

**Choosing Best Fit Framework For Web  
Application “EVLve”**

Department of Software Engineering  
Capstone Project Phase B, 23-2-D-1

**Supervisor:**

Mr. Alexander Keselman,  
alex@vir-tec.net

**Authors:**

Daniel Moshe Ben David,  
Daniel.Moshe.Ben.Dav@e.braude.ac.il

Idan Daar,  
Idan.Daar@e.braude.ac.il

## Table Of Contents

|  |    |
|--|----|
| Abstract.....                                  | 3  |
| 1. Introduction.....                           | 3  |
| 1.1. Scope of the Project.....                 | 3  |
| 1.2. Project Stakeholders.....                 | 3  |
| 1.3. Project Code.....                         | 4  |
| 2. Project Review and Process Description..... | 4  |
| 2.1. Project Accomplishments Description.....  | 4  |
| 2.2. Architecture.....                         | 7  |
| 2.2.1. Node.js.....                            | 7  |
| 2.2.2. Express.js.....                         | 7  |
| 2.2.3. React.....                              | 8  |
| 2.2.4. Preact.....                             | 8  |
| 2.2.5. MongoDB.....                            | 8  |
| 2.2.6. Bootstrap.....                          | 8  |
| 2.2.7. Tailwind.....                           | 9  |
| 2.2.8. MapBox API.....                         | 9  |
| 2.2.9. Open Charge Map API.....                | 9  |
| 2.3. Research Process.....                     | 10 |
| 2.4. Challenges and Solutions.....             | 11 |
| 2.5. Discarded ideas.....                      | 12 |
| 2.6. Testing Process.....                      | 13 |
| 2.6.1. First Contentful Paint (FCP).....       | 13 |
| 2.6.2. Time to First Byte (TTFB).....          | 14 |
| 2.6.3. Cumulative Layout Shift (CLS).....      | 15 |
| 2.6.4. Largest Contentful Paint (LCP).....     | 16 |
| 2.6.1. Functional Testing.....                 | 16 |
| 2.6.2. Performance Testing.....                | 18 |
| 2.7. Results and Conclusions.....              | 20 |
| 3. User Documentation.....                     | 23 |
| 3.1. User's Guide.....                         | 23 |
| 3.1.1. Registration and Login.....             | 24 |
| 3.1.2. Map Page.....                           | 26 |
| 3.1.2. Navigation to Station.....              | 27 |
| 3.1.3. Admin Control Panel.....                | 28 |
| 3.2. Maintenance Guide.....                    | 28 |
| 3.2.1. Hardware Requirements.....              | 28 |
| 3.2.2. Software Requirements.....              | 29 |
| 3.2.2.1. Operating System.....                 | 29 |
| 3.2.2.2. Tools.....                            | 29 |
| 3.2.3. Installation.....                       | 29 |
| 3.2.4. Maintenance.....                        | 30 |
| 3.3. Data Structures.....                      | 31 |
| 4. References.....                             | 34 |

## **Abstract**

This paper presents a study on the selection of the best fit framework for web application development. The motivation behind this work is to provide web developers with a comprehensive solution to make an informed decision on which framework to choose for their web application development. Our study aims to address this gap in the literature by implementing a web application using a few framework combinations and evaluating which one performs best.

## **1. Introduction**

### **1.1. Scope of the Project**

Our project, “EVoIve”, is a web application designed for electric vehicle owners, allowing them to locate nearby charging stations around their current location and receive information for a specific position. By developing this application using a few different frameworks combinations, we hope to provide developers with a clear understanding of the benefits and disadvantages of different web development frameworks and ultimately help them make an informed decision on which one to use.

### **1.2. Project Stakeholders**

The potential stakeholders on our platform are:

- **Electrical car owners** - The application allows them to get information about electric charging stations. They can see the places around them that have available charging stations, get the address and start navigation to the selected station.
- **Web developers** - Web developers are the potential stakeholders who can benefit from our project, as it provides them with a comparative analysis of different web development frameworks and helps them to choose the best one for their needs. Our project also demonstrates how to use various frameworks to build a web application that integrates with an external API and provides a user-friendly interface.

## 1.3. Project Code

[Evolve](#)

## 2. Project Review and Process Description

### 2.1. Project Accomplishments Description

The main goal of our project is to compare different web development frameworks and choose the best one for building a web application for electric vehicle owners. The web application allows users to locate nearby charging stations and get information and navigation for a specific station. We divided the project into several phases, each addressing a specific aspect of the development process:

#### Framework Selection Process

We started by researching existing web development frameworks and narrowing down our options to JavaScript-based frameworks such as React, Node.js, Bootstrap, and Tailwind. We then conducted a comparative analysis of these frameworks based on various criteria such as size, performance, popularity, documentation, and features. We decided to try two combinations of frameworks and evaluate which one performs better.

#### Web Application Design Process

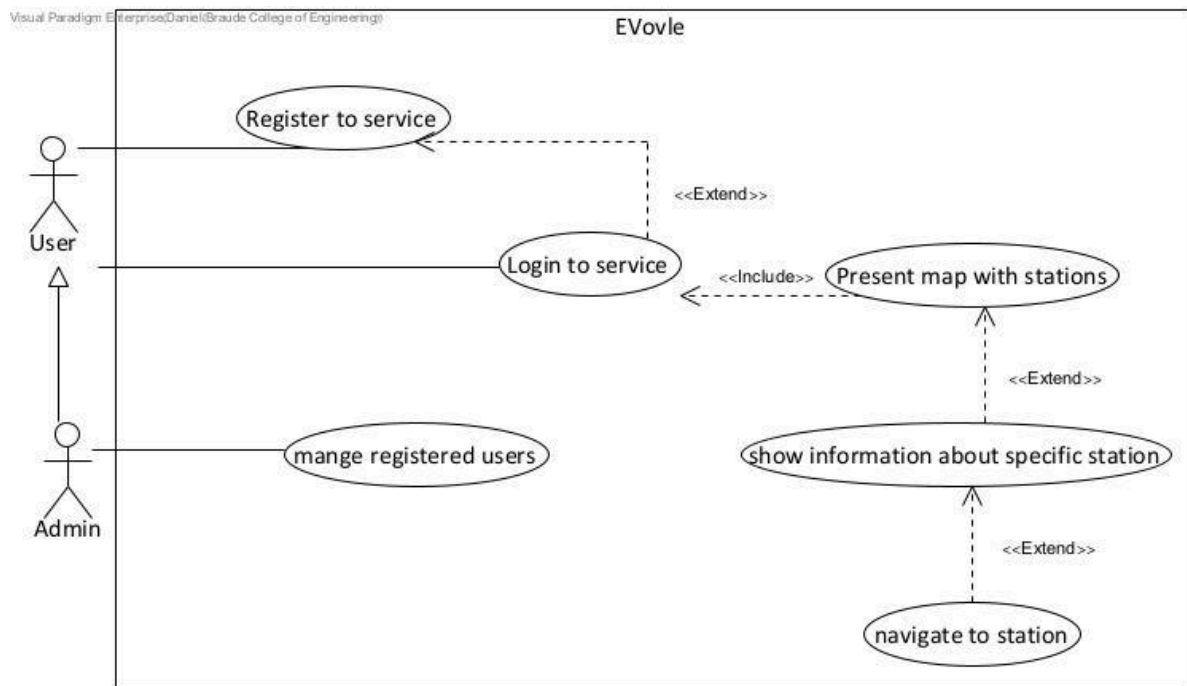
We then designed the web application using use case and activity diagrams to define the user requirements and the flow of the application. We also created a graphical user interface mockup to illustrate the appearance and functionality of the application. We used the Open Charge MAP API to access the data of electric vehicle charging stations around the world.

#### Web Application Implementation Process

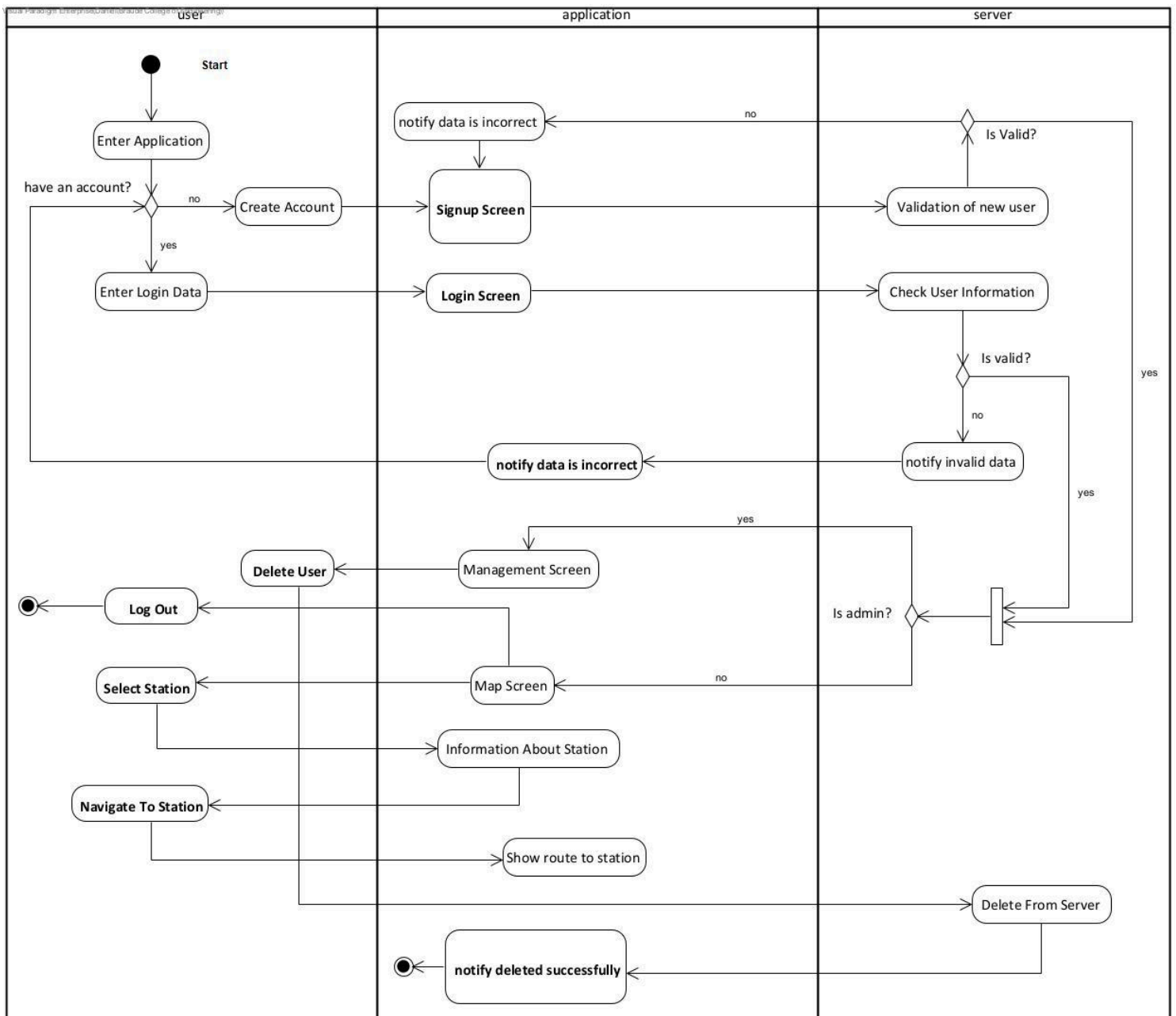
We implemented the web application using two combinations of frameworks: React and Bootstrap, and Preact and Tailwind. We used Node.js and MongoDB for the server-side and database components. We followed the best practices of web development and ensured the quality and readability of our code.

## Web Application Evaluation Process

We evaluated the web application using system testing and white box testing methods to verify the correctness and accuracy of the software. We also measured the performance of the web application using metrics such as file size, build time, and rendering time.

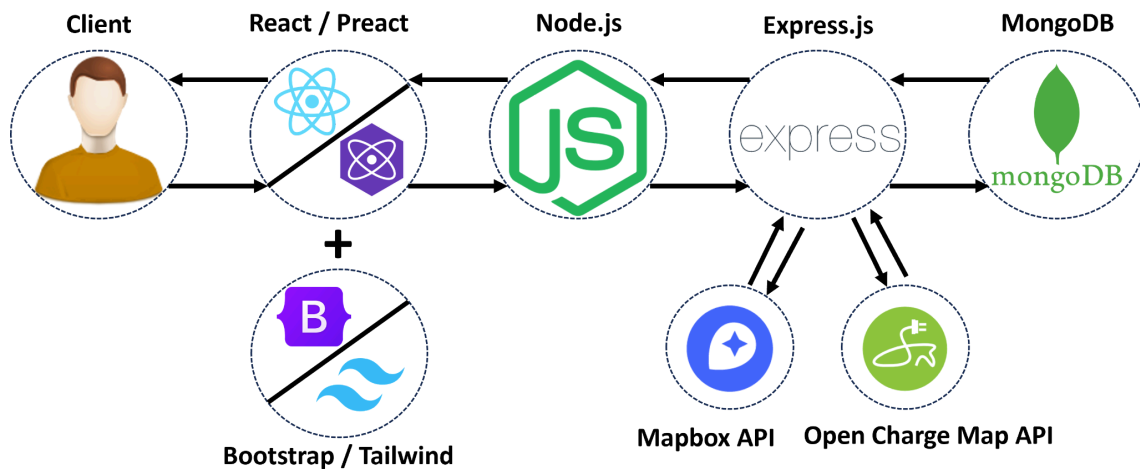


**Figure 1: system's use case**



**Figure 2: system's activity**

## 2.2. Architecture



**Figure 3: System's architecture**

The user interface of our system was constructed using the M.E.R.N. (MongoDB, Express.js, React, and Node.js) stack architecture. This technology stack enabled us to write both client-side and server-side code in JavaScript, providing a seamless development experience. We utilized React for building user interfaces and Express.js and Node.js to effectively communicate with our MongoDB database.

### 2.2.1. Node.js

Node.js is an open-source, cross-platform runtime environment that excels in constructing rapid and scalable server-side and networking applications. It operates atop the V8 JavaScript runtime engine and employs an event-driven, non-blocking I/O structure, rendering it a proficient choice for real-time applications.

### 2.2.2. Express.js

Express.js, an open-source framework for Node.js, is tailored for creating web applications and APIs. It operates under the MIT License and is primarily focused on the back-end of web development.

### **2.2.3. React**

React is a popular JavaScript library that simplifies the process of building interactive user interfaces for web applications. It's a core component of modern web development, allowing developers to create dynamic and responsive UIs efficiently. With React, you can build reusable components, manage the state of your application, and efficiently update the DOM. It's a versatile tool for web developers, making it easier to create complex, interactive web applications with a focus on component-based design and efficient rendering.

### **2.2.4. Preact**

Preact is a lightweight JavaScript library for crafting fast and efficient user interfaces in web applications. It serves as a streamlined alternative to React, offering a similar component-based approach but with a smaller footprint. Preact enables developers to create responsive and dynamic web interfaces with a strong focus on minimizing overhead and optimizing performance. It's a valuable tool for those looking to build web applications with a lean and speedy framework, making it a great choice for projects where performance and efficiency are paramount.

### **2.2.5. MongoDB**

MongoDB is a NoSQL database program that is available as open source, works across different platforms, and is designed for handling document-based data. It employs JSON-like documents with the flexibility of optional schemas and offers diverse search capabilities, including field-based, range queries, and regular expression searches. MongoDB ensures data reliability and consistency through high availability features like replica sets.

### **2.2.6. Bootstrap**

Bootstrap, which was launched in 2011, has become the most widely used CSS framework within a short period. It gained recognition for its intelligent CSS framework design. Bootstrap is a component-based framework and follows an object-oriented CSS approach. This framework includes a collection of pre-built components that can be utilized to rapidly design standard websites. Additionally, Bootstrap maintains consistency across multiple browsers and devices, making it simple to learn and customize for website development, particularly for those familiar with CSS.



### **2.2.7. Tailwind**

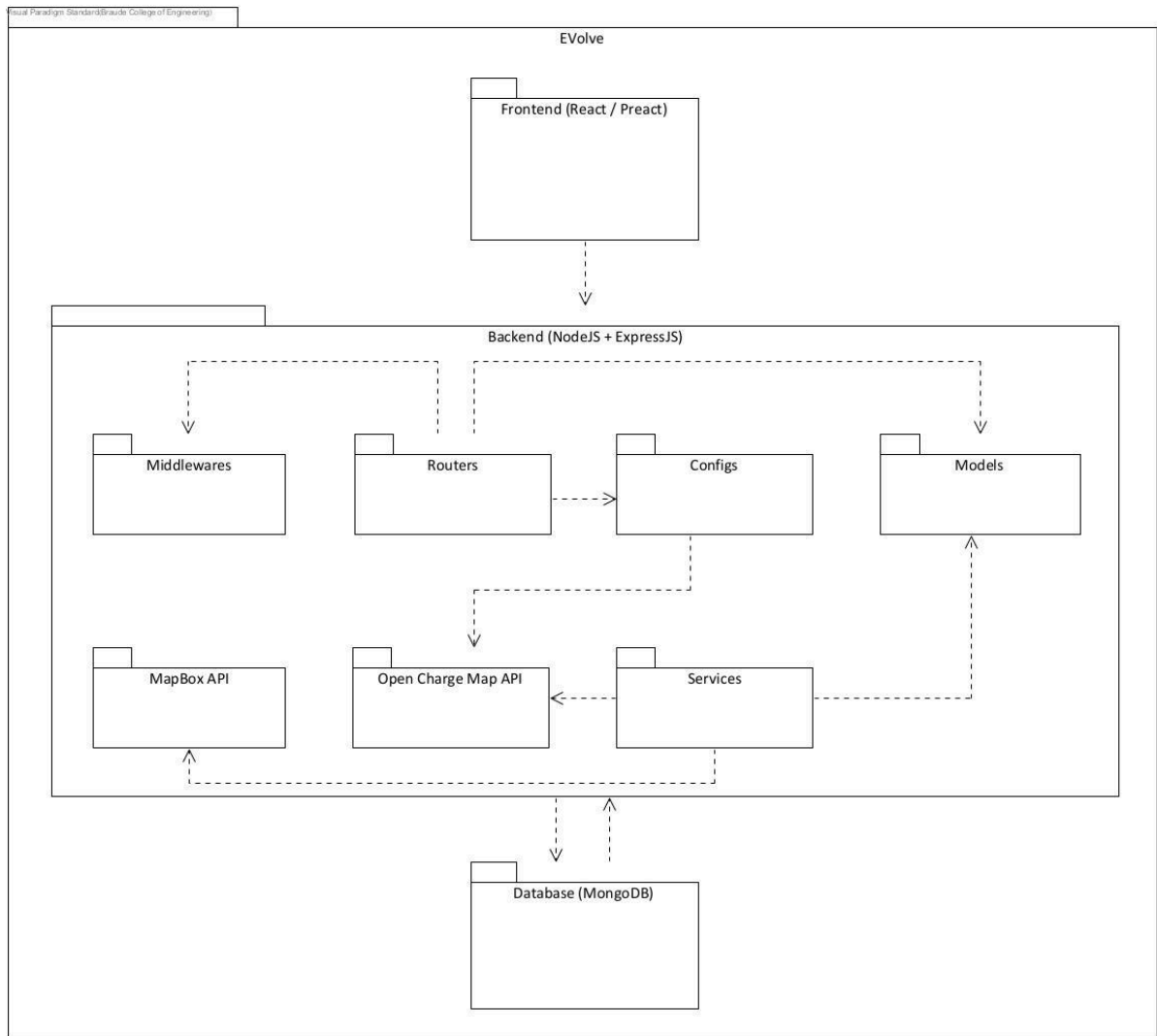
Tailwind CSS was introduced in 2017 and quickly gained recognition for significantly expediting development. It is classified as a Utility-First CSS Framework and provides unparalleled customization capabilities, allowing each website component to be styled as per Tailwind UI requirements. Unlike other frameworks, Tailwind CSS does not offer pre-made components for design purposes. Instead, users can create their own custom components using utility classes. As opposed to writing extensive CSS code, this framework emphasizes creating several classes for HTML elements. Tailwind CSS is exceptionally versatile and can easily transform the appearance and behavior of website elements.

### **2.2.8. MapBox API**

Mapbox API is a powerful tool for creating custom maps and location-based services. It's a key part of Mapbox's platform, enabling developers to integrate interactive maps into apps and websites. With Mapbox API, you can use geospatial data to build engaging maps for various applications, from navigation to e-commerce. It's a versatile tool for location-based solutions and helps businesses make the most of geospatial data.

### **2.2.9. Open Charge Map API**

Open Charge Map API is a vital resource for developers looking to access information about electric vehicle charging stations. This API is part of the Open Charge Map platform and enables seamless integration of EV charging data into applications and services. With Open Charge Map API, developers can offer users real-time details on nearby charging stations, availability, and pricing. It's a versatile tool for electric vehicle-related applications, contributing to the growth of the EV ecosystem and providing valuable information to users on the go.



**Figure 4: System's package diagram**

## 2.3. Research Process

- **Define the research problem and objectives:** The research problem is how to choose the best fit framework for web application development. The objectives are to compare and evaluate different web development frameworks in terms of performance, size, features, and popularity, and to implement a web application using a few framework combinations.
- **Conduct a literature review:** we reviewed existing academic and professional articles that address the problem of choosing web

development frameworks. They also review the background and related work on web development frameworks, such as client-side and server-side frameworks, CSS frameworks, and APIs.

- Design the research methodology: we used a comparative analysis method to compare and contrast different web development frameworks, such as React, Preact, Bootstrap, and Tailwind. We also used diagrams, such as use case and activity diagrams, to illustrate the functionality and flow of the web application.
- Implement the web application: The implementation of a web application using the chosen framework combinations, following the software engineering process. The web application is designed for electric vehicle owners, allowing them to locate nearby charging stations around their current location and receive information for a specific position.
- Evaluate and verify the web application: We used functional testing and performance testing to evaluate and verify the web application. We based our testing on various criteria, such as code size, complexity, build time, performance, and user experience, to measure the quality and efficiency of the web application.

## 2.4. Challenges and Solutions

- **Google Maps API:** One of the challenges we encountered was integrating the API into our web application. We faced difficulties in implementing certain features and ensuring smooth functionality. Google also asked us to insert a payment method even to use the free version of the API as well as Google limit the use of the API for free. To overcome this challenge, we researched other APIs that provide the same service as Google just in a free of charge way and without limitations. we found an API called MapBox that provides great, fast and well documented service and used it instead.
- **MongoDB Database:** Another challenge we faced was working with MongoDB. We encountered difficulties in setting up the database, establishing connections, and managing data efficiently. To address these challenges, we dedicated time to learning MongoDB and its best practices. We

also sought guidance from experienced developers and utilized online documentation and tutorials. With perseverance and continuous improvement, we were able to overcome the challenges related to the MongoDB database.

- **Choosing the First Frameworks to Develop:** Selecting the initial frameworks to develop our web application was a crucial decision. We had to consider factors such as ease of use, scalability, performance, and community support. We conducted thorough research, compared different frameworks, and evaluated their pros and cons. After careful consideration, we decided to use Node.js, Express.js, React, and Preact as our primary frameworks. This decision was based on their popularity, extensive documentation, and the availability of resources and support.

## **2.5. Discarded ideas**

- **Angular-Based Version:** Initially, we considered developing an Angular-based version of our web application. However, after evaluating the requirements and considering the learning curve associated with Angular, we decided to focus on React and Preact. This decision was made to ensure faster development and better utilization of available resources.
- **All Country Stations Presentation:** Another idea that was discarded was to present all the EV charging stations in Israel while focusing the map on the user's current location, this option lets the user look around him and choose any station in his route. After we implemented this option we found that the request from the API to receive an array of all country stations takes a very long time. We decided, after a few tries, to change the request from the API and ask for the stations in a range of 100 km from the users location only. This change improved the loading time of the page significantly.

## 2.6. Testing Process

To ensure the robustness and reliability of our system, we conducted two primary forms of evaluation: functional testing and performance testing. These tests were designed to ensure the reliability of our application both on the client side and the server side.

The functional testing was built upon the tests that we had previously written and documented in the “Capstone Project Phase A” project book. These tests were expanded and redefined to cover more aspects of the system and ensure that all functionalities were thoroughly tested.

The performance testing was conducted using a built-in tool in the Chrome browser called Lighthouse that allows users to measure and analyze the performance of the application using different metrics. We measured four web vitals: First Contentful Paint (FCP), Time to First Byte (TTFB), Cumulative Layout Shift (CLS) and Largest Contentful Paint (LCP).

### 2.6.1. First Contentful Paint (FCP)

The First Contentful Paint (FCP) metric gauges the duration from the user's initial navigation to the page until any portion of the page's content is displayed on the screen. In this context, "content" encompasses text, images (including background images), <svg> elements, or non-white <canvas> elements.

To enhance user satisfaction, websites should aim for a First Contentful Paint of 1.8 seconds or faster. To ascertain that this objective is met for the majority of users, a useful benchmark is to evaluate the 75th percentile of page loads, categorized by mobile and desktop devices.



Figure 5: FCP benchmark times

## 2.6.2. Time to First Byte (TTFB)

TTFB, or Time To First Byte, is a metric gauging the duration from the request for a resource to the commencement of the arrival of the first byte of the response. TTFB encompasses several phases in the request process:

1. Redirect time
2. Service worker startup time (if applicable)
3. DNS lookup
4. Connection and TLS negotiation
5. Request up to the point of the arrival of the first byte of the response

Reducing latency in both connection setup and backend processes contributes to a diminished TTFB. It's crucial to note that TTFB precedes user-centric metrics like First Contentful Paint (FCP) and Largest Contentful Paint (LCP). To ensure a favorable user experience, it is advisable for your server to promptly respond to navigation requests, allowing the 75th percentile of users to experience an FCP within the designated "good" threshold. As a general guideline, most websites should target a Time To First Byte of 0.8 seconds or less.



Figure 6: TTFB benchmark times

### 2.6.3. Cumulative Layout Shift (CLS)

CLS, or Cumulative Layout Shift, is a measurement of the most significant collection of layout shift scores resulting from any unforeseen layout shifts throughout the entire lifespan of a page.

A layout shift occurs whenever a visible element changes its position between consecutive rendered frames. (Refer to the details below for an explanation of how individual layout shift scores are computed.)

A session window, constituting a burst of layout shifts, occurs when one or more individual layout shifts happen rapidly, with intervals of less than 1 second between each shift and a maximum total window duration of 5 seconds.

The largest burst refers to the session window with the highest cumulative score of all layout shifts within that window.

To optimize the user experience, websites should aim for a CLS score of 0.1 or lower.



**Figure 7: CLS benchmark times**

## 2.6.4. Largest Contentful Paint (LCP)

The Largest Contentful Paint (LCP) metric indicates the time it takes for the largest image or text block, visible within the viewport, to be rendered from the moment the user initially navigates to the page.

For an optimal user experience, websites are recommended to achieve a Largest Contentful Paint of 2.5 seconds or faster.



**Figure 8: LCP benchmark times**

### 2.6.1. Functional Testing

| No. | Test Subject   | Expected Result                                     | Result |
|-----|--|---|--------|
| 1.  | User enters an email in the wrong format and press submit. | Pop up with an error message.                       | Pass   |
| 2.  | User tries to login with the wrong user name.              | Alert message “wrong username or password” appears. | Pass   |
| 3.  | User tries to login with the wrong password.               | Alert message “wrong username or password” appears. | Pass   |
| 4.  | User tries to register as Admin with the wrong secret key. | Alert message “Invalid Admin” appears               | Pass   |
| 5.  | User tries to register with the wrong email format.        | Pop up with an error message.                       | Pass   |



|     |   |  |      |
|-----|---|--|------|
| 6.  | An already registered user tries to register using the same email address.              | Alert message “Something went wrong” appears.  | Pass |
| 7.  | New user fills all the sign up forms correctly and presses the “sign up” button.        | Alert message “Registration successful” appears.   | Pass |
| 8.  | User enters his email and password correctly and press submit.                          | Application moves to a “map screen” and presents the closest stations to the user's location.      | Pass |
| 9.  | User clicks on a station icon.  | A floating window with information about the station appears.                                      | Pass |
| 10. | User clicks the “Get Direction” button in the station window.                           | Navigation to the station from the user's current location starts.                                 | Pass |
| 11. | Admin enters his username and password and presses “submit”.                            | Application moves to a “manage screen” and presents a table with all registered users information. | Pass |
| 12. | Admin clicks on “delete” in the user's information row.                                 | Alert message “are you sure you want to delete this user” appears.                                 | Pass |
| 13. | Admin clicks the “approve” button when he asks if he is sure he wants to delete a user. | Alert message “deleted” appears and table updates.   | Pass |
| 14. | Admin clicks the “Logout” button in the management screen.                              | Application moves back to the main sign in page.   | Pass |
| 15. | User clicks the “Logout” button in the map screen.                                      | Application moves back to the main sign in page.   | Pass |

## 2.6.2. Performance Testing

We looked for a popular and reliable tool that provides a way to test our application performance.

Afterwards, we discussed our supervisor with our goal and by his recommendation we decided to use Google’s Lighthouse tool.

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO, and more.

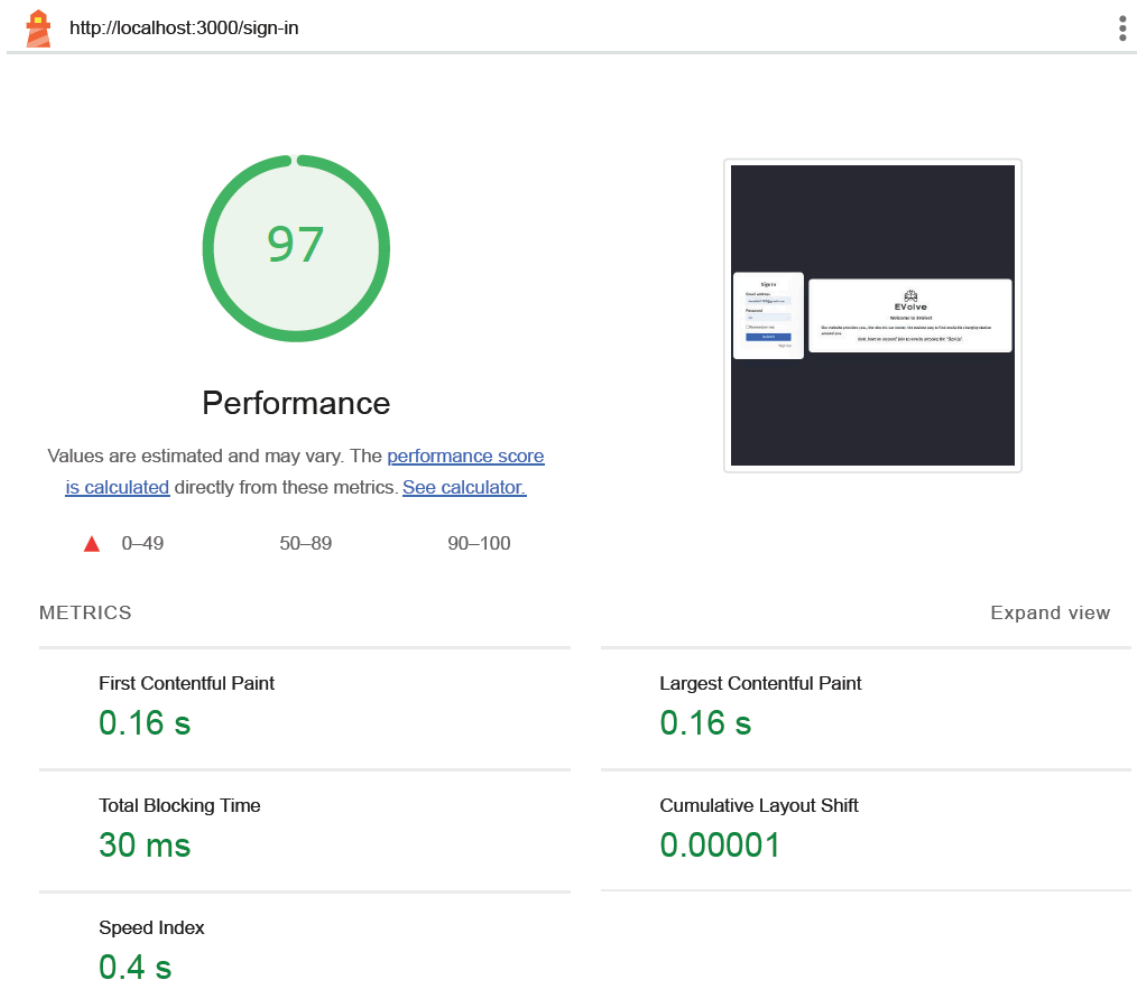


Figure 9: Lighthouse performance test on sign-in page

After we used the Lighthouse tool performance test on both application versions and on each page of the application, we summed up our results in a table for easy comparison.

|                           | React + Bootstrap                           | Preact + Tailwind                          |
|---------------------------|---|--|
|                           | Login / Register page                       |  |
| FCP<br>TTFB<br>CLS<br>LCP | 259.6 ms<br>4.8 ms<br>0.01 ms<br>271.3 ms   | 162.9 ms<br>2.6 ms<br>0.01 ms<br>162.9 ms  |
|                           | Admin control panel page                    |  |
| FCP<br>TTFB<br>CLS<br>LCP | 623.7 ms<br>317.3 ms<br>0.03 ms<br>706.1 ms | 216.8 ms<br>7.09 ms<br>0.03 ms<br>275.7 ms |
|                           | Map page                                    |  |
| FCP<br>TTFB<br>CLS<br>LCP | 199.09 ms<br>8.9 ms<br>0.02 ms<br>204.7 ms  | 180.02 ms<br>5.4 ms<br>0.02 ms<br>190.5 ms |

## 2.7. Results and Conclusions

In this section, we will summarize the main outcomes and findings of our project, as well as reflect on the challenges, decisions, and lessons learned throughout the process.

### Project Outcomes

The main goal of our project was to compare different web development frameworks and choose the best one for building a web application for electric vehicle owners. The web application allows users to locate nearby charging stations and get information and navigation for a specific station. We achieved this goal by implementing the web application using two combinations of frameworks: React and Bootstrap, and Preact and Tailwind. We then evaluated the web application using functional testing and performance testing methods to verify the correctness and accuracy of the software. We also measured the performance of the web application using metrics such as file size, build time, and rendering time.

Based on our evaluation and analysis, we concluded that the Preact and Tailwind combination performed better than the React and Bootstrap combination in terms of performance, size, and user experience. Preact and Tailwind offered a faster, lighter, and more responsive web application, while maintaining the same functionality and features as React and Bootstrap. Therefore, we recommend Preact and Tailwind as the best fit framework for web application development, especially for projects where performance and efficiency are paramount.

### Challenges and Solutions

During the development process, we encountered several challenges that required us to adapt and overcome. Some of the challenges and solutions were:

- **Google Maps API:** We faced difficulties in integrating the API into our web application, such as implementing certain features and ensuring smooth functionality. Google also asked us to insert a payment method even to use the free version of the API, and limited the use of the API for free. To overcome this challenge, we researched other APIs that provide the same service as Google, but free of charge and without limitations. We found an API called MapBox that provides great, fast and well documented service and used it instead.

- **MongoDB Database:** We encountered difficulties in setting up the database, establishing connections, and managing data efficiently. To address these challenges, we dedicated time to learning MongoDB and its best practices. We also sought guidance from experienced developers and utilized online documentation and tutorials. With perseverance and continuous improvement, we were able to overcome the challenges related to the MongoDB database.
- **Choosing the First Frameworks to Develop:** Selecting the initial frameworks to develop our web application was a crucial decision. We had to consider factors such as ease of use, scalability, performance, and community support. We conducted thorough research, compared different frameworks, and evaluated their pros and cons. After careful consideration, we decided to use Node.js, Express.js, React, and Preact as our primary frameworks. This decision was based on their popularity, extensive documentation, and the availability of resources and support.

### Decisions and Considerations

Throughout the project, we made several decisions that influenced the outcome and quality of our work. Some of the decisions and considerations were:

- **Discarding Angular-Based Version:** Initially, we considered developing an Angular-based version of our web application. However, after evaluating the requirements and considering the learning curve associated with Angular, we decided to focus on React and Preact. This decision was made to ensure faster development and better utilization of available resources.
- **All Country Stations Presentation:** Another decision that we made was to present only the EV charging stations within a range of 100 km from the user's current location, instead of all the stations in Israel. We made this decision after we realized that the request from the API to receive an array of all country stations takes a very long time. We decided, after a few tries, to change the request from the API and ask for the stations in a range of 100 km from the user's location only. This change improved the loading time of the page significantly.

- **Testing Metrics and Methods:** We also decided on the testing metrics and methods that we used to evaluate and verify our web application. We chose to use functional testing and performance testing, as well as four web vitals: First Contentful Paint (FCP), Time to First Byte (TTFB), Cumulative Layout Shift (CLS), and Largest Contentful Paint (LCP). We made these choices based on the relevance and importance of these metrics and methods for measuring the quality and efficiency of the web application.

### Lessons Learned and Future Improvements

Finally, we will reflect on the lessons learned and the possible future improvements for our project. Some of the lessons learned and future improvements are:

- **Working as a Team:** One of the most valuable lessons that we learned from this project was how to work effectively as a team. We learned how to communicate, collaborate, and coordinate our tasks and responsibilities, as well as how to support and motivate each other. We also learned how to use various tools and platforms, such as GitHub, VSCode, and Zoom, to facilitate our teamwork and workflow. We believe that these skills and experiences will be beneficial for our future projects and careers.
- **Researching and Learning New Technologies:** Another lesson that we learned from this project was how to research and learn new technologies and frameworks. We learned how to conduct a comparative analysis of different web development frameworks, as well as how to use online resources and documentation to learn and implement them. We also learned how to adapt and troubleshoot when we encountered difficulties or errors. We think that these abilities and knowledge will be useful for our future development and learning endeavors.
- **Improving the User Interface and Experience:** One of the possible future improvements for our project is to enhance the user interface and experience of our web application. We think that our web application can be improved by adding more features and functionalities, such as filtering and sorting options, user reviews and ratings, and social media integration. We also think that our web application can be improved by making it more accessible and responsive, such as adding voice and gesture commands, dark mode, and

multilingual support. We believe that these improvements will make our web application more attractive and engaging for the users.

### **3. User Documentation**

#### **3.1. User's Guide**

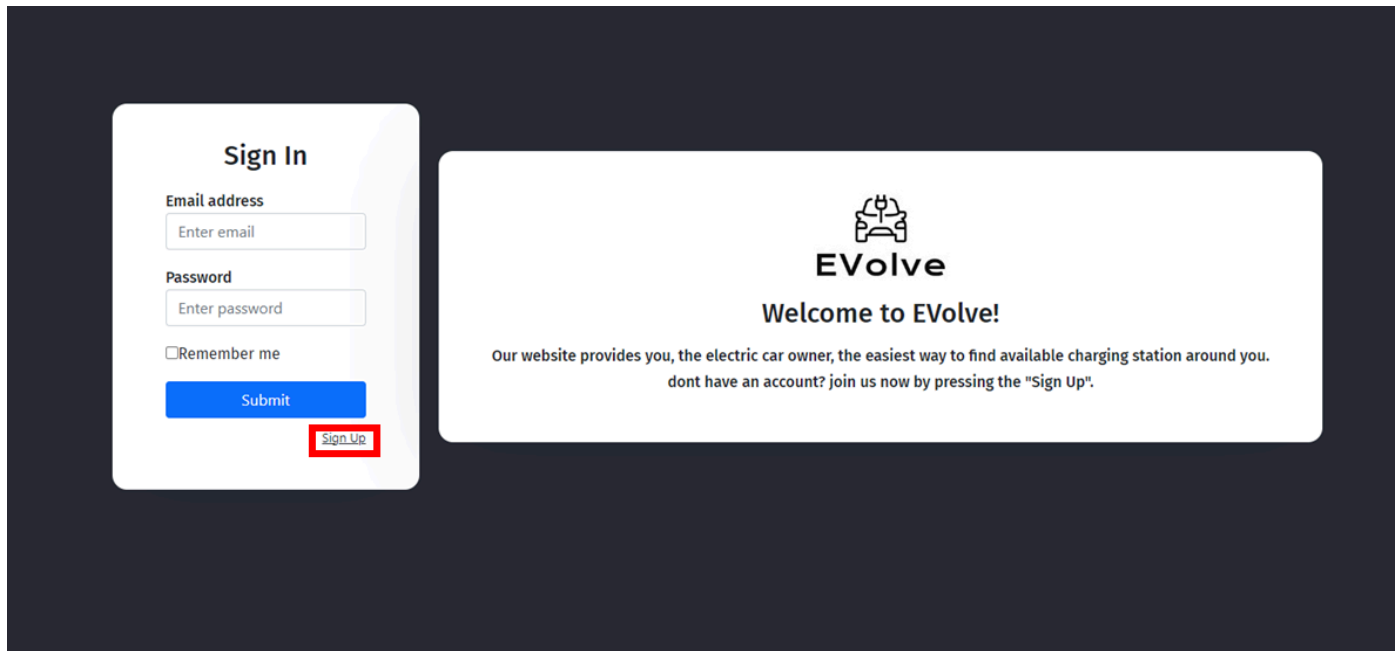
EVolve is a web application designed for electric vehicle owners, allowing them to locate nearby charging stations around their current location and receive information for a specific station. EVolve aims to provide a convenient and reliable service for electric vehicle owners, as well as to promote the use of clean and sustainable energy.

The user guide for this project has been divided into the following sections:

- **Registration and Login:** This section explains how to create an account and log in to the application.
- **Map Screen:** This section describes how to use the map screen to find and navigate to charging stations, as well as to view information and reviews for each station.
- **Manage Screen:** This section is only applicable for admin users, and it explains how to manage the registered users and delete them from the database.

### 3.1.1. Registration and Login

Upon visiting our website, you will encounter our home page. To access information about charging stations near your current location, it is necessary to complete the registration process. Simply click on the

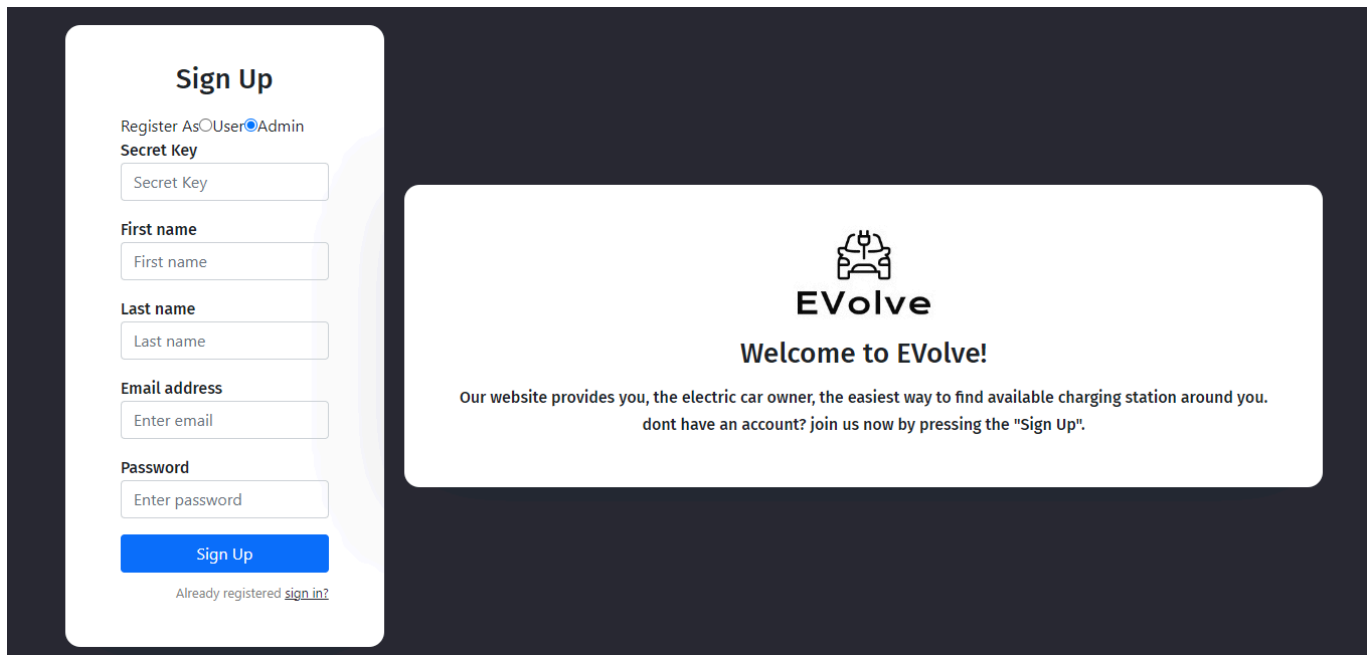


**Figure 10: Home page's sign up button**

“Sign Up” button to get started:

You will then be forwarded to the registration page, where you'll have to fill out your name and Email address and choose a password, if you wish to become an Admin you'll also have to enter a secret key to get an admin permissions.





The image shows a web page with a dark blue background. On the left, there is a white rounded rectangle containing a 'Sign Up' form. The form has a title 'Sign Up' in bold. Below it, there is a text 'Register As' followed by two radio buttons: 'User' (unselected) and 'Admin' (selected). Below this is a 'Secret Key' label and a text input field. Then, there are 'First name' and 'Last name' labels, each followed by a text input field. Next is an 'Email address' label and a text input field. Then, a 'Password' label and a text input field. At the bottom of the form is a blue button with the text 'Sign Up'. Below the button, there is a link 'Already registered [sign in?](#)'. On the right, there is a white rounded rectangle containing the 'EVOlve' logo (a car with a charging plug on top) and the text 'EVOlve' in bold. Below the logo is the text 'Welcome to EVOlve!'. Underneath that, there is a paragraph: 'Our website provides you, the electric car owner, the easiest way to find available charging station around you. dont have an account? join us now by pressing the "Sign Up".'

**Sign Up**

Register As ☐ User ☒ Admin

**Secret Key**

Secret Key

**First name**

First name

**Last name**

Last name

**Email address**


Enter email

**Password**

Enter password

**Sign Up**

Already registered [sign in?](#)

  
**EVOlve**

**Welcome to EVOlve!**

Our website provides you, the electric car owner, the easiest way to find available charging station around you. dont have an account? join us now by pressing the "Sign Up".

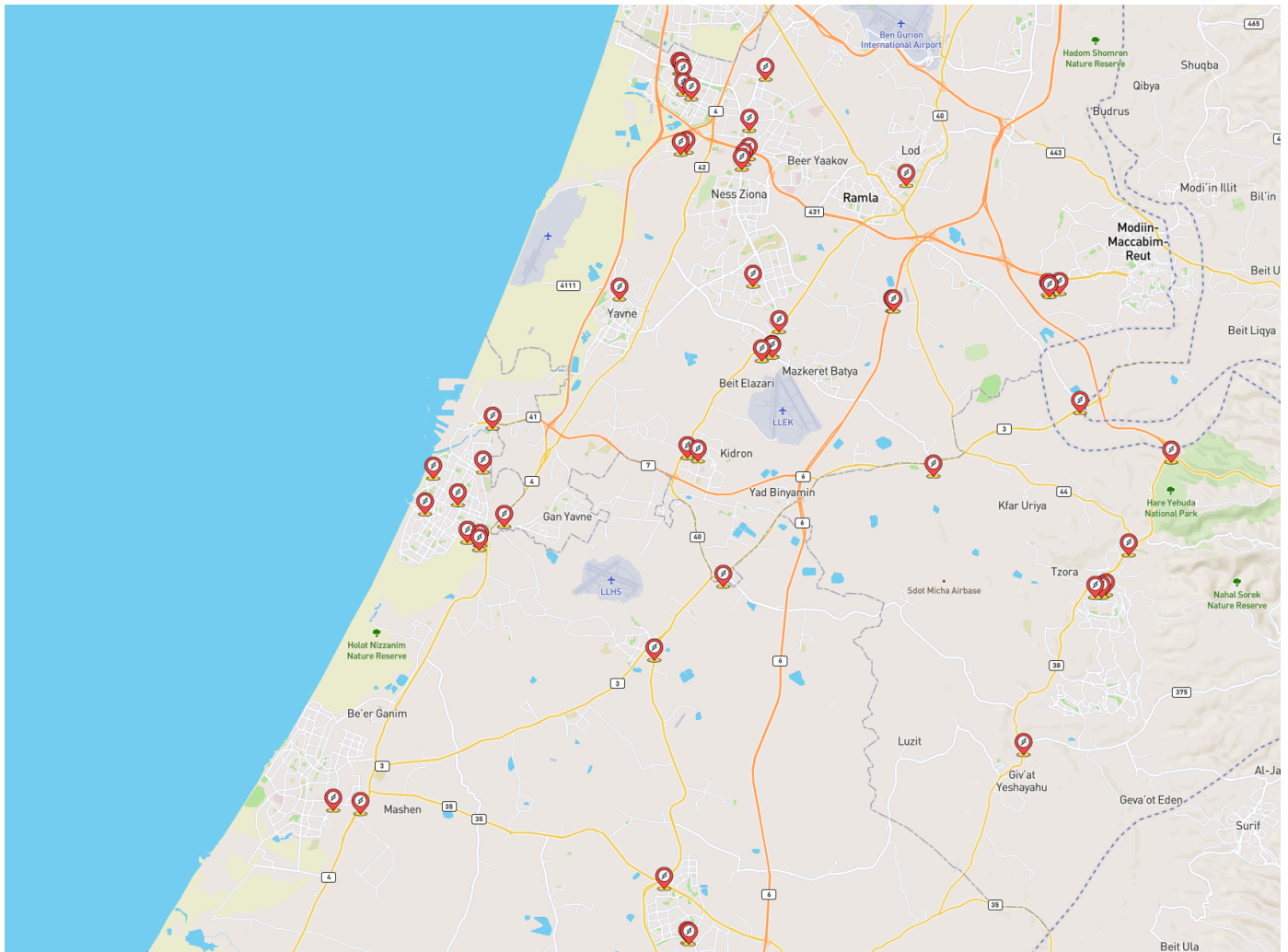
**Figure 11: Sign up page**

Following registration, you will be redirected to the home page where you must provide your details to log in. Upon successful login, you will be automatically directed to the map page displaying the closest charging stations around you. Alternatively, if you are registered as an admin, you will be directed to the admin's control panel.

### 3.1.2. Map Page

After a user's successful login, you will be redirected to a map page, you will be asking for permission to use your location.

After receiving your permission the system will locate you and present a charging station within a range of 100 km from you.



**Figure 12: Map page with charging stations**

### 3.1.2. Navigation to Station

To start a navigation to a station, the user has first to choose the one that he wants to get to. by clicking on each of the markers on the map a small popup window will appear with information about the chosen station and its location. to start a navigation the user simply has to click the “Get directions” button. a new window will appear with route from the current location to the station with an option to navigate.

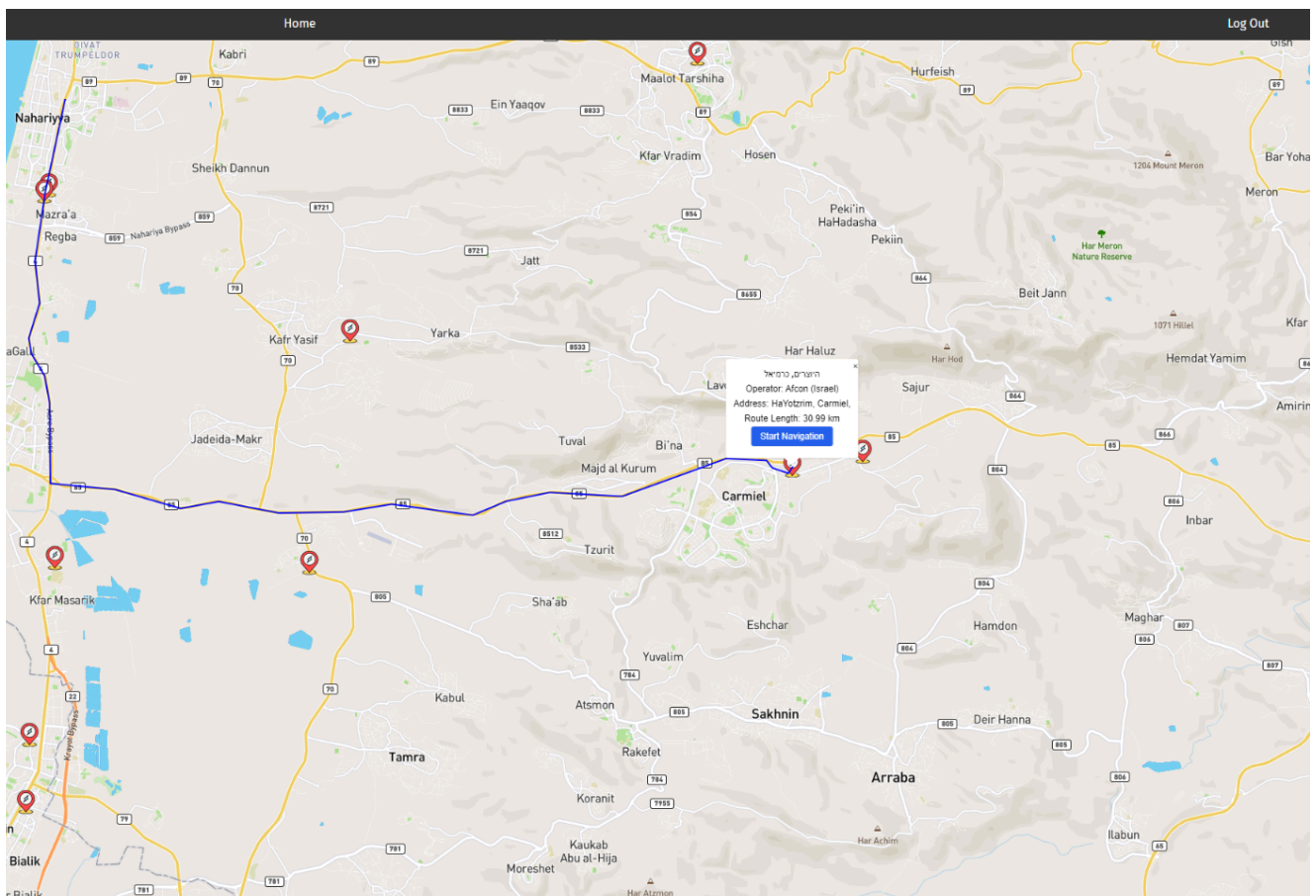


Figure 13: Get directions in map page

### 3.1.3. Admin Control Panel

After an Admin successful login, you will be redirected to the managed page, this page provides a table with all the users registered to the website. The page provides information about the registered users such as name and email and also an option to delete a user.

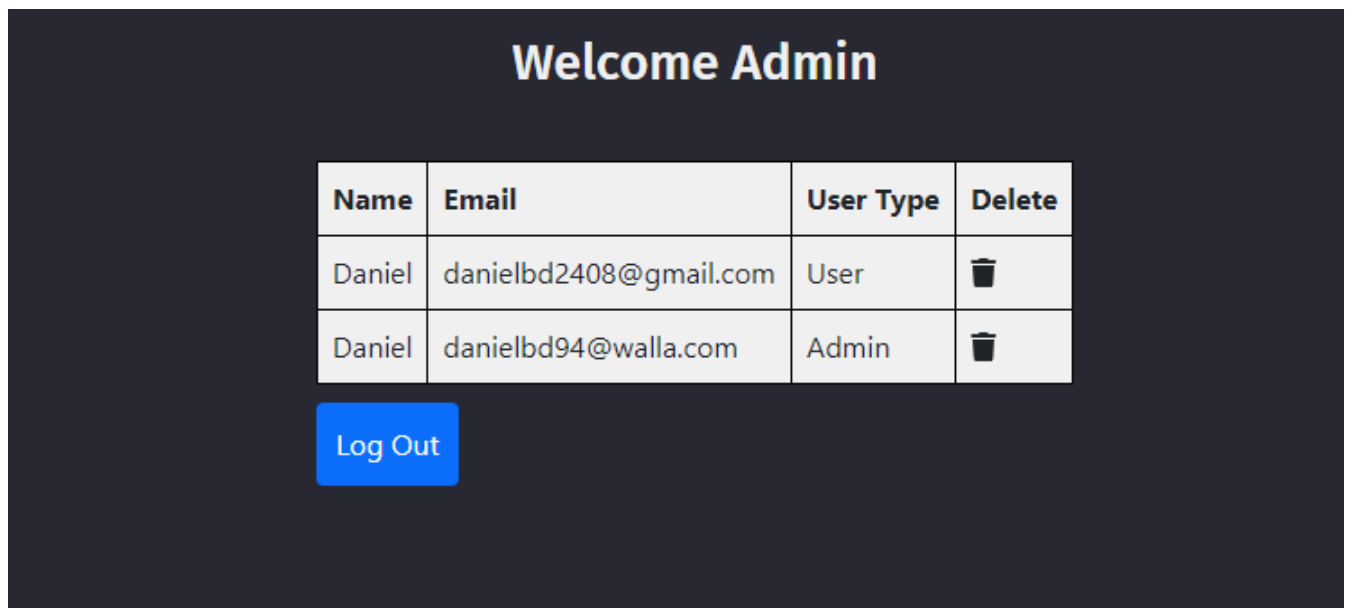


Figure 14: Admin's control panel

## 3.2. Maintenance Guide

In this section, we will discuss the software and hardware requirements that you need to have in order to continue developing our platform.

### 3.2.1. Hardware Requirements

- Processor: 1.8 GHz or faster processor.
- RAM: 4 GB RAM or more.
- Disk Space: 8 GB or more is recommended.

## 3.2.2. Software Requirements

### 3.2.2.1. Operating System

- Windows: Windows 7 SP1 or later.
- macOS: macOS 10.13 or later.
- Linux: Ubuntu 18.04 LTS or later, Debian 9 or later, Fedora 29 or later, or equivalent Linux distribution.

### 3.2.2.2. Tools

- Node.js: Latest stable version.
- Database: MongoDB Compass is recommended but not required (It's possible to work using MongoDB's website). Furthermore, you'll need to be added by us as an admin to the database. For this, you'll need a MongoDB account.
- Development Tools: Code editor or IDE of your choice (Visual Studio Code is highly recommended).

## 3.2.3. Installation

- Basic tools: If not already installed, please download VSCode, Node.js, and Git. Use the following links for the downloads:

- [Node.js](#)
- [VSCode](#)
- [Git](#)

- **Installation verification:** Verify the successful installation of Node.js by running the commands “node -v” and “npm -v” in the CLI. Confirm Git's successful installation by running the command “git --version”.
- **Pull the project:** Create a folder for the project. Then, through the VSCode version control tool, initialize a repository and pull Evolve from GitHub.
- **Install required libraries:** The required packages for the frontend and backend need to be installed separately. Using the CLI, run the “npm install” command once in the server side folder and again in the client side folder. If any errors occur during the installation, you can use the “npm install –force” command.
- **Run the server and user interface (UI):** From the CLI, run the command "npm start" from the server side folder. From another CLI, run the command "npm start" from the client side folder. Once the UI finishes loading, a window should appear with the website in your default browser. If, for any reason, the website doesn't automatically open, you can manually access it from any browser using the address “http://localhost:3000/”.

### 3.2.4. Maintenance

Our platform is mostly stable, therefore maintenance wise there is one main point you should follow for keeping the site running.

- In the event that an API/library becomes deprecated, it should be replaced with a suitable and up-to-date API/library.

### 3.3. Data Structures

- In addition to the properties we defined, MongoDB might also assign the additional properties:

| Table name   | Property             | Type      |
|--|----------------------|-----------|
| MongoDB general properties(might appear in each table) | _id(always assigned) | Object id |
|  | __v                  | Number    |
|  | createdAt            | Date      |
|  | updatedAt            | Date      |

- Users table:

| Table name | Property | Type   |
|------------|----------|--------|
| UserInfo   | fname    | String |
|            | lname    | String |
|            | email    | String |
|            | password | String |
|            | userType | String |

- Charging stations array:

|            |            |            |     |            |
|------------|------------|------------|-----|------------|
| Station(0) | Station(1) | Station(2) | ... | Station(N) |
|------------|------------|------------|-----|------------|

N = number of stations within a range of 100 km from the user's current location.

- Station(i):

| Property               | Type    |
|------------------------|---------|
| AddressInfo            | Array   |
| Connections            | Array   |
| DataProvider           | Array   |
| DataProviderID         | Number  |
| DataProvidersReference | String  |
| DataQualityLevel       | Number  |
| DateCreated            | Date    |
| DateLastConfirmed      | Date    |
| DateLastStatusUpdate   | Date    |
| DateLastVerified       | Date    |
| DatePlanned            | Date    |
| GeneralComments        | String  |
| ID                     | Number  |
| IsRecentlyVerified     | Boolean |
| MediaItems             | Array   |
| MetadataValues         | Array   |
| NumberOfPoints         | Number  |
| OperatorID             | Number  |
| OperatorInfo           | Array   |
| OperatorsReference     | Number  |
| ParentChargePointID    | Number  |
| PercentageSimilarity   | Number  |
| StatusType             | Array   |



|                               |               |
|-------------------------------|---------------|
| <b>StatusTypeID</b>           | <b>Number</b> |
| <b>SubmissionStatus</b>       | <b>Array</b>  |
| <b>SubmissionStatusTypeID</b> | <b>Number</b> |
| <b>UUID</b>                   | <b>String</b> |
| <b>UsageCost</b>              | <b>Number</b> |
| <b>UsageType</b>              | <b>Array</b>  |
| <b>UsageTypeID</b>            | <b>Number</b> |
| <b>UserComments</b>           | <b>Array</b>  |

## 4. References

1. [Switching to Preact \(from React\)](#)
2. [THE OPEN CHARGE MAP API](#)
3. [Mapbox for EV](#)
4. [React Measuring Performance](#)
5. [First Contentful Paint \(FCP\)](#)
6. [Time to First Byte \(TTFB\)](#)
7. [Cumulative Layout Shift \(CLS\)](#)
8. [Largest Contentful Paint \(LCP\)](#)
9. [Google Lighthouse](#)
10. [Tailwindo: Convert Bootstrap CSS code to Tailwind CSS code](#)