Fleet Management System

Prepared by

Abdullah Haidar (91799)

Kamal Al Sayed (91985)

Loaui Beydoun (84736)

LEBANESE UNIVERSITY

FACULTY OF SCIENCES I

DEPARTMENT OF COMPUTER SCIENCES

2018 – 2019

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

We are thrilled to present to you the report about our completely new and revolutionary Fleet management system. We would like to express our gratitude & respect to our honorable teachers for their constant guidance, advice, encouragement & every possible help. We also thank our parents and our family members for giving us the moral support in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes .In addition, we will mention our friends who also inspired and helped us to finish our work. We hope that this project paper has been prepared for the fulfillment of the project requirement. We would also like to thank our authority of Lebanese University.

ABSTRACT

Knowing where the vehicles are, what the drivers doing and monitoring every event in real time is the key parameters for a well-managed decision-making process. In this paper, a novel approach for control and monitoring of a fleet management system using three elements including GPS based vehicle locators, a mobile application and web-based software is proposed to show exact position of the desired vehicle on different maps and take detailed reports of the mission, travelled path, fuel consumption rate, speed limits, and other necessary information according to the customers' requests. The most significant features of the proposed system are its inventory management, vehicle and drivers covering, high accuracy of locations, route optimization and easy operation by the different users at any location.

Keywords

Vehicle Tracking, Fleet Management System (FMS), Cost Reduction,

Optimized Routes, Trip Planning, Inventory Management, Driver Performance Evaluation,

Fleet Safety, Vehicle Maintenance, GPS , Maps

CHAPTER 1: INTRODUCTION

1.1 Overview

The aim of this student project is to develop a fleet management system that can be used for the monitoring of different kinds of motor vehicles such as cars, vans, trucks, buses, motorcycles, etc... The final product consists of three applications: a website, an API, and a mobile application.

The website is an online fleet tracking and information portal that gives the manager access to insightful reports and analytics. It provides the tools needed for the live tracking of the vehicles, trip planning as well as the management of the company's drivers, clients and deliveries. Moreover the website contains extra features such as routes optimization and management of the vehicles inventory.

The mobile application will be used by both the drivers and clients. The drivers can keep track of their assigned deliveries and the clients may order from the company and monitor their deliveries.

The report aims to give a detailed description and documentation of the design and implementation of the entire system. The rest of this chapter gives a general introduction to the project.

Chapter two of this report discusses the background research involved to gain a better understanding of the management of the various types of fleets. It also brings notice to some similar existing application and compares it to our project.

Chapter three discusses the requirements and specifications initially intended to this project. It states all of the functional requirements by specifying the roles of each user of the system.

Using UML diagrams, chapter four brings the conceptual model that clarifies the project entities, their underlying handled data, roles in the application and associations with each other. We will also showcase the different techniques that helped in the conception of the project.

In chapter five we showcase the results of the project using sample screenshots from the application while discussing the main features in each one of them.

The final chapter brings a conclusion to this report and gives a summary of the overall project progress and results. Additionally, it also suggests further work that can improve what has already been accomplished.

1.2 Aims and objectives

In the text below follows the brief description of the initial aims and objectives for the project given in the list of potential projects. Eventually these aims were altered to benefit a more general purpose of such a project and accordingly to what time allowed. Furthermore, the altered aims also to focus much more on the route optimization for the deliveries and management of drivers and clients.

Initial brief:

"*Fleet management is the management of commercial motor vehicles such as cars, vans, trucks, specialist vehicles, and trailers Private vehicles used for work purposes. The aims of the project is to develop an application that allows managing:*

- *Live Fleet tracking (GPS)*
- *Fuel Consumption tracking*
- *Distance Tracking*
- *Time Tracking*
- *Accident Claim*
- *Delivery Workflow*
- *Monthly report*

"

The altered aims for the project are as follows:

- Live tracking of the vehicles with all related information including : location, driver, fuel level, odometer, speed, current deliveries …

- Drivers' performance evaluation according to their driving behavior.
- Management of the vehicle inventory including maintenance plans and service logs.
- Fuel consumption tracking in addition to a fuel log that keep track of all refuel invoices.
- Clients making orders from the company.
- Monitor trips to implement better route planning and job assignment, the optimization of routes include optimal distance, optimal time, and optimal fuel consumption.
- Management of the drivers and clients information including their deliveries.
- Several type of reports that can be generated for different periods of time. Reports include drivers' evolution, costs evaluation and fuel consumptions relative to drivers' performance.

CHAPTER 2: Background

2.1 Purpose of Fleet Management Systems

The use of Fleet Management System ensures that the operation of company vehicles totally align with company goals rather than as source of headache, which typically happens in its absence. The improper use of business vehicles can easily add substantial losses to your business operation, not to mention the time wasted by managers to drivers looking over the issues in search of meaningful ways to address those losses.

It is very vital for any fleet company to not only be able to track its vehicle any time, but also to monitor the driving behavior of the drivers. Highly developed fleet management systems collect a full range of data in real-time and for transport and fleet managers. By combining received data from the vehicle tracking system and the driver mobile application, it is possible to form a profile for any given driver (average speed, harsh breaks, severity of turns, idling). This data can be used to highlight drivers with dangerous habits and to suggest remedial training applicable to the issues. This data also ensures that the driver will not be using the company's vehicle to make personal trips, which might cause the company more fuel and maintenance costs.

2.2 Fleet Management Domain

The global Commercial telematics market size was estimated to be worth USD 17.54 billion in 2016. Technological advancements such as real-time engine diagnostics, GPS tracking, fatigue alert, and drive lane assist are changing the current driving experience, and penetration of such features in mid-range vehicles is expected to boost the market.

Rising fuel prices coupled with growing number of vehicles has resulted in frequent traffic congestions where telematics comes into the picture, which provide alternate routes to prevent congestions thereby improving the fuel economy of the vehicle. These factors are expected to result in an increasing adoption rate of telematics in developed as well as developing nations.

Automobile manufacturers are implementing smart driving systems to minimize human interaction with vehicles to reduce the number of accidents. Over the recent few years, several insurance companies are utilizing telematics to comprehend the driver's driving style to issue a premium, and claim benefits to the owner also called as the usage-based car insurance (UBI).

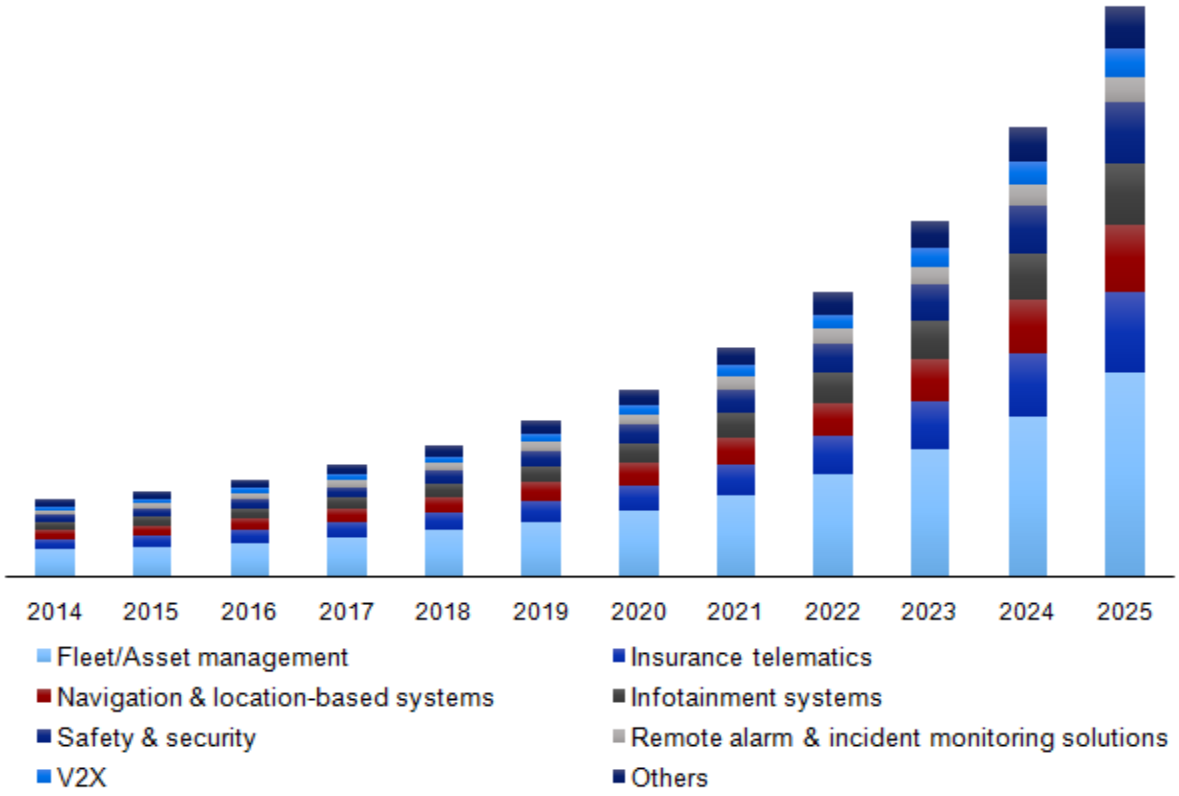

Figure 1 U.S. Commercial telematics market, by solution, 2014 - 2025 (USD Billion)

By combining on-board diagnostics with GPS, the exact location and speed of the vehicle can be determined. This information can also be utilized to determine the internal performance of the vehicle. This is has resulted in changing insurance policies from pay as you drive (PAYD) to pay how you drive (PHYD). In addition, business models such as manage how you drive (MHYD) are likely to gain momentum over the next decade.

2.3 Similar Applications

In order to gain an appropriate understanding of fleet management, it is necessary to research existing software related to fleet management. There are several products revolved around fleet management, most notably Odoo Fleet and Blue Tree Drivers Performance Scoring Application.

2.3.1 Odoo Fleet

Odoo is an open source software that consists of different applications. One of the is developed for fleet management.

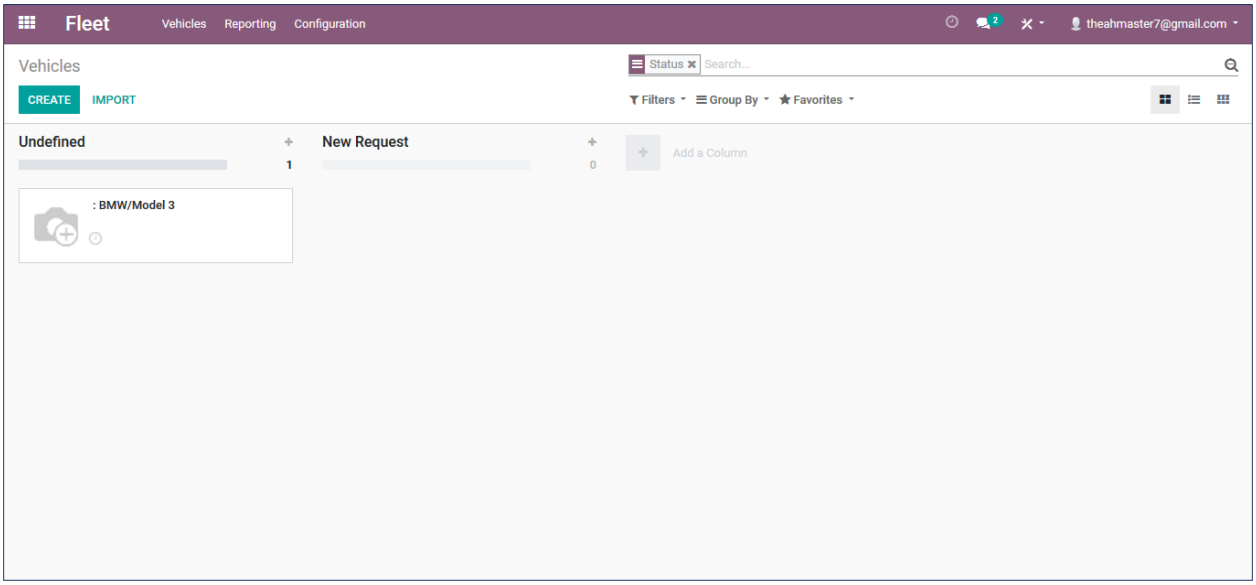

Figure 2 Odoo Fleet Dashboard

Odoo allows the user to manage different information about his company's vehicles including: odometer, costs, contracts, fuel logs, service logs and models.

The system also provides two types of reports costs and indicative costs. The user may also configure the model make of the year, service types, contract types, and vehicle status and vehicle tags.

However this application lacks many important features such as the management of drivers and trip planning. There is no driver performance evaluation or route optimization for the trips. Moreover is there is no interaction between the clients and the manager.

2.3.1 Blue Tree Drivers Performance Scoring Application.

The Driver Performance Scoring App is a comprehensive driver performance app, scoring drivers on the three most important aspects of a driver's behavior that impacts any fleet: performance, compliance and safety. This results in significantly reduced fuel, insurance and maintenance costs, as well as reduced fines and violation incidences. Fleets can select which aspects of driver behavior they wish to monitor – and the weight they wish to attribute to each aspect.

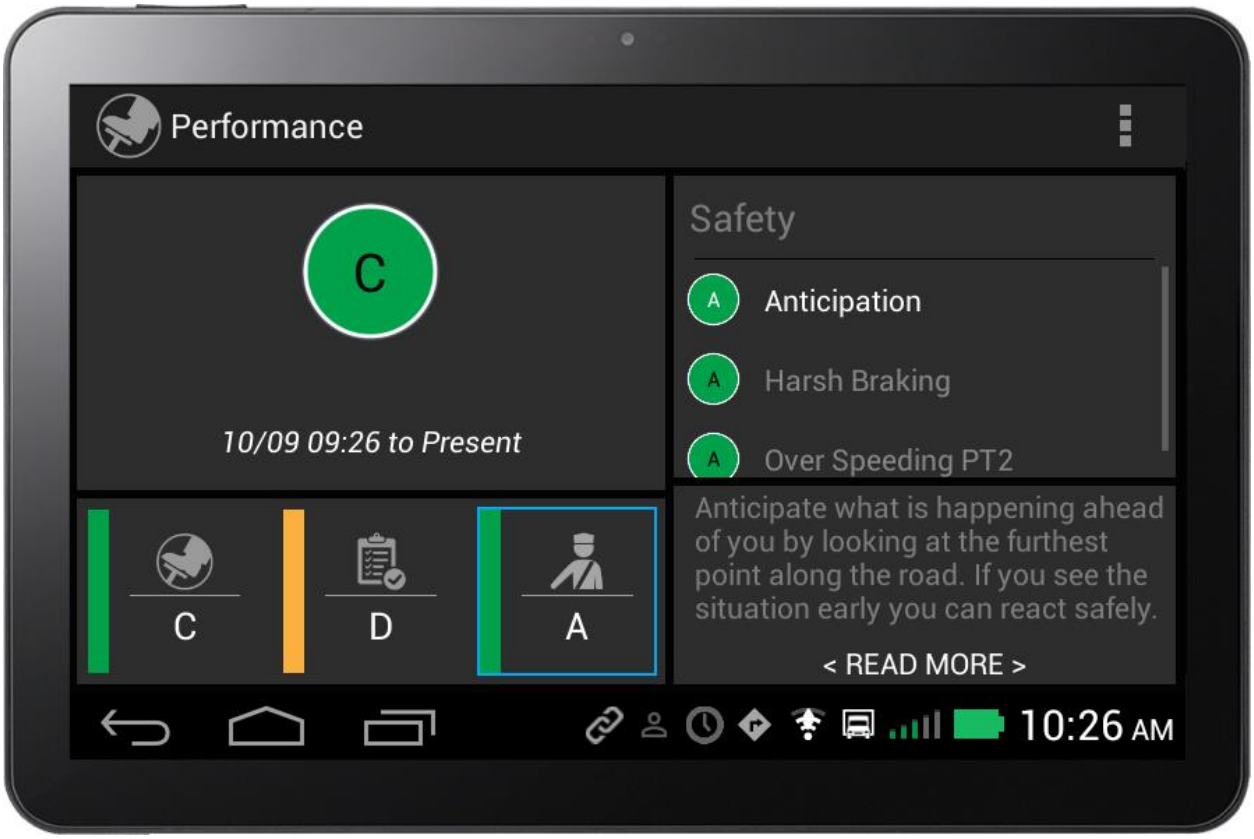

Figure 3 Blue Tree Drivers Dashboard

- Performance

Measure the driver's ability to drive the vehicle in a manner that promotes good fuel consumption and reduced wear and tear on the vehicle. Elements which contribute to this score include harsh acceleration and deceleration; over-revving; idling; coasting; use of cruise control; harsh braking; speeding; brake usage; and hard cornering.

- Compliance

This category measures the driver's ability to respect and adhere to driving and working legislation and road speeds. If infringements are detected, then the driver's score for this category is reduced and near real-time feedback is offered to the driver. Similarly, each instance of over-speeding contributes to a reduction in the driver's score in this category.

- Safety

Safety relates to how the driver operates the vehicle to promote road safety and reduce the potential for accidents. By connecting directly to the engine, Blue Tree can determine how a driver is operating the vehicle and assign them a safety rating. Drivers are scored on the following aspects of their driving behavior: anticipation; harsh acceleration and deceleration; harsh braking; speeding and hard cornering.

This criteria for the evaluation of the drivers' performance was adopted by our application. But still this system lacks important features such as clients management and trip planning.

CHAPTER 3: Requirements Analysis and Specifications

2.1 Introduction

The details of functional features of this application are described below in addition to a use case diagram at the end which might provide a clear summary of the requirements. The users of this application are: manager, drivers and clients and their roles are stated below.

2.2 Manager

Create users: Can create drivers' and clients' accounts by assigning username, password, name, and other personal and professional information.

Edit users: User's name, address, phone, contact no, e-mail and other details can be edited any time by selecting from existing list of users.

Add vehicles: Can add vehicle by specifying the type, model and make of the vehicle. Other information like purchase date, fuel consumption rate, odometer level, CO2 emission rate, etc can also be included. Moreover he may add an image and a map icon (will represent the vehicle on map).

Live tracking: The manager can track his vehicles and view the movement of the vehicles on his map. Other information about the vehicle will be shown to the manager such as fuel level, current driver and vehicle status, current load, etc.

Create maintenance plan: A maintenance plan is a set of scheduled activities that must be applied to a vehicle or group of vehicles. It helps the manager keep track of all the services made to his vehicles and when is the due time for any service. He can create a maintenance plan by specifying a title and a set of activities, and with each activity he must specify how often this service should be applied. The manager will be notified with the due time for all unapplied services every time he logs in.

Fuel log: The manger can track all the refuel operations for his vehicles. He can add a fuel log entry by specifying the vehicle, provider, quantity, price, type of fuel and date.

Add bills: The manager can add bills for the different services offered by to his vehicles, by providing the type of service (oil change, breaks repair…) along with the date, cost and provider

of the service. He can view all these information any time and may filter it according to different factors such as vehicle type, service type, date or provider ….

Create map locations: The company probably has lots of important locations that the manager would like to save and view on his map every time, such as warehouses, centers, providers … he can create a map location by specifying its name, type , image and exact location. All these map locations will be loaded every time he opens the map.

Reply to orders: The manager can view all the orders made to his company with details such as source and destination of the order and the quantity of items for the order. He can see the new orders on his map and add a trip for this order in two ways: he can choose the vehicle and driver for this trip or can let the system choose the optimal route and best driver and vehicle for him. The new route will be represented on the map and the driver will be notified with all its details.

View deliveries: The manager can view the deliveries of any of his vehicles, drivers or clients with all its details including time, source and destination, status, etc.

Cancel deliveries: The manager can cancel any of the drivers' deliveries.

View drivers ranks: The manager can view the ranks of his drivers ordered from the top to the last with the score of each one of them. The rank evaluation is conducting according to the driver's performance reports.

Compare drivers' performance evolution with time: The manager can view charts representing the evolution of the drivers' performance with time. The charts contain three main evaluation principles: performance score, compliance score and safety score. More details about these principles will be provider in the next chapter.

View costs reports: The manager can view pie charts representing the distribution of the costs of the company of any specific year.

View fuel consumption reports: The manager can view the fuel consumption rates for any period of time. The charts include both the optimal fuel consumption and the actual fuel consumption rates. The chart also shows an additional line plot representing the drivers' performance evolution so that the manager can compare the change of the fuel consumption rates with the evolution of the drivers' performance evolution.

2.3 Driver

View deliveries: The drivers can view all of their deliveries with their details on the map, which include: the answered deliveries which has been assigned to them, started deliveries that they are currently working on and finished deliveries (they can view the delivery history).

Start Delivery: Once a delivery is assigned to a driver, he can start the delivery by sending the initial vehicle information.

Update Delivery Information: Once a delivery is started the driver can send all the information of the vehicle during this delivery (location, fuel level, harsh breaking ...).

Finish Delivery: Once a driver is done with the delivery, he can finish the delivery by sending the final vehicle information.

View Score: The driver can view the score of each delivery he made, and the score will be divided according three factors: performance score, compliance score and safety score.

View Rank: The driver can view his rank among all the other drivers.

2.4 Client

Create Account: The client can account by specifying his information. He can use this account to log in later on.

Make an Order: The client can make an order by choosing the company, and the location and quantity of items he wants.

View deliveries: The client can view all of his deliveries with their details on the map.

Figure 4 Usecase diagram

CHAPTER 4: Design and Implementation

4.1 Introduction

In the previous chapters, we have talked about the features that should be offered by our application... This reveals that the following entities are implied in the process… In this chapter we give, using UML diagrams, the conceptual model that clarifies these entities, their underlying handled data, roles in the application and associations with each other. We will also showcase the different techniques that helped in the conception of the project.

4.2 Design and UML Diagrams

4.2.1 UML Class Diagram

In the diagram below:

- Every company has only one manager.
- Every company has a list of drivers, vehicles, deliveries and map locations.
- Every driver belongs to one company and has a list of deliveries.
- Every delivery belongs to a company, and has one vehicle, one driver and one client associated with it.
- Each maintenance plan has a list of vehicles and a list of activities that are offered to the vehicles.
- Each activity has a service and a period so that after this period the service must be repeated, the service can be associated to different activities with each activity having different period for this vehicle.

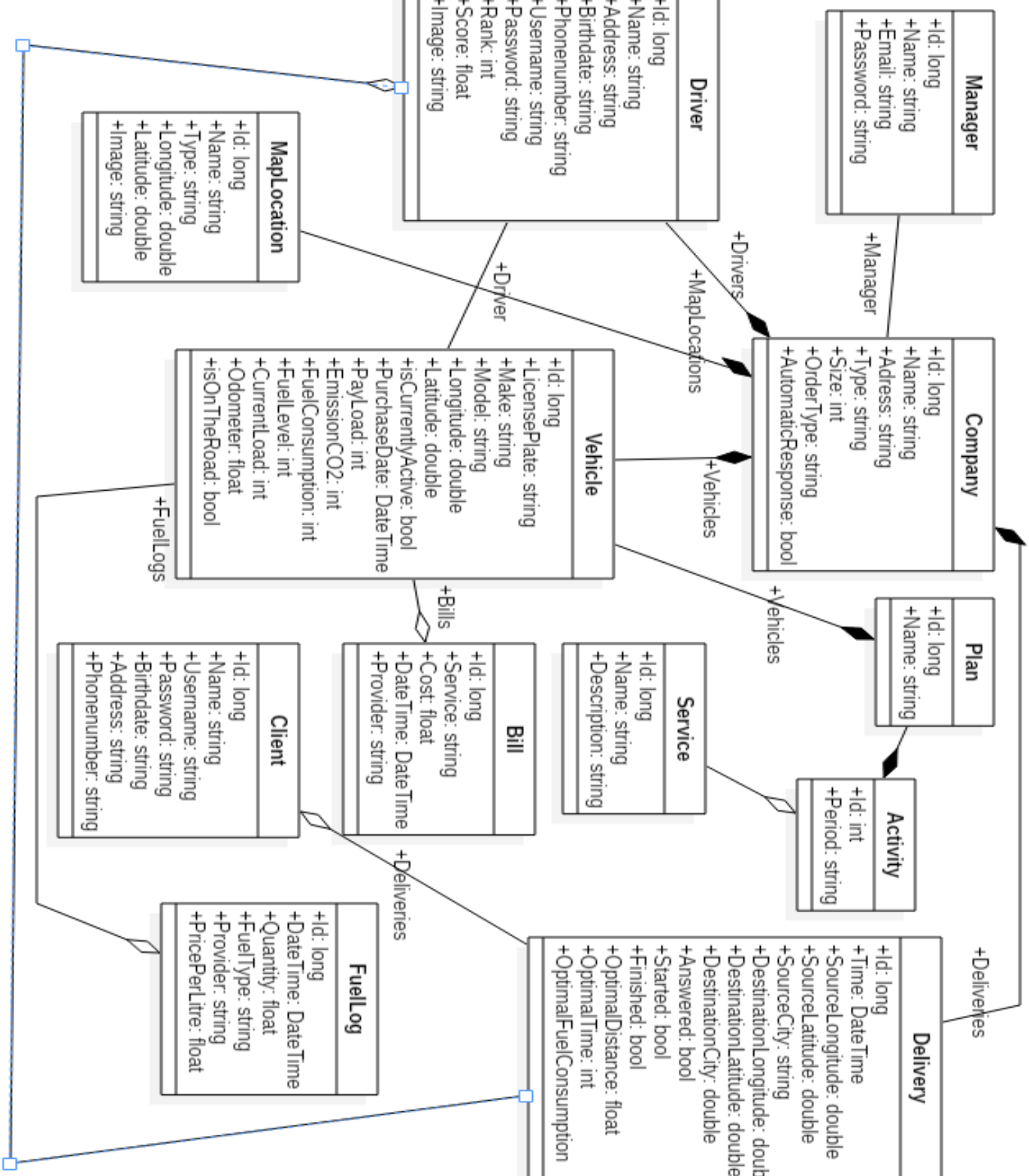

Figure 5 Class diagram

4.2.2 UML Sequence Diagrams

4.2.2.1 Sequence diagram for "Client Makes an Order"

The following diagram show the different steps done when the client makes an order. The delivery controller creates a new delivery and assigns the order information to the new delivery. Then the controller checks if the company has automatic response on, if so it calls the method getNearesVehicle() to find the nearest vehicle to the order location. The getNearestVehicle() retrieves all the active vehicle of the company and then forms the request to be downloaded by the WebClient from the HereMap API. The Json result is then parsed and the nearest vehicle is obtained and returned to the controller which assigns the vehicle to the delivery. Finally the delivery is added to the database and sent back to the client.

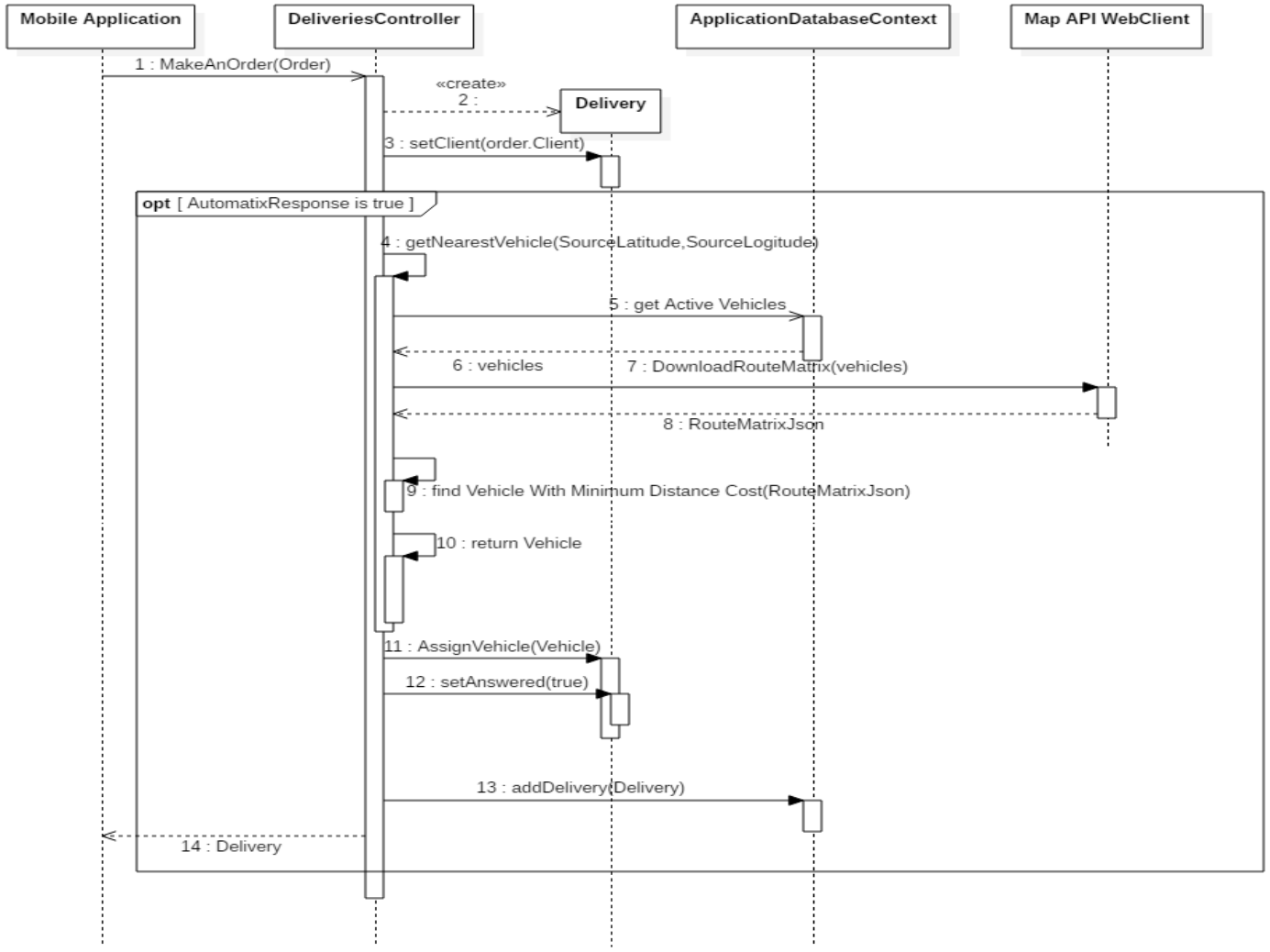

Figure 6 Make an order sequence diagram

4.2.2.2 Add delivery sequence diagram:

The following diagram show the different steps of how the manager adds a delivery. He first retrieves the delivery information from the document, then he requests to find the optimal driver according to the delivery info. After that he gets the source and destination cities. Finally the delivery is added to the database through AJAX post request.

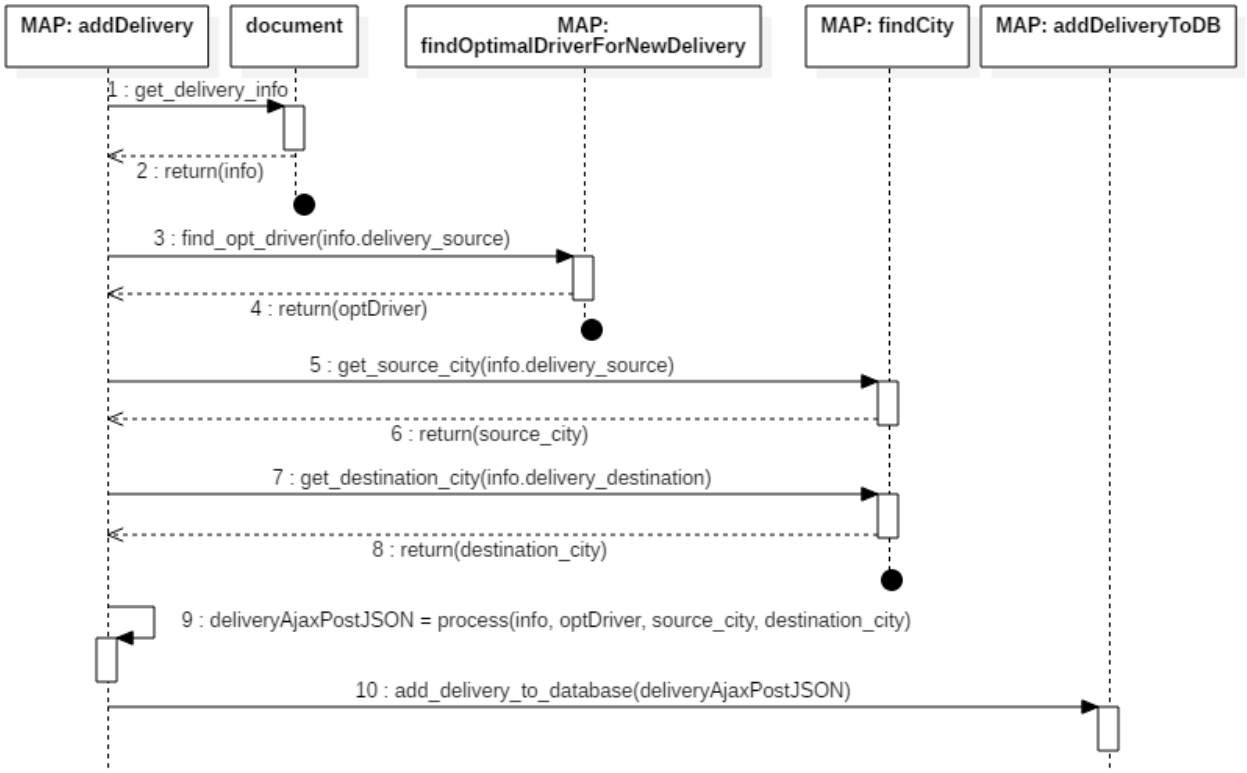

Figure 7 Add delivery sequence diagram

4.2.2.3 Find optimal driver sequence diagram:

The following sequence diagram shows how the best driver is chosen for a delivery.

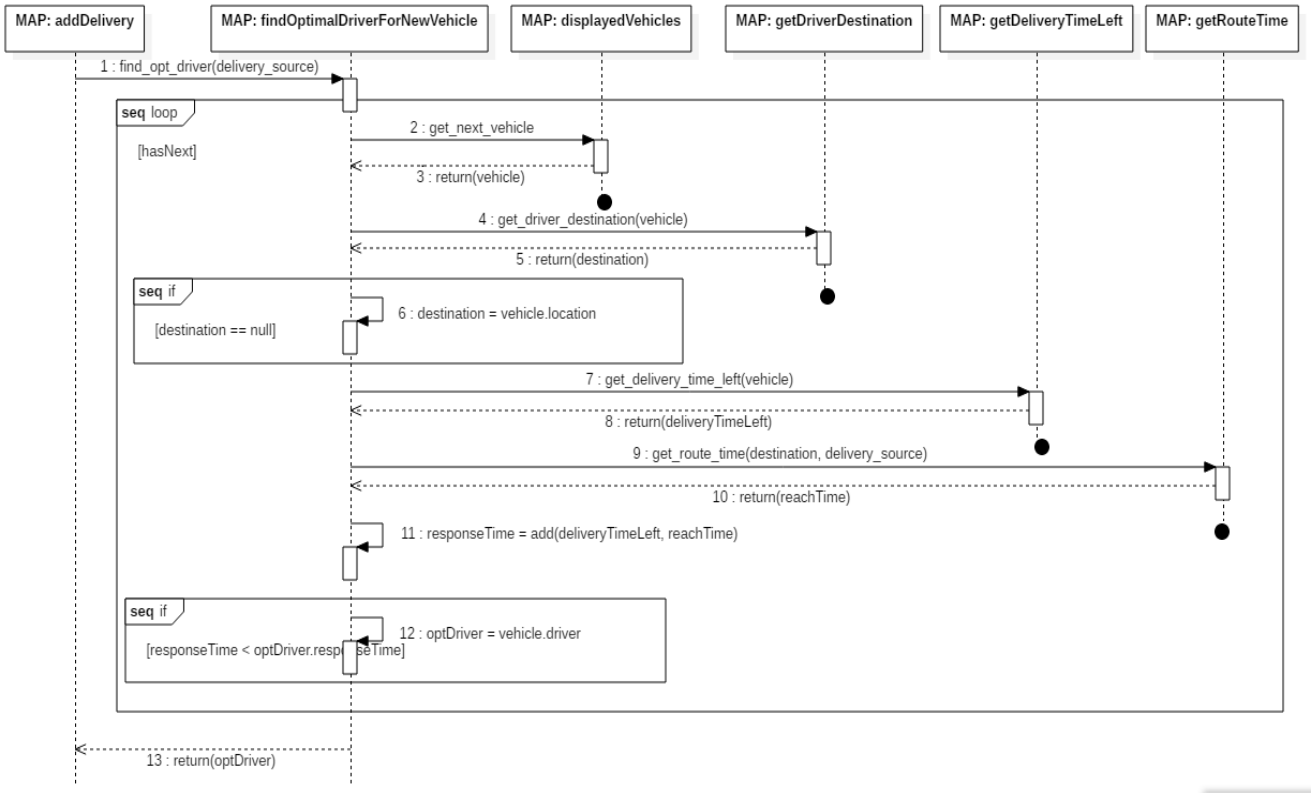

Figure 8 Find optimal driver sequence diagram

4.3 Tools and technologies used

To implement the system we used several software and techniques. In this section we will bring the main software, programming languages, application programming interfaces and libraries used in the implementation of the fleet website, fleet API and the mobile application.

4.3.1 Fleet website

The website is a Model-View-Controller web application implemented using the Asp.NET Core 2.2 framework. The server side programming language used is C# which is an object oriented programming languages and relatively easy to learn for programmers with intermediate experience in object oriented programming. The client side programming language includes both html and JavaScript. The testing of the website was mainly done on Mozilla Firefox.

- **Visual Studio**

The chosen software was Visual Studio 2019 since it provides the best development experience for asp.net web developers. Visual studio is very helpful in several ways, for instance the code generation saves lots of time and helps in learning more about C# and entity frame work. Also the way visual studio is able to combine all the tools needed for web development makes it a very simple and pleasant experience for new learners. The website was tested locally using Visual Studio's own server the IIS Express server.

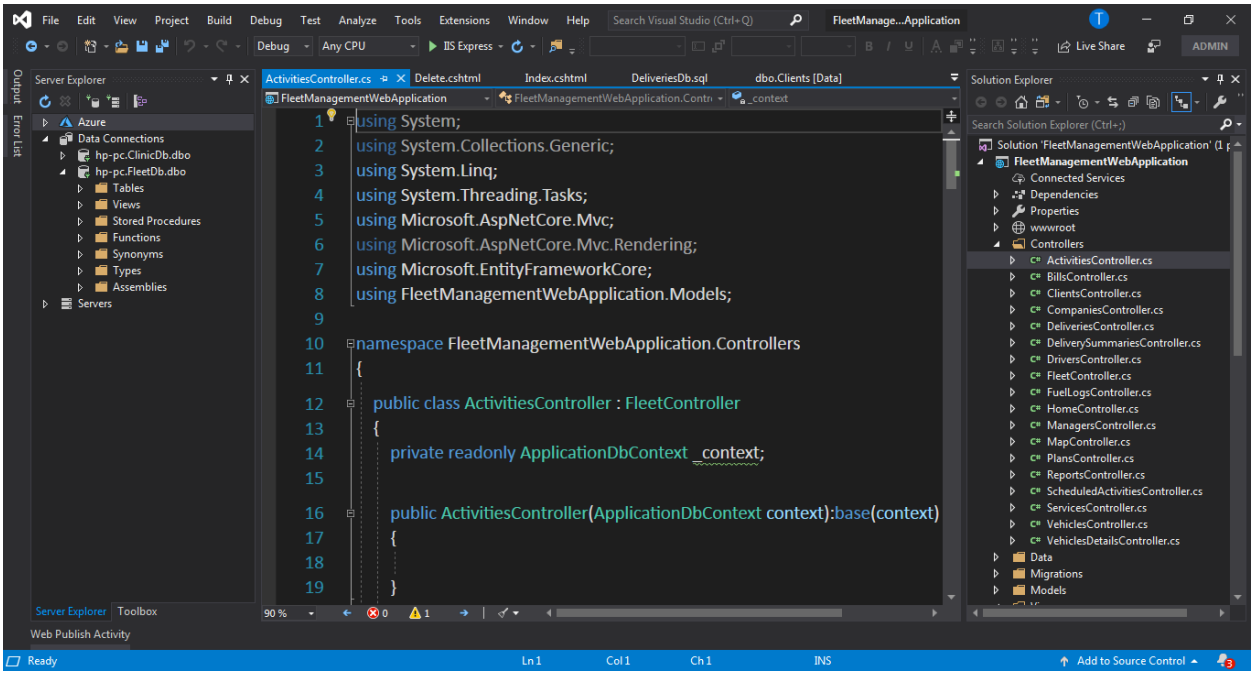

Figure 9 Visual studio

❖ Data Annotations

A valuable and code saving feature in C# which has many benefits. The advantage of using data annotation feature is that by applying Data Attributes, we can manage the data definition in a single place and do not need re-write the same rules in multiple places. The Data Annotation attributes falls into three categories:

1. **Validation Attributes**: Used to enforce validation rules.
2. **Display Attributes**: Used to specify how data from a class /member is displayed in the UI.
3. **Modelling Attributes**: Used to specify the intended use of class member and the relationship between classes.

Here is an example of the use of data annotations in the validation of model attributes:

```
23 references
public class Manager
{
    [Required]
    9 references | 0 exceptions
    public long Id { get; set; }
    [Required]
    [StringLength(100)]
    12 references | 0 exceptions
    public string Name { get; set; }
    [Required]
    [EmailAddress]
    14 references | 0 exceptions
    public string Email { get; set; }
    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    13 references | 0 exceptions
    public string Password { get; set; }
    0 references | 0 exceptions
    public List<Vehicle> Vehicles { get; set; }
```

Figure 10 C# data annotations in models

- **Sessions**

Session is a State Management Technique. A Session can store the value on the Server. It can support any type of object to be stored along with our own custom objects. A session is one of the best techniques for State Management because it stores the data as client-based, in other words the data is stored for every user separately and the data is secured also because it is on the server.

The sessions in this project are used to store data from page to another, and to keep the users personal information:

```
try
{
    Manager manager = _context.Managers.First(m => m.Email == Email && m.Password == Password);
    Company company = _context.Companies.First(c => c.Manager.Id == manager.Id);
    HttpContext.Session.SetInt32("LoggedIn", 1);
    HttpContext.Session.SetInt32("Id", (int)manager.Id);
    HttpContext.Session.SetString("Name", manager.Name);
    HttpContext.Session.SetInt32("CompanyId", (int)company.Id);
    HttpContext.Session.SetString("CompanyName", company.Name);
    HttpContext.Session.SetString("OrderType", company.OrderType);
    return RedirectToAction("Index");
}
```

Figure 11 Sessions in c#

- **Entity framework and Linq**

Entity framework (hereafter, EF) is the framework ORM (object-relational mapping) that Microsoft makes available as part of the .NET development (version 3.5 SP1 and later). Its purpose is to abstract the ties to a relational database, in such a way that the developer can relate to the database entity as to a set of objects and then to classes in addition to their properties.

```
var query = from summary in _context.DeliverySummaries
            join d in _context.Deliveries on summary.Delivery.Id equals d.Id
            where d.Company.Id == CompanyId && summary.StartTime.Year==year
            group new { summary, d } by  new { summary.StartTime.Month} into g
            select new
            {
                Month = g.Key,
                ActualFuel = g.Sum(p => (p.summary.StartFuelLevel -p. summary.EndFuelLevel)),
                OptimizedFuel= g.Sum(p=>p.d.OptimalFuelConsumption)
            };

var query1 = from summary in _context.DeliverySummaries
             where summary.Delivery.Company.Id == CompanyId && summary.StartTime.Year == year
             group  summary  by summary.StartTime.Month  into g
             select new
             {
                Month = g.Key,
                Score=g.Average(summary=>(summary.PerformanceScore+summary.SafetyScore+summary.ComplianceScore)/3)
            };
```

Figure 12 Entity frame work and LINQ

- **Bootstrap**

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. It provide quick prototypes to build an entire app with Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

- **JavaScript**

The core JavaScript language consists of some common programming features that allow you to do things like store useful values inside variables, run code in response to certain events occurring on a web page and much more…In the map page for instance, we send an AJAX request to the map controller to get the active vehicles and then store them in JSON arrays.

```
$.ajax({
    type: "POST",
    url: "/Map/GetVehicles",
    data: ajaxPostRequestJSON,
    error: () => { console.log("Failed to get update") },
    success: async function (result) {
        const vList = JSON.parse(result.result);
        for (veh in vList) {
            let v = vList[veh];
            if (displayedVehicles[v.Id]) {
                let flag = await updateVehicleInMap(v.Id, v.Latitude, v.Longtitude);
                if (flag) await updateVehicleDeliveries(v.Id, v.Deliveries);
            }
            else {
                let vDelList = [];
                if (!(v.Deliveries == null) && arrayLength(v.Deliveries) > 0) {
                    for (del in v.Deliveries) {
                        let d = v.Deliveries[del];
                        vDelList.push({
                            "deliveryID": `${d.deliveryID}`,
                            "startLatitude": `${d.SourceLatitude}`,
                            "startLongitude": `${d.SourceLongtitude}`,
                            "endLatitude": `${d.DestinationLatitude}`,
                            "endLongitude": `${d.DestinationLongtitude}`,
```

Figure 13 JavaScript and JQuery

What is even more exciting however is the functionality built on top of the core JavaScript language. So-called Application Programming Interfaces (APIs) provide you with extra superpowers to use in your JavaScript code. The Map API used made the integration of the maps possible and providing many essential routing information.

- **Maps API**

The map packages are essential for this project in different ways. It provides the map graphics which is an essential tool for trip planning and vehicle tracking, in addition to the map API routing functionalities which provides valuable routing information such as geocoding and shortest routes.

The original plan was to use Google Map API but unfortunately the later was not free anymore, so we opted for other free Map APIs. After extensive search we found the most suitable APIs that includes all the functionalities required for the project: HereMap API and OpenLayers API.

1. HereMap API

The HERE JavaScript API offers easy-to-use libraries of classes and methods with great processing speed and faster loading times for both desktop and mobile. Here are some of the features offered by this API:

JAVASCRIPT API	FEATURES
Map & Satellite Tiles	The service underlying the HERE Maps API for JavaScript is the HERE Map Tile API, which provides map images in a choice of styles, including satellite, terrain and traffic. Also try our Map Tile API
Venue Maps	Call venues using PNG/JS tiles. Includes floor levels or full JSON models with polygons/geometry, elevators/escalators, entry/exit doors and store/POI information. Also try our Venue Maps API
Car & Pedestrian Routing	Turn-by-turn drive and walk directions with instructions in many languages. Users can set preferences such as shortest or fastest route, restrictions, tolls or motorways and more. Also try our Routing API
Geocoding	Resolve addresses to geo-coordinates and vice-versa using the service module (mapsjs-service.js) for easy integration into a mapping app. Also try our Geocoder API
Places	Search, explore and see points of interest. Results include name, address, category and contact info. Also try our Places API
Real Time & Historical Traffic Tiles	Display traffic info on the map, such as traffic incidents, real time and historic traffic events, for cities around the world. Also try our Traffic API
Fleet Telematics Custom Locations	Store, manage and retrieve custom POIs and polygons. Also try our Fleet Telematics API.
Fleet Telematics Advanced Data Sets	Get additional HERE Map Content, including height and slope values, curvature, speed limits and traffic lights. Also try our Fleet Telematics API.

Figure 14 HereMap API features

In the diagram below we state the different map functions that we implemented using the HereMap API. All of these functions are defined and implemented in the JavaScript library map_init.js.

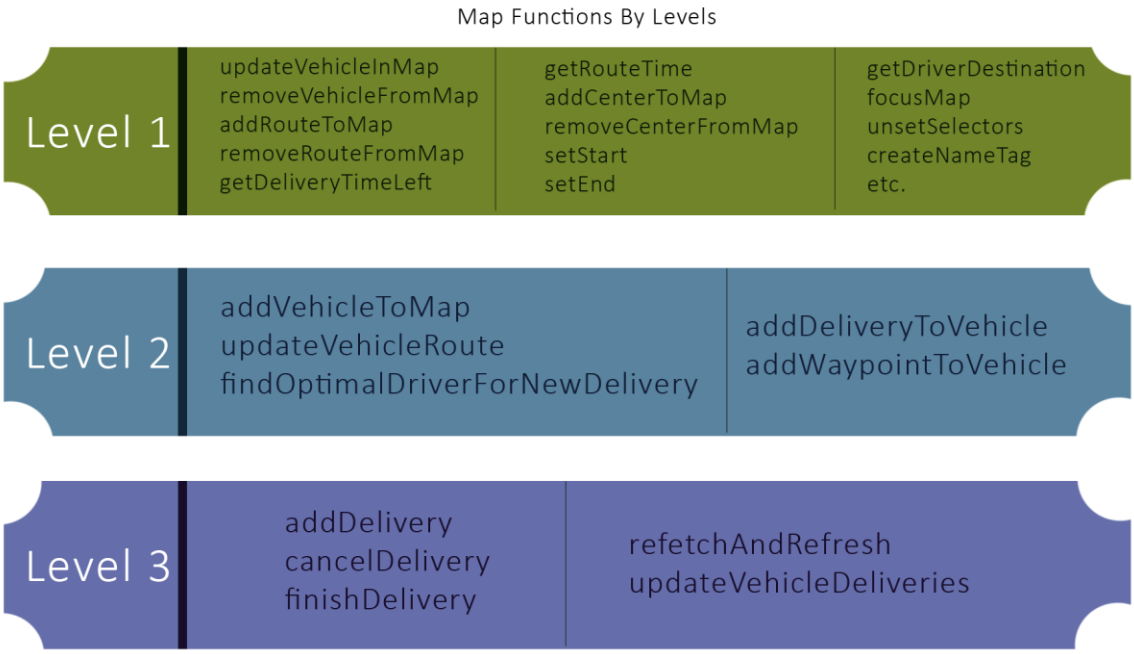

Figure 15 Map levels diagram

In the figure below we show how the request url is formed using JavaScript before being sent to the Here Map API. After the HttpClient returns the Json object, the result is parsed and the traffic time is obtained.

```
function getRouteTime(route) {
    return new Promise(resolve => {
        console.log("---------- IN GET ROUTE TIME ----------");
        console.log(route);
        let requestURL = `https://route.api.here.com/routing/7.2/calculateroute.json?app_id=ORWs1MBbnXAyzlgdPGpw&app_code=ftEQwIdO

        for (waypoint in route) {
            requestURL += `${waypoint}=geo!${route[waypoint].latitude},${route[waypoint].longitude}&`;
        }

        requestURL += `mode=fastest;car;traffic:enabled`;

        const client = new HttpClient();
        client.get(requestURL, function (response) {
            const responseJSON = JSON.parse(response);
            const trafficTime = responseJSON.response.route[0].summary.trafficTime;

            console.log("---------- OUT GET ROUTE TIME ----------");
            resolve(trafficTime);
        });
```

Figure 16 HereMap API request

2. OpenLayers

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles, vector data and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds. It is completely free, Open Source JavaScript package.

The OpenLayers API was mainly used for its geocoding services, where the address of a location is obtained from the coordinates of a location and vice versa. Here is an example where OpenLayers API is used:

```
document.querySelector("#startLatitudeOptRouteInput").value = coord.lat;
document.querySelector("#startLongitudeOptRouteInput").value = coord.lng;

let url1 = `https://nominatim.openstreetmap.org/reverse?format=json&lon=${coord.lng}&lat=${coord.lat}`;
var jsonPromise1 = $.getJSON(url1);

jsonPromise1.done(function (data) {

    if (data.address.hasOwnProperty('city'))
        startCity = data.address.city;
    else
        startCity = data.address.state;
    alert(startCity + " \n " + coord.lat + "," + coord.lng);
});
jsonPromise1.fail(function (reason) {
    alert("Error");
});
```

Figure 17 OpenLayers API request

❖ **JavaScript Charts Libraries**

The data visualization for the reports was made possible using the JavaScript libraries for charts graphics. The libraries with the most suitable and relevant features were used in the display of the statistics. The libraries used are Chart.js and AmCharts.

In the figure below C# arrays are transformed to Json arrays to be displayed by the JavaScript charts.

```
var YPerformance = Newtonsoft.Json.JsonConvert.SerializeObject(Performance);
var YSafety = Newtonsoft.Json.JsonConvert.SerializeObject(Safety);
var YCompliance = Newtonsoft.Json.JsonConvert.SerializeObject(Compliance);
var XDays = Newtonsoft.Json.JsonConvert.SerializeObject(Days);
var XMonths = Newtonsoft.Json.JsonConvert.SerializeObject(Months);
var XYears = Newtonsoft.Json.JsonConvert.SerializeObject(Years);
```

Figure 18 Arrays from C# to Json

```
const ctx = document.getElementById('chartCanvas').getContext('2d');
const data = {
  // Labels should be Date objects
  labels:dates,
  datasets: [{
    fill: false,
    label: 'Perormane Score',
    data:@YPerformance,
    borderColor: 'red',
    backgroundColor: 'red'
  },
  {
    fill: false,
    label: 'Compliance Score',
    data: @YCompliance,
    borderColor: 'blue',
    backgroundColor: 'blue'

  },
```

Figure 19 Data given to the charts

4.3.2 Fleet API

The main job of the fleet API is to respond to the various requests from the vehicle and the mobile application. The API is a MVC web API developed using ASP.Net Core 2.2, it receives requests with data in Json format and replies back with result in Json format also. The different API actions along with the input and output of each one are stated in **Appendix A**. The software used for testing the API requests is Postman as shown below.

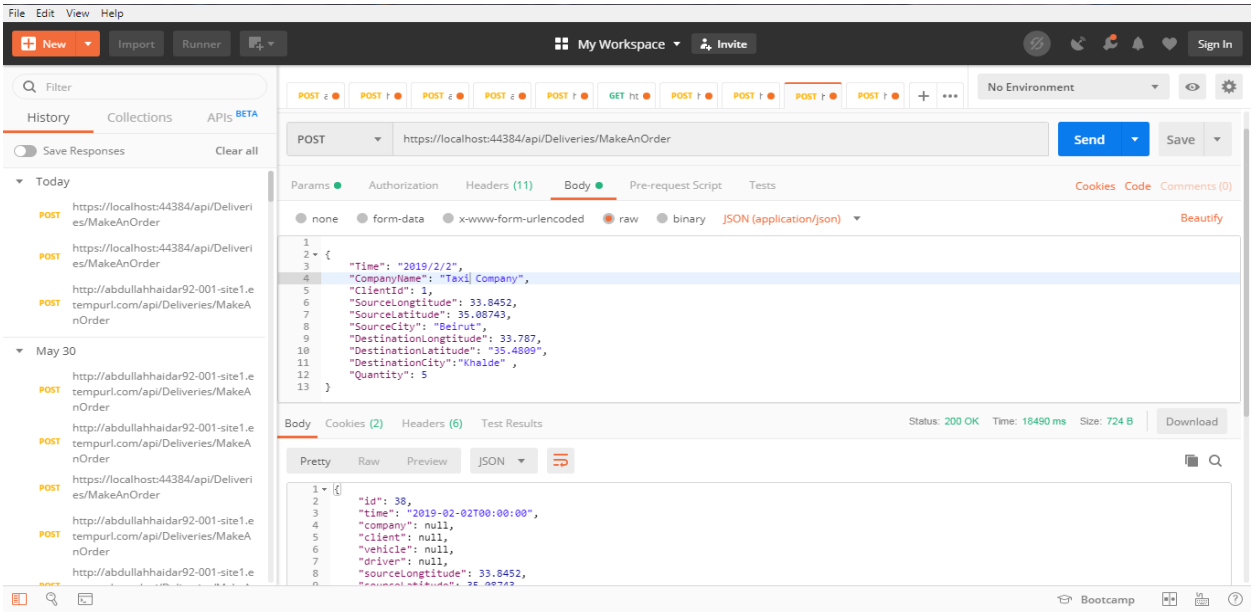

Figure 20 Postman

4.3.3 Hosting

The Fleet API and Fleet website were hosted on SmarterAsp.com servers. SmarterAsp offers a two month free trial version for ASP.NET or PHP applications:

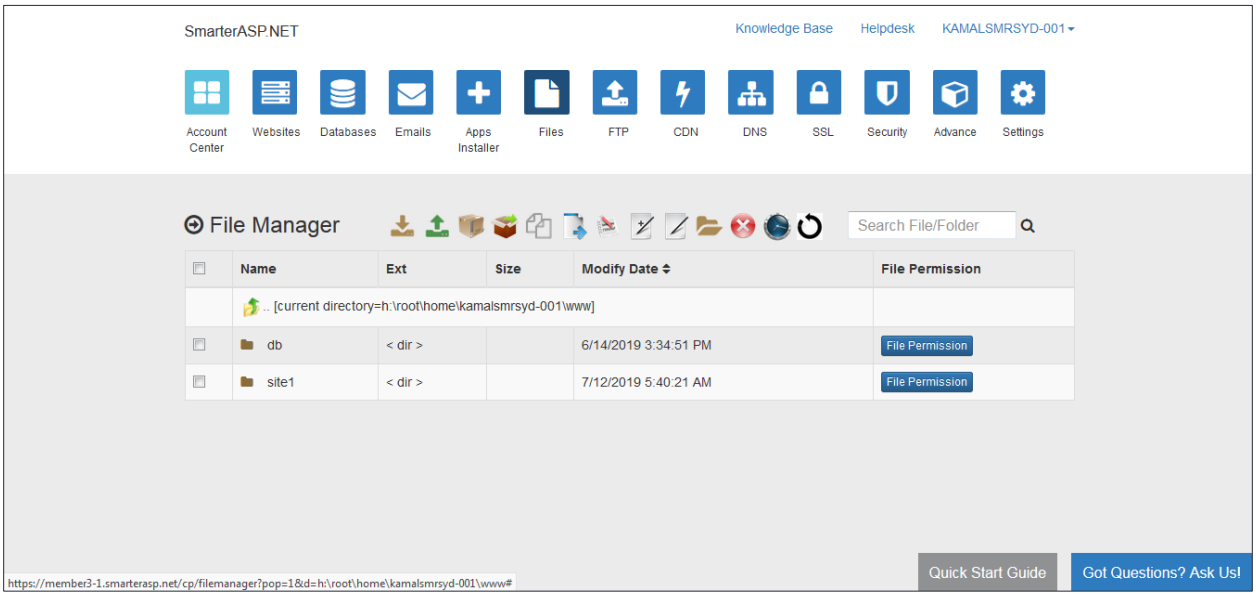

Figure 21 SmarterAsp.com control panel

4.3.4 Mobile Application

The mobile application is designed for android operating systems using Java. The tool used is Android Studio.

4.3.5 Database

The database was implemented in Microsoft SQL 2014, and is hosted on a remote server with the API through SmarterAsp.com. The creation of tables and any changes to the database is done through entity framework migrations, and using package manager console tool that is included in Visual Studio. The design of the database is identical to the class diagram in figure 5.

CHAPTER 5: RESULTS AND DISCUSSION

The following snapshots show the final results of the Fleet Management Website and the Fleet Mobile Application.

5.1 Fleet Management Website

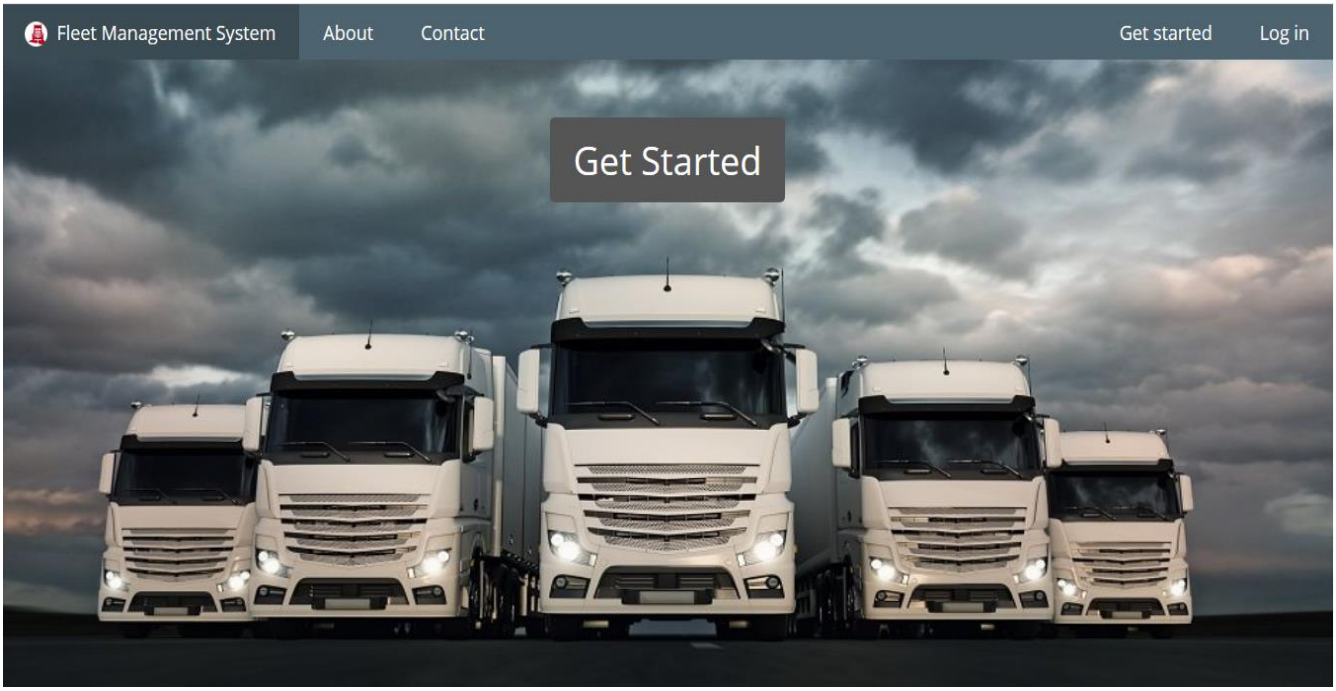

Figure 22 Home page

From the home page the user can choose between two options: he can login to the application if already has an account, or go ahead and get started by creating a user account and then creating a company profile.

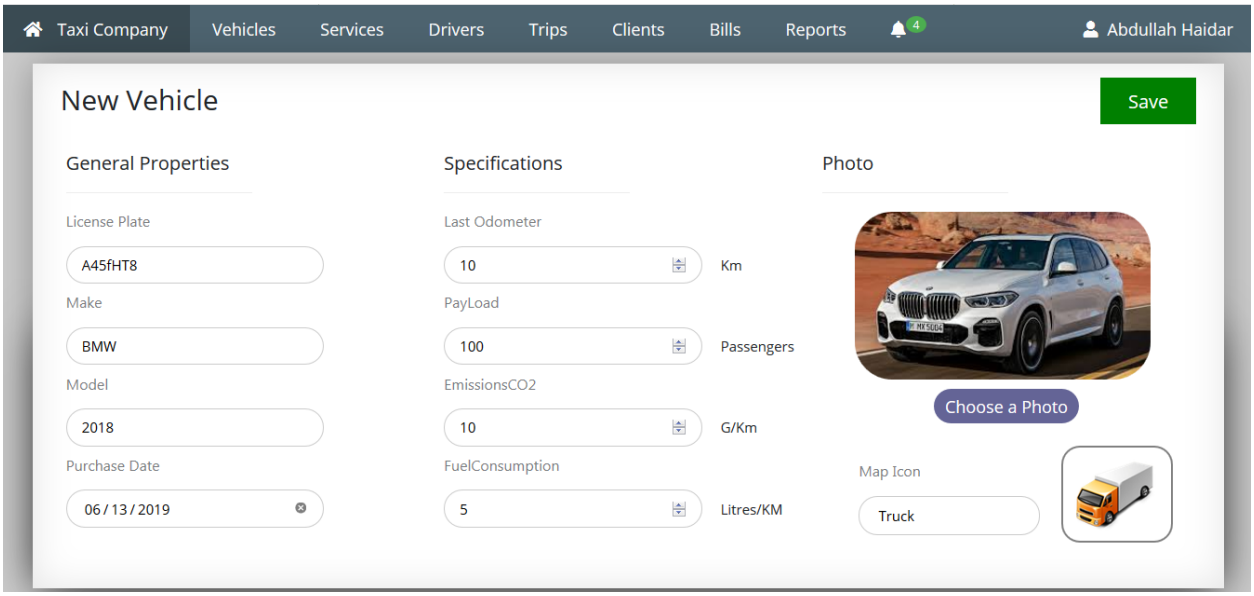

Figure 23 Add vehicle page

The manager can add a vehicle by specifying the general properties and specifications of the vehicle.

The general properties include the License Plate, Make, Model and Purchase Date of the vehicle.

The Specifications include the odometer level (at the time the vehicle is added), the payload, and the rate of CO2 emission and rate of fuel consumption.

The manager can also add an image to the vehicle and a map icon to represent the vehicle on the maps, the map icon can be a car, truck, bus or a motorcycle.

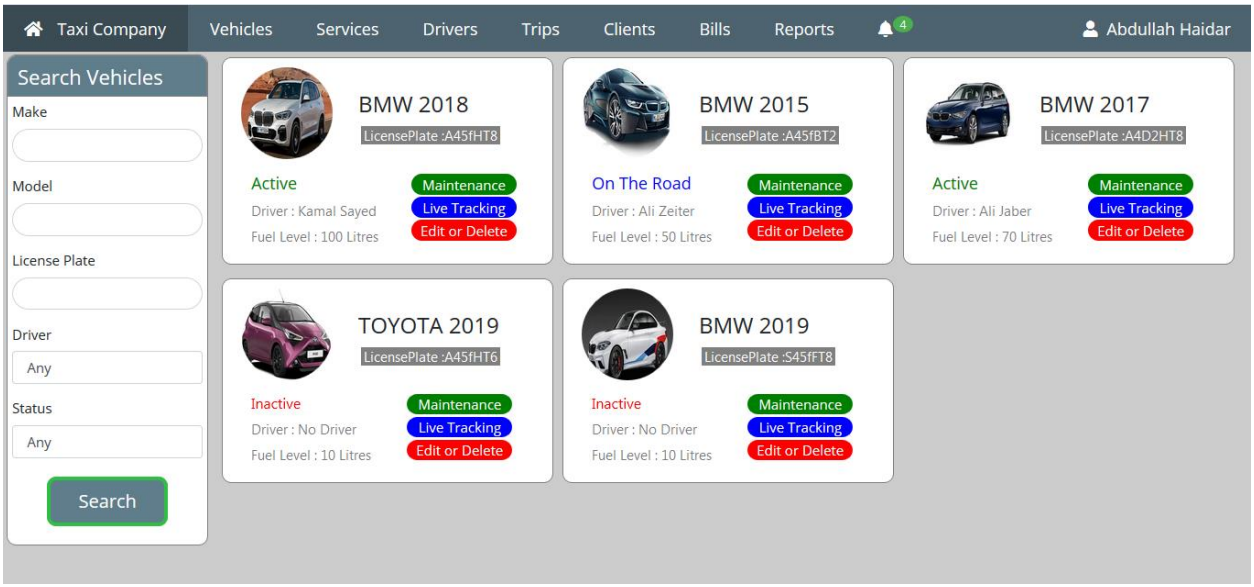

Figure 24 Manage vehicles

In the home page the manager can view all the vehicles of his company. For every vehicle the information shown are the status of the vehicle (Active, On The Road or Inactive), the current driver, the fuel level and the license plate.

For every vehicle he has three options:

- Maintenance: where he can check the car properties, scheduled activities, costs, fuel logs and delivery history.
- Live Tracking: where he can track the vehicle live on the map with all the continuous updates about the cars specifications received from the vehicle log.
- Edit or Delete: where he can choose to edit the properties of the vehicle or delete the vehicle with all its related data.

The manager can also filter the vehicles according to License plate, make, model, driver or status.

He can also view all the notifications which are either about new orders or reminders for vehicle checkups.

The manager can manage the vehicle by clicking on maintenance, which opens the following pages:

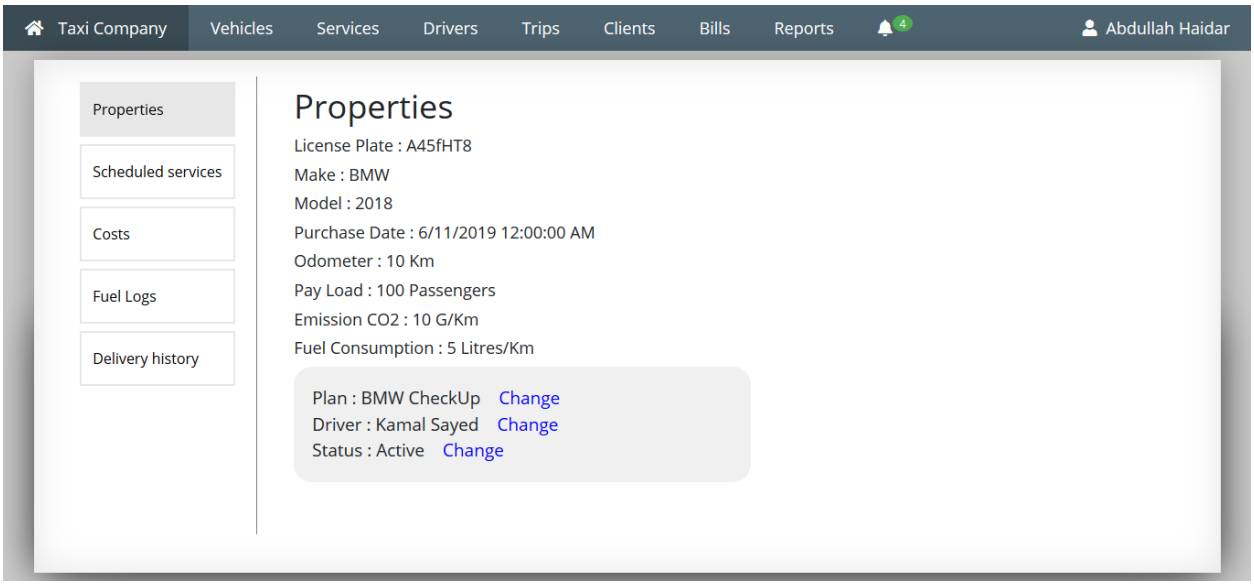

Figure 25 Vehicle properties

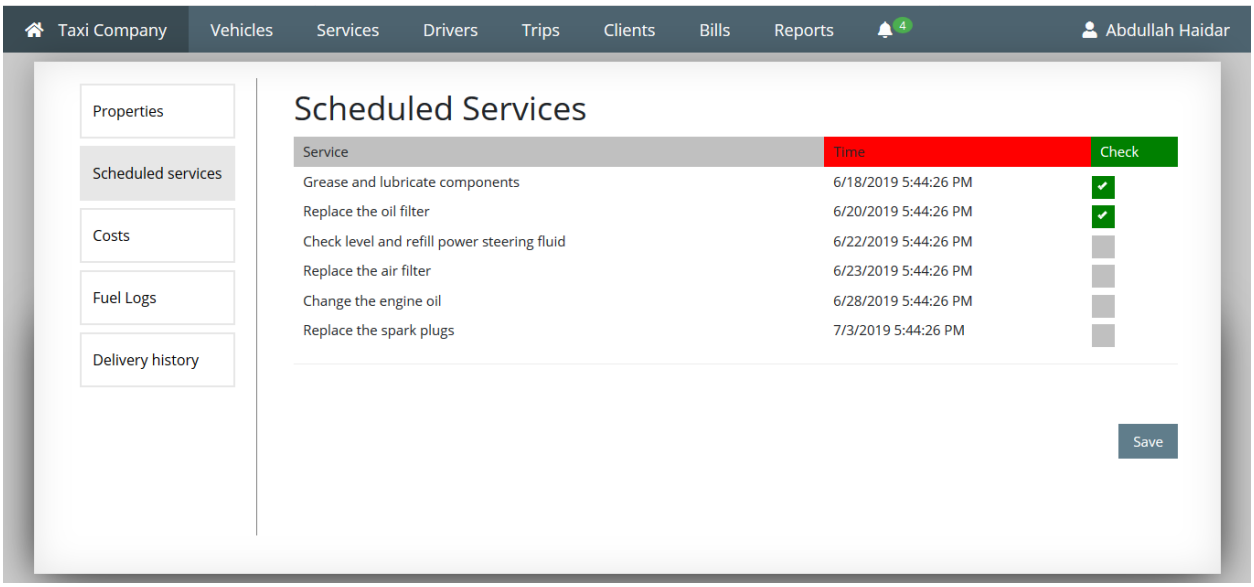

Figure 26 Vehicle's scheduled activities

In the scheduled services section the manager can review all the services scheduled for the vehicle with the appointed time.

The manger can choose the services that are already done by checking the box and clicking save.

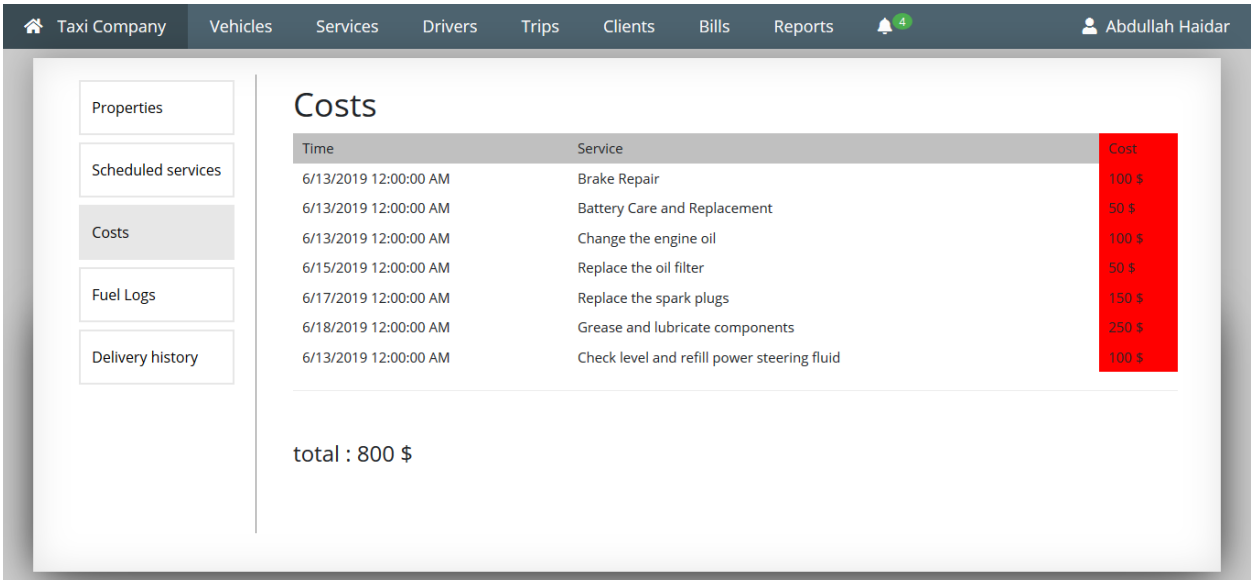

Figure 27 Vehicle's costs

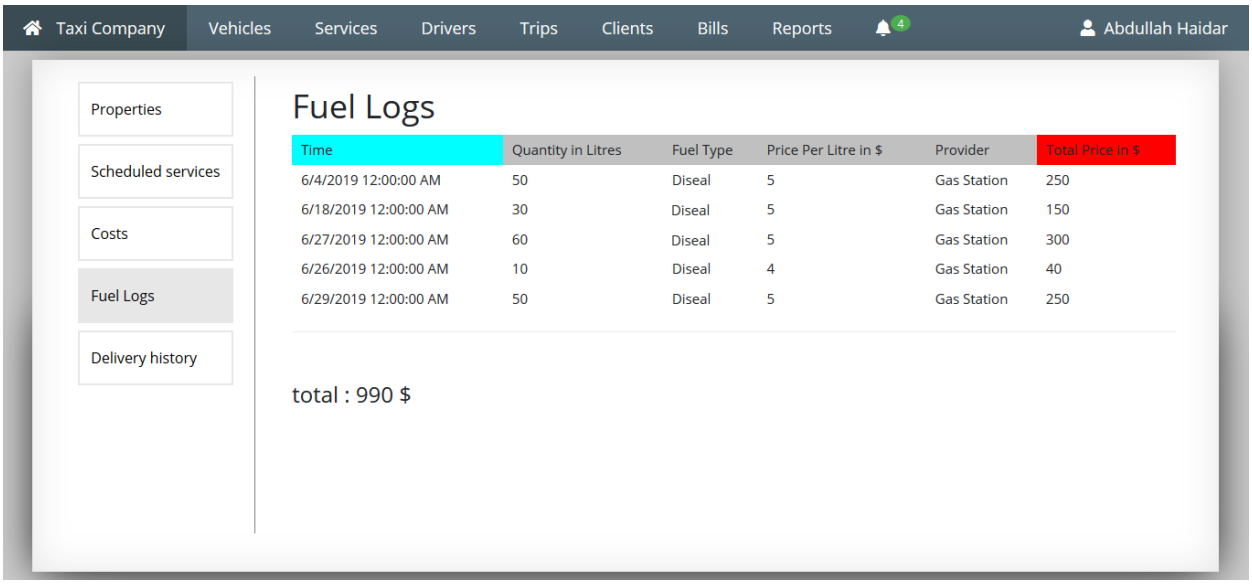

Figure 28 Vehicle's fuel logs

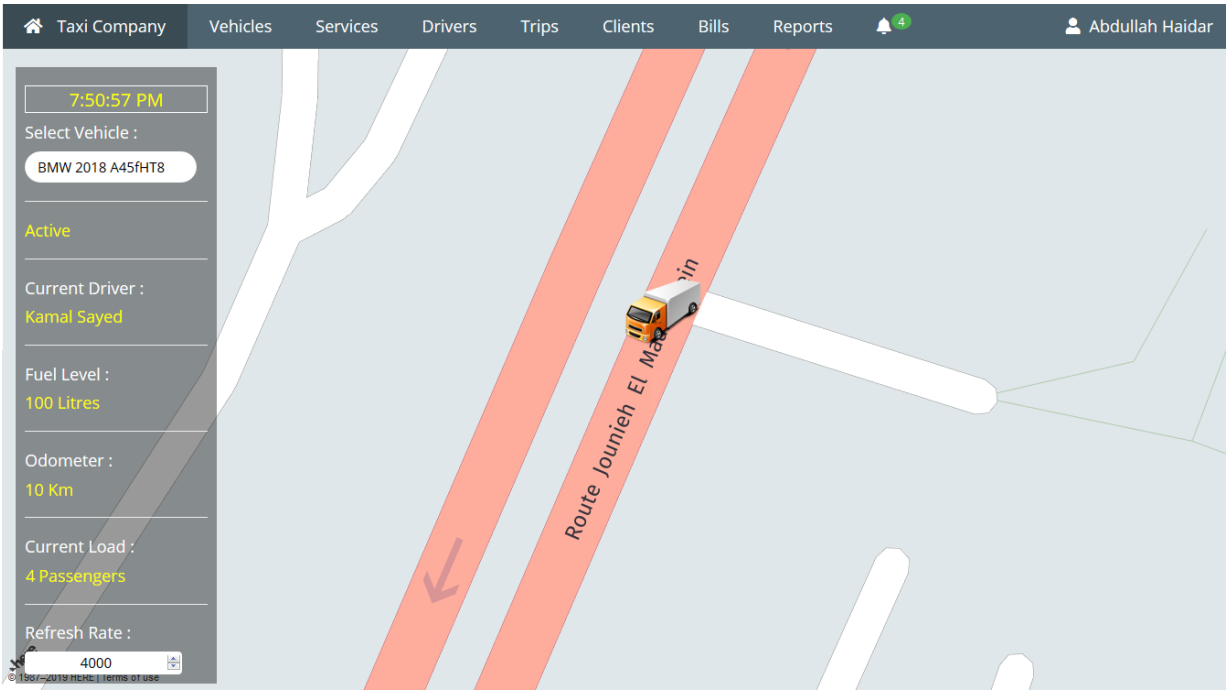

Figure 29 Vehicles' live tracking

All of the company's vehicles can be tracked on map with the vehicle location and vehicle information updated continuously.

The manager can select the vehicle he wants to track and then monitor the vehicle movement during its trip.

The information displayed are the status of the vehicle (active, on the road, or inactive), the current driver, the current fuel level, the current odometer and the current load.

The manager can change the rate of refresh for the update of the vehicle location and information.

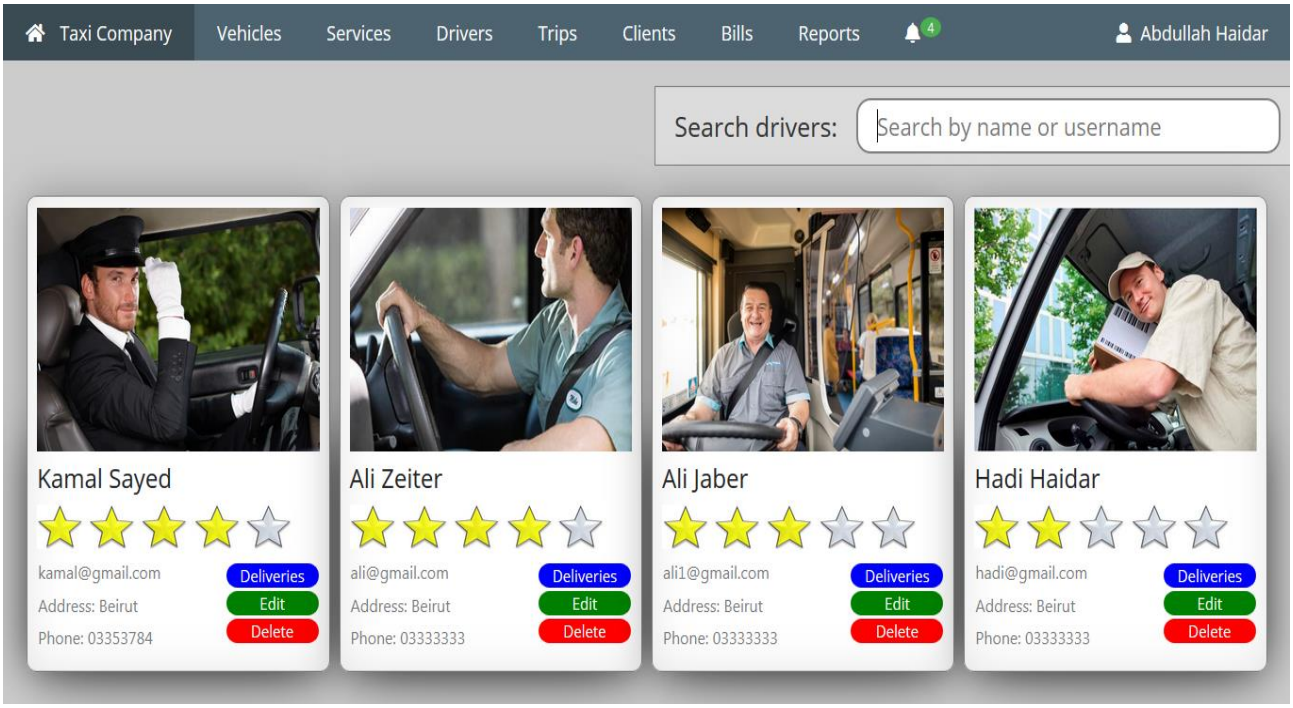

Figure 30 Drivers

The manager can view all the drivers of the company with their scores, deliveries and details. He may choose to edit their profile or remove them completely. Removing a driver removes all the other information linked to him like deliveries and delivery summaries…

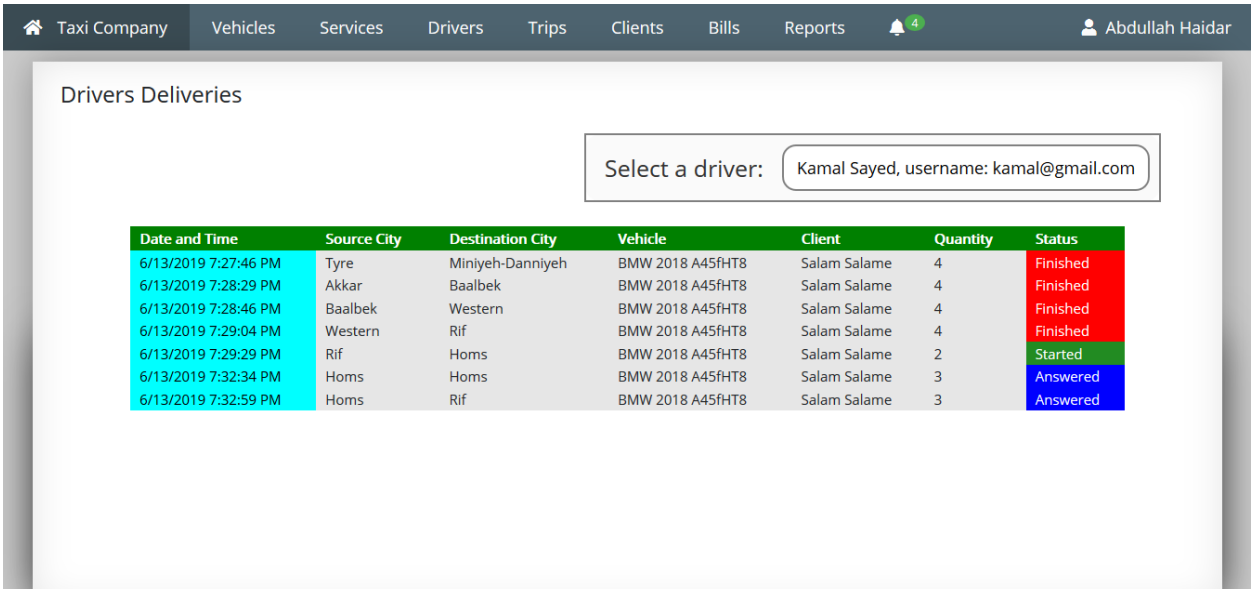

Figure 31 Drivers deliveries

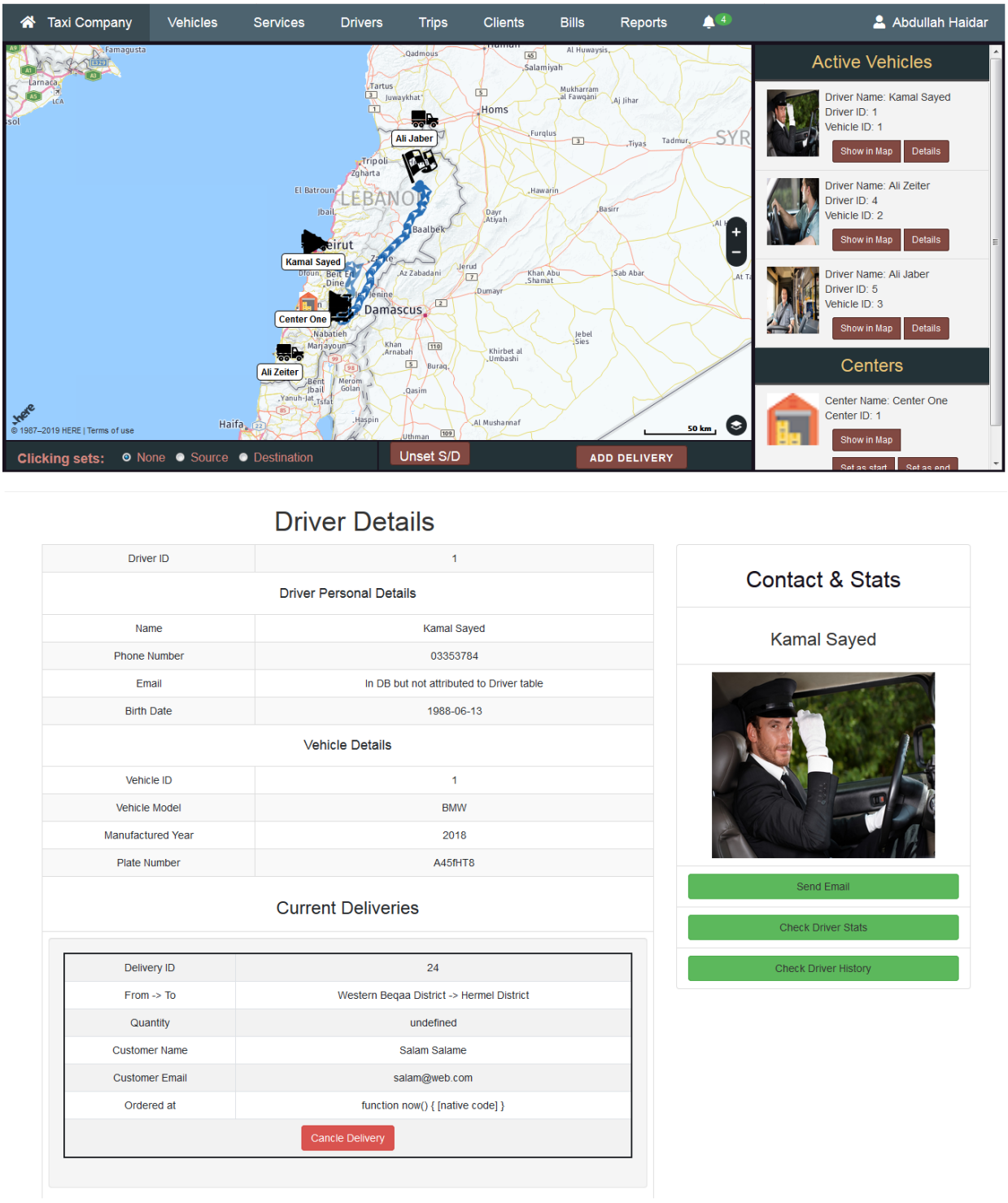

Figure 32 Plan trips

The manger can open the map where he can see all the active vehicles with their deliveries. The page also includes the centers belonging to the company with other map locations that the manager has already created. If the manager opens this page by clicking on an order request (from the notification

section) the details of the orders will be automatically added to the map, including the source and destination locations, the client and quantity of the order.

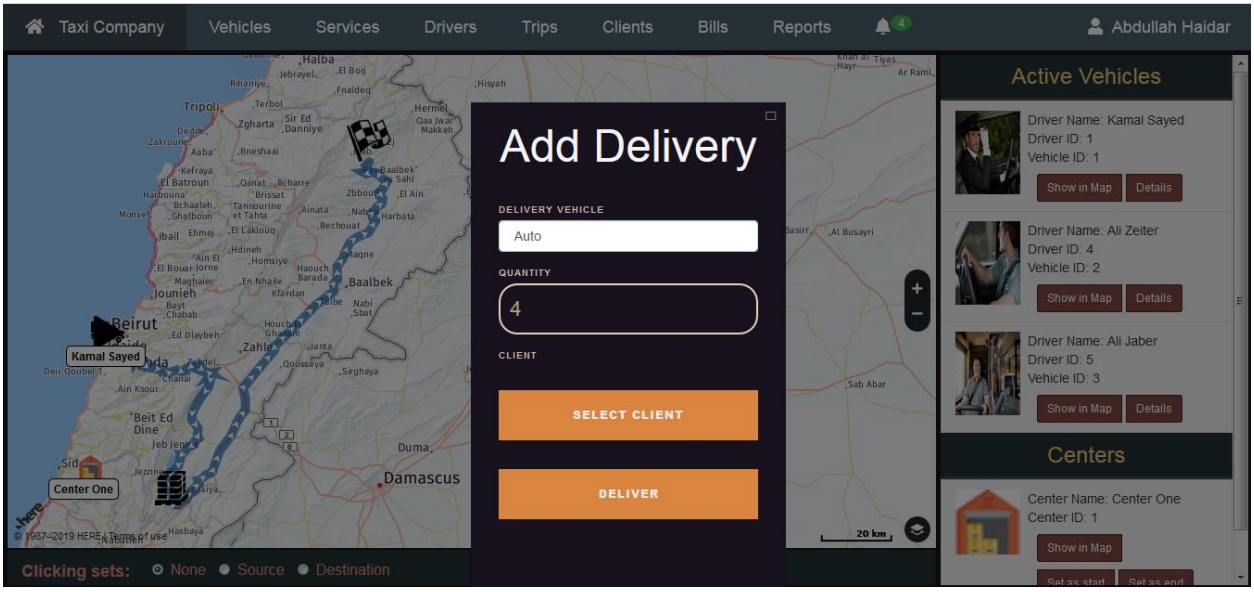

Figure 33 Add delivery in map page

If the manager wants to add a delivery he should specify the source and destination for the delivery (choose the source or destination radio button and then click on the location on the map).

He then may select to a vehicle for delivery or simply allow the system to choose the optimal vehicle for the delivery.

Finally he enters the quantity and selects the client for the delivery.

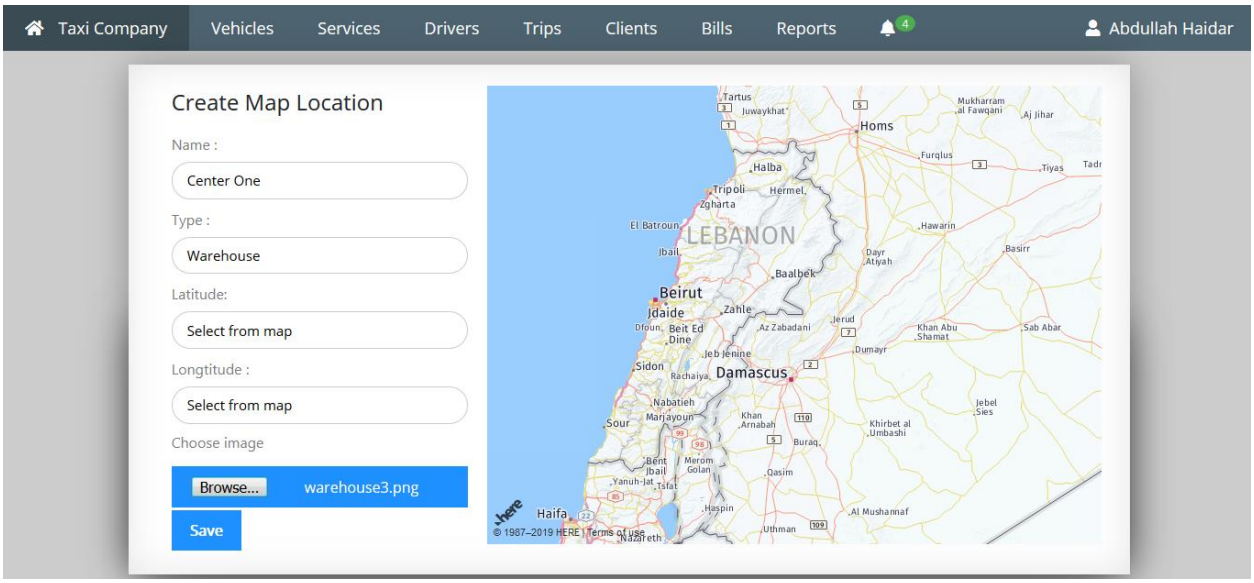

Figure 34 Add map location

The manager can create a map location for the important places belonging to the company. These places can be a warehouse, drop-off centers, etc. He specifies a name, type, image and the location from the map.

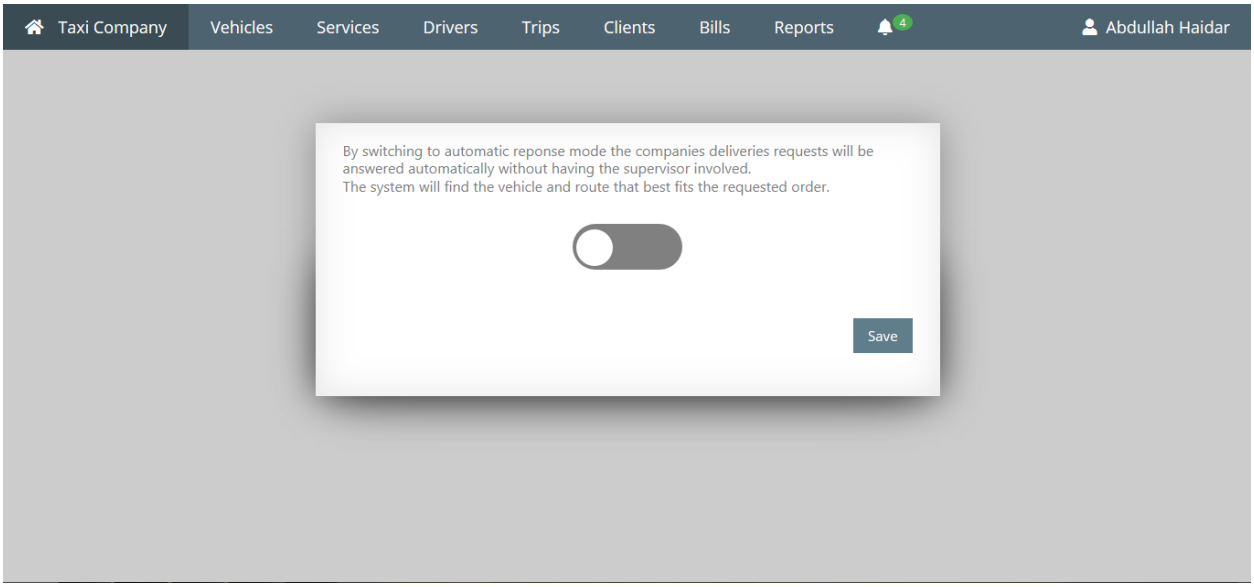

Figure 35 Automatic response page

The manager can change the automatic response option for his company. When he switches the automatic response to on all the order requests will be answered directly by the API.

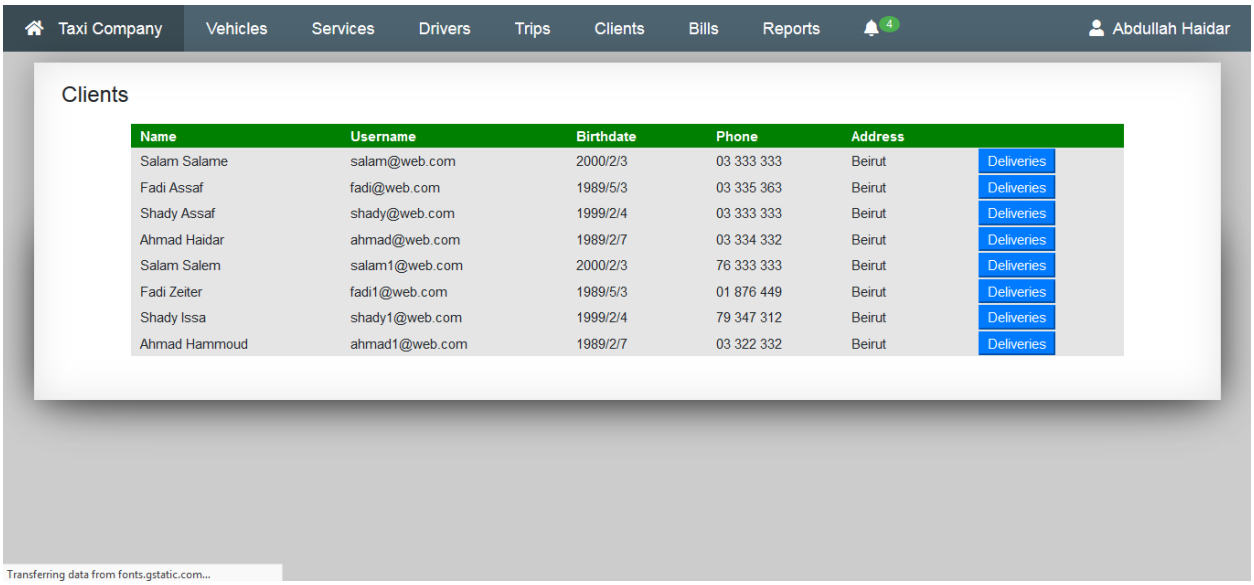

Figure 36 Clients page

The manager can see all the clients of his company with their details. Moreover he can all of their deliveries with the delivery status and details. The clients deliveries page is similar to driver deliveries page.

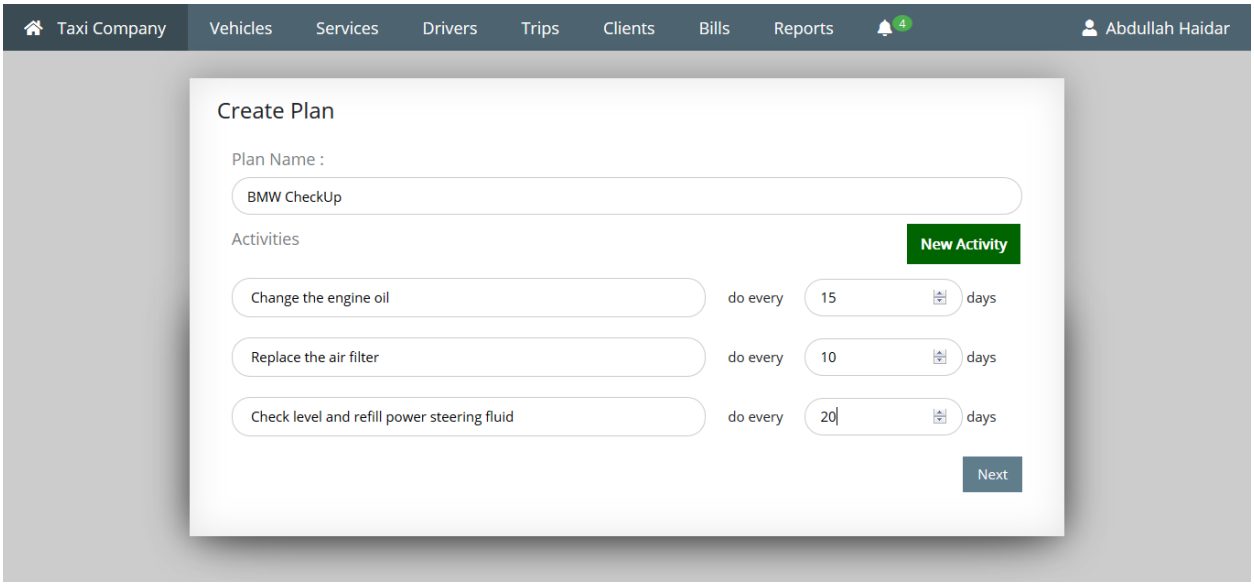

Figure 37 Create maintenance plan

The manger can create a maintenance plan by choosing a title and specifying the activities associated with the plan. For each activity he must provide the type of the activity and the period for which the activity will be repeated after.

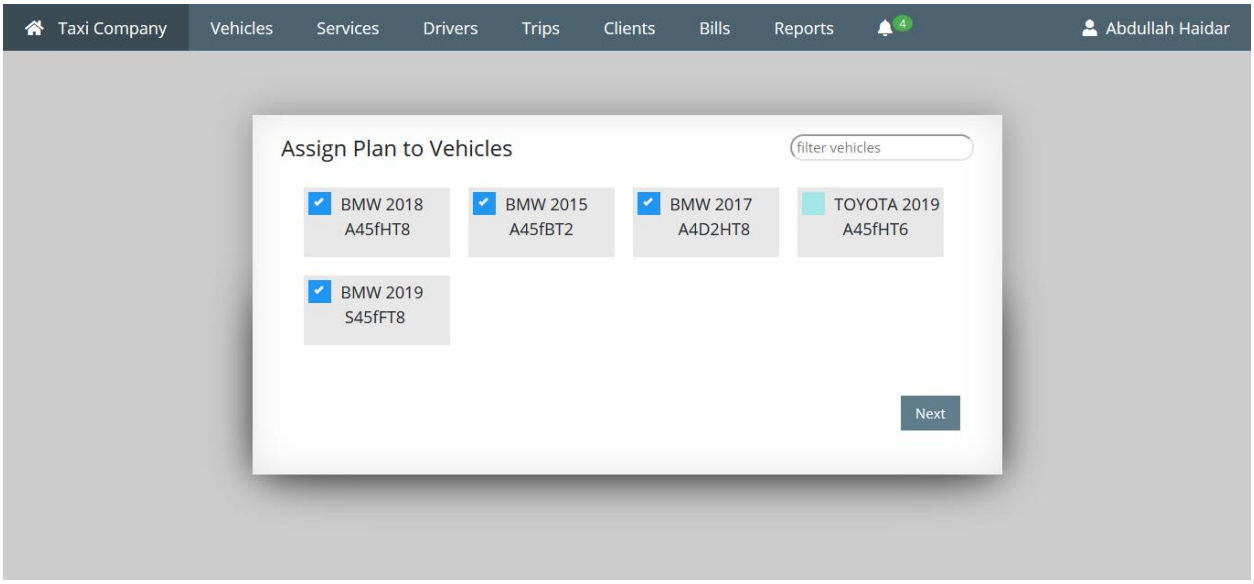

Figure 38 Add plan to vehicles

After creating a maintenance plan the manager should specify all the vehicles to which this plan will be applied. He can change these vehicles later on.

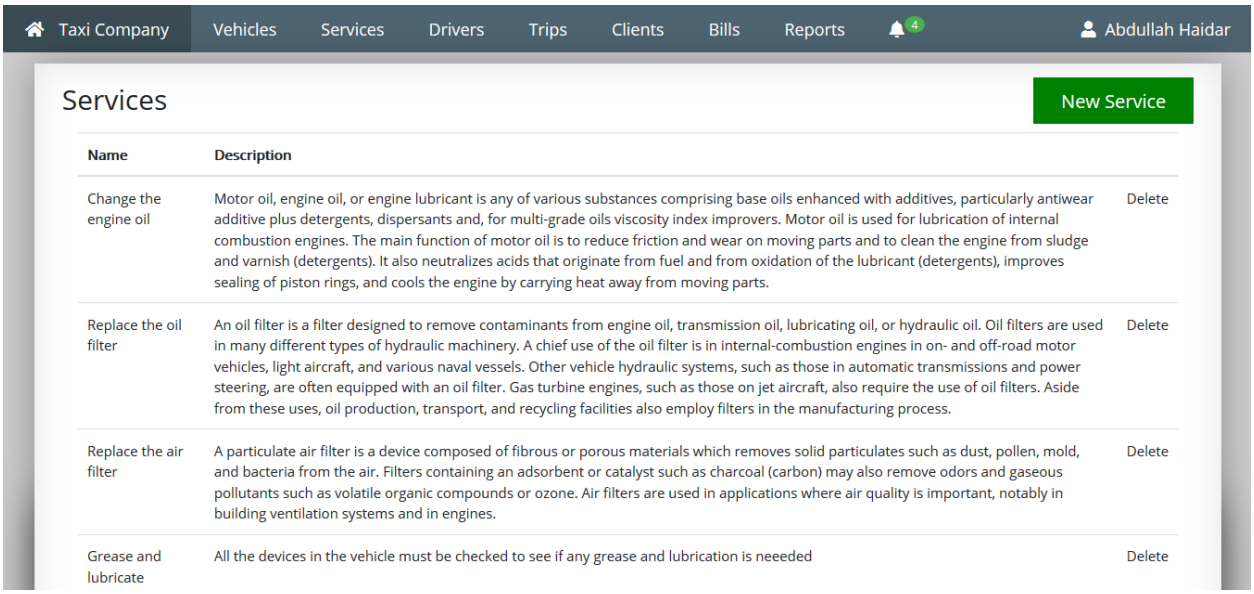

Figure 39 Services

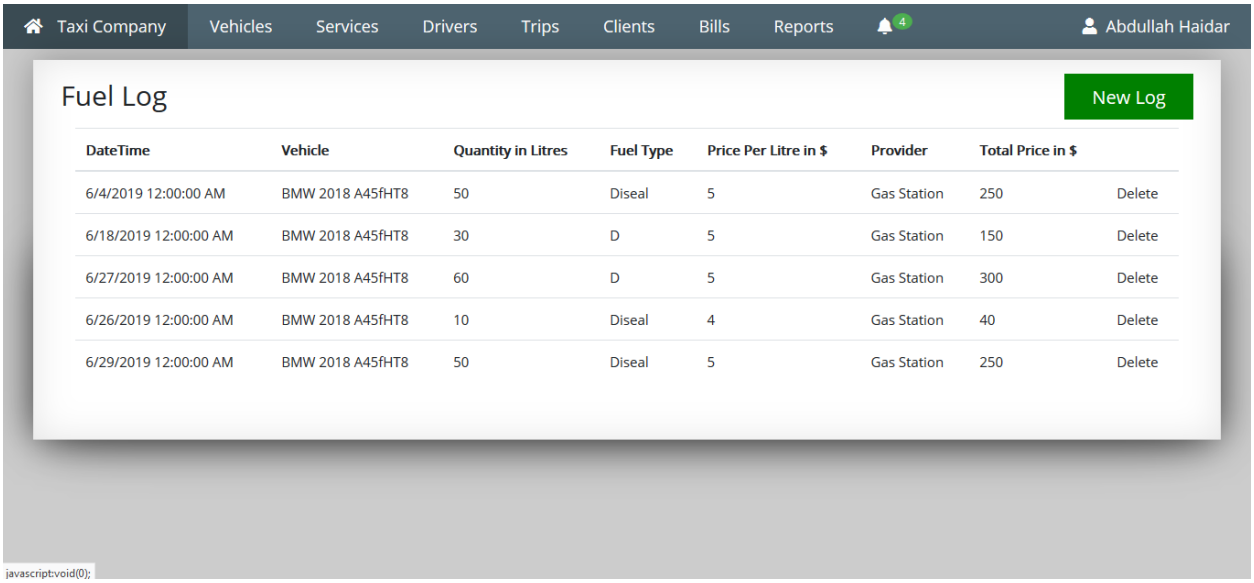

Figure 40 Fuel log

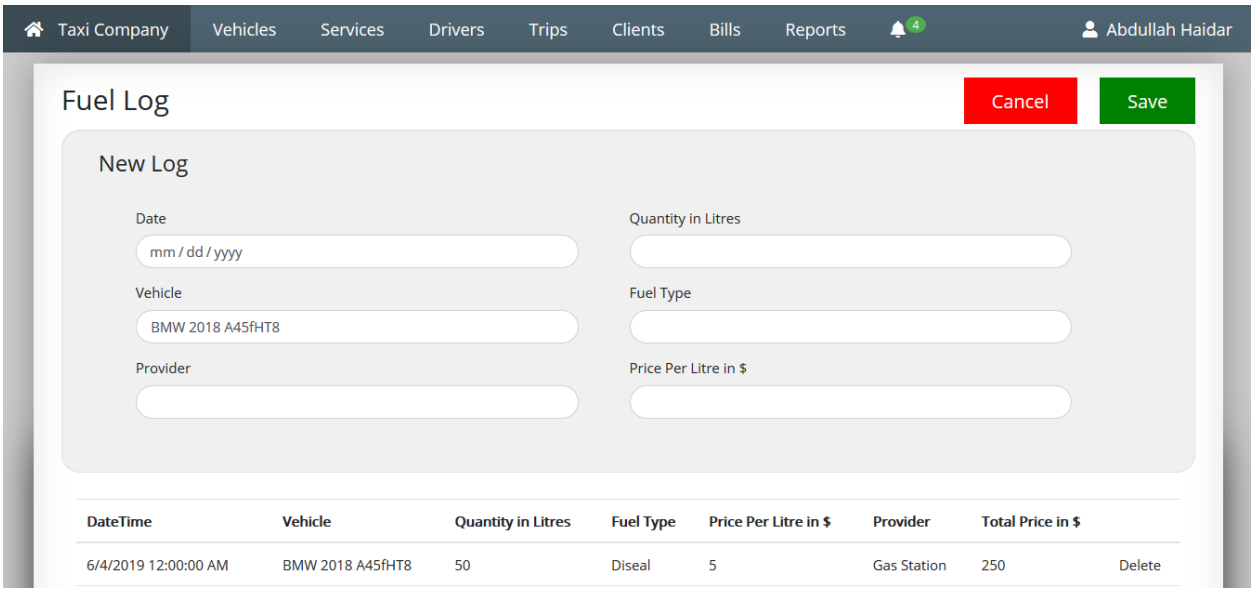

Figure 41 New fuel log

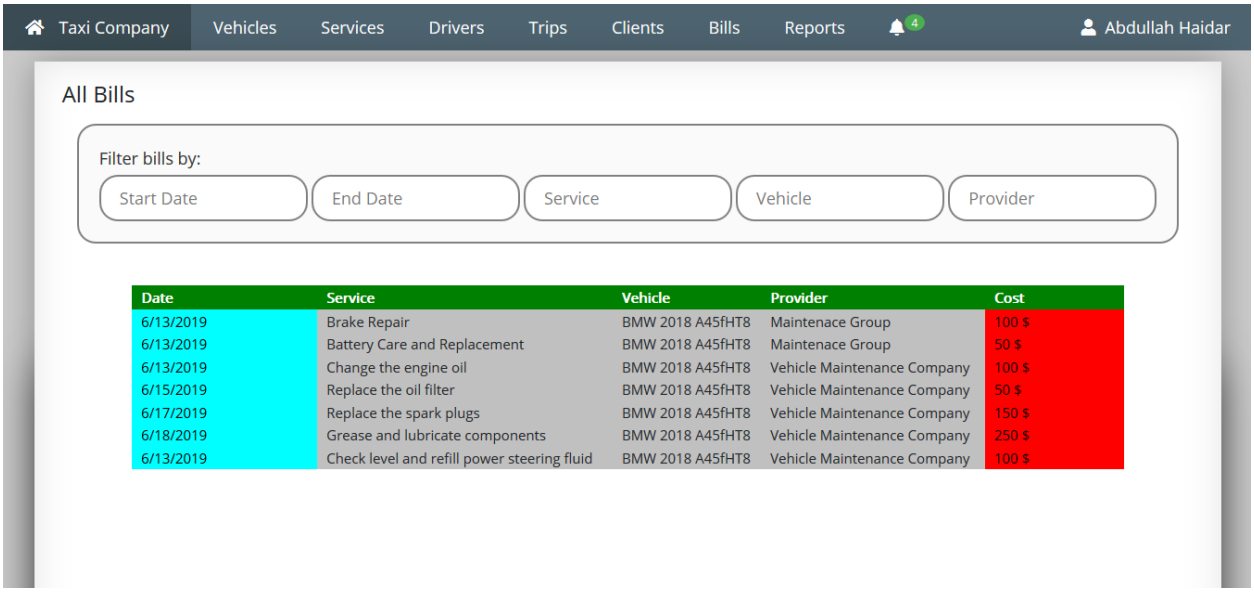

Figure 42 Bills

The bills page contain all the bills registered by the manager. The manager can filter these results by date (choose start date and end date), by service type, by vehicle or by provider.

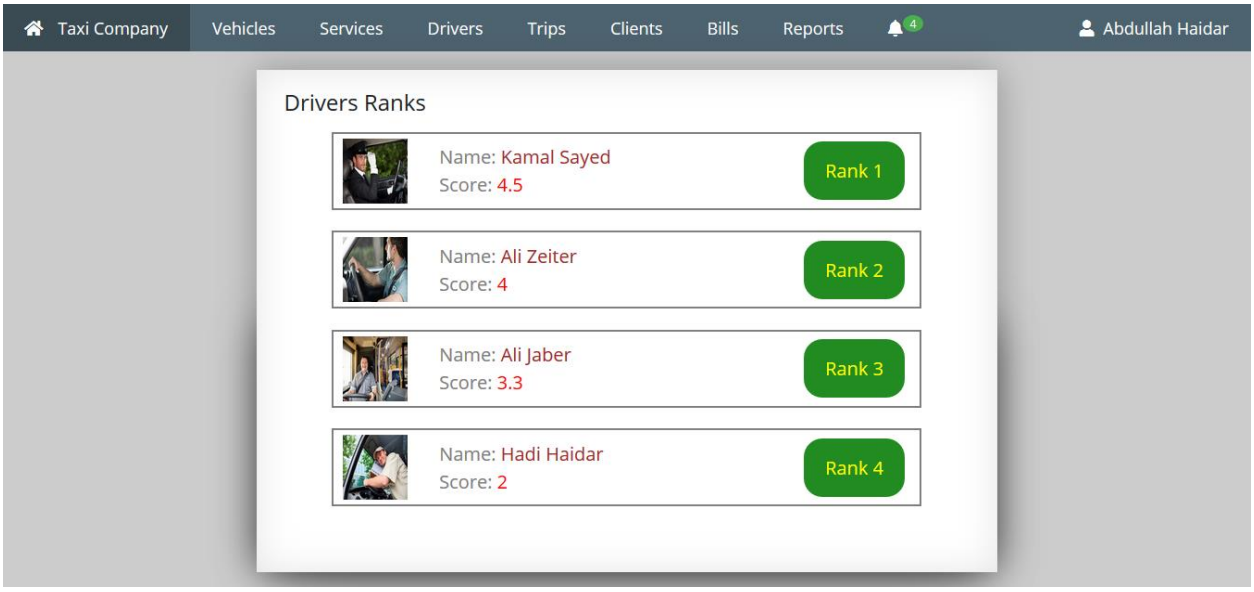

Figure 43 Drivers ranks

The drivers ranks page show the current of the drivers sorted from the top ranker to the lower one. The scores of each driver are calculated from three scores: the performance score, the compliance score and the safety score.

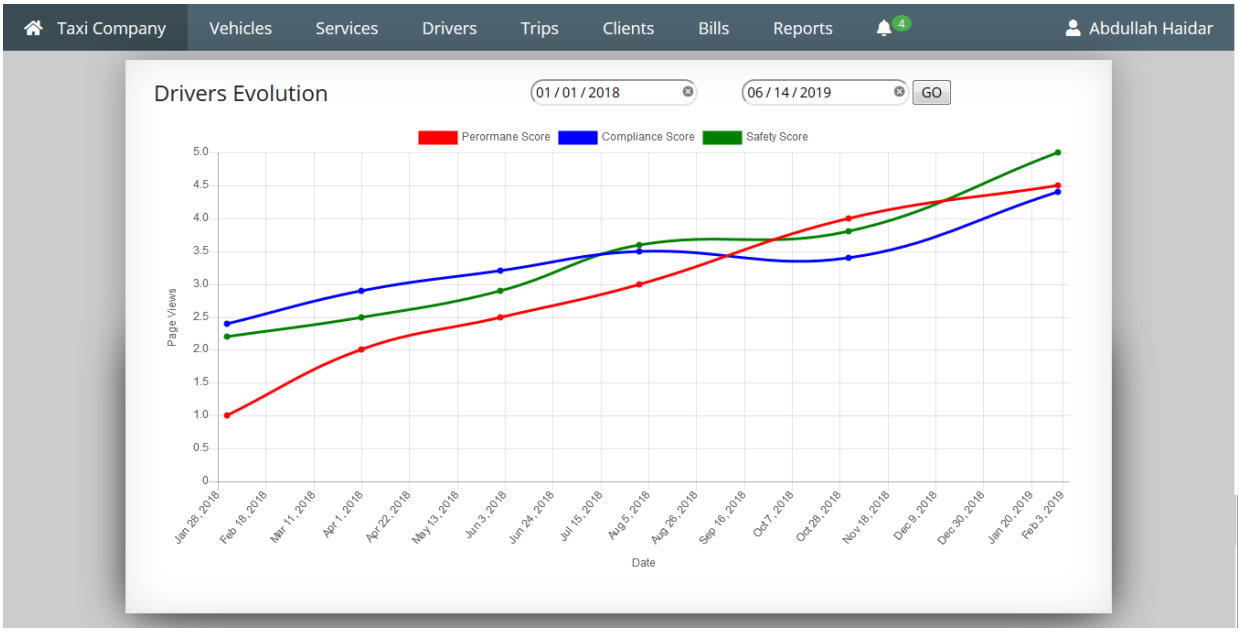

Figure 44 Drivers evolution chart

The drivers' evolution chart allows the manager to get an insight of the performance of the drivers and compare its evolution with time. The chart shows three line plots: the performance score chart, the compliance score chart and the safety score chart. The manager can choose to show only the statistics of a certain period by entering a start date and an end date.

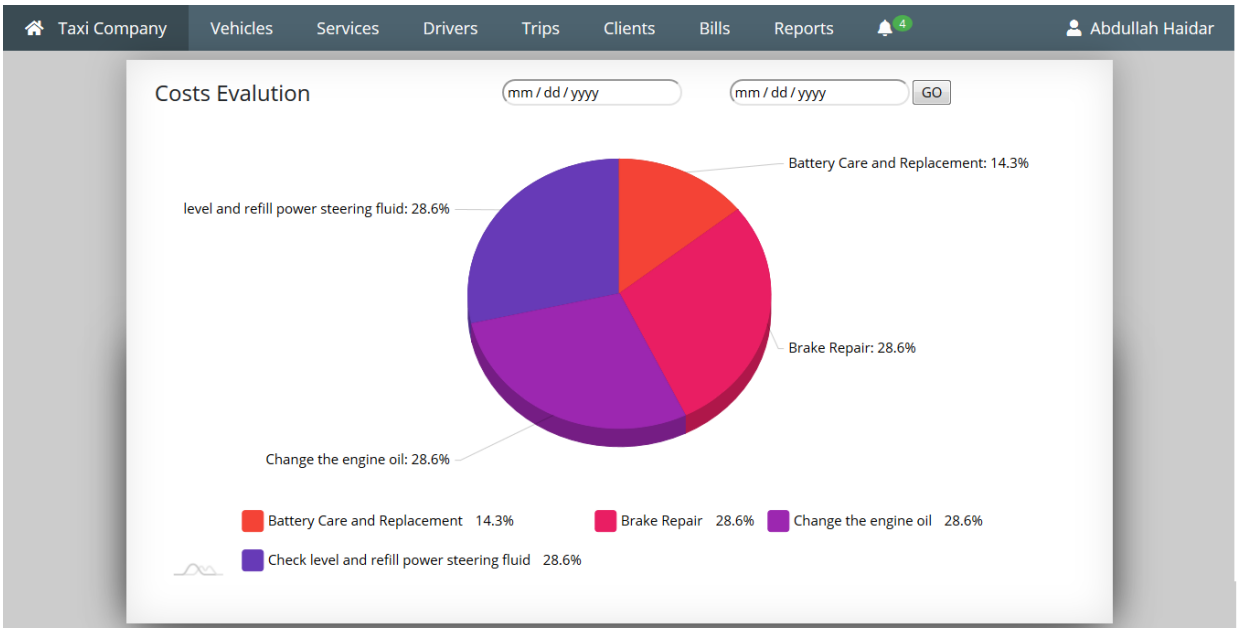

Figure 45 Costs evaluation chart

The costs evaluation chart shows the distribution of the costs of the company according to the services given to the vehicles. Just like with the drivers evolution chart the manager can choose to narrow the statistics by date.

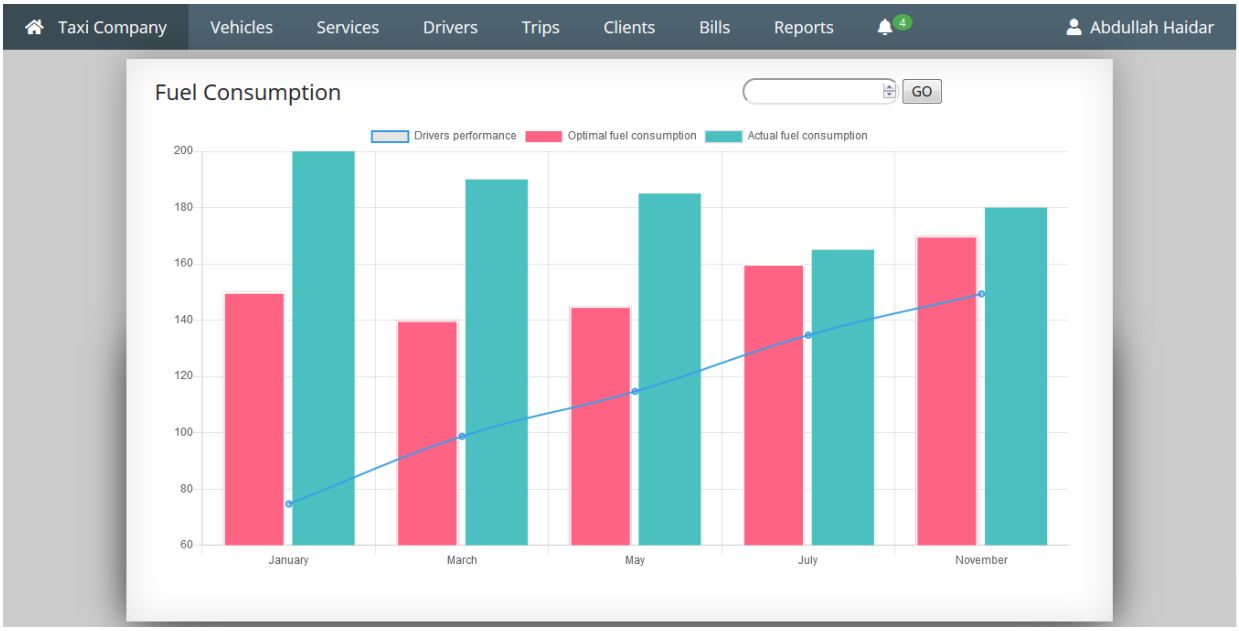

Figure 46 Fuel consumption chart

The fuel consumption chart allows the manager to compare the actual fuel consumption rates with the optimal fuel consumption rate that was expected for the trips. It allows him to point the places where fuel is wasted or spent more than it should be. The chart also allows him to compare the waste of fuel with the drivers' performance evaluation so he could take better decisions concerning his drivers.

5.2 Mobile application

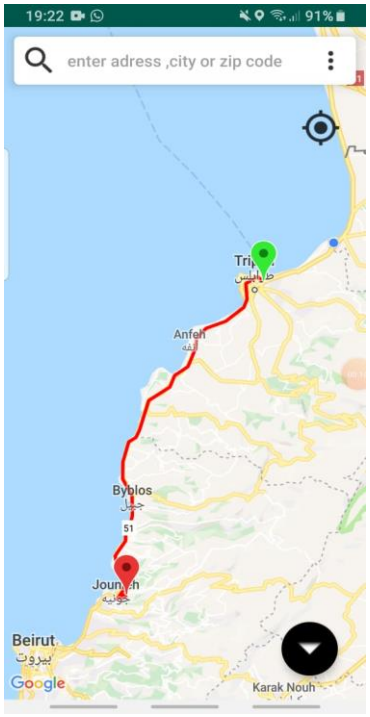

Figure 47 Drivers map

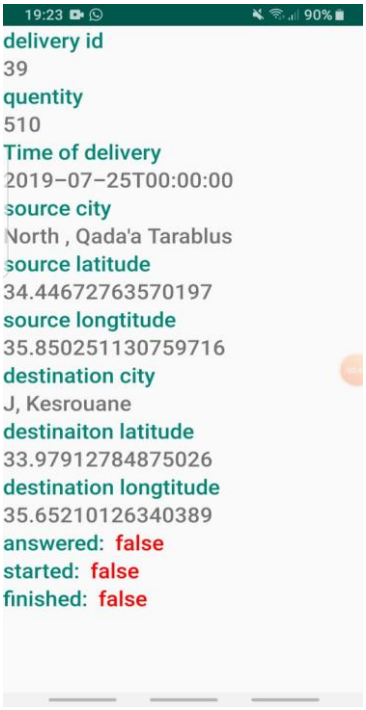

Figure 47 Delivery details

Figure 49 Driver can start and track deliveries

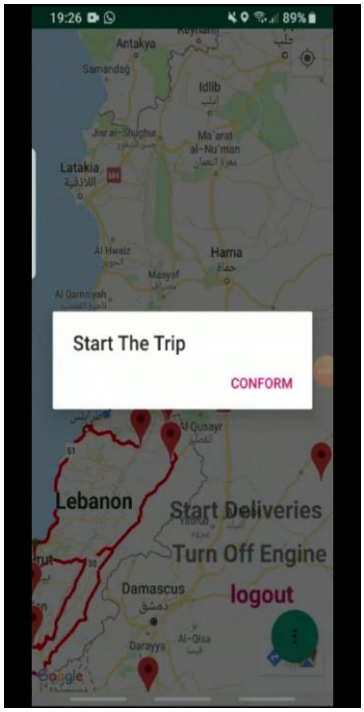

Figure 50 Starting a trip

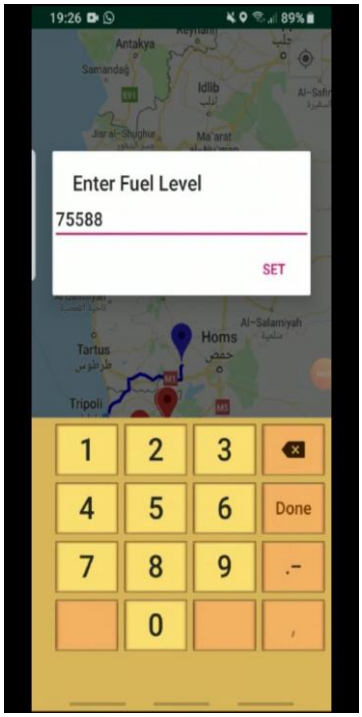

Figure 51 Updating fuel level

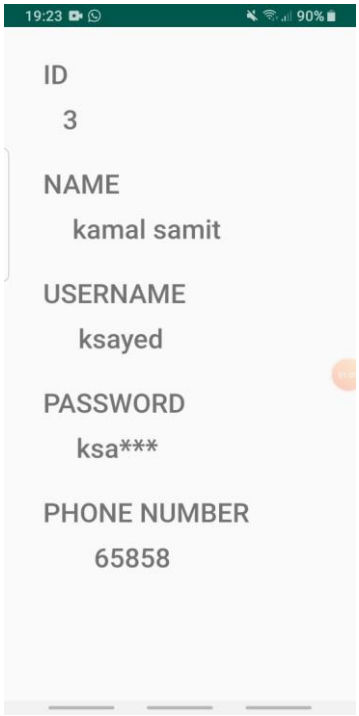

Figure 52 Driver account

CHAPTER 6: CONCLUSION AND EVALUATION

6.1 Further work

With the near completion of this project there are several possibilities to expand it. Below follows a brief description of areas which can be expanded upon to make the project more useful and reliable.

6.1.1 Integrating sensors technology

Until now all the vehicle information from fuel level, odometer, and brakes… is sent by the driver using the mobile application installed on his phone. However this practice is not efficient for many reasons: The driver could easily manipulate the system by changing this information to his own advantage, also the information recorded might not be accurate or transmitted in real time.

Sensor technology provides new insights for fleet tracking and maintenance, giving managers a full control on the vehicles and the information it transmits. All the information sent by the drivers phone can be recorded by sensors such as fuel level sensors, odometer sensor, brakes sensor … and transmitted with an onboard computer through cloud services.

6.1.1 Improving safety and security

Maintaining the highest safety conditions when having a business that relies on driving is a top priority for any fleet manager. Having a system in place for managing risks and collecting data can be very helpful in preventing accidents. This project can be altered in a way it ensures that every operation done is in compliance with safety regulations. More features can also be added to further improve the safety of the staff by monitoring the drivers' health and driving to ensure the drivers has the right skills and are able to perform perfectly.

6.2 Evaluation and Conclusion

The project has finished the tasks it set to accomplish. The features included in the project allows any fleet manager to monitor his fleet in a very efficient manner and to reduce all the unnecessary costs.

This project has been similar learning experience to that of first learning programming languages. At first, programming with c# and entity frame work was slow, however by the end of the project programming using these new technologies went at a much faster rate. Also the amount of time required to learn all the map API techniques was underestimated.

To conclude, the project of this project is of very good quality despite some potential bugs, especially that the three of us had no experience working with the used technologies such as: C#, Asp.Net, entity frame work, Map APIs, remote hosting …

CHAPTER 7: References

- HereMap API Documentation:

https://developer.here.com/documentation

- OpenLayer API Documentation:

https://openlayers.org/en/latest/apidoc/

- Asp.Net Core Microsoft Documentation:

https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

- Asp.Net API Core Microsoft Documentation:

https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-2.2

- Charts.js Documentation:

https://www.chartjs.org/docs/latest/getting-started/

- AmCharts Documentation:

https://www.amcharts.com/docs/v4/

- SmarterAsp Knowledge Base:

https://www.smarterasp.net/support/kb/root.aspx

- Bootstrap Documentation:

https://getbootstrap.com/docs/4.3/getting-started/introduction/

- W3C Schools Tutorials:

https://www.w3schools.com/

- GPS Coordinates:

https://www.gps-coordinates.net/

- Mozilla JavaScript Documentation:

https://developer.mozilla.org/en-US/docs/Web/JavaScript

APPENDIX A: API Actions

The Fleet API responds to the following actions. The input and expected output of each one is specified in details.

/api/DeliverySummaries/StartDeliverySummary

Input:

```
{
   "DeliveryId": 170,
   "StartTime": "2019/2/3T00:00:00",
   "StartFuelLevel": 4,
   "StartOdometer": 2,
    "Latitude":33.547484303646414,
   "Longtitude":35.420248437499993
}
```

Output:

```
{"deliverySummaryId":25}
```

api/DeliverySummaries/FinishDeliverySummary

Input

```
{
   "DeliverySummaryId": 4,
    "EndTime": "2019/2/3T00:50:00",
    "EndFuelLevel": 2,
    "EndOdometer": 120
}
```

Output

```
{
   "performanceScore": 4.16666651,
    "complianceScore": 5,
    "safetyScore": 4.25,
   "deliveryScore": 4.472222,
  "overallScore": 4.472222,
  "rank": 1,
  "nbOfDrivers": 3  }
```

/api/DeliverySummaries/UpdateDeliveryInfo

Input:

```
{
   "DeliverySummaryId": 25,
   "HarshAccelerationAndDeceleration": 1,
   "HarshBreakings": 1,
   "HardCornering": 1,
   "Speedings": 1,
   "SeatBelt": 1,
   "OverRevving": 1,
   "Odometer": 1,
   "EngineRunning": 1,
   "Latitude":33.56,
   "Longtitude":35.677
}
```

Output:

```
{"success":true}
```

api/DeliverySummaries/FinishDeliverySummary

Input {

```
   "Time": "2019/2/2",
   "CompanyName": "Taxi Company",
   "ClientId": 9,
   "SourceLongtitude": 35.589894796875,
   "SourceLatitude": 33.80672895624442,
   "SourceCity": "Aleyh",
   "DestinationLongtitude": 35.59195473339844,
   "DestinationLatitude": "33.346208737432754",
   "DestinationCity":"Marjayoun" ,
   "Quantity": 5  }
```

/api/Clients/PostClient

Input


```
{
   "Username": "ahmad@client.com",
   "Password": "Aa12345$",
   "Name": "Ahmad",
   "Birthdate": "1/1/2003",
   "Address": "Beirut",
   "Phonenumber": "03127648"
}
```


/api/Drivers/GetDriver

Input:


```
{
   "Username": "Kamal@gmail.com",
   "Password": "Aa12345$"
}
```


Output:


```
{
   "id": 1,
   "username": "kamal@gmail.com",
   "password": "Aa12345$",
   "name": "Kamal Sayed",
   "birthdate": "1998-06-15",
   "address": "Beirut",
   "phonenumber": "03365133",
   "rank": 1,
   "score": 0,
   "image": "driver1.jpg",
   "company": {
      "id": 1,
      "name": "Furniture Company",
      "address": "Beirut",
      "type": "Furniture",
```


api/Deliveries/AnsweredDeliveries

Input:


```
{
   "Username": "Kamal@gmail.com",
   "Password": "Aa12345$"
}
```


Output:


```
[
   {
      "id": 10,
      "time": "2019-06-15T07:38:04.2334415",
      "sourceLongtitude": 36.463949609374993,
      "sourceLatitude": 34.76438993722649,
      "sourceCity": "Homs",
      "destinationLongtitude": 38.364584374999993,
      "destinationLatitude": 33.303200870327309,
      "destinationCity": "Homs",
      "quantity": 6,
      "answered": true,
      "started": false,
      "finished": false,
      "optimalDistance": 0,
      "optimalTime": 0,
      "optimalFuelConsumption": 0
   },
   {
      "id": 11,
      "time": "2019-06-15T07:38:23.9685702",
      "sourceLongtitude": 36.238729882812493,
      "sourceLatitude": 33.1515644469665,
      "sourceCity": "Daraa",
      "destinationLongtitude": 37.370321679687493,
      "destinationLatitude": 34.184754325134293,
      "destinationCity": "Homs",
```

In a similar fashion the remaining actions are defined:

api/Clients/GetClient


```
{
   "Username": "ahmad@client.com",
   "Password": "Aa12345$"
}
```


/api/Deliveries/StartedDeliveries


```
{
   "Username": "kamal@gmail.com",
   "Password": "Aa12345$"
}
```


api/Deliveries/FinishedDeliveries


```
{
   "Username": "kamal@gmail.com",
   "Password": "Aa12345$"
}
```


api/Deliveries/GetDelivery


```
{
   "id":2,
   "Username": "kamal@gmail.com",
   "Password": "Aa12345$"
}
```