFilter
* where o.idCustomer=${query}`)
* return result.recordset;
* }
* this.showMachinInHaveToCustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`select DISTINCT m.\*
* from OrdersFromCustomers o join DalitsOrderFromCustomer d
* on o.id=d.idOrderFromCustomer join MachinAndParties m
* on d.idMachin=m.id
* where o.idCustomer=${query} and m.isFilter!='true'
* UNION
* select DISTINCT m.\*
* from VisiteOfCustomer v join OrdersFromCustomers o
* on o.id=v.idOrderFromCustomer join
* MachinAndParties m
* on v.nameFilter=m.nameMachin
* where o.idCustomer=${query}
* `)
* return result.recordset;
* }
* this.showDatesByMachin = async (query1, query2) => {
* console.log(query1);
* console.log(query2);
* let connection = await sql.connect(config)
* let result = await connection.request().query(`   select oa.ArrivalDate  as  dataOfOrder ,o.id from OrderArrivalToCustomer oa join DalitsOrderFromCustomer d
* on oa.idDitelsOrderFromCustomer=d.id join OrdersFromCustomers o
* on o.id=d.idOrderFromCustomer join MachinAndParties m
* on m.id=d.idMachin
* where o.idCustomer=${query1}  and m.nameMachin=${query2} and oa.ArrivaCount>0
* `)
* return result.recordset;
* }
* this.showDatesByFilter = async (query1, query2) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`
* select v.dateOrderFromCustomer as  dataOfOrder ,o.id
* from VisiteOfCustomer v join OrdersFromCustomers o
* on o.id=v.idOrderFromCustomer
* where o.idCustomer=${query1} and  v.nameFilter=${query2}
* `)
* return result.recordset;
* }
* this.showCorrectionHistory = async () => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`
* select cu.fullName,o.id as idOrder,c.nameMachin,c.dateOfCorrection,co.descriptionCorrection,c.price
* from ${table} c
* join Correction co
* on c.idCorrection=co.id  join OrdersFromCustomers o
* on o.id=c.idOrderFromCustomer join Customers cu
* on cu.id=o.idCustomer
* `)
* return result.recordset;
* }
* this.showCorrectionHistoryByIdCustomer = async (query) => {
* let connection = await sql.connect(config)
* let result = await connection.request().query(`
* select c.id,c.dateOfCorrection as dateOrder,c.nameMachin,co.descriptionCorrection,c.price as finalPrice from ${table} c
* join OrdersFromCustomers o
* on o.id=c.idOrderFromCustomer join Correction co
* on co.id=c.idCorrection
* where o.idCustomer=${query}
* `)
* return result.recordset;
* }
* }
* מחלקת הוספת נתונים:
* let sql = require('mssql')
* var config = {
* port: 1433,
* user: 'chaya',
* password: '207322868',
* server: 'DESKTOP-B5S1T71',
* database: 'waterClean2'
* }
* module.exports = function MyObject(Providers) {
* var table = Providers
* this.addOne = async (values) => {
* let connection = await sql.connect(config);
* let result = await connection.request().query(`insert into ${table}  values (${values}) `)
* let ans = await connection.request().query(`select @@IDENTITY `)
* return ans.recordset[0];
* }
* this.addAgoVisit = async (values1,values2,values3) => {
* let connection = await sql.connect(config);
* let result = await connection.request().query(`insert into AgoVisiteOfCustomer(idOrderFromCustomer,dateOrderFromCustomer,nameFilter,cost,[count],nameFilterAgo)
* select top 1 idOrderFromCustomer,dateOrderFromCustomer,nameFilter,cost,[count] ,nameFilterAgo from ${table}
* where idOrderFromCustomer=${values1} and dateOrderFromCustomer=${values2} and nameFilter=${values3}`)
* let ans = await connection.request().query(`select @@IDENTITY `)
* return ans.recordset[0];
* }
* }

• מחלקת עדכון נתונים:

let sql = require('mssql')

var config = {

    port: 1433,

    user: 'chaya',

    password: '207322868',

    server: 'DESKTOP-B5S1T71',

    database: 'waterClean2'

 }

module.exports = function MyObject(col) {

    var table = col

    this.updateStrock = async (name,count) => {

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set countOfStrock=countOfStrock+${count}

        where id=(select id

        from ${table}

        where nameMachin='${name}')`)

        return result;

    }

    this.updataStrockCustom = async (name,count) => {

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set countOfStrock=countOfStrock-${count}

        where id=(select id

        from ${table}

        where nameMachin='${name}')`)

        return result;

    }

    this.updateOne=async(id,colum,change)=>{

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update ${table}

        set ${colum}= ${change} where id=${id}`)

        return result;

    }

    this.updetstrock = async (name,count) => {

        console.log(name)

        console.log(count)

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set countOfStrock=countOfStrock+${count}

        where id=(select id

        from ${table}

        where nameMachin='${name}')`)

        return result;

    }

    this.updetCountstrock = async (name,count,countAgo) => {

        console.log(name)

        console.log(count)

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set countOfStrock=countOfStrock-${count}+${countAgo}

        where id=(select id

        from ${table}

        where nameMachin='${name}')`)

        return result;

    }

    this.upDeteDalits  = async (id,count,finalPrice) => {

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set count=${count}, finalPrice=${finalPrice}

        where id=${id}`)

        return result;

    }

    this.upDatevisit  = async (id,count) => {

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set count=${count}

        where idOrderFromCustomer=${id}`)

        return result;

    }

    this.upDateagovisit  = async (id,count) => {

        let connection = await sql.connect(config);

        let result = await connection.request().query(`update  ${table}

        set count=${count}

        where idOrderFromCustomer=${id}`)

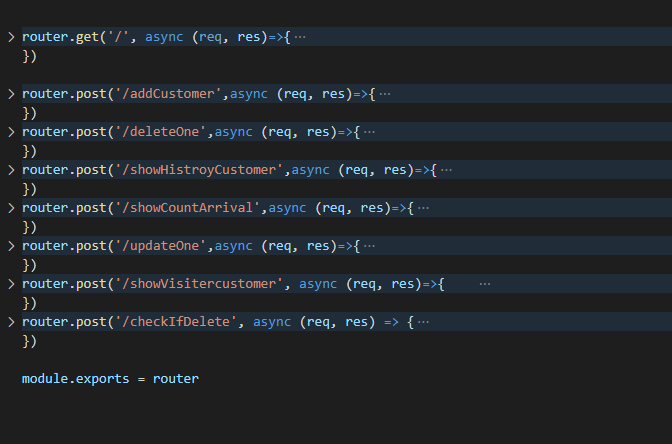
        return result;

    }

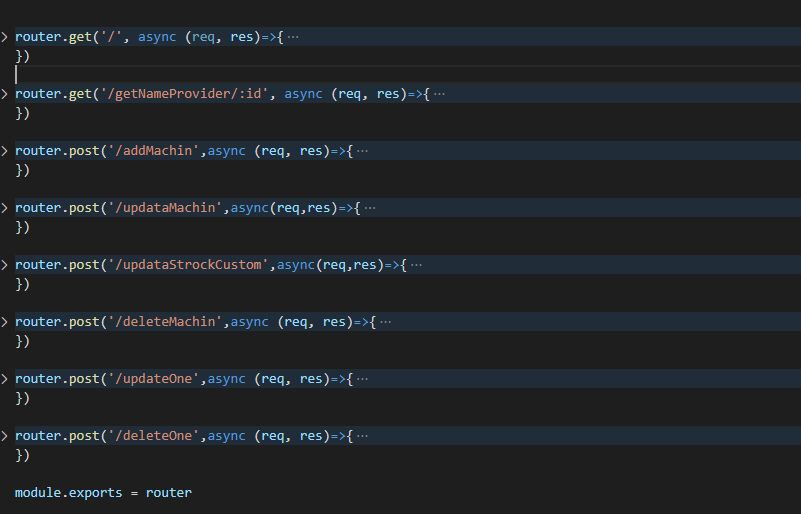
}

**מחלקות ב-BL:**

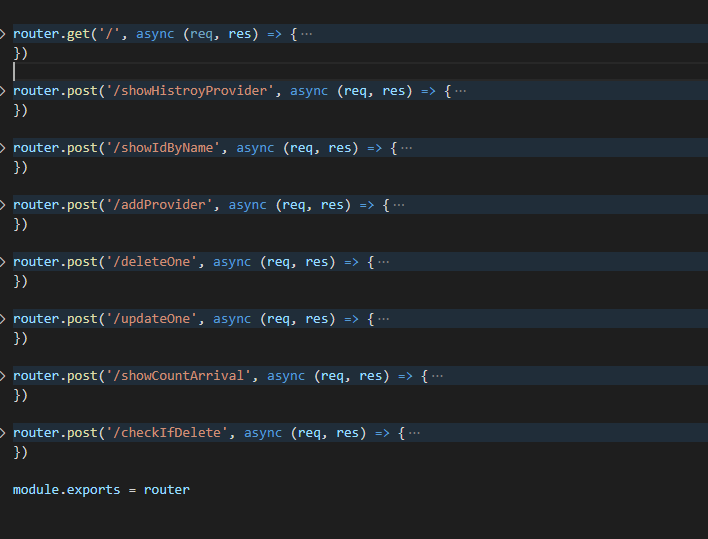
* מחלקת Customers:



* מחלקת MachineAndParts:



* מחלקת Provider:



###### 1.6.2 תיאור הפונקציות:

הפונקציה checkIfDelete מקבלת סוכן למחיקה ובודקת האם ניתן למחוק אותו (כל התשלומים הסתיימו , כל ההזמנות הגיעו ליעדם, ונגמרו המוצרים במלאי) .

router.post('/checkIfDelete', async (req, res) => {

    let order = new get('OrdersFromProviders')

    let dataOfOrder = await order.showAlldateIsHaveOrderfromProvider(`'${req.body.myProvider.fullName}'`)

    for (let item of dataOfOrder) {

        //תשלומים

        let order = new get('PayToProviders')

        let bateTotal = await order.getBateTotal(`where p.idOrderFromProvider=${item.id}`)

        if (bateTotal.length == 0 || bateTotal[0].bateTotal != 0)

            res.json(1);

        //הגעה

        order = new get('DalitsOrderFromProviders')

        let history = await order.showHistroyProvider(`o.dateOrderFromProvider='${item.date}' and o.idProvider=${req.body.myProvider.id}`)

        for (let h of history) {

            order = new get('DalitsOrderFromProviders')

            let countArrival = await order.showCountArrivalToProvider(`o.dateOrderFromProvider='${h.dateOrder}'and o.idProvider=${req.body.myProvider.id}`)

            console.log("countArrival", countArrival);

            for (let c of countArrival)

                if (h.countToOrder != c.countArrival)

                    res.json(2);

        }

    }

    //מלאי

    let MachinAndParties = new get('MachinAndParties')

    let machin = await MachinAndParties.showMachinByIdProvider(`${req.body.myProvider.id}`)

    for (let i = 0; i < machin.length; i++) {

        if (machin[i].countOfStrock > 0)

            res.json(3);

    }

    res.json(0);

})

הפונקציה addAgoVisitAndDelVisit מקבלת פרמטרים לעדכון תורים שכבר התבצעו

מוסיפה אותם לטבלת תורים שכבר התבצעו ומוחקת אותם מטבלת ביקורים .

router.post('/addAgoVisitAndDelVisit',async (req, res)=>{

    let visit = new add('VisiteOfCustomer')

    let agovisit=new del('VisiteOfCustomer')

    let ans2 = await visit.addAgoVisit(`${req.body.id}`,`'${req.body.date}'`,`'${req.body.nameFilter}'`)

    let ans3 = await agovisit.delVisit(`${req.body.id}`,`'${req.body.date}'`,`'${req.body.nameFilter}'`)

    res.json(ans2)

})

## 3.1 מדריך למשתמש:

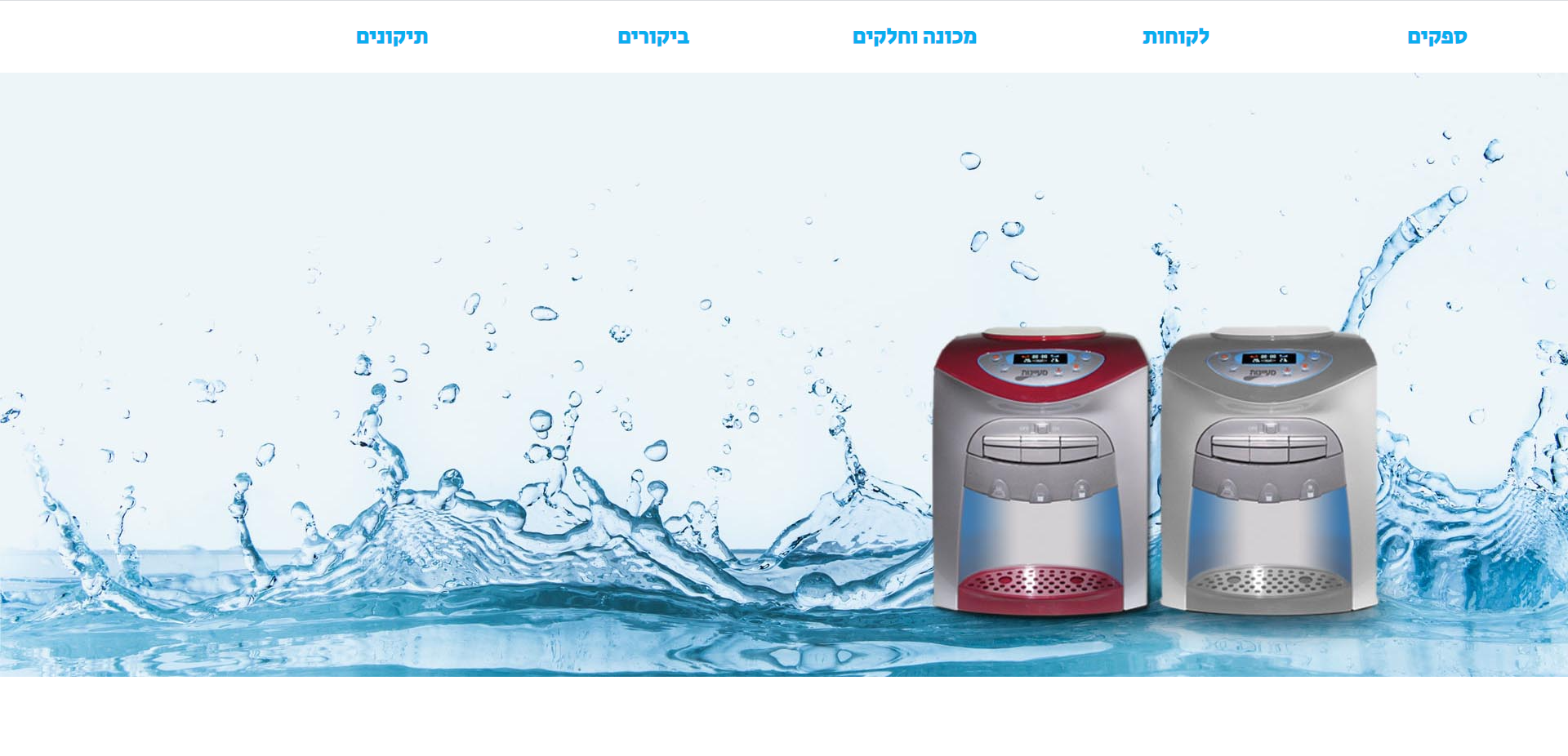
### הוראות כלליות לשימוש באתר:

#### **מדריך למנהל האתר**

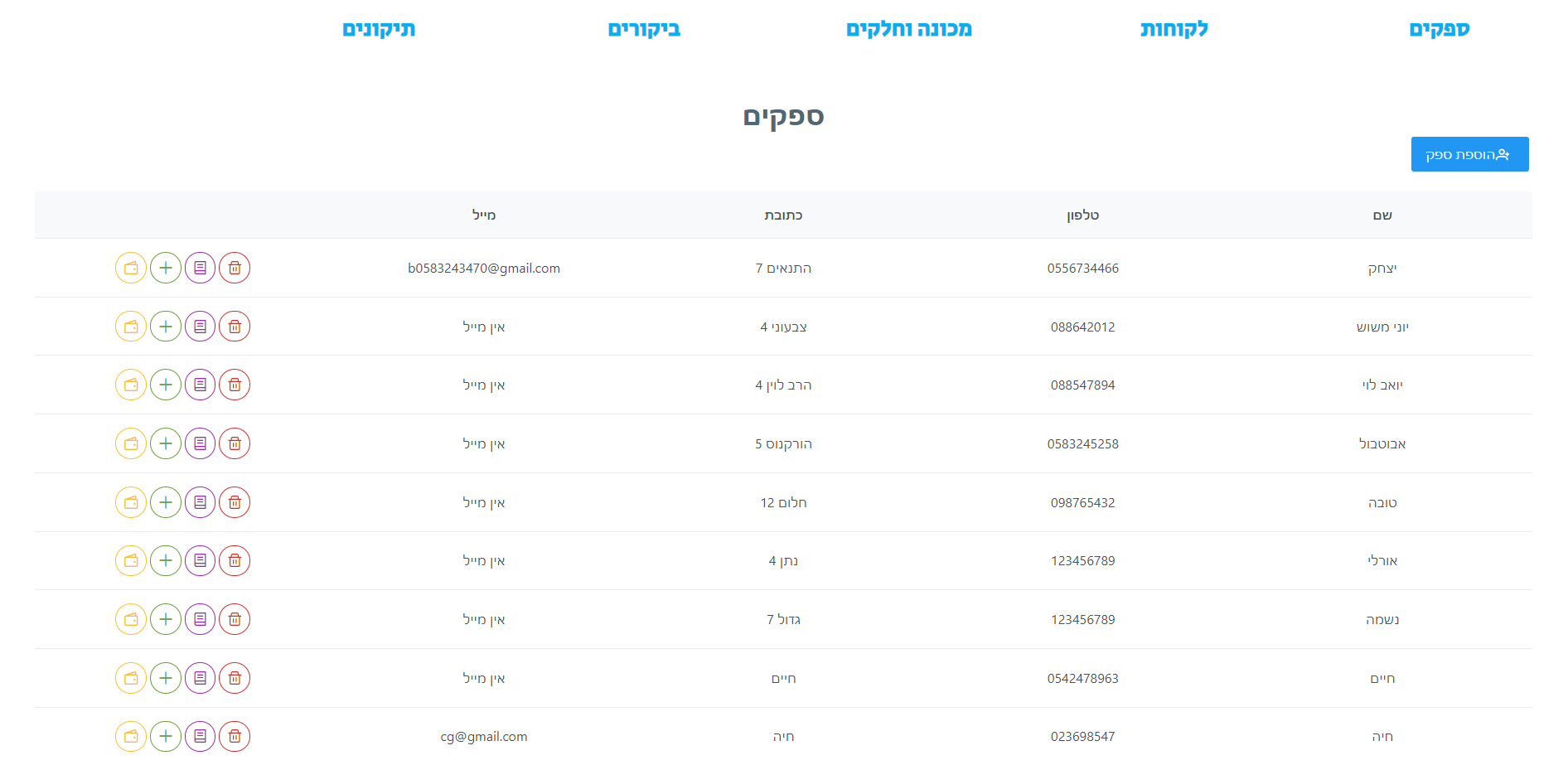
כנס למערכת תוכל לצפות בסוכנים ובלקוחות וכן בהזמנות ובביקורים השונים.

### מסכים:

#### מסך ראשי:



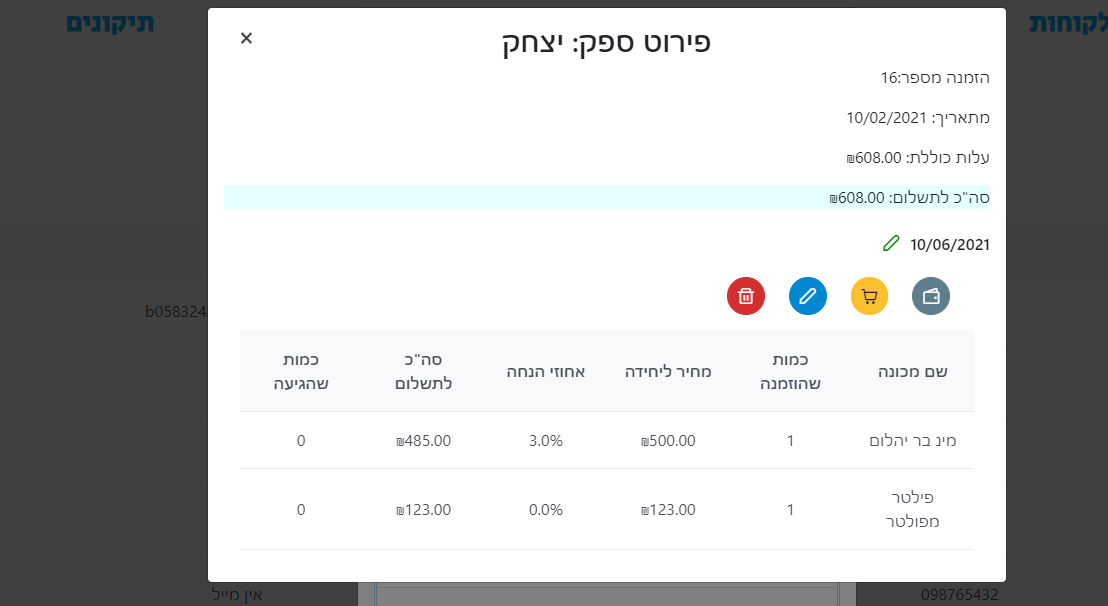
#### ספקים



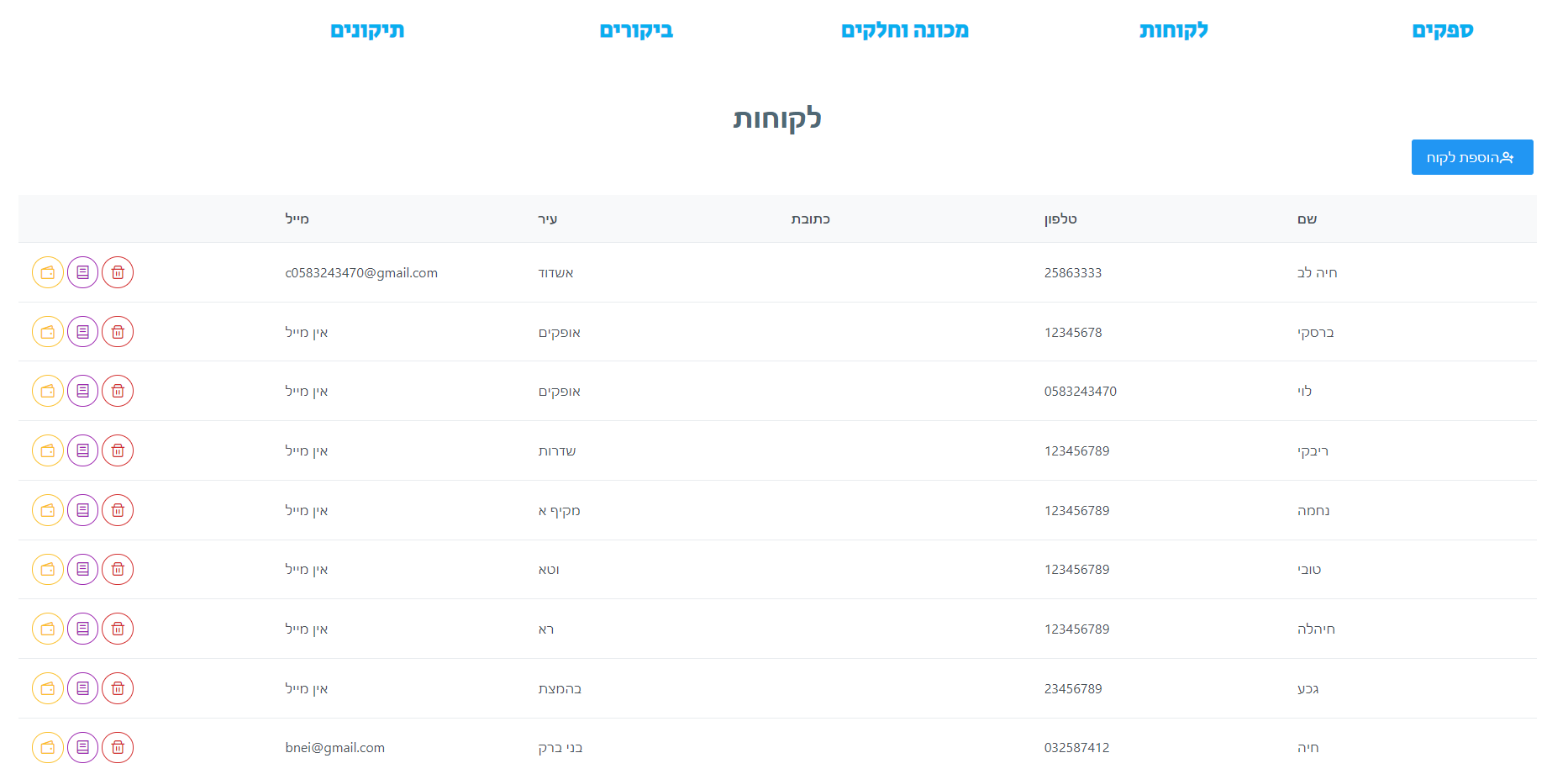
#### הסטרית הזמנות מספק

#### 

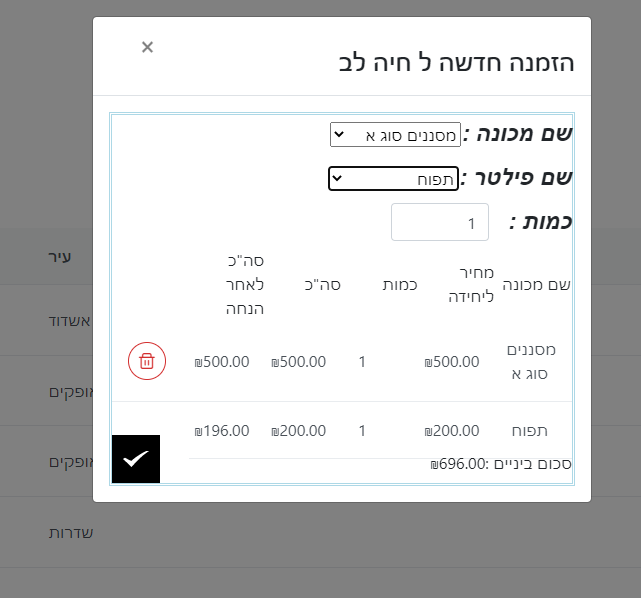
#### פרטי הזמנה מספק



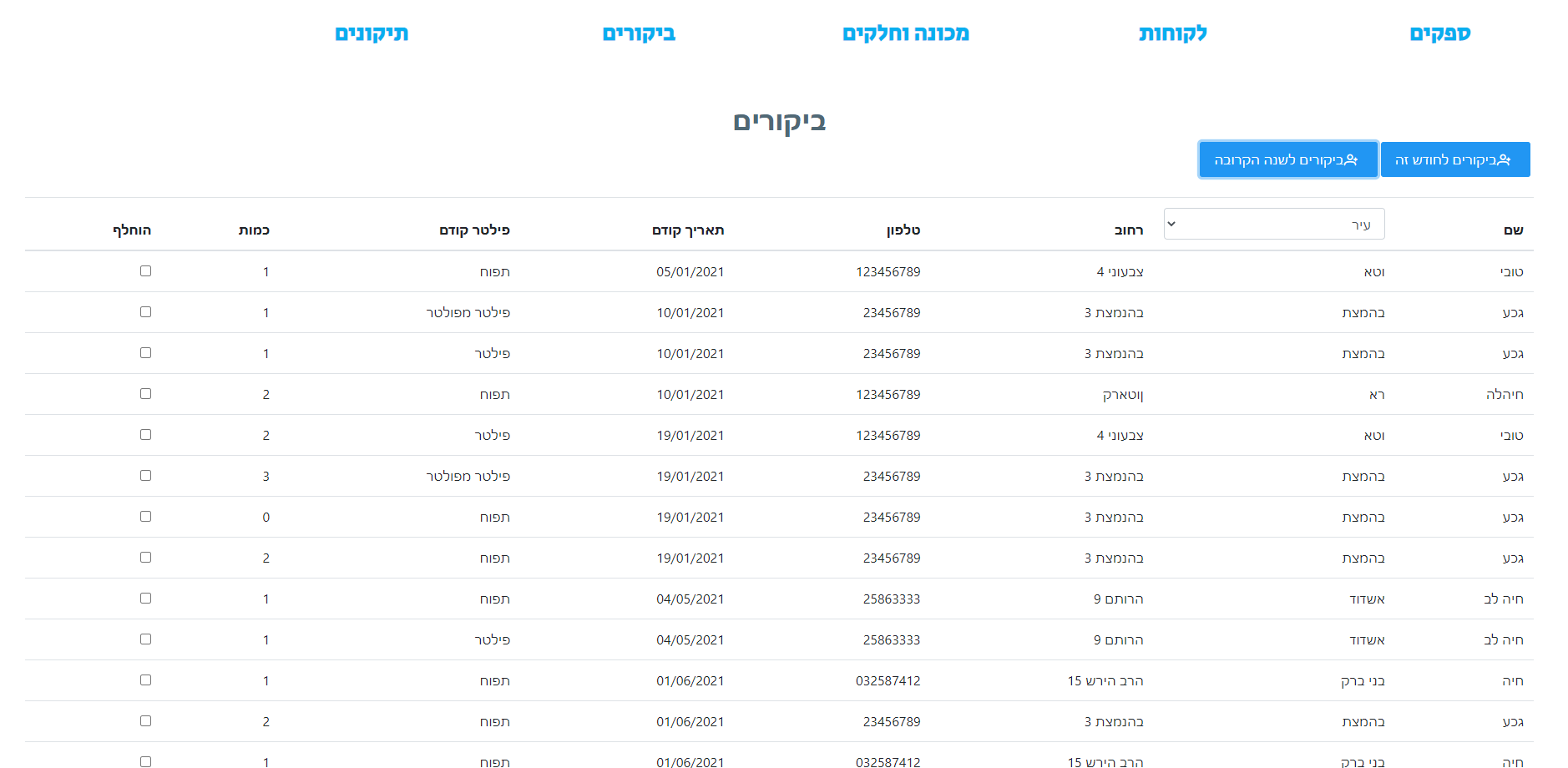
מסך לקוחות



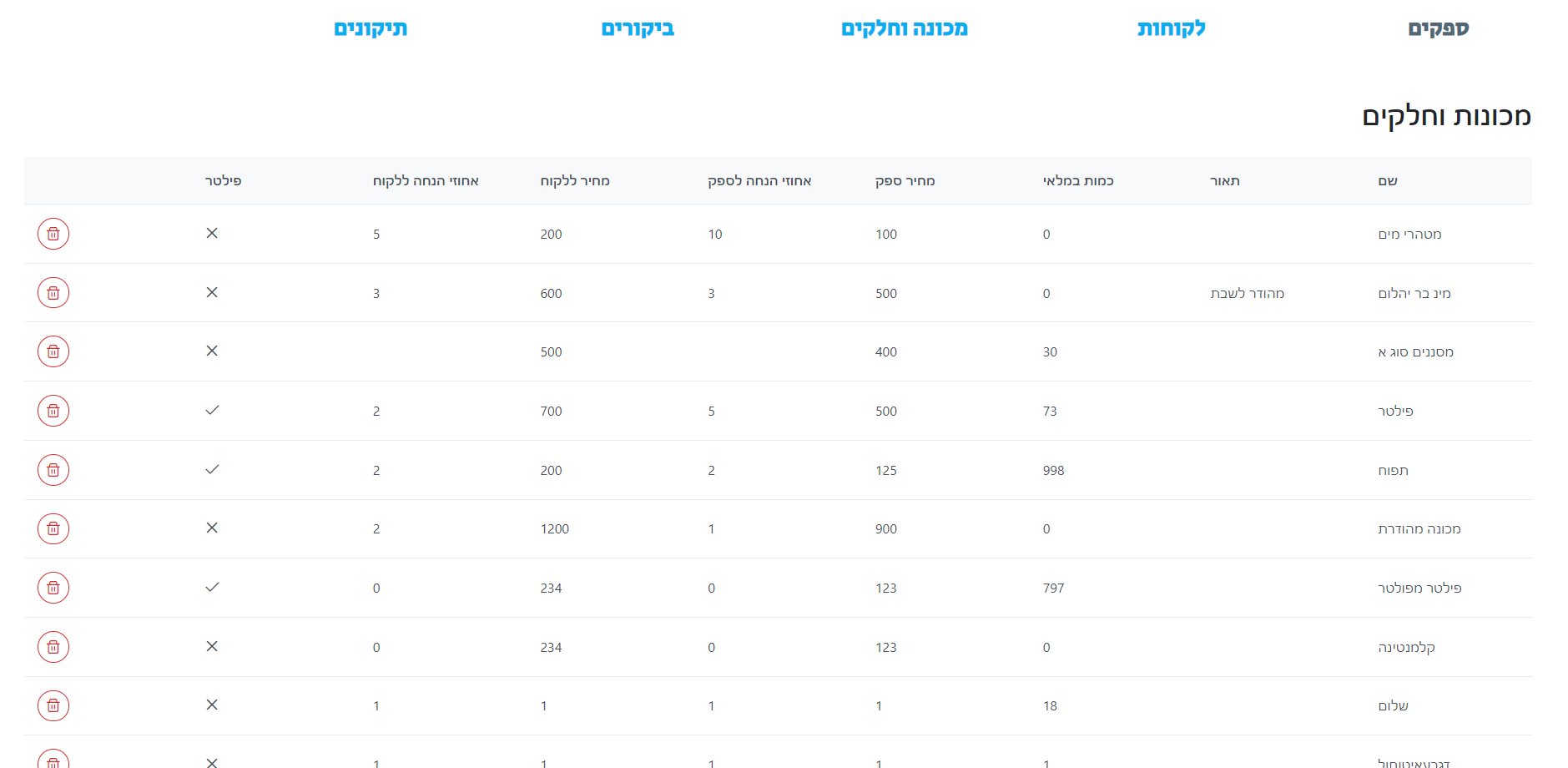
מסך הזמנה חדשה ללקוח



מסך ביקורים



מסך מכונות ופלטרים



## סיכום ומסקנות:

השקענו רבות בבניה נכונה של המערכת ועל תקשורת נכונה עם השרת, ראשית הפרויקט הקנה לנו מרחב הסתכלות ופרספקטיבה רחבה על פרויקט מושלם, משלב ההצעה ועד לתוצר המוגמר תוך דרך ארוכה הכוללת באגים ופתרונם ויצירתיות רבה על מנת להפוך את האתר ליעיל ,חוויית ומקצועי ככל האפשר. במהלך הפרויקט למדנו הרבה ,ואנו מרגישות כי הוא תרם לנו רבות . תחומים שהכרנו באופן שטחי ,נעשו ידידותיים בהרבה ממה שחשבנו .השקענו זמן ומאמץ מרובים בלמידת נושאים וטכנולוגיות חדשים .גילינו שמהנה גם אם לא קל ללמוד ולהכיר תחומים חדשים ולנסות דרכי חשיבה מגוונות ומפתיעות שלא חשבנו עליהם ממבט ראשון. למדנו לחשוב בהגיון וביעילות על מנת להשיג תוצאות במהירות.

ומעל הכול הפנמנו שסבלנות והתמדה-אין כמותם בכל משימה הנראית קשה עד בלתי אפשרית...

## נספחים:

{

  "name": "example-app",

  "version": "0.0.0",

  "scripts": {

    "ng": "ng",

    "start": "ng serve",

    "build": "ng build",

    "test": "ng test",

    "lint": "ng lint",

    "e2e": "ng e2e"

  },

  "private": true,

  "dependencies": {

    "@angular-devkit/build-angular": "^0.1101.1",

    "@angular/animations": "^11.2.3",

    "@angular/cdk": "^11.1.0",

    "@angular/cli": "^11.1.1",

    "@angular/common": "^11.1.0",

    "@angular/compiler": "^11.1.0",

    "@angular/compiler-cli": "^11.1.0",

    "@angular/core": "^11.1.0",

    "@angular/forms": "^11.1.0",

    "@angular/language-service": "^11.1.0",

    "@angular/localize": "^11.1.0",

    "@angular/platform-browser": "^11.1.0",

    "@angular/platform-browser-dynamic": "^11.1.0",

    "@angular/router": "^11.1.0",

    "@coreui/angular": "~2.11.1",

    "@coreui/coreui": "^2.1.16",

    "@coreui/coreui-plugin-chartjs-custom-tooltips": "^1.3.1",

    "@coreui/icons": "^2.0.0-rc.0",

    "@coreui/icons-angular": "1.0.0-alpha.3",

    "@ng-bootstrap/ng-bootstrap": "^9.1.0",

    "@types/jasmine": "^3.6.3",

    "@types/jasminewd2": "^2.0.8",

    "@types/node": "^14.14.22",

    "bootstrap": "^4.6.0",

    "chart.js": "^2.9.4",

    "classlist.js": "^1.1.20150312",

    "codelyzer": "^6.0.0",

    "core-js": "^3.8.3",

    "express": "^4.17.1",

    "flag-icon-css": "^3.5.0",

    "font-awesome": "^4.7.0",

    "jasmine-core": "~3.6.0",

    "jasmine-spec-reporter": "~5.0.0",

    "karma": "~5.2.0",

    "karma-chrome-launcher": "~3.1.0",

    "karma-coverage": "~2.0.3",

    "karma-jasmine": "~4.0.0",

    "karma-jasmine-html-reporter": "^1.5.0",

    "ng-bootstrap": "^1.6.3",

    "ng2-charts": "^2.4.2",

    "ngx-bootstrap": "^6.2.0",

    "ngx-paypal": "^7.0.0",

    "ngx-perfect-scrollbar": "^10.1.0",

    "ngx-spinner": "^10.0.1",

    "primeflex": "^2.0.0",

    "primeicons": "^4.1.0",

    "primeng": "^11.3.0-rc.1",

    "protractor": "~7.0.0",

    "rxjs": "^6.6.3",

    "simple-line-icons": "^2.5.5",

    "sweetalert2": "^10.16.7",

    "ts-helpers": "^1.1.2",

    "ts-node": "^8.10.2",

    "tslib": "^2.0.0",

    "tslint": "~6.1.0",

    "typescript": "~4.0.5",

    "web-animations-js": "^2.3.2",

    "zone.js": "~0.10.3"

  },

  "devDependencies": {

    "@angular-devkit/build-angular": "^0.1101.1",

    "@angular/cli": "^11.1.1",

    "@angular/compiler-cli": "^11.1.0",

    "@angular/language-service": "^11.1.0",

    "@types/jasmine": "^3.6.3",

    "@types/jasminewd2": "^2.0.8",

    "@types/node": "^14.14.22",

    "codelyzer": "^6.0.0",

    "jasmine-core": "~3.6.0",

    "jasmine-spec-reporter": "~5.0.0",

    "karma": "~5.2.0",

    "karma-chrome-launcher": "~3.1.0",

    "karma-coverage": "~2.0.3",

    "karma-jasmine": "~4.0.0",

    "karma-jasmine-html-reporter": "^1.5.0",

    "protractor": "~7.0.0",

    "ts-node": "^8.10.2",

    "tslint": "~6.1.0",

    "typescript": "~4.0.5"

  }

}

## ביבליוגרפיה:

אתרים בנושא תכנות:

* stackoverflow.com
* webmaster.org.il
* [.djamware.com](https://www.djamware.com/post/5a48517280aca7059c142972/ionic-3-angular-5-firebase-and-google-maps-location-tracking)
* .joshmorony.com
* .msdn.microsoft.com