Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Systems Engineering

Master's Thesis

Delivery Route Planner web Application

Ahmed Tebakhi

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. Ahmed Tebakhi

Systems Engineering and Informatics
Informatics

Thesis title

Delivery Route Planner web Application

Objectives of thesis

This diploma thesis focuses on designing and implementing a web application for automated planning that gets Route Planning Solutions for local delivery company in Prague. The main goal of this application is to reduce spending time on manual scheduling, sorting out delivery management or specific route optimization details.

Methodology

The thesis will consist of two parts; the theoretical part will based on study Dijkstra's Algorithm to finds the shortest path between a given node (which is called the ”source node”) and all other nodes in a graph. This algorithm uses the weights of the edges to find the path that minimizes the total distance (weight) between the source node and all other nodes.

The practical part will start with analyzing the requirements, based on the requirements the database will be implemented using SQL Server, based on Shortest Paths APIs also, then the application will be designed, implemented and finally tested. The standard tools and methods of software engineering will be used during the whole process. Based on both theoretical and practical parts, the conclusion will be formulated and possible future development and improvements proposed.

The proposed extent of the thesis

55-70 pages

Keywords

Route planning, Shortest path, Travelling salesman problem, Web application, logistic optimization.

Recommended information sources

Akella, Ravindra, et al. Enterprise Application Development with C# 9 And . NET 5 : Enhance Your C# and . NET Skills by Mastering the Process of Developing Professional-Grade Web Applications, Packt Publishing, Limited, 2021. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=6507711.

Dincer, Alper, and Balkan Uraz. Google Maps API Cookbook, Packt Publishing, Limited, 2013. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=1572949.

Mancas, Christian. Conceptual Data Modeling and Database Design: a Fully Algorithmic Approach, Volume 1 : The Shortest Advisable Path, Apple Academic Press, Incorporated, 2015. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=4003224.

Shortest Path Solvers. from Software to Wetware, edited by Andrew Adamatzky, Springer International Publishing AG, 2018. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=5372081.

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

doc. Ing. Ludmila Dömeová, CSc.

Supervising department

Department of Systems Engineering

Electronic approval: 16. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Head of department

Electronic approval: 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 30. 03. 2023

Declaration

I declare that I have worked on my master’s thesis titled “Delivery Route Planner web Application" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 2023

[illegible]

Ahmed Tebakhi

Acknowledgement

First of all, I would like to thank my family for their continuous encouragement, support and love. They have always been there for me in times of my need in all aspects of life.

I would like to show my gratitude to my thesis supervisor doc. Ing. Ludmila Dömeová, CSc, for her great support and understanding in the completion of this thesis. Prof. Ludmila Dömeová was always clear and willing to answer whenever I had a question about my thesis research and writing.

I would also like to thank my teachers, my classmates and the academic staff of the Systems Engineering and Informatics of the Czech University of Life Sciences, Prague, who always supports me in the learning process.

Finally, I would like to thank my friends for their continuous support and motivation to complete the thesis. I also want to wish them a bright future and success in their life.

Delivery Route Planner web Application

Abstract

Delivery web applications have become integral to people's lives, especially in today's fast-paced world. With the convenience of ordering anything from the comfort of their homes or workplaces, people rely on delivery apps for everything from food to groceries, medicine, and even furniture. Delivery apps offer a hassle-free way to get what people need without leaving their homes or dealing with the stresses of traffic and parking. The ease of use and reliability of delivery apps have made them essential tools for people to manage their busy lives efficiently.

Delivery Route Planner web Application helps businesses optimize their delivery operations through a Delivery Route Planner web application. Based on delivery addresses, the software generates an optimized route plan based on factors such as traffic, distance, and time windows. It provides real-time tracking of drivers and deliveries and analytics and reporting tools for monitoring and optimizing operations. User-friendly interface with real-time driver and delivery statuses and analytics and reporting tools for monitoring performance allows businesses to easily manage delivery addresses, view driver and delivery statuses, and manage their deliveries. Users can also customize parameters such as delivery time windows and driver availability to suit their requirements. Fuel costs can be reduced, driver overtime can be minimized, and customer satisfaction can be increased by optimizing delivery routes. Small start-ups and large enterprises alike can use Delivery Route Planner, which is scalable. Business owners looking to improve their delivery operations and increase profit can benefit significantly from Delivery Route Planner. Business operations can be streamlined, costs can be reduced, and products can be delivered faster and more efficiently by leveraging route optimization technology.

Keywords: Route planner solution, travel salesman problem, MySQL, database, JavaScript, React, Node.js, Web App, TomTom API, Delivery logistic system.

Webová aplikace plánovač doručovacích tras

Abstrakt

Doručovací webové aplikace se staly nedílnou součástí života lidí, zejména v dnešním uspěchaném světě. Díky pohodlí objednat si cokoli z pohodlí domova nebo pracoviště se lidé spoléhají na aplikace pro doručování všeho, od jídla po potraviny, léky a dokonce i nábytek. Doručovací aplikace nabízejí bezproblémový způsob, jak získat to, co lidé potřebují, aniž by museli opouštět své domovy nebo řešit stres z provozu a parkování. Snadné použití a spolehlivost doručovacích aplikací z nich udělaly základní nástroje pro lidi, aby mohli efektivně řídit svůj rušný život.

Webová aplikace Delivery Route Planner pomáhá podnikům optimalizovat jejich doručovací operace prostřednictvím webové aplikace Delivery Route Planner. Na základě dodacích adres software vygeneruje optimalizovaný plán trasy na základě faktorů, jako je provoz, vzdálenost a časová okna. Poskytuje sledování řidičů a dodávek v reálném čase a analytické a reportovací nástroje pro monitorování a optimalizaci provozu. Uživatelsky přívětivé rozhraní se stavy ovladačů a dodávek v reálném čase a analytickými a reportovacími nástroji pro monitorování výkonu umožňuje podnikům snadno spravovat dodací adresy, zobrazovat stavy řidičů a dodávek a spravovat své dodávky. Uživatelé si také mohou přizpůsobit parametry, jako jsou dodací lhůty a dostupnost řidiče, aby vyhovovaly jejich požadavkům. Optimalizací dodacích tras lze snížit náklady na palivo, minimalizovat přesčasy řidiče a zvýšit spokojenost zákazníků. Malé start-upy i velké podniky mohou využívat Plánovač tras doručení, který je škálovatelný. Majitelé podniků, kteří chtějí zlepšit své doručovací operace a zvýšit zisk, mohou výrazně těžit z Plánovače tras doručení. Obchodní operace lze zefektivnit, snížit náklady a dodávat produkty rychleji a efektivněji využitím technologie optimalizace trasy.

Klíčová slova: Řešení plánovače tras, problém s cestovním prodejcem, MySQL, databáze, JavaScript, React, Node.js, Web App, TomTom API, Logistický systém dodávek.

Table of Contents

1. Introduction

The transportation and logistics industry is undergoing a rapid transformation, driven by the growth of e-commerce and the increasing need for fast and efficient delivery services. Delivery companies are pressured to optimize operations and reduce delivery times while maintaining high customer service standards. One of the critical challenges delivery companies face is planning delivery routes. With multiple stops to make and various constraints to consider, route planning can be a time-consuming and complex task. To address this challenge, advanced route planning systems have been developed that use algorithms to calculate the most efficient delivery routes. These systems consider distance, traffic, delivery volume, and time windows to create optimized routes that minimize delivery times and costs. However, such systems can be expensive and complex, making them inaccessible to smaller delivery companies.

A more cost-effective and accessible solution is a web application that enables delivery companies to create optimized delivery routes. A web application can be accessed from any device with an internet connection, allowing delivery companies to plan their routes from anywhere. It can also be designed to be user-friendly and easy to use, making it accessible to delivery drivers and operations staff with minimal training. In this diploma thesis, we will present the design and development of a web application for delivery route planning.

The thesis aims to provide a user-friendly and efficient tool for delivery companies to improve their operations and enhance customer satisfaction. The web application will be designed to consider the specific needs and requirements of delivery companies, such as the number of stops, delivery windows, and the size of the delivery vehicles.

Providing a cost-effective and accessible solution for delivery companies of all sizes, the web application for delivery route planning can potentially revolutionize the delivery industry. By improving their delivery operations, delivery companies can reduce costs, improve customer satisfaction, and gain a competitive edge in the rapidly evolving transportation and logistics industry.

Modern businesses constantly look for ways to improve their delivery processes to save money and time while improving the customer experience. A delivery route planner web application is one of the most effective ways to achieve this. By using this software tool, businesses can plan their delivery routes most efficiently, considering factors such as traffic, delivery time windows, and vehicle capacity. Thus, businesses can reduce delivery costs, improve delivery times, and provide better customer service. This article aims to examine the benefits of using a delivery route planner web application and how it can help businesses streamline their delivery process.

2. Objectives and Methodology

2.1 Objectives

This diploma thesis focuses on designing and implementing a web application for automated planning that gets Route Planning Solutions for a local delivery company in Prague. The main goal of this application is to reduce spending time on manual scheduling, sorting out delivery management, or specific route optimization details.

2.1 Methodology

The theoretical portion of the thesis will be centered on examining Dijkstra's Algorithm, which determines the shortest path between a specified node (referred to as the "source node") and every other node in a graph. In order to determine the path that minimizes the overall distance (weight) between the source node and all other nodes, this technique uses edge weights.

The practical part will start with analyzing the requirements; based on the requirements, the database will be implemented using SQL Server and Shortest Paths APIs. Then, the application will be designed, implemented, and finally tested. The standard software engineering tools and methods will be used throughout the process. The conclusion will be formulated based on both theoretical and practical parts, and possible future development and improvements will be proposed.

3. Literature Review

3.1 Dijkstra’s Algorithm: The Shortest Path Algorithm

3.1.1 Purpose and Use Cases

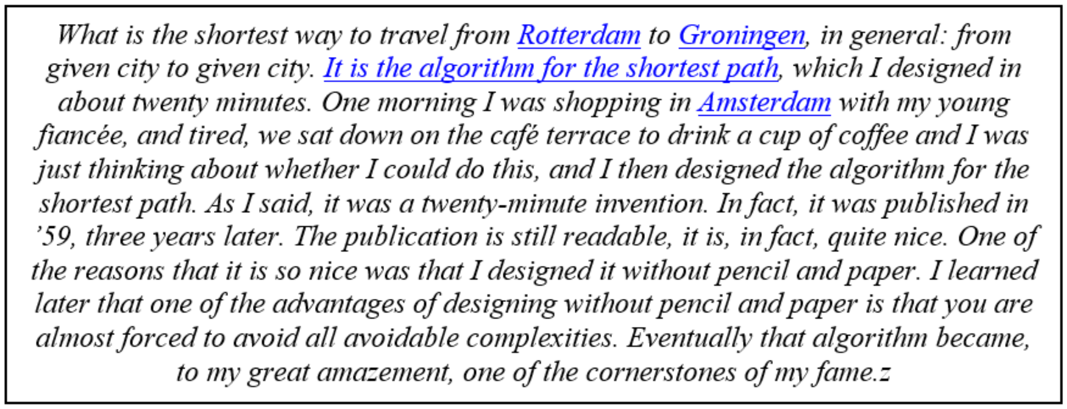
A graph's shortest path can be found with Dijkstra's Algorithm. The shortest path tree is generated by finding the shortest path between a node (called the "source node") and all other nodes in the graph.

GPS devices use this algorithm to find the shortest path between their location and destination. It has broad applications in industry, especially in domains that require modeling networks.

3.1.2 Dijkstra's History

Mathematician and computer scientist Edsger Wybe Dijkstra was from the Netherlands. He passed away in Nuenen on August 6, 2002, and was born on May 11, 1930, in Rotterdam, Netherlands. Before accepting a professorship at the University of Texas at Austin in 1984, Dijkstra held academic positions as a researcher and professor in the Netherlands. The Dijkstra algorithm, one of the most well-known in computer science, was developed by Dijkstra while working as a programmer at the Mathematical Center in Amsterdam in 1956 to showcase the capabilities of a brand-new computer named ARMAC. His goal was to select both a computer-generated problem and a solution that non-computer experts could grasp. His most outstanding achievement was coming up with the. His invention of the structured programming paradigm for creating computer programs made him most famous.

Brilliant Dutch computer scientist and software engineer Dr. Edsger W. Dijkstra developed and published this approach. Dr. Dijkstra described the algorithm's creation process and motivations in an interview from 2001 (sauer-utley, 2020):



What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city. It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in ’59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.z

Figure 1 describes Dr. Dijkstra’s speech (Dijkstra, 2001).

3.1.3 Introduction to Graphs

Real-time objects, persons, or other entities are called nodes (or vertex) in graphs, data structures designed to describe connections between a few of these elements. The connections between nodes are referred to as edges. Moreover, two nodes are only linked if an edge exists between them.

Generally, graphs are suited to real-world applications, such as graphs illustrating a transportation system/network, where nodes represent facilities that transfer or obtain products. Edges show routes or subways connecting nodes.

Graphs can represent complex relationships between objects and are thus helpful for modeling and analyzing real-world systems, such as transportation networks.

One of the most common applications of graphs is to represent transportation systems. In this context, nodes represent facilities that transfer or obtain products, and edges represent routes or subways that connect those facilities. For example, a graph could be used to model a city's subway system, with each station represented as a node, and each subway line represented as an edge.

This representation allows us to analyze the transportation system in several ways, such as using graph algorithms to find the shortest path between two stations or identify the most efficient way to route products between different facilities.

Graphs can also be used to model social networks, where nodes represent individuals and edges represent their relationships. This kind of representation can help us to understand how information and influence spread through a network or to identify key individuals who play essential roles in the network.

In addition to transportation and social networks, graphs can be applied to other real-world systems. For example, they can be used to model the structure of molecules in chemistry, the flow of electrical currents in circuits, or the connections between websites on the internet.
One of the strengths of graphs as a modeling tool is their flexibility.

They can be used to represent many different kinds of relationships and can be adapted to suit the specific needs of a particular application. Additionally, many powerful algorithms have been developed for analyzing and manipulating graphs, making them valuable tools for data analysis and decision-making.

In conclusion, graphs are a versatile and powerful tool for modeling and analyzing real-world systems. Their ability to represent complex relationships between objects makes them well-suited to transportation and social network applications. With the right tools and techniques, graphs can help us understand and optimize these systems in various ways..

3.1.4 Dijkstra Algorithm implementation

Dijkstra's algorithm starts with the source node, the node that will be chosen, and it looks at the complete graph to find the shortest path between the source node and other nodes in the graph. The method keeps track of the shortest path found so far between each node and the source code and updates these values when a new shortest path is found.

After finding the short path between two nodes, we can make the node visited and be added to the path once the algorithm has found the quickest route across the source code to another node.

The process is repeated until all of the nodes in the graph have been added to the path, creating a path that connects the source node to every other node while also taking the most logically sound route to each node (Cormen, 2001).

Now describe the algorithm implementation procedure step by step:

- All nodes must initially be marked as "unvisited."
- Give the chosen starting node a distance of 0 and the remaining nodes a distance of infinite.
- Fix the initial node to become the current node now.
- Add the current node's distance to the weight of the edge connecting the neighboring node and the current node to calculate the distances between the current node and its unexplored neighbors.
- Compare the most recent measurement to the distance already allocated to the neighboring node, then set that distance as the new current distance for that node.
- Consider all of the current node's unvisited neighbors. After that, mark the current node as visited, and then.
- When an algorithm ends, and the destination node has been marked as visited, stop.
- Instead, select the unvisited node marked with the shortest distance and set it as the new current node before repeating the procedure from step 4 with the selected node.

3.1.5 Dijkstra algorithm Pseudocode code

First, we called the distance from the source to the destination u is represented by dist[u] in the following pseudocode algorithm, which is an array including the origin distances from the source to other vertices. The shortest path from the origin point to the specified destination is contained in the prior array via references to previous-hop nodes. In the vertex set Q, the code u vertex in Q with min dist[u] looks for the vertex u with the lowest dist[u] value. The function edges(u, v) returns the length of the edge connecting (or, more precisely, the distance between) the neighboring nodes. The length of the path passing through [u] from the root node to neighbor node v is represented by the variable alt on line 14. This alternative path will be used instead of the shortest path recorded for [v] if it is cost less.

```
1  function Dijkstra(Graph, source):
 2
 3      for each vertex v in Graph.Vertices:
 4          dist[v] ← INFINITY
 5          prev[v] ← UNDEFINED
 6          add v to Q
 7      dist[source] ← 0
 8
 9      while Q is not empty:
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
18
19      return dist[], prev[]
```

Figure 2 describes Dijkstra algorithm Pseudocode code (McConnell, 2004).

If we are only interested in the shortest path between the destination's origin and destination point, we can decide the result after ten lines if (u) = destination point. So, using reverse iteration, we can read the shortest route between source and target:

```
1  S ← empty sequence
2  u ← target
3  if prev[u] is defined or u = source:          // Do something only if
the vertex is reachable
4      while u is defined:                       // Construct the shortest
path with a stack S
5          insert u at the beginning of S        // Push the vertex onto
the stack
6          u ← prev[u]
```

Figure 3 describes Dijkstra algorithm Pseudocode code (McConnell, 2004).

The collection of vertices making up one of the shortest pathways from source to target is now known as sequence S or the empty sequence if no path is present.

Finding the shortest routes between the source and the objective would be a more general task (there might be several different ones of the same length). Then, we would store all nodes meeting the relaxation condition instead of just one node in each item of prev. As the edge cost is identical in all scenarios, we would add both r and source to prev [target] if, for instance, both r and source connect to the target and lie on different shortest paths through the target.

3.2 TomTom — Mapping and Location Technology

TomTom is a leading provider of navigation and location-based services, offering a range of APIs that enable developers to integrate TomTom's technology into their applications. There has been growing interest in using TomTom's APIs in various industries, including transportation, logistics, and urban planning. As a result, several studies and reviews of the TomTom API have been conducted in academic and industry literature (Manson, 2017).

These reviews generally focus on the strengths and limitations of the API, including its accuracy, reliability, ease of use, and potential applications in various industries. Some studies have also compared the TomTom API to other navigation and location-based services, such as Google Maps and OpenStreetMap.

Overall, the literature review suggests that the TomTom API is a competent and versatile tool for integrating location-based services into applications, with various potential applications in various industries.

3.2.1 TomTom's Map Display API

TomTom's Map Display API is a web-based service allowing developers to embed interactive maps and location-based services. The API offers a comprehensive set of tools and functionalities that enable developers to create customized maps, add markers and annotations, and display dynamic information related to specific locations.

With the Map Display API, developers can integrate maps and location-based services into various applications, including fleet management systems, ride-hailing apps, logistics platforms, and more. The API provides access to a global map database, which includes detailed street-level maps, satellite imagery, and traffic data, enabling developers to create highly detailed and accurate maps.

The Map Display API is designed to be easy to use and integrate into existing applications. It supports multiple programming languages and platforms, including JavaScript, Python, and Android. It provides a range of customization options, such as map styles, markers, and overlays, allowing developers to create maps that match the branding and design of their applications.

In addition to its core functionalities, the Map Display API offers advanced features, such as real-time traffic data, geocoding and reverse geocoding, and routing and navigation services. These features enable developers to create sophisticated location-based applications that provide users with real-time information about traffic conditions, optimal routes, and nearby points of interest.

Overall, TomTom's Map Display API provides developers with a powerful and flexible tool for creating interactive maps and location-based services that enhance the user experience of their applications.

3.2.2 Matrix Routing services

Matrix Routing is a powerful feature of the TomTom API that allows developers to calculate travel times and distances between multiple origins and destinations. This technology is beneficial for optimizing logistics operations and planning efficient routes for fleets of vehicles (Van der Zijpp, 1996).

The Matrix Routing API takes a set of origins and destinations as input and returns a matrix of travel times and distances between all pairs of points. This matrix can calculate the shortest or fastest route between any two points or optimize a complex routing problem involving multiple stops and vehicles.

The input to the API can be provided in several formats, including addresses, coordinates, or a combination of both. The API also allows users to specify additional parameters such as travel mode (driving, walking, or cycling), traffic conditions, and vehicle profiles.

One of the critical advantages of Matrix Routing is that handling extensive data is efficient and easy. The API is designed to process up to 1,000 origins and destinations in a single request, which can significantly simplify the development of complex routing algorithms.

The Matrix Routing API uses a combination of TomTom's proprietary routing algorithms and real-time traffic data to calculate the travel times and distances between pairs of points. The algorithms consider road types, speed limits, turn restrictions, and traffic signals to calculate the most efficient route between each pair of points.

The real-time traffic data is gathered from various sources, including GPS devices, mobile apps, and traffic sensors. This data estimates the current traffic conditions on each route segment and adjust the travel times accordingly.

The output of the Matrix Routing API is a matrix of travel times and distances between all pairs of points, represented as a two-dimensional array. This matrix can calculate the shortest or fastest route between any two points or optimize a complex routing problem involving multiple stops and vehicles.

In summary, Matrix Routing is a powerful feature of the TomTom API that allows developers to calculate travel times and distances between multiple origins and destinations. This technology is beneficial for optimizing logistics operations and planning efficient routes for fleets of vehicles. The API is designed to handle large datasets efficiently.

It uses routing algorithms and real-time traffic data to calculate the most efficient route between each pair of points.

Route Directions API is a web service that determines routes between several sites. The API can determine the best direct, time-saving, or economic path between two or more points. The API is well-documented and straightforward to use. The TomTom API has all the information you require to get going.

The Route Directions API has a wide range of applications. Here are a few illustrations:

- Determine the shortest path between various places.
- Determine the shortest path between various places.
- Determine the delivery driver's path that is the most effective.
- The most effective path for a road trip.

The Route Directions API provides multiple parameters to calculate various routes. The whole list of parameters and a brief description are provided below. Our API is covered in great detail on the API documentation page.

- Type: The route type to calculate. Short, balanced, and fewer maneuvers are all possible values. Short is the default value.
- Avoid: avoid the kinds of places to stay away from. Highways, tolls, locations, and ferries are examples of potential values. None is the default value.
- Details: Whether or not to provide more details about the route, like the overall mileage and expected journey time. Elevation, route details, and instruction details are all potential values.
- Traffic: the traffic model used to choose routes, utilized only for driving, trucking, and other motorized vehicle modes.
- Units: the units to be used for time and distance measurements. Imperial and metric measurements are options. Metric is the default setting.
- Waypoint: the places between which to compute the route. Each waypoint is formatted as latitude and longitude.

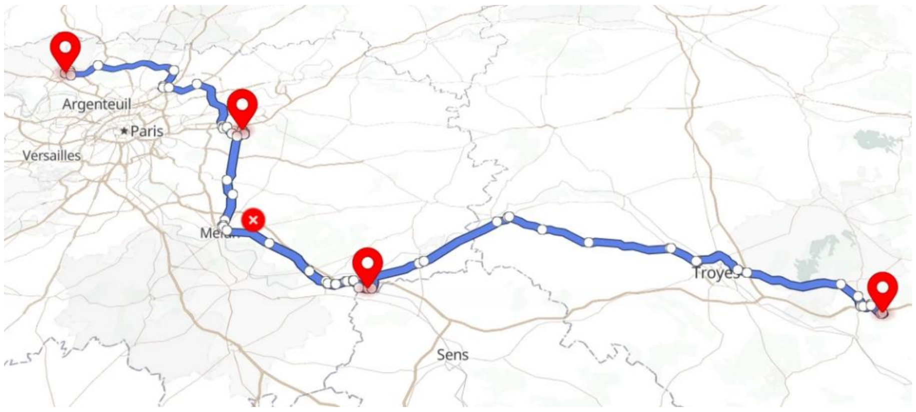


Figure 4 shows the route Directions and points using TomTom API (Geoapify, 2022).

3.3 The Travel Salesman Problem

The Traveling Salesman Problem is a classic optimization problem in computer science and mathematics. The problem involves finding the shortest route that visits a set of cities and returns to the starting city. This problem is of great interest in logistics, transportation, and supply chain management, as it has practical applications in route planning and scheduling (Fevrin, n.d.).

The TSP can be formulated as follows: Given a list of n cities and the distances between each pair of cities, what is the shortest route that visits each city exactly once and returns to the starting city (Cook, n.d.)?

The approach is to generate all possible routes, calculate their total distances, and then select the route with the shortest distance. However, this approach quickly becomes infeasible for larger problem sizes as the number of possible routes grows exponentially with the number of cities.

To address this challenge, various algorithms have been developed to solve the TSP efficiently. One popular approach is to use heuristics and metaheuristics, which use rules of thumb or randomized search strategies to find suitable solutions in a reasonable amount of time.

One such heuristic is the Nearest Neighbour algorithm, which starts at a random city and repeatedly visits the nearest unvisited city until all cities have been visited. This algorithm is easy to implement and often yields reasonable solutions, but it can also produce suboptimal results.

Another popular algorithm is the 2-Opt algorithm, which involves iteratively swapping pairs of edges in the tour to improve its length. This algorithm can be used as a post-optimization step after applying another algorithm to generate an initial solution.

Metaheuristics such as simulated annealing, genetic algorithms, and ant colony optimization have also been applied to the TSP with varying levels of success. These algorithms often find high-quality solutions but can be computationally expensive and difficult to tune.

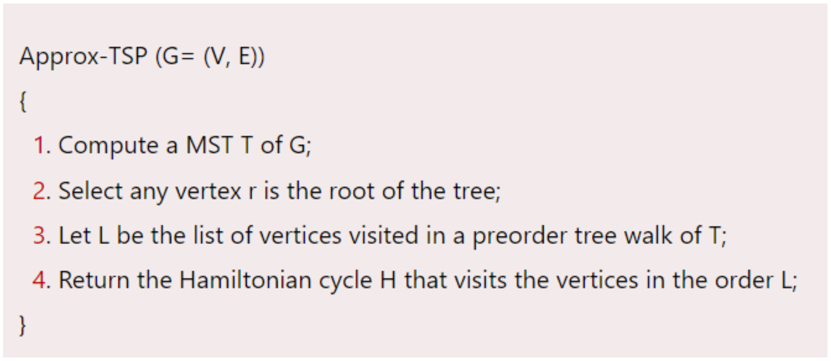
Despite the advances in algorithmic techniques for solving the TSP, it remains a challenging problem, particularly for large problem sizes. Researchers continue to investigate new approaches to the TSP and apply its principles to other optimization problems in logistics, transportation, and supply chain management.

In conclusion, the Traveling Salesman Problem is a classic optimization problem with practical applications in logistics, transportation, and supply chain management. Various algorithms have been developed to solve the TSP efficiently, including heuristics, metaheuristics, and optimization techniques. While the problem remains challenging for large problem sizes, researchers continue investigating new approaches to the TSP and applying its principles to other optimization problems.

3.3.1 Traveling-salesman Problem implementation

In the traveling salesman problem, a driver must visit n points. The salesman wishes to make a tour or cycle path, visiting each point exactly once and finishing at the point he started. This algorithm does not accept any negative cost from point to point. The goal is to find a path of minimum cost for drivers. We assume that every two points are connected. Such problems are called the Traveling-salesman problem (TSP). We can model the points as a complete graph of n vertices, where each edge represents a point. Assuming that cost function c satisfies the triangle inequality, we can use the following approximate algorithm.

Define u, v, and w. as three vertices.



```
Approx-TSP (G= (V, E))
{
  1. Compute a MST T of G;
  2. Select any vertex r is the root of the tree;
  3. Let L be the list of vertices visited in a preorder tree walk of T;
  4. Return the Hamiltonian cycle H that visits the vertices in the order L;
}
```

Figure 5 describes Traveling-salesman Problem formula (Anon., 2018).

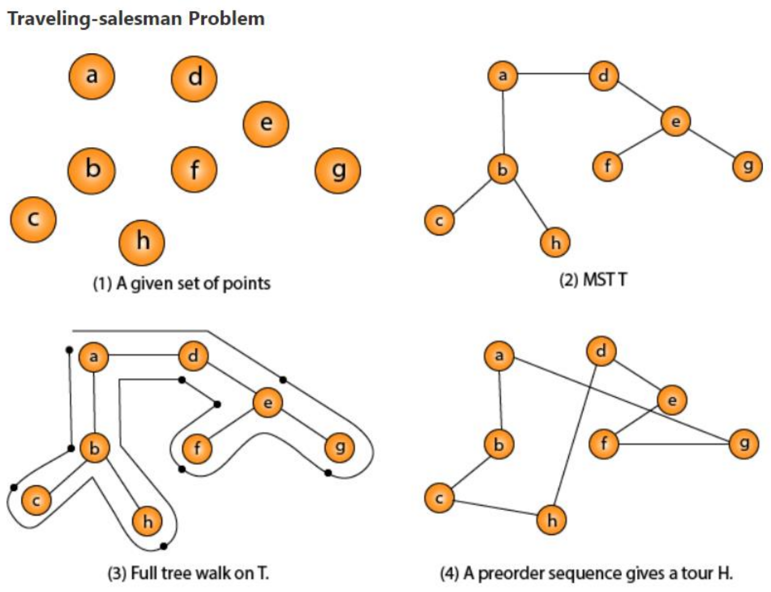

Traveling-salesman Problem
a
d
e
b
f
g
c
h
(1) A given set of points
(2) MST T
(3) Full tree walk on T.
(4) A preorder sequence gives a tour H.

Figure 6 describes Traveling-salesman Problem graph (Anon., 2018).

3.4 React JavaScript Library

React is an open-source JavaScript library used for building user interfaces. It was developed by Facebook and released to the public in 2013. Since then, React has become one of the most popular libraries for building web applications, with a large and active community of developers.

At its core, React is designed to make it easy to build large-scale, complex user interfaces. It provides a set of reusable components that can be combined to create custom UI elements. These components are designed to be modular so they can be easily reused across different parts of an application.

One of the key benefits of using React is its performance. React is built around a virtual DOM (Document Object Model), a lightweight representation of the actual DOM. This means that updates to the UI can be processed more efficiently, which results in faster rendering times and a smoother user experience (Porcello, 2020).

Another critical feature of React is its ability to manage application state. React provides a simple and efficient way to manage data within an application, making it easier to keep track of changes and ensure that everything stays in sync. That makes building more complex applications that require dynamic data and user interactions easier.

React also provides several other useful features, such as server-side rendering, which allows applications to be pre-rendered on the server before being sent to the client. That can help improve performance and SEO and provide a better user experience.

In addition to these core features, React has a vast and active community of developers who have created a wide range of plugins, libraries, and tools to extend its capabilities. That makes it easy to find solutions to common problems and integrate React with other technologies.

React is a flexible library for building modern web applications. Its modular design, performance benefits, and state management capabilities make it an excellent choice for building complex user interfaces. Moreover, with a large and active community of developers, plenty of resources and tools are available to help you get started and build unique applications (Inc., 2023).

3.4.1 Most Important React libraries

3.4.1.1 Redux library

In particular, Redux is a state management library for React-based JavaScript applications. It was developed in 2015 as a solution to the issue of handling complicated states in large applications Dan Abramov and Andrew Clark.

At its core, Redux operates on the principle of a single immutable state tree. That means an application's state is stored in a single object that cannot be directly mutated. Instead, any changes to the state must be made through dispatching actions, which are plain JavaScript objects that describe what happened in the application (e.g., "user clicked a button," "data was loaded from the server").

Reducers are pure functions that take action and the current state and produce a new state object reflecting the changes that action described. The application's user interface is then updated using the new state.

Redux also provides middleware, which can be used to add additional functionality to the dispatch process. That can include things like logging, async operations, and more.
Overall, Redux provides a powerful and flexible way to manage application state and is widely used in the React ecosystem (CyberWolves, 2021).

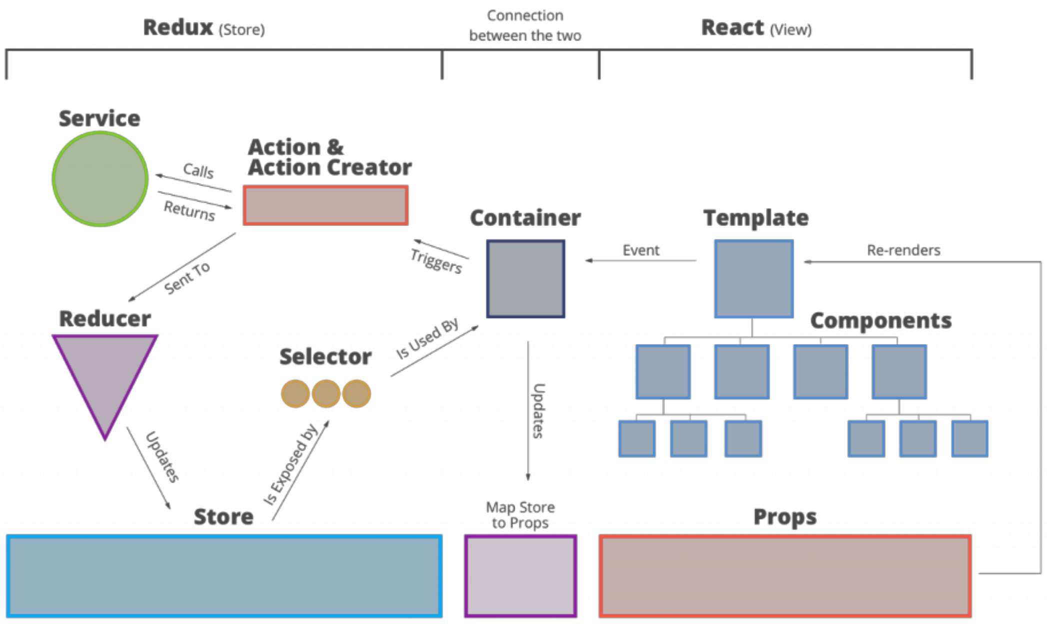


Figure 7 describes connection between react and redux (geosolutionsgroup, 2020).

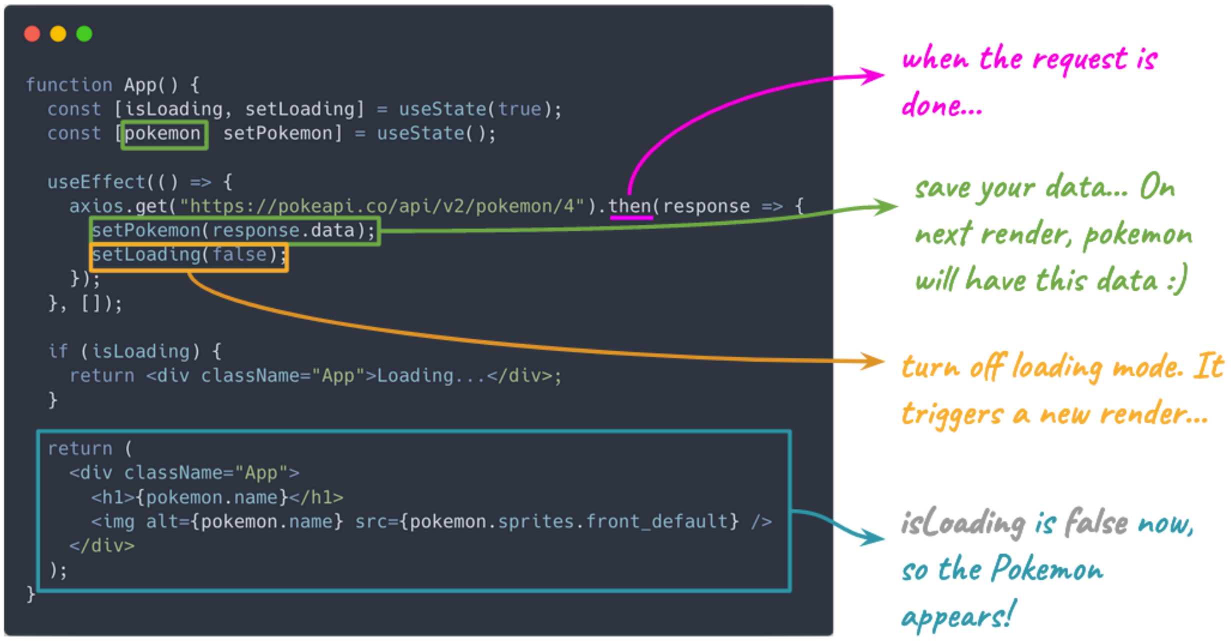
3.4.1.2 Axios library

Axios is the most JavaScript library for making HTTP requests from web applications. It is often used in front-end applications built with frameworks like React and Vue but can also be used in server-side Node.js applications.

Axios provides a simple and consistent API for making HTTP requests and supports a wide range of features, including:

- Promise-based API: Axios returns promises which allow us to use async/await syntax for handling responses.
- Interceptors: Axios provides a mechanism for intercepting requests and responses, allowing you to modify or handle them as needed.
- Request and response transformations: Axios allows you to transform the data being sent or received using functions that can be useful for formatting data or handling errors.
- Automatic serialization: Axios can automatically serialize request and response data in various formats, including JSON, form data, and more.
- Cancelation: Axios provides a built-in mechanism for canceling requests, which can help prevent unnecessary requests from being sent.

Overall, Axios is a powerful and flexible library for making HTTP requests from JavaScript applications and is widely used in the web development community (Anon., 2023).



```
function App() {
  const [isLoading, setLoading] = useState(true);
  const [pokemon, setPokemon] = useState();

  useEffect(() => {
    axios.get("https://pokeapi.co/api/v2/pokemon/4").then(response => {
      setPokemon(response.data);
      setLoading(false);
    });
  }, []);

  if (isLoading) {
    return <div className="App">Loading...</div>;
  }

  return (
    <div className="App">
      <h1>{pokemon.name}</h1>
      <img alt={pokemon.name} src={pokemon.sprites.front_default} />
    </div>
  );
}
```

when the request is done...
save your data... On next render, pokemon will have this data :)
turn off loading mode. It triggers a new render...
isLoading is false now, so the Pokemon appears!

Figure 8 describes how axios making http request (Nico, 2020).

3.4.1.4 TomTom international/web-sdk-maps

Tomtom-international/web-sdk-maps is a JavaScript library that provides tools and components for integrating maps and location-based services into web applications. TomTom, a leading location data and services provider, developed and maintained it.

The library is built on top of the TomTom Maps SDK, which provides high-quality map data and a robust set of APIs for working with location data. It includes features such as:

· Map display and interaction: The library provides a flexible set of tools for displaying and interacting with maps, including support for markers, overlays, and custom styling.

· Geocoding and search: The library includes APIs for geocoding (i.e., converting addresses to latitude/longitude coordinates) and searching for locations based on keywords or categories.

· Routing and navigation: The library provides APIs for calculating routes between locations and displaying turn-by-turn navigation instructions.

· Traffic and weather information: The library includes APIs for accessing real-time traffic and weather information, which can be used to provide more accurate and relevant location-based services.

Overall, tomtom-international/web-sdk-maps provides a robust set of tools for integrating location-based services into web applications and is a popular choice for developers who need to work with maps and location data (support, 2023).

3.5 Tailwindcss Library

Tailwind CSS provides a set of pre-defined styles and classes that can be easily applied to any HTML element. Adam Wathan, Steve Schoger, and Jonathan Reinink created it. It has gained popularity in the web development community due to its ease of use and flexibility.

The primary goal of Tailwind CSS is to provide developers with a set of low-level utility classes that can be used to create custom designs quickly and easily. These utility classes can be combined to create more complex styles and are designed to be highly configurable and customizable.

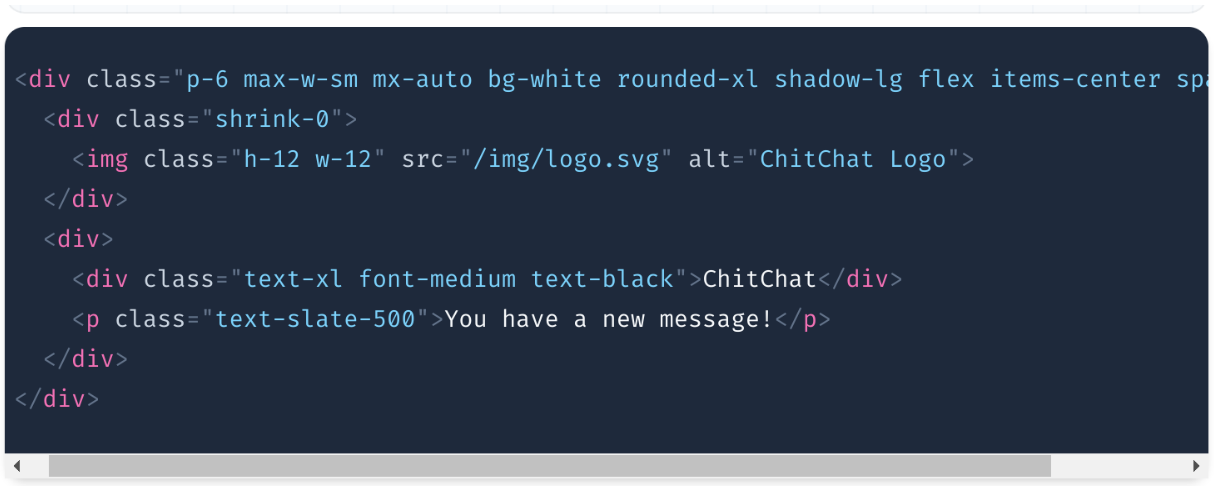
One of the main advantages of Tailwind CSS is its simplicity. The framework is designed to be easy to use and understand, even for developers new to CSS. The classes are named descriptively, making it easy to understand what each one does, and the framework provides much flexibility for customization.

Tailwind CSS has a minimal footprint, which can be easily integrated into any project without adding many bloats. That is especially important for performance-sensitive applications, where every kilobyte of data can significantly affect load times.

Another advantage of Tailwind CSS is its responsive design capabilities. The framework provides a range of responsive classes that can be used to create designs that work well on different screen sizes and devices. That means developers can create responsive designs quickly and easily without writing many custom CSS codes (Ovuoba, 2021).

Tailwind CSS also includes a range of plugins that can be used to extend its functionality. For example, plugins for forms, typography, and even dark mode exist.

Overall, the Tailwind CSS framework provides developers with many utility classes and customization options. Its plainness and ease of use make it an excellent choice for developers who want to create custom designs quickly and efficiently without writing much custom CSS code. Moreover, its small footprint and responsive design capabilities make it an excellent choice for performance-sensitive applications and mobile-first designs.



```
<div class="p-6 max-w-sm mx-auto bg-white rounded-xl shadow-lg flex items-center sp
  <div class="shrink-0">
    <img class="h-12 w-12" src="/img/logo.svg" alt="ChitChat Logo">
  </div>
  <div>
    <div class="text-xl font-medium text-black">ChitChat</div>
    <p class="text-slate-500">You have a new message!</p>
  </div>
</div>
```

Figure 9 shows an example of tailwind CSS library code (tailwindcss, 2021).

3.6 MySQL Database

MySQL is a relational database management system widely used by developers and businesses of all sizes. It was initially developed by Swedish developers Michael Widenius and David Axmark in 1995 and later acquired by Oracle Corporation in 2010. MySQL is written in C and C++ programming languages and is available for various operating systems, including Windows, macOS, and Linux.

MySQL is a strong database management system that stores organizes, and manages large amounts of data. It uses a structured query language (SQL) to manage data and allows users to define the relationships between different data sets. This makes it easy to search, sort, and filter data and create complex reports and analytics.

One of the main advantages of MySQL is its speed and scalability. It is designed to handle large amounts of data and high-traffic websites, making it an excellent choice for businesses with growing needs. MySQL also supports clustering, which allows multiple servers to work together to handle large amounts of traffic and data.

Another advantage of MySQL is its flexibility. It is compatible with various programming languages, including PHP, Java, Python, and C++. It makes it easy to integrate MySQL into existing systems and create custom applications that can interact with the database.

MySQL also provides a range of security features, including user authentication and access control. It makes it easy to manage user access and keep sensitive data secure.

MySQL has a large and active community of developers contributing to its development and support. It means that many resources are available for developers who need help with MySQL, including documentation, forums, and tutorials.

Overall, MySQL is a powerful and flexible database management system widely used by developers and businesses worldwide. Its speed, scalability, and flexibility make it an excellent choice for businesses with growing needs, and its security features make it a safe choice for storing sensitive data. With a large and active community of developers, plenty of resources are available to help developers get started and support them as they build applications with MySQL (Technologies, 2022).

3.7 Unified Modeling Language

UML is a standard notation software engineers use to create visual models of software systems. It was created by Grady Booch, James Rumbaugh, and Ivar Jacobson in the 1990s and has since become a widely accepted standard in the software development industry.

The purpose of UML is to provide a standardized way to represent complex software systems. Using UML, software engineers can create visual diagrams that describe the structure, behavior, and interactions of different components within the system. It can make the software development process more efficient and effective.

One of the main advantages of UML is that it is a standardized notation that can be easily understood by developers, designers, and other stakeholders involved in the software development process. It helps to ensure that everyone is on the same page and that there is a common understanding of the system being developed.

UML provides wide range of diagrams that can be used to represent different aspects of software systems. These diagrams include use case diagrams, which describe the system's functionality from the user's perspective; class diagrams, which explain the structure of the system and the relationships between different classes; and sequence diagrams, which describe the interactions between different components of the system over time.

Another advantage of UML is that it is a flexible notation that can be adapted to different development methodologies and programming languages. It means that it can be used to model systems developed using agile methodologies and those developed using more traditional approaches.

Various software development tools, including modeling tools, code generators, and testing tools, also support UML. These tools can make the software development process more efficient and effective by automating many software modeling and testing tasks.

Overall, UML is a powerful and flexible notation widely used by software engineers to model complex software systems. Its standardized notation, a wide range of diagrams, and support for different development methodologies and programming languages make it a valuable tool for software development teams worldwide. Using UML, software engineers can create visual models of software systems that can help ensure that everyone involved in the development process has a common understanding of the system being developed.

There are several types of UML diagrams, each with its specific purpose. Here is an overview of the principal UML diagrams:

1. Use Case Diagram: This diagram represents the user's interaction with the system. It shows the different use cases or scenarios where the user interacts with the system and the actors involved in those interactions.

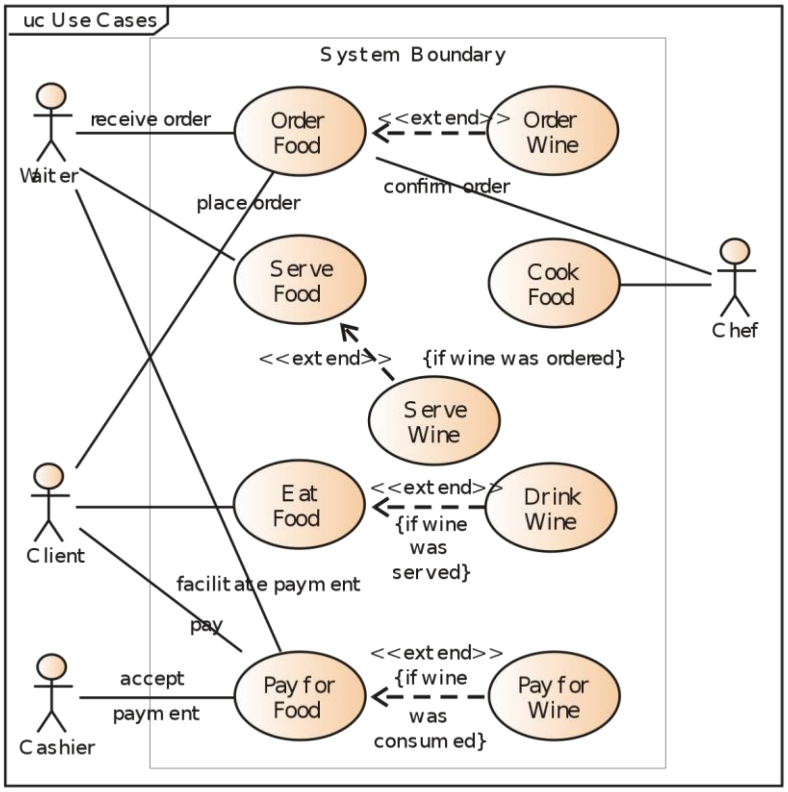


Figure 10 represents an example of use case diagram (Kawabata, 2007).

2. Class Diagram: This diagram represents the system's classes, interfaces, and relationships. It shows the attributes and methods of each class and their relationships with other classes.

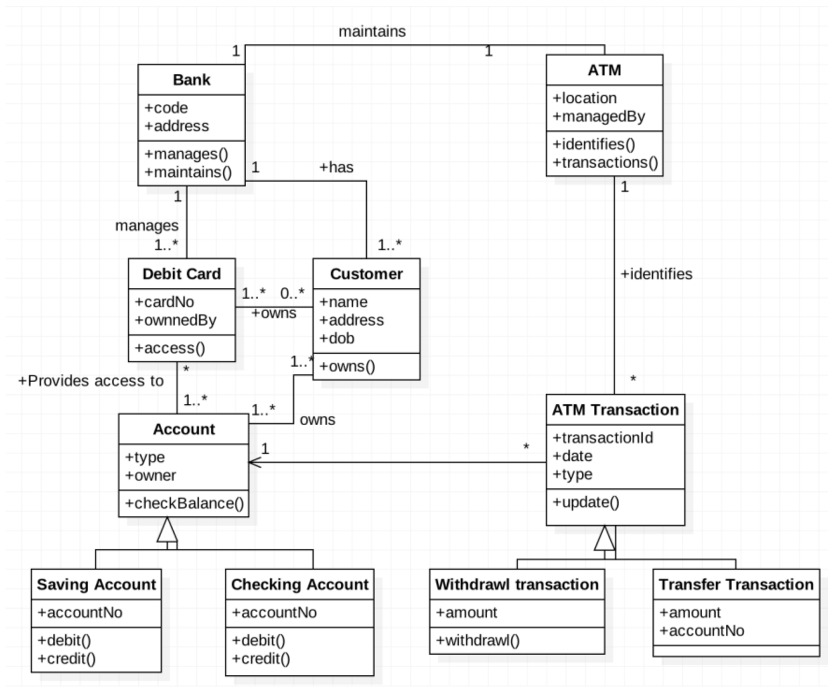


Figure 11 represents an example of a class diagram (Lopez-Rojas, n.d.).

3. Sequence Diagram: This diagram represents the chronological interactions between objects or classes. It shows the sequence of messages exchanged between objects or classes.

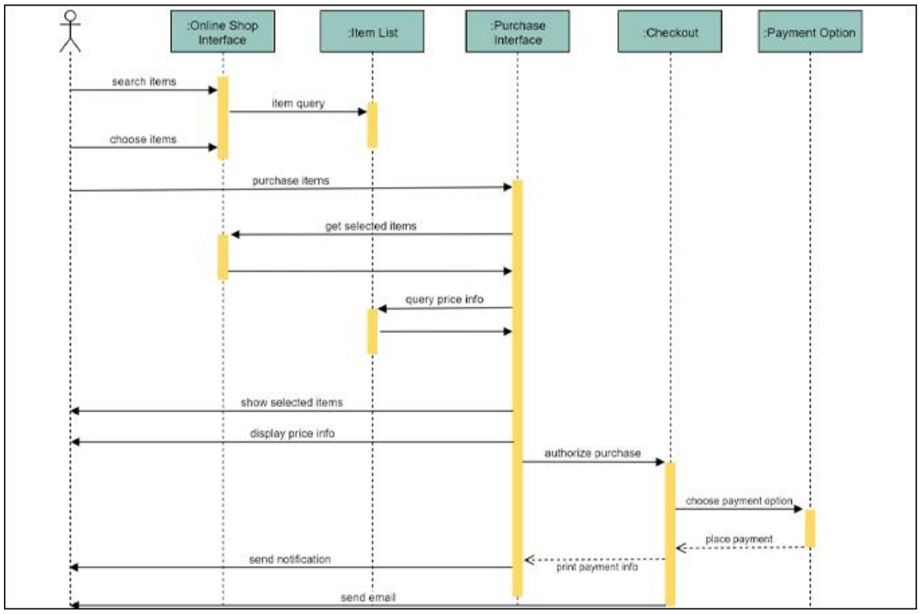


Figure 12 represents an example of a Sequence Diagram (Kawabata, 2007).

4. State Diagram: This diagram represents the different states of an object and the events that trigger transitions between those states. It shows the states, events, and actions during state transitions.

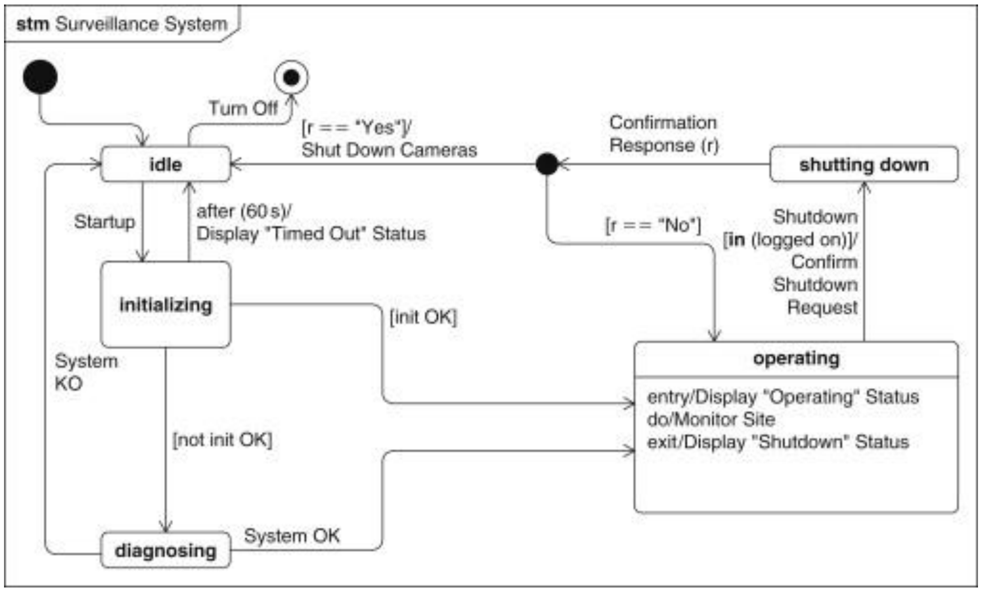


Figure 13 represents an example of a State Diagram (Kawabata, 2007).

5. Activity Diagram: This diagram represents a system's flow of activities or processes. It shows the different activities and the conditions or decisions that affect the flow of the activities.

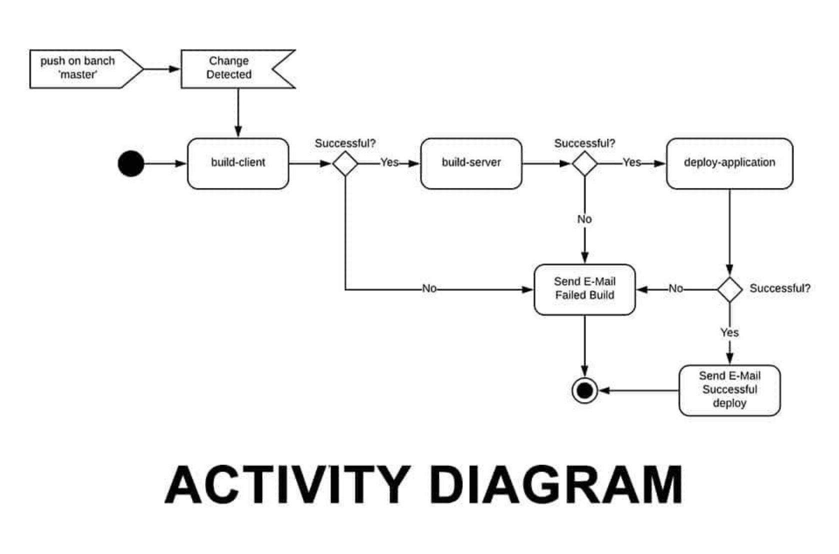


Figure 14 represents an example of an Activity Diagram (Kawabata, 2007).

There are more types of UML diagrams as Object diagrams, Collaboration diagrams, Component diagrams, and Deployment Diagram.

3.8 UI/UX Design

UI design, or user interface design, is designing the visual elements and user interactions of software applications, websites, and other digital products. UI design aims to create intuitive, user-friendly, and aesthetically beautiful user interfaces.

One of the critical principles of UI design is usability. A good user interface should be simple to use and navigate using labeling that is clear and straightforward, buttons, and other interface elements. UI designers strive to create intuitive interfaces that require minimal training for users to understand.

Another important aspect of UI design is visual design. A good UI designer will create visually appealing interfaces with a consistent design language and style. It includes selecting appropriate colors, typography, and images and creating a visual hierarchy to guide the user's attention to the essential elements of the interface.

Accessibility is also an essential consideration in UI design. A skilled user interface (UI) designer will ensure that people with disabilities can use the interface. It may involve using appropriate color contrasts, providing alternative text for images, and ensuring the interface is navigable using a keyboard alone.

UI design is an iterative process, with designers creating and testing multiple interface versions before arriving at a final design. It involves gathering feedback from users and making adjustments based on their feedback.

One of the challenges of UI design is creating interfaces that work well across various devices and screen sizes. With the increasing popularity of mobile devices, UI designers must ensure that the interface works well on both desktop and mobile devices, with appropriate scaling and layout adjustments.

Creating user-centered digital goods or services, simple to use and successful in accomplishing their objectives is known as UX design or user experience design. Whether a user interacts with a website, app, or other digital product, UX design aims to create a good experience.

UX design is a connection process that involves ongoing testing and refinement to create a product that meets the user's needs. By focusing on the user's needs and goals, UX designers can create products that are easy to use and effective in achieving their intended purpose (Matco, 2023).

4. Practical Part

In this chapter, we delve into the planning, analysis, design, and implementation of the Delivery Route Planner web application, which offers companies a suite of powerful tools, including driver matrix routing control, the ability to add and edit driver data, and order display. Additionally, we explore other essential functions that enhance website performance, such as user authentication and login capabilities, as well as secure data storage in the database.

4.1 Functional Requirements

Functional requirements are specific outcomes that a system must achieve. Their implementation involves facilitating communication between the application system and its users, regardless of whether the system is executed. The following list comprises the primary functional requirements of this web application:

- User registration and login.
- Ability to enter the origin and destination addresses.
- Display the delivery route from the origin to the destination.
- Calculation of the delivery cost based on the delivery option chosen and the distance between origin and destination.
- Ability to add drivers and add orders with the destination.
- Display a map with all destinations that have been added.

4.2 Non-Functional Requirements

Non-functional requirements refer to the quality attributes of a system that help assess the system's quality. Failing to meet non-functional requirements can result in the system's inability to meet and satisfy user needs. The success of the software system depends on its ability to meet critical non-functional standards such as responsiveness, usability, security, portability, and others.

- Performance: The application should be able to handle a large number of users and provide quick responses.
- Reliability: The application should be available and accessible 24/7 with minimal downtime.
- Security: The application should provide access to user information and payment details.
- Usability: The application should be easy to use and navigate for users with varying levels of technical expertise.
- Compatibility: The application should be compatible with web browsers and mobile devices.
- Scalability: The application should be able to handle an increasing number of users and delivery requests without compromising performance or reliability.

4.3 Analysis model

The necessary analysis of system design for this application is discussed using different UML diagrams for the visualization; all database relationships are correct; ensure that all factors and environment are suitable for launching the application without any problems. All libraries and functions used must be compatible with each other.

4.4 Static model

The following figure shows the overview of a class diagram. It depicts the static relationships between objects present in the application. Each class represents the objects with their related attributes and methods/operations. Before building the diagram, it is necessary to recognize relationships between the classes, their communication, and their links, associations, and generalizations with other classes in the diagram.

As shown in the figure below, the classes and their relationships are designed considering the project description of the web application. That includes the classes of user, administrator, customer, driver, route, location, and order. The relationship among the classes and their respective multiplicities are identified in the diagrams.

We can see in figure no; the class user inherits or generalizes three other classes, i.e., Administrator, Driver, and Customer. These three classes inherit all the attributes and method from the parent User class and also has their particular methods. The parent class User abstracts the common attributes from the child classes even though the child classes have their attribute. The other type of relationships we see here are three of the composition association between the order class and customer class, location class and order class, and route class and location class.

This type of association shows the strong relationships between the class as it implies the ownership from the whole to the part. In this case, the customer class has composition relationships with the order class. Destroying the customer class leads to destroying the order class. Order class cannot exist without the customer class. And the same thing for all location and route classes.

In the following class diagram, the class administrator has an Association relationship with the driver, that it can add a new driver with driver class type. Also administrator class has an Association relationship with the route class so that it can add a new route with the route class type.

The following Diagram explains all the relationships between the classes required by this application, as it explains the relationship between the user and user types and the relationship between an administration-type class and other classes such as Route, Driver, and others.

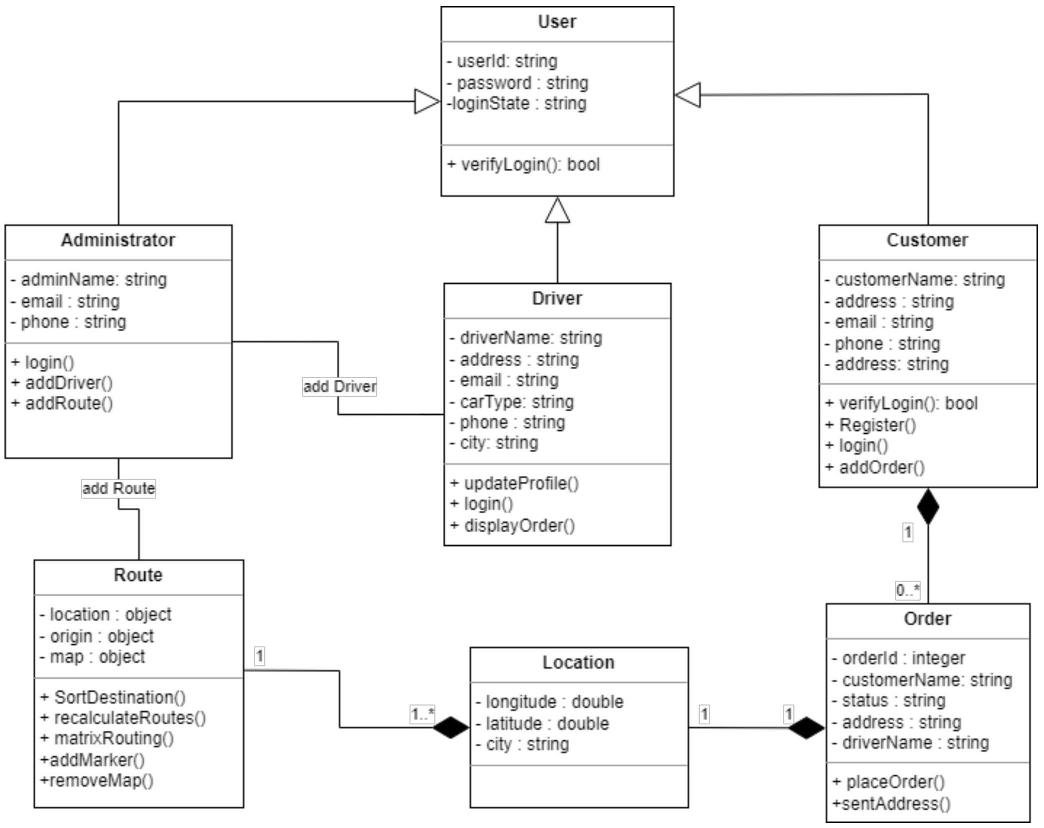


Figure 15 display class diagram and their relationships(own work).

4.5 Dynamic model

4.5.1 Use case diagram:

The following figure for this web application shows cases involving the customer, administrator, and driver as an actor. As the customer first visits the web application, they are prompted to log in to be able to make an order.

The login use case is an included relationship that shows the dependency with verify password use case. The verify password use case is also implemented every time the login use case is implemented.
The admin is available to see the available drivers, and they can add routes and orders task for each.

The drivers can display all tasks they should be done. Add route use case shows the dependency relationship on driver information and locations with origins points. The admin can also add a driver to our system by using add driver use case.

The customers can add orders to the system by using add order use case. The order use case shows the dependency relationship between order details and addresses.

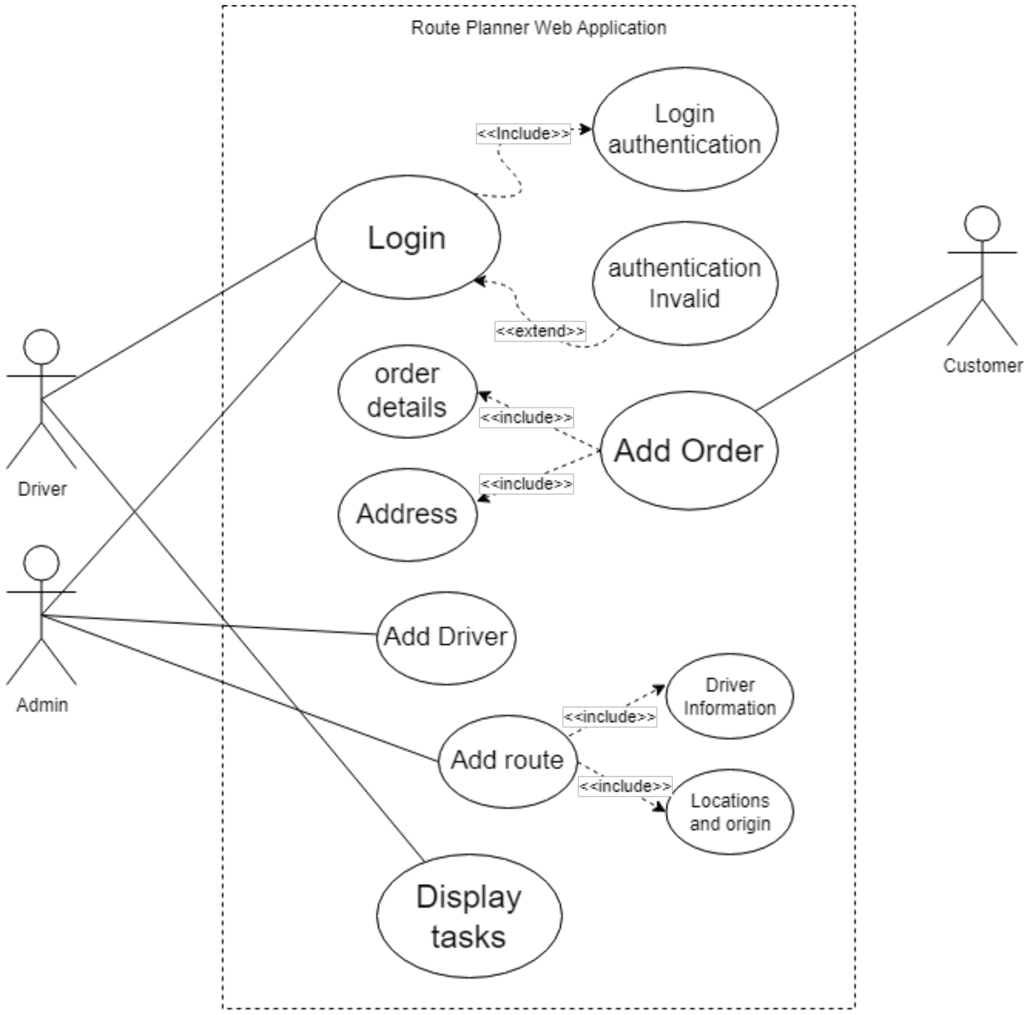


Figure 16 Use case diagram and their relationships(own work).

4.5.2 Sequence diagram:

Sequence diagrams are the type of UML diagrams that show how the objects in the system interact. These diagrams show interactions in the order that takes place. In other words, they show the sequence of events. The first figure shows the implementation of the sequence diagram. The process starts as the user visits the site and log-in to access our web application. The scenario starts with the user attempting to log-in by inputting the username and password. If the log-in credential provided by the user is authenticated and authorized, the user can see and interact with the available services in the system. The second figure shows the process for adding a new route plan to our system. It starts with selecting a driver from the database to add a route for him/her; then the admin asks for a request from the database to return all information he needs to make a new route; after the admin gets all locations then, he can select the points he wants to make a route for them, then send these points to the server to modify and running a specific function to make a new route, after server finish routing the path, display the route for admin. He can decide to confirm the routing or do it again.

- Sequence diagram for login operation

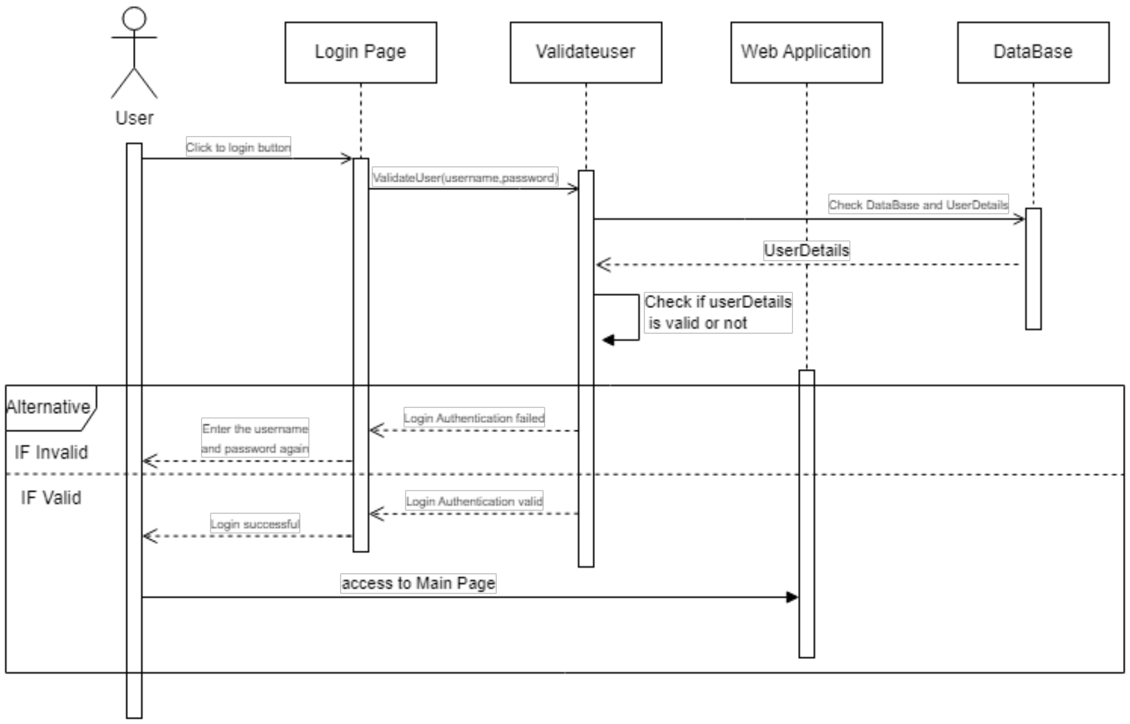

User
Login Page
Validateuser
Web Application
DataBase
Click to login button
ValidateUser(username,password)
Check DataBase and UserDetails
UserDetails
Check if userDetails is valid or not
Alternative
IF Invalid
Login Authentication failed
Enter the username and password again
IF Valid
Login Authentication valid
Login successful
access to Main Page

Figure 17 display Sequence diagram for login operation(own work).

- Sequence diagram for routing operation

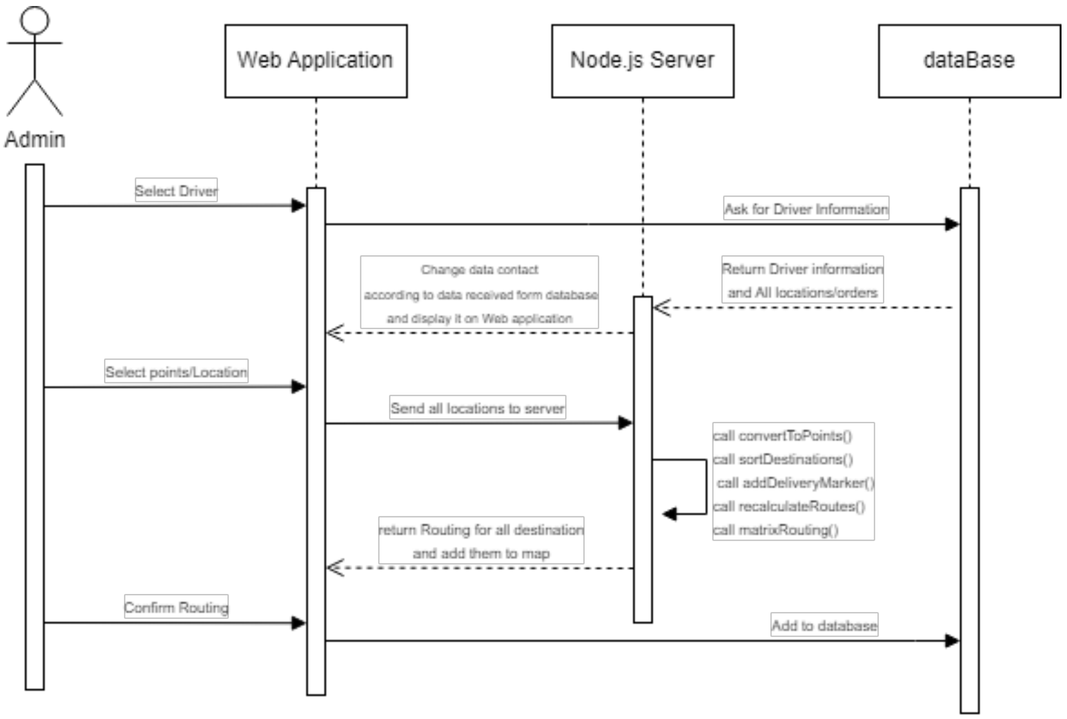

Admin
Web Application
Node.js Server
dataBase
Select Driver
Ask for Driver Information
Change data contact according to data received form database and display it on Web application
Return Driver information and All locations/orders
Select points/Location
Send all locations to server
call convertToPoints()
call sortDestinations()
call addDeliveryMarker()
call recalculateRoutes()
call matrixRouting()
return Routing for all destination and add them to map
Confirm Routing
Add to database

Figure 18 display Sequence diagram for routing operation(own work).

5. Implementation of web application

The implementation process is divided into three main parts, UI/UX design, Client-side, server-side, and Database connection. We will use Adobe Illustrator for UI design, React app platform for the client side, node.js for the server side, and MySQL workbench for the database.

5.1 UI/UX design

5.1.1 Login Page

Logging in is required to access the application. There are two inputs on the login page, the username, and password, and then a button to send the entered information to our database and to confirm the username is there, and also to verify the password, either yes or no, and if the data entered is incorrect, the page will ask the user to enter it again, but if it is correct, it will take the user to the main page.

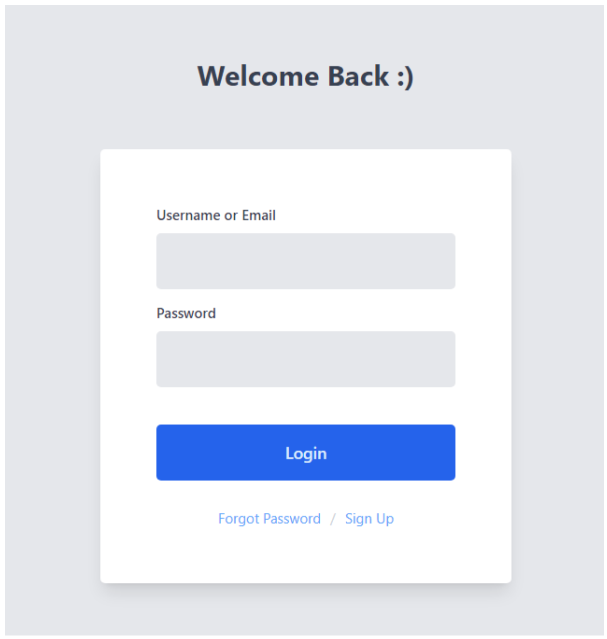


Figure 19 shows login page design(own work).

5.1.2 Main Page

Upon logging in successfully, the program sends the user to the home page, which contains the following:

- Main menu: all pages that the user can access are listed on the menu, and each time he or she clicks on an item, a new page will be displayed
- Sidebar: contain all drivers available so the user can check all tasks or order for each driver.
- Map: display all locations or points for each driver, different from driver to driver.

- Logout button: this button lets the user log out of the application.
- Profile button: this button route the user to the profile page for the user, then he can edit or modify his personal information.

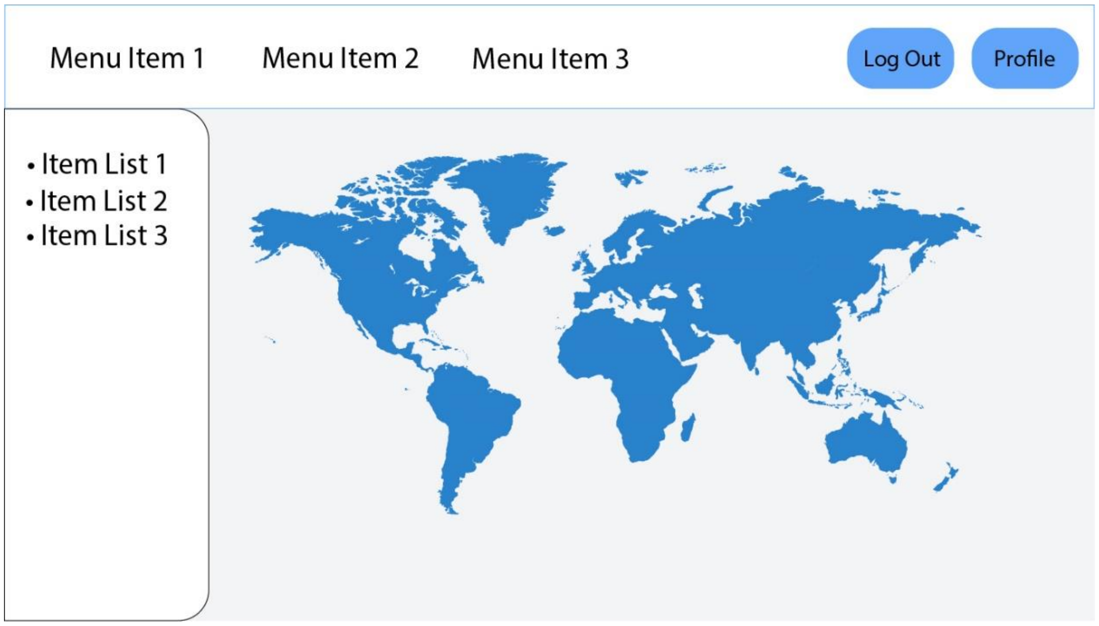


Figure 20 shows the Main Page design(own work).

5.1.3 Add Order page

On this page, customers can add orders with customer names, phone numbers, email, longitude, latitude, and address information to add this information to the database.

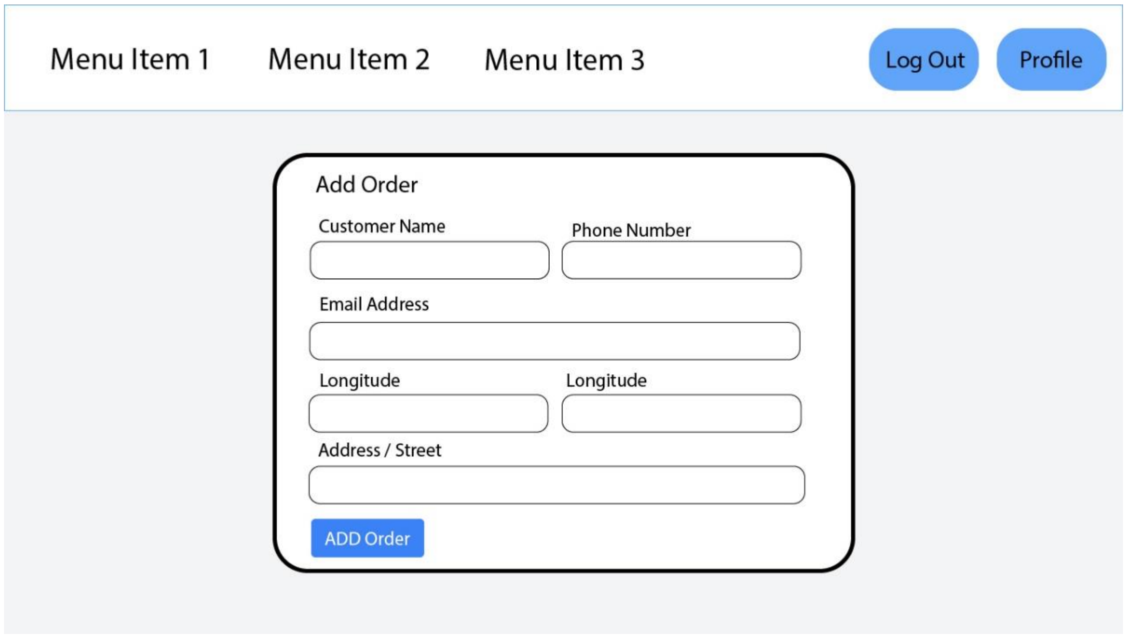


Figure 21 shows the Add Order Page design(own work).

5.1.4 Add driver page

Admin users can only access this page, and they can add more drivers to the system. This page has ten inputs, including usernames, passwords, confirm passwords, first names, last names, car types, emails, phone numbers, addresses, and cities. This page has ten inputs, including usernames, passwords, confirm passwords, first names, last names, car types, emails, phone numbers, addresses, and cities—moreover, one button to add this information to the database.

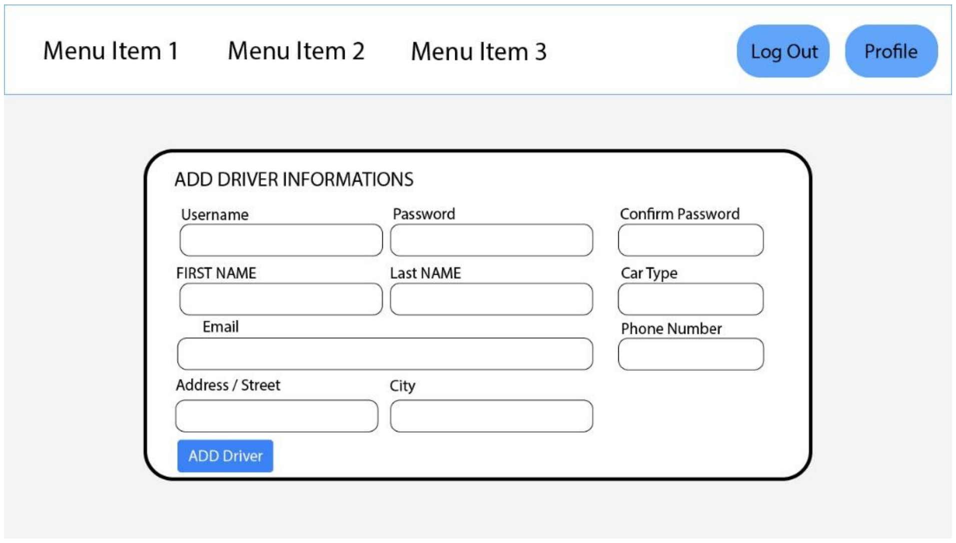


Figure 22 shows the Add Driver Page design(own work).

5.1.5 Order List page

Admin users can only access this page. On this page, the admin can see all the orders that need to be done. On this page, the admin can add any of these orders to one of the couriers for this company, which will be added directly to the couriers. He can also review all orders and check whether the data is correct or incorrect.

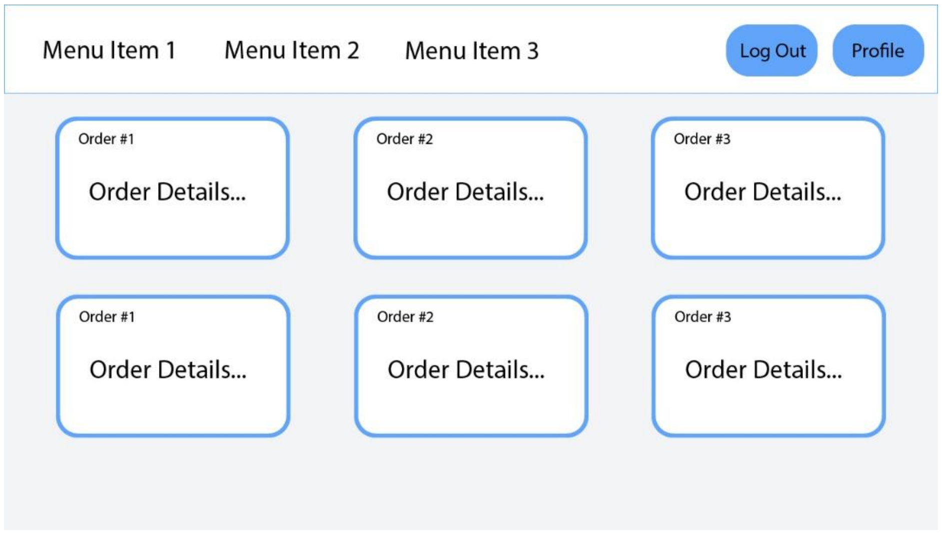


Figure 23 show the Order List Page design(own work).

5.2 Client-side

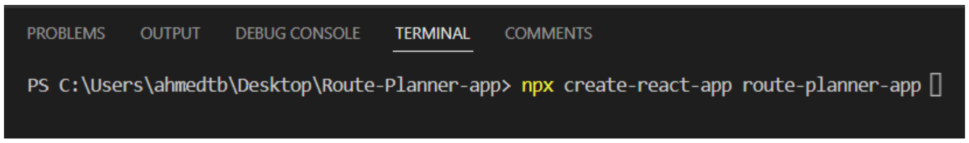
5.2.1 React application platform installation

To create a React app, you can use Create React App, a command-line interface tool that creates a new React project with a single command. Here are the steps:

- First, install Node.js and npm installed on your computer:

- Second, install Visual Studio Code – a code editor that is redefined and optimized for building and debugging modern web and cloud applications.

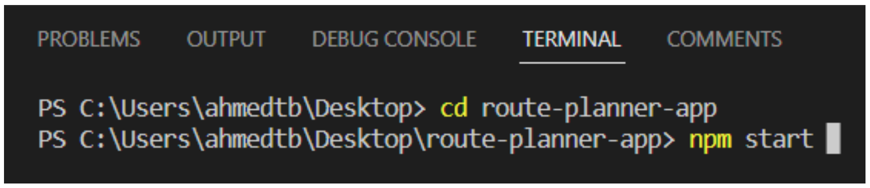
- Third, Install or set up react library package. To create a new react app, we need to use this command:



```
PS C:\Users\ahmedtb\Desktop\Route-Planner-app> npx create-react-app route-planner-app
```

source code 1 shows how to install react app(own work).

Then we can run the react app by this command after going to the path for react app:



```
PS C:\Users\ahmedtb\Desktop> cd route-planner-app
PS C:\Users\ahmedtb\Desktop\route-planner-app> npm start
```

source code 2 shows how to run react app(own work).

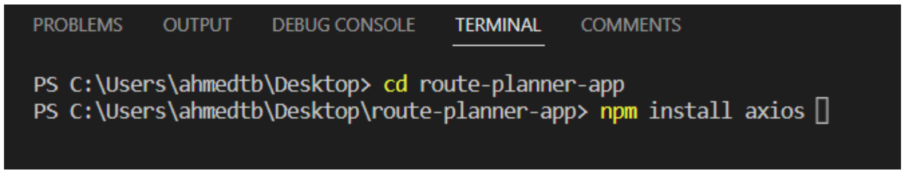
- Install the tailwind CSS library package

Instead of creating a CSS file for each component, we can use the tailwind CSS library and add it to our application. That provides styling the code on the same page.

5.2.2 Axios Installation

I used Axios for making HTTP requests from a web browser or Node.js server, automatic transformation of data, and error handling.

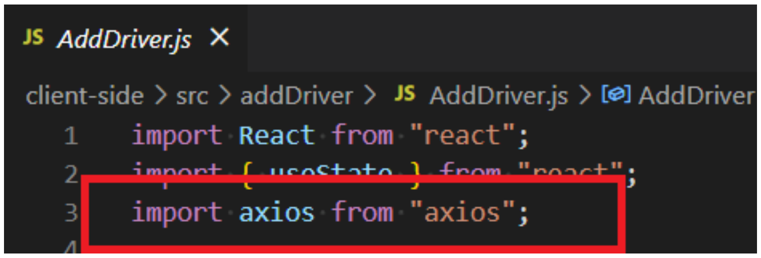
- Open the terminal on visual code editor and run this following command:



```
PS C:\Users\ahmedtb\Desktop> cd route-planner-app
PS C:\Users\ahmedtb\Desktop\route-planner-app> npm install axios
```

source code 3 shows how to install axios library(own work).

- Now it's ready for use, and I can use it by import Axios to specific component:



```
import React from "react";
import { useState } from "react";
import axios from "axios";
```

source code 4 shows an example how to import axios(own work).

5.2.3 TomTom API installation

In the web application, we used tomtom API to build our application with accurate Maps, Places, and Routing. To use this API, we must register on the tomtom website https://developer.tomtom.com/ and get the API key.

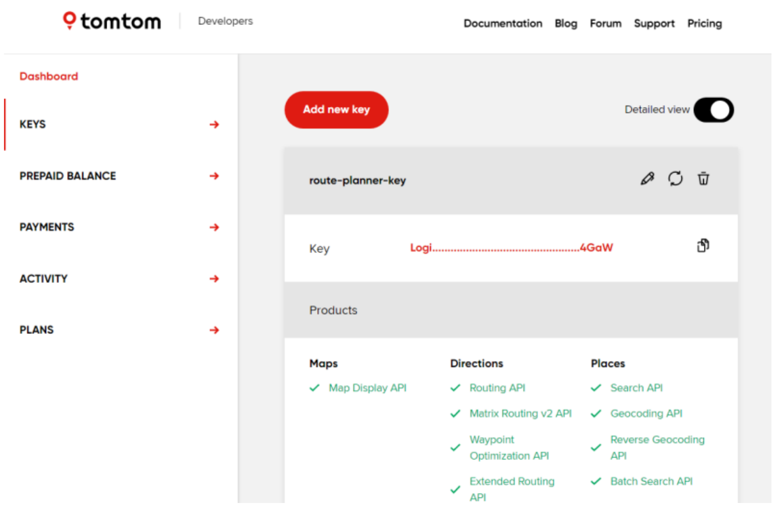
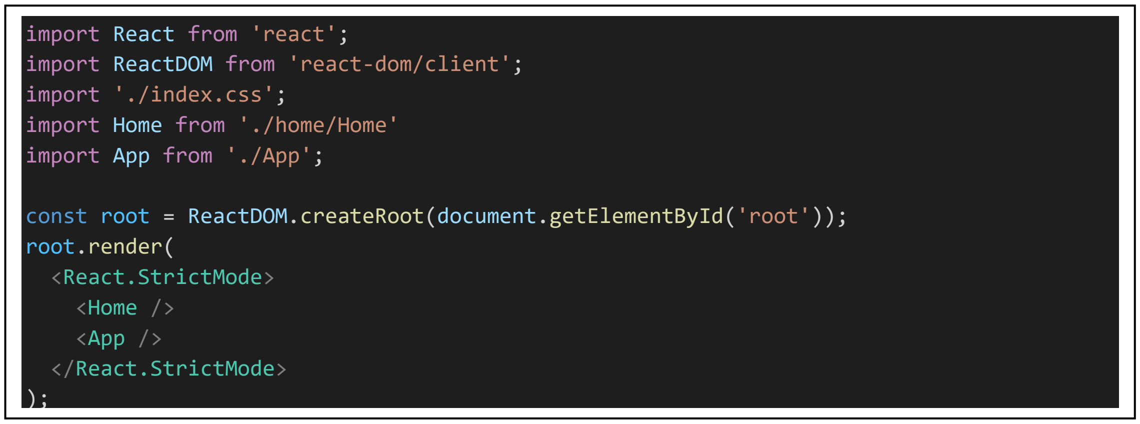


Figure 24 shows TomTom API dashboard and how to get API key (tomtom, n.d.).

5.2.4 Create React Components

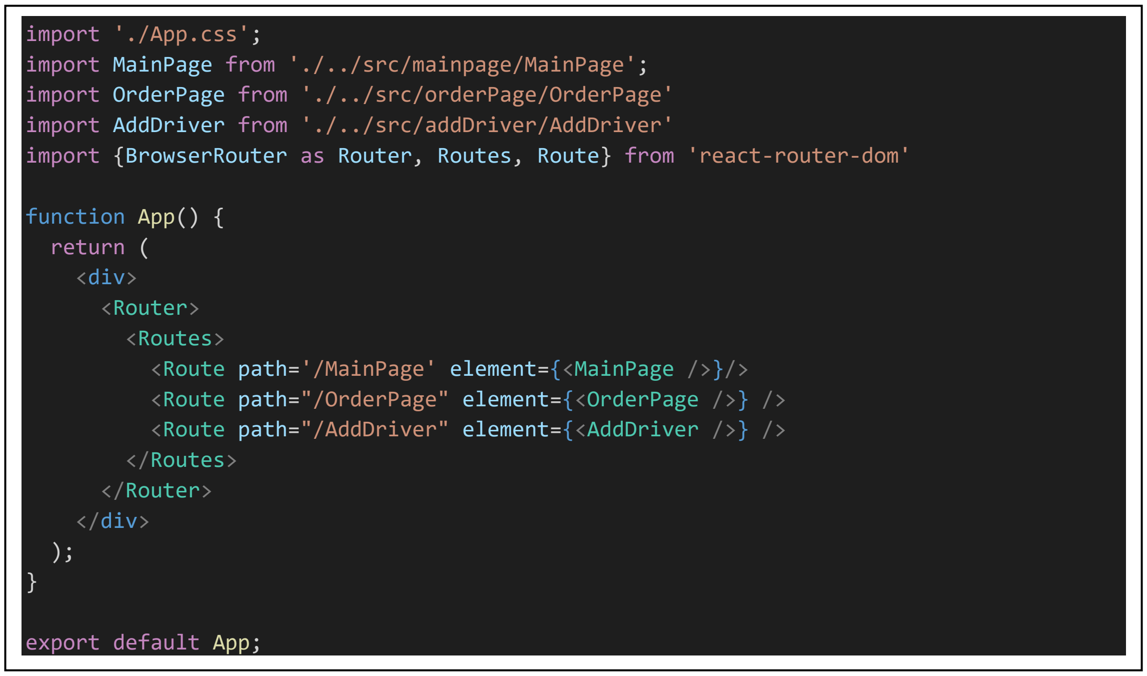
1) Index.js component: this file used for render the application using react hook called react DOM.



```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import Home from './home/Home'
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Home />
    <App />
  </React.StrictMode>
);
```

source code 5 display the index.js component(own work).

2) App.js component : this file used as a router, by this file we can route and access for all others pages in our web application.

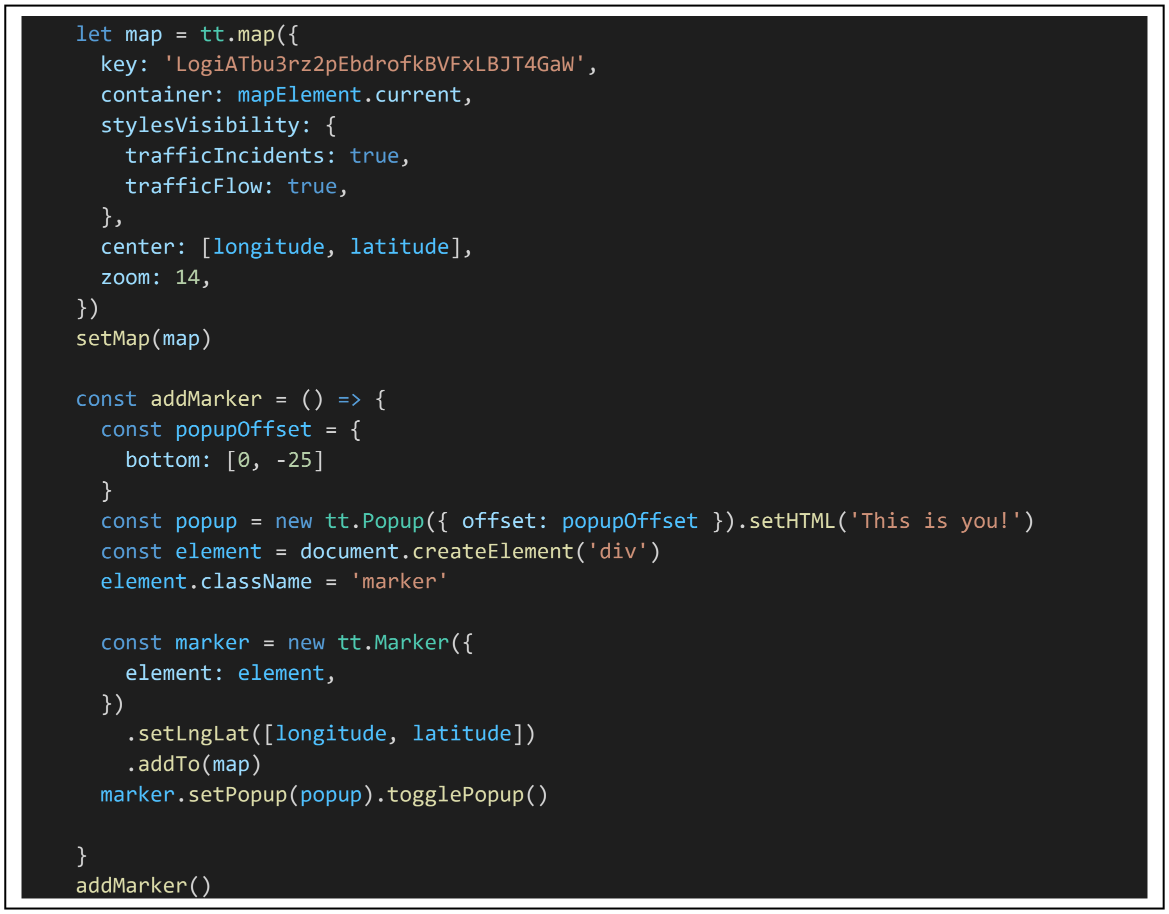


```
import './App.css';
import MainPage from './../src/mainpage/MainPage';
import OrderPage from './../src/orderPage/OrderPage'
import AddDriver from './../src/addDriver/AddDriver'
import {BrowserRouter as Router, Routes, Route} from 'react-router-dom'

function App() {
  return (
    <div>
      <Router>
        <Routes>
          <Route path='/MainPage' element={<MainPage />}/>
          <Route path="/OrderPage" element={<OrderPage />} />
          <Route path="/AddDriver" element={<AddDriver />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```

source code 6 display the App.js component(own work).

3) RoutePage Component: this component contains almost the all function we need in this application. By this page we can add maps, delivery markers, drawRoute line, add Location or addresses, recalculate short path, and sortDestinations.

Add Map function: this function calls the tomtom API library and imports it inside the code page; after that, we can call a function called map and insert all parameters we need. Also, we can call a function called addMarker to add a marker for the origin location.



```
let map = tt.map({
  key: 'LogiATbu3rz2pEbdrofkBVFxLBJT4GaW',
  container: mapElement.current,
  stylesVisibility: {
    trafficIncidents: true,
    trafficFlow: true,
  },
  center: [longitude, latitude],
  zoom: 14,
})
setMap(map)

const addMarker = () => {
  const popupOffset = {
    bottom: [0, -25]
  }
  const popup = new tt.Popup({ offset: popupOffset }).setHTML('This is you!')
  const element = document.createElement('div')
  element.className = 'marker'

  const marker = new tt.Marker({
    element: element,
  })
    .setLngLat([longitude, latitude])
    .addTo(map)
  marker.setPopup(popup).togglePopup()

}
addMarker()
```

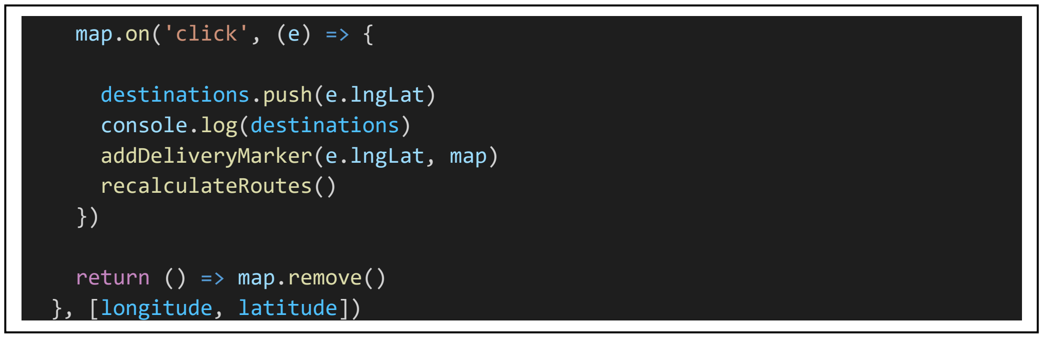
source code 7 display how to add map to our code(own work).

By this function, we can add a map with specific variables:

- TT.map is an object containing parameters for the map; here, we added traffic incidents and traffic flow, center, and zoom.
- AddMarker: function to add markers for a driver. In this way, the driver can know his location on the map.
- Popup: function to add text above driver marker.
- SetLngLat: by this function, we can add the location or address for a driver. This function takes two variables [longitude and latitude].

Map.on click function: Using TomTom API, we can add points or locations to the map by three methods. In our application, we used the click on map function to add points for our drivers. In this function, we called all the following functions:

- Add the longitude and latitude to an array called destinations.
- AddDeliveryMarker to send the longitude and latitude to add these points on the map.
- We called recalculateRoutes to make a new route between all points we added.

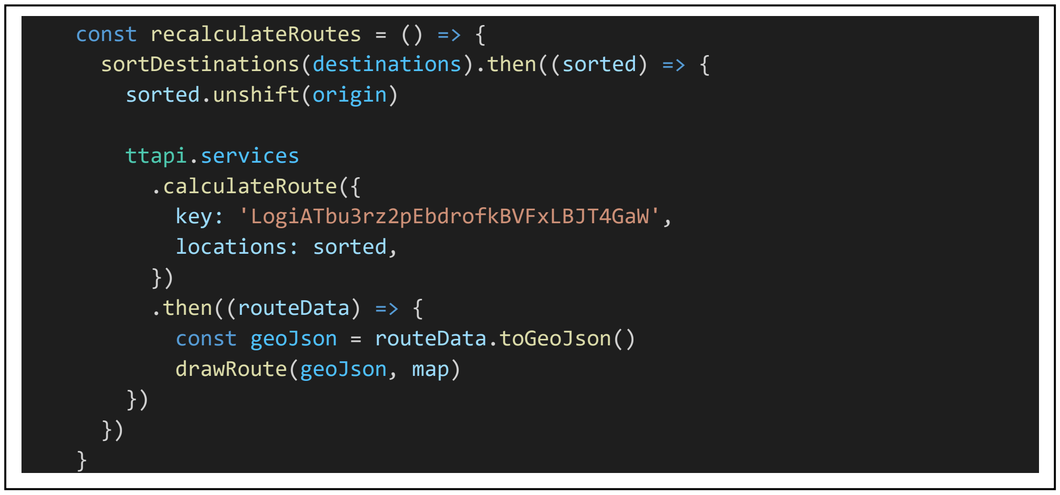


```
    map.on('click', (e) => {

      destinations.push(e.lngLat)
      console.log(destinations)
      addDeliveryMarker(e.lngLat, map)
      recalculateRoutes()
    })

    return () => map.remove()
  }, [longitude, latitude])
```

source code 8 display how to add longitude and latitude(own work).

RecalculateRoutes function. This function is used to recalculate the time cost between all nodes that we added. We also used TomTom API services to add all parameters we needed.

- We called the sortDestinations function and gave it a destination array variable that includes all our destinations. Then ask for sorting.
- TT.map API. Services: An object containing a key to establish route services and locations.
- I used routeData function to add geoJson to map.

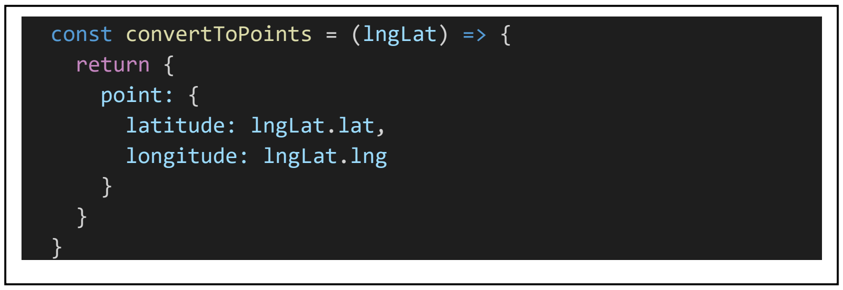


```
    const recalculateRoutes = () => {
      sortDestinations(destinations).then((sorted) => {
        sorted.unshift(origin)

        ttapi.services
          .calculateRoute({
            key: 'LogiATbu3rz2pEbdrofkBVFxLBJT4GaW',
            locations: sorted,
          })
          .then((routeData) => {
            const geoJson = routeData.toGeoJson()
            drawRoute(geoJson, map)
        })
      })
    }
```

source code 9 display recalculateRoutes function(own work).

SortDestinations function is the most essential function in our application.

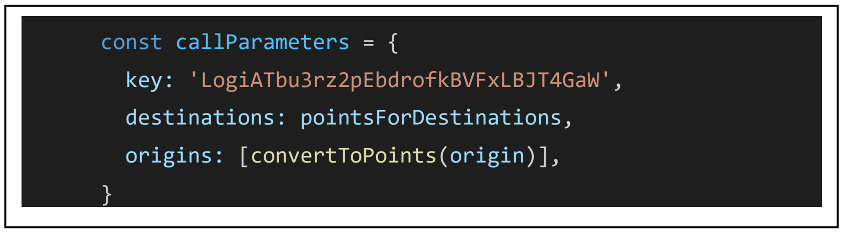
- First, we called the pointsForDestinations function, this function called function defined by the tomtom API called location.map. We passed destinations. PointsForDestinations function returns a function called convertToPoints. We passed the destination to this function. ConvertToPoints function takes the lngLat as input, then convert it to an object called points containing two parameters [latitude, longitude].



```
const convertToPoints = (lngLat) => {
  return {
    point: {
      latitude: lngLat.lat,
      longitude: lngLat.lng
    }
  }
}
```

source code 10 represent convertToPoints function(own work).

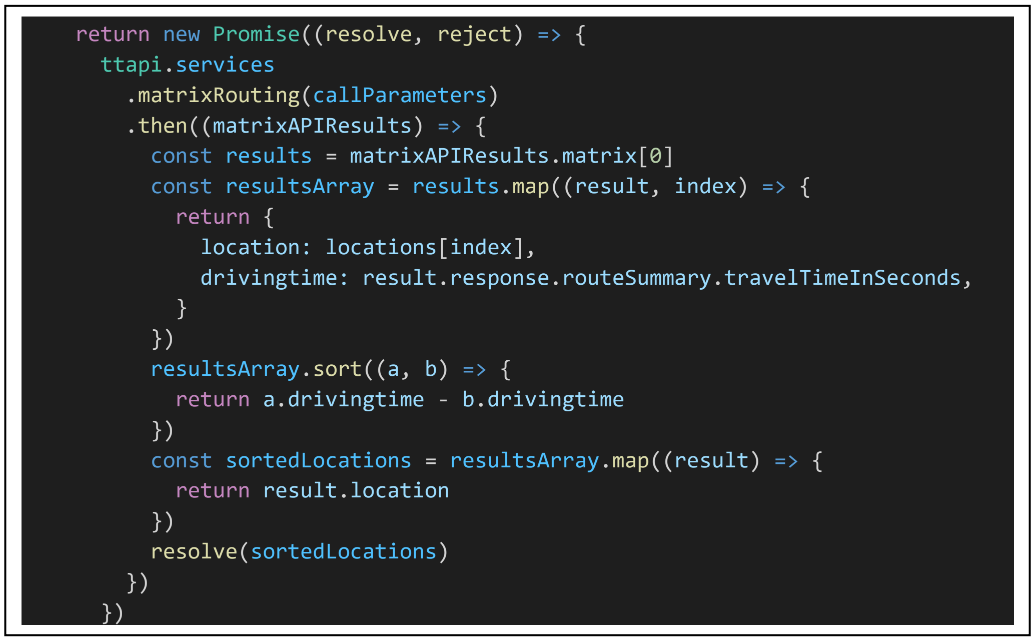
- Second, we define a variable called callParameters with an object type; this variable contains the following parameters.



```
    const callParameters = {
      key: 'LogiATbu3rz2pEbdrofkBVFxLBJT4GaW',
      destinations: pointsForDestinations,
      origins: [convertToPoints(origin)],
    }
```

source code 11 represent callParameters function(own work).

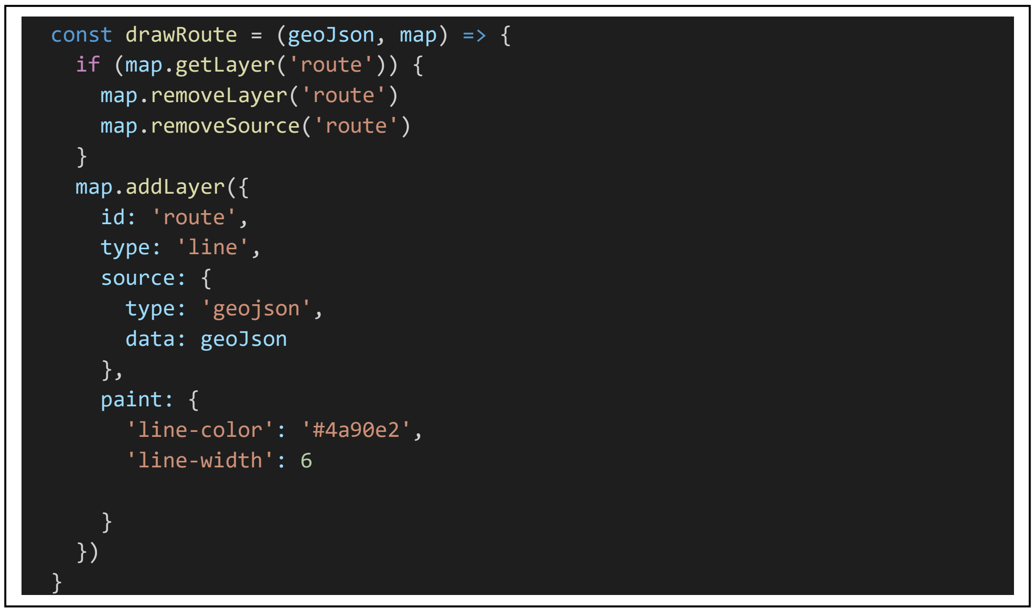
- Finally, after we defined all parameters and variable that we need, now we can return new Promise using tomtom api services with two inputs called resolve and reject.
    - matrixRouting function:
        - The Matrix Routing service enables the calculation of a matrix of route summaries for a set of routes defined with origin and destination locations.
        - For every given origin, this service calculates the cost of routing from that origin to every given destination.
        - The set of origins and the set of destinations can be thought of as the column and row headers of a table, while each cell in the table contains the costs of routing from the origin to the destination for that cell.
        - The following costs are computed for each route:
            - Travel times
            - Distances



```
    return new Promise((resolve, reject) => {
      ttapi.services
        .matrixRouting(callParameters)
        .then((matrixAPIResults) => {
          const results = matrixAPIResults.matrix[0]
          const resultsArray = results.map((result, index) => {
            return {
              location: locations[index],
              drivingtime: result.response.routeSummary.travelTimeInSeconds,
            }
          })
          resultsArray.sort((a, b) => {
            return a.drivingtime - b.drivingtime
          })
          const sortedLocations = resultsArray.map((result) => {
            return result.location
          })
          resolve(sortedLocations)
        })
      })
```

source code 12 represent matrixRouting function(own work).

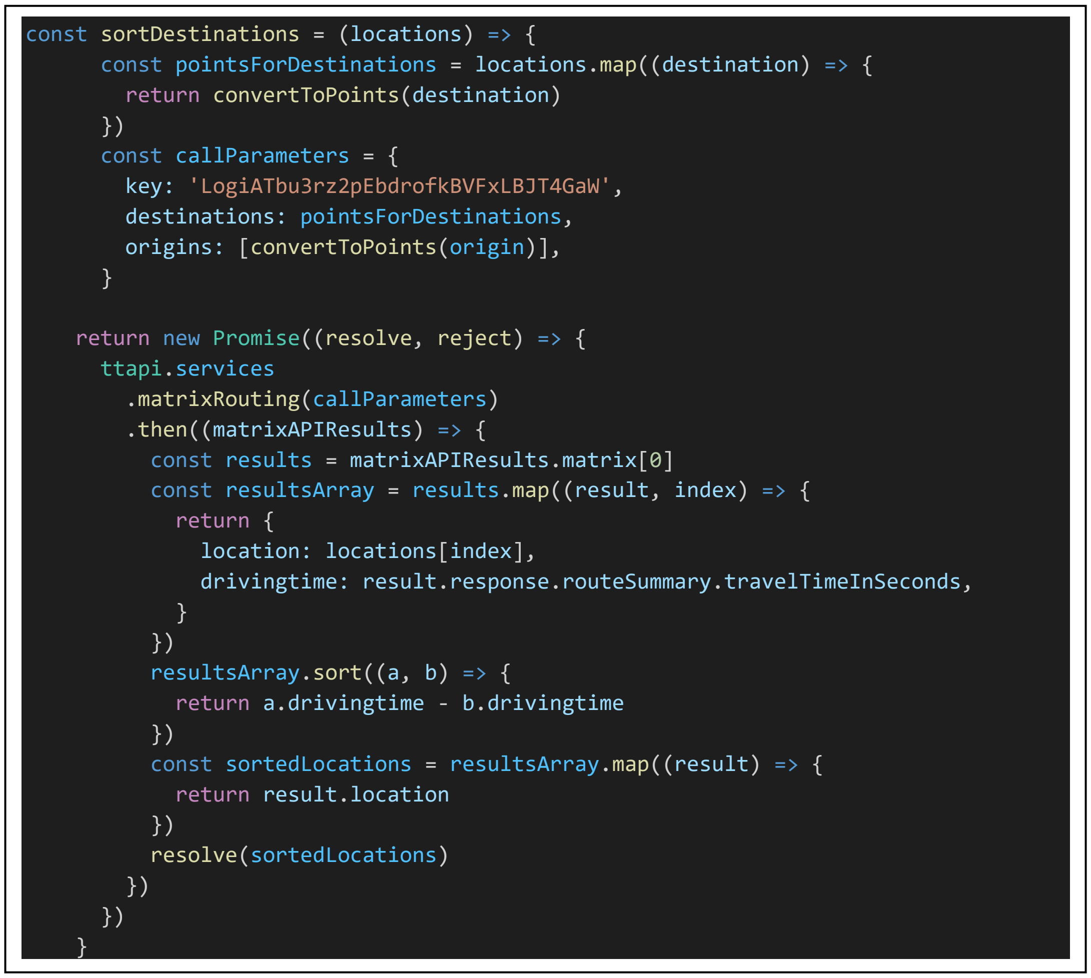
- DrawRoute function: this function used to add layer or lines between all node, we need to draw a path for drivers then they can know the driver route planner route for each.



```
  const drawRoute = (geoJson, map) => {
    if (map.getLayer('route')) {
      map.removeLayer('route')
      map.removeSource('route')
    }
    map.addLayer({
      id: 'route',
      type: 'line',
      source: {
        type: 'geojson',
        data: geoJson
      },
      paint: {
        'line-color': '#4a90e2',
        'line-width': 6

      }
    })
  }
```

source code 13 represent drawRoute function(own work).

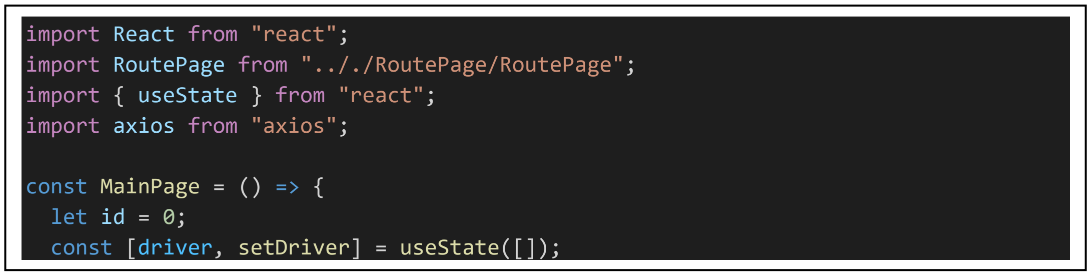
- In the end, we combine all functions in one function called sortDestinations.



```
const sortDestinations = (locations) => {
      const pointsForDestinations = locations.map((destination) => {
        return convertToPoints(destination)
      })
      const callParameters = {
        key: 'LogiATbu3rz2pEbdrofkBVFxLBJT4GaW',
        destinations: pointsForDestinations,
        origins: [convertToPoints(origin)],
      }

    return new Promise((resolve, reject) => {
      ttapi.services
        .matrixRouting(callParameters)
        .then((matrixAPIResults) => {
          const results = matrixAPIResults.matrix[0]
          const resultsArray = results.map((result, index) => {
            return {
              location: locations[index],
              drivingtime: result.response.routeSummary.travelTimeInSeconds,
            }
          })
          resultsArray.sort((a, b) => {
            return a.drivingtime - b.drivingtime
          })
          const sortedLocations = resultsArray.map((result) => {
            return result.location
          })
          resolve(sortedLocations)
        })
      })
    }
```

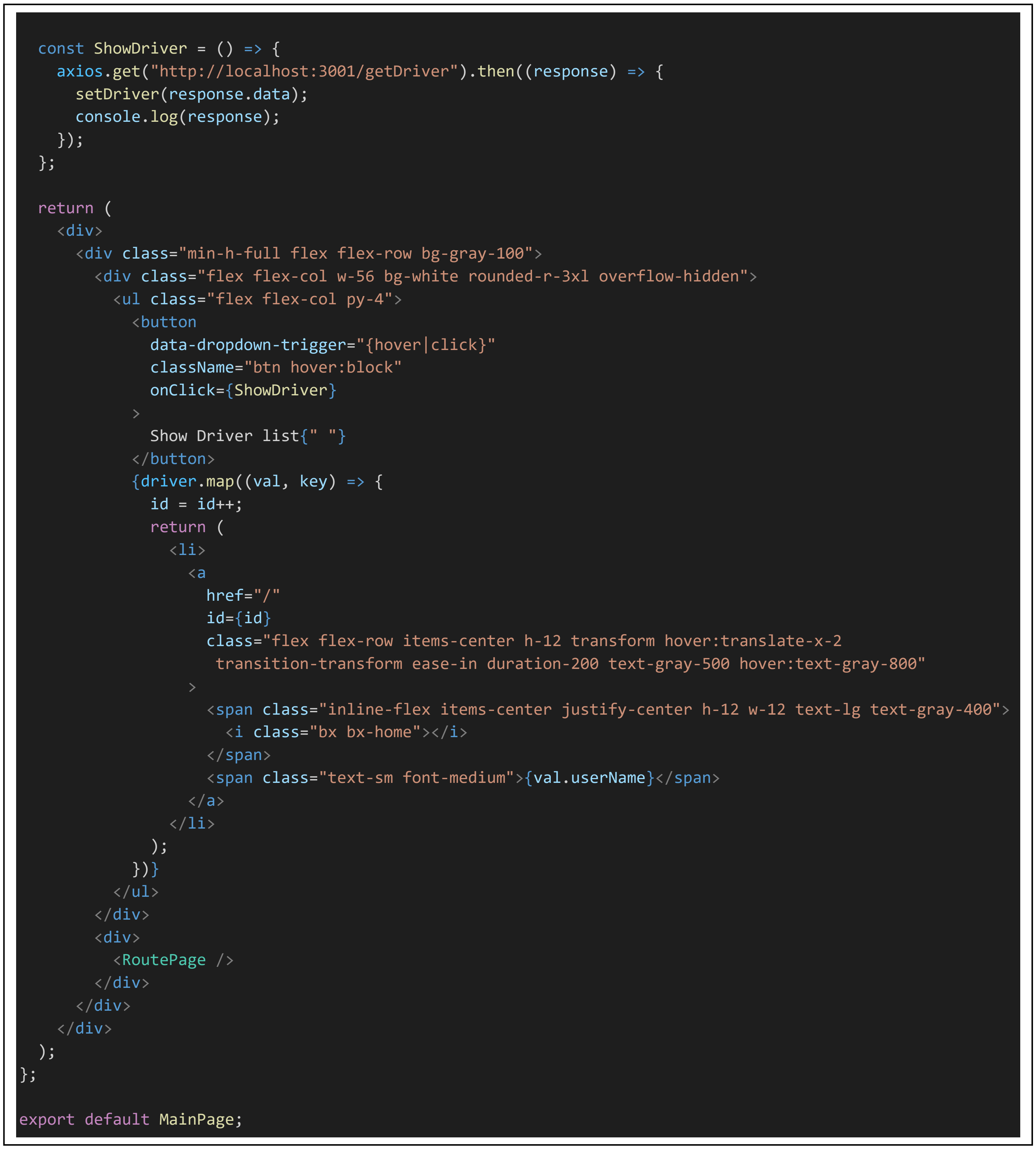
source code 14 represent sortDestinations function(own work).

- MainPage component: In this component, we displayed all the following components:
  - Main menu: contains all pages that the admin can access, like add driver page, order list page,
  - Sidebar: contains all drivers we have; this sidebar gives access to admin to display all driver maps and reach every driver page.
  - Map: This page shows the main map containing all points that need to make a route for the driver.



```
import React from "react";
import RoutePage from "../../RoutePage/RoutePage";
import { useState } from "react";
import axios from "axios";

const MainPage = () => {
  let id = 0;
  const [driver, setDriver] = useState([]);
```

source code 15 represent part 1 of MainPage component(own work).



```
  const ShowDriver = () => {
    axios.get("http://localhost:3001/getDriver").then((response) => {
      setDriver(response.data);
      console.log(response);
    });
  };

  return (
    <div>
      <div class="min-h-full flex flex-row bg-gray-100">
        <div class="flex flex-col w-56 bg-white rounded-r-3xl overflow-hidden">
          <ul class="flex flex-col py-4">
            <button
              data-dropdown-trigger="{hover|click}"
              className="btn hover:block"
              onClick={ShowDriver}
            >
              Show Driver list{" "}
            </button>
            {driver.map((val, key) => {
              id = id++;
              return (
                <li>
                  <a
                    href="/"
                    id={id}
                    class="flex flex-row items-center h-12 transform hover:translate-x-2
                     transition-transform ease-in duration-200 text-gray-500 hover:text-gray-800"
                  >
                    <span class="inline-flex items-center justify-center h-12 w-12 text-lg text-gray-400">
                      <i class="bx bx-home"></i>
                    </span>
                    <span class="text-sm font-medium">{val.userName}</span>
                  </a>
                </li>
              );
            })}
          </ul>
        </div>
        <div>
          <RoutePage />
        </div>
      </div>
    </div>
  );
};

export default MainPage;
```

source code 16 represent part 2 of MainPage component(own work).

- AddDriver component: we used three methods in this component
    - Tailwindcss library to create a form that includes all input information for the driver.
    - Axios library to make the connection between the interface page and the database.
    - Database to store all driver information in the database.

```
import React from "react";
import { useState } from "react";
import axios from "axios";
```

source code 17 represent Import react, useState, and axios(own work).

```
const AddDriver = () => {
  const [userName, setUserName] =useState("");
  const [password, setPassword] = useState("");
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [carType, setCarType] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [address, setAddress] = useState("");
  const [city, setCity] = useState("");
```

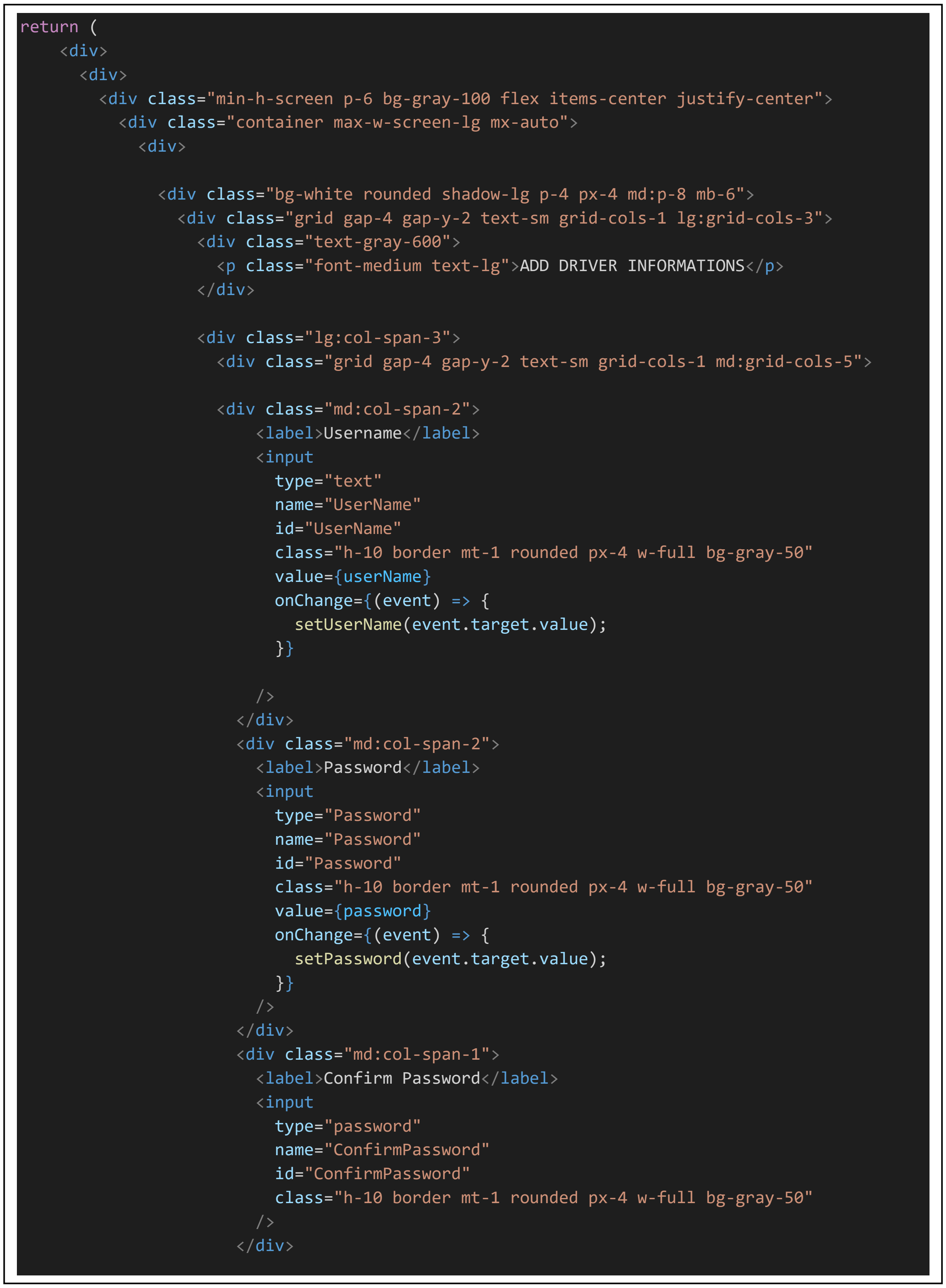
source code 18 for Defined all inputs as a variable using useState hook to save all data and changes(own work).

```
  const DriverAdd = () => {
    axios
      .post("http://localhost:3001/AddDriver", {
        userName : userName,
        password: password,
        firstName: firstName,
        lastName: lastName,
        carType : carType,
        email: email,
        phone: phone,
        address: address,
        city: city,
      })
      .then(() => {
        console.log("success");
      });
  };
```

source code 19 for using axios post method to send all data that we inserted to database url called

locathost:3001/AddDriver(own work).

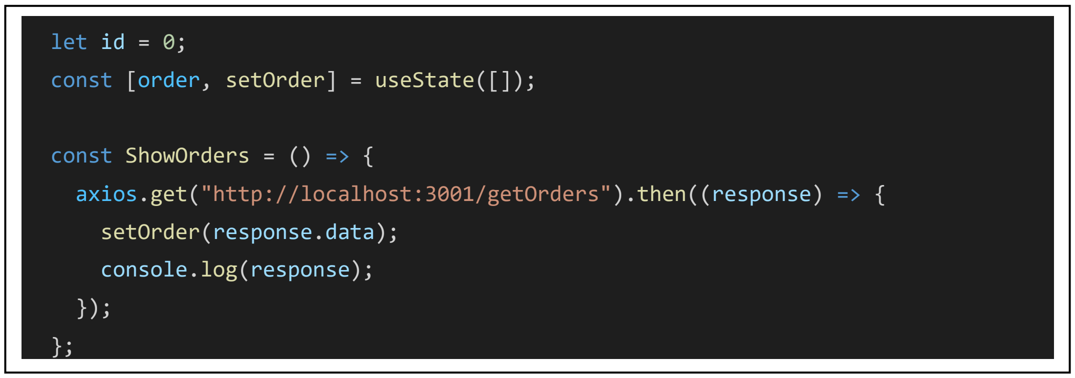
The following code contains <divs> designing by tailwind CSS to display driver information form. This form contains Username, Password, Confirm Password, FIRST NAME, Last NAME, Car Type, Email Address, Phone Number, Address / Street, and City.



```
return (
    <div>
      <div>
        <div class="min-h-screen p-6 bg-gray-100 flex items-center justify-center">
          <div class="container max-w-screen-lg mx-auto">
            <div>

              <div class="bg-white rounded shadow-lg p-4 px-4 md:p-8 mb-6">
                <div class="grid gap-4 gap-y-2 text-sm grid-cols-1 lg:grid-cols-3">
                  <div class="text-gray-600">
                    <p class="font-medium text-lg">ADD DRIVER INFORMATIONS</p>
                  </div>

                  <div class="lg:col-span-3">
                    <div class="grid gap-4 gap-y-2 text-sm grid-cols-1 md:grid-cols-5">

                    <div class="md:col-span-2">
                        <label>Username</label>
                        <input
                          type="text"
                          name="UserName"
                          id="UserName"
                          class="h-10 border mt-1 rounded px-4 w-full bg-gray-50"
                          value={userName}
                          onChange={(event) => {
                            setUserName(event.target.value);
                          }}

                        />
                      </div>
                      <div class="md:col-span-2">
                        <label>Password</label>
                        <input
                          type="Password"
                          name="Password"
                          id="Password"
                          class="h-10 border mt-1 rounded px-4 w-full bg-gray-50"
                          value={password}
                          onChange={(event) => {
                            setPassword(event.target.value);
                          }}
                        />
                      </div>
                      <div class="md:col-span-1">
                        <label>Confirm Password</label>
                        <input
                          type="password"
                          name="ConfirmPassword"
                          id="ConfirmPassword"
                          class="h-10 border mt-1 rounded px-4 w-full bg-gray-50"
                        />
                      </div>
```

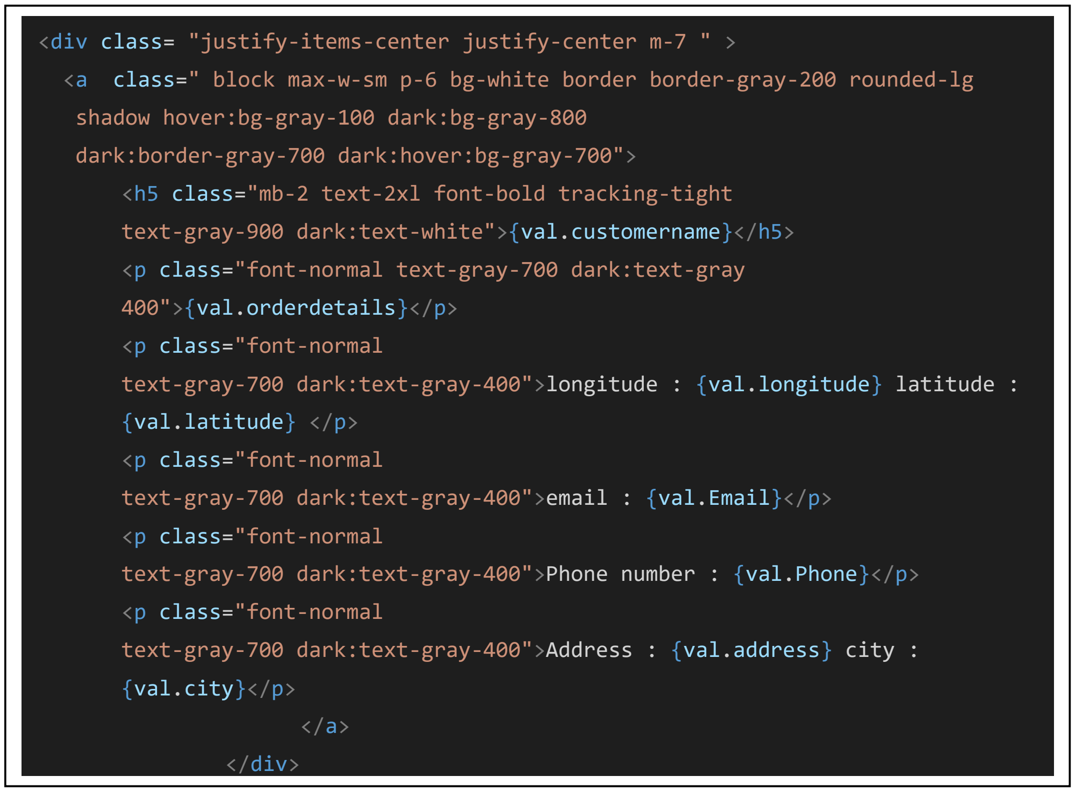
source code 20 represent html and tailwindcss for driver information form(own work).

- OrderList Page: this page created to display all orders in our database. It a simple component contain function called showOrders that get data from table called order in the database using axios library.



```
let id = 0;
const [order, setOrder] = useState([]);

const ShowOrders = () => {
  axios.get("http://localhost:3001/getOrders").then((response) => {
    setOrder(response.data);
    console.log(response);
  });
};
```

source code 21 for using axios get method to get orders data that we inserted to the database(own work).



```
<div class= "justify-items-center justify-center m-7 " >
  <a  class=" block max-w-sm p-6 bg-white border border-gray-200 rounded-lg
   shadow hover:bg-gray-100 dark:bg-gray-800
   dark:border-gray-700 dark:hover:bg-gray-700">
       <h5 class="mb-2 text-2xl font-bold tracking-tight
       text-gray-900 dark:text-white">{val.customername}</h5>
       <p class="font-normal text-gray-700 dark:text-gray
       400">{val.orderdetails}</p>
       <p class="font-normal
       text-gray-700 dark:text-gray-400">longitude : {val.longitude} latitude :
       {val.latitude} </p>
       <p class="font-normal
       text-gray-700 dark:text-gray-400">email : {val.Email}</p>
       <p class="font-normal
       text-gray-700 dark:text-gray-400">Phone number : {val.Phone}</p>
       <p class="font-normal
       text-gray-700 dark:text-gray-400">Address : {val.address} city :
       {val.city}</p>
                     </a>
               </div>
```

source code 22 display orders data that get from the database(own work).

- We used the .map function, this function for variables with an array type, allows us to reach all data inside this array. That function looks like for loop. We used this function to catch all orders data and display it on web application using <divs> with tailwindCSS styiling.

5.3 Server side

Web browsers use the Hypertext Transfer Protocol (HTTP) to communicate with web servers. Whenever a link is clicked, a form is submitted, or a search is performed, the browser sends an HTTP request to the target server (Mancas, 2015).

A straightforward architecture for a dynamic website is illustrated in the diagram below. Like in the previous diagram, browsers send HTTP requests to the server, which then processes the requests and returns appropriate HTTP responses.

Requests for static resources, such as CSS, JavaScript, images, and pre-created PDF files, are handled similarly to static sites, as they are files that do not change (Deshpande, 2021).

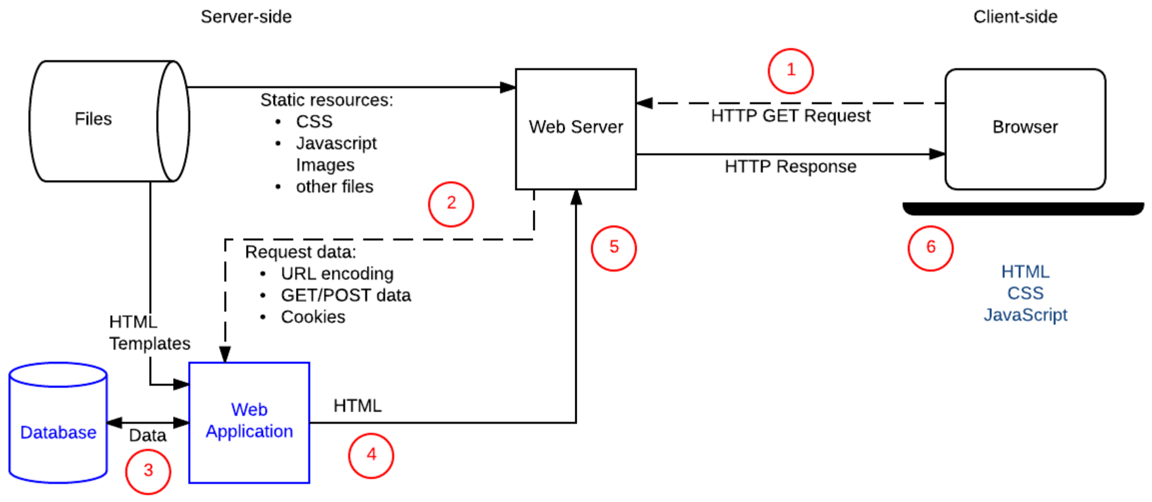


Figure 25 shows relationships between browser, server, and database (Mancas, 2015).

In our application, we need to send data to the database and also request data from the database. To do that, we need to create a connection between React files or components and MySQL workbench.
We used all of the expresses, require, Cors, and MySQL libraries for React to connect the database and our files.

- Require function: The function is intended to add separate pieces of code ("modules") to the current scope, a feature not part of the JavaScript/ECMA Script language until the ES2015 specification.
- Cors: Cross-Origin Resource Sharing is a mechanism that allows services to communicate with each other. You must have encountered the infamous error that bothered almost everyone dabbling with web development.
- Express is a back-end web application framework of node js.
- MySQL: Functions allow us to enhance the capabilities of MySQL.

Index.js file: this file contains all the following function and library below:

- Configuration express, MySQL, and Cors functions.

```
const express = require("express");
const app = express();
const mysql = require("mysql");
const cors = require("cors");
app.use(cors());
app.use(express.json());
```

source code 23 display configuration of express, mysql, and cors library(own work).

- Create connection to the database by MySQL function.

```
const db = mysql.createConnection({
  user: "root",
  host: "localhost",
  password: "1234",
  database: "thesis_database",
});
```

source code 24 display configuration connection to the database(own work).

- Add driver function, after clicked on submit button to insert driver information by interface form in client-server, we used app.post require function to catch all information and save it in variables.

```
app.post("/AddDriver", (req, res) => {
  const userName = req.body.userName;
  const password = req.body.password;
  const firstName = req.body.firstName;
  const lastName = req.body.lastName;
  const carType = req.body.carType;
  const email = req.body.email;
  const phone = req.body.phone;
  const address = req.body.address;
  const city = req.body.city;
```

source code 25 represent store data came from inteface front-end to the database(own work).

- After saved all data in variables, we called next function to insert these data to database.

```
  db.query(
    "INSERT INTO drivers (userName , password , firstName , lastName , carType ,
email, phone,address,city ) VALUES (?,?,?,?,?,?,?,?,?)",
    [userName,password,firstName, lastName,carType,email,phone, address,city],
     (err,result) => {
        if(err){
            console.log(err);
        }
        else {
            res.send("driver inserted");
        }
```

source code 26 represent sending data to database(own work).

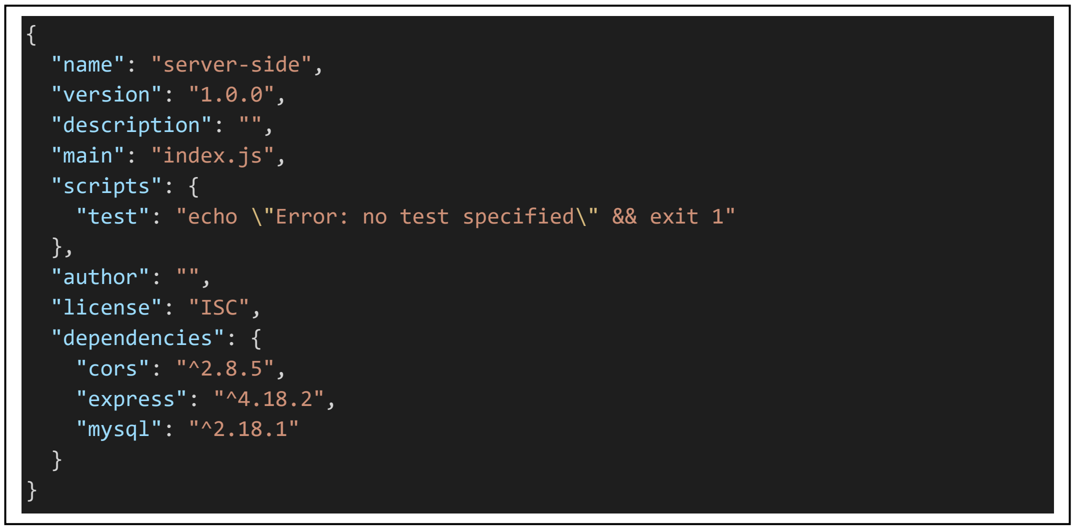
- In the following code, we represent how the system create driver information form and how catch the information from inputs, then save it into variables that sent it to database using http requset created by axios library. We used aslo html with tailwindcss to design the driver information form that dispalyed on the interface website or web application.

```
import React from "react";
import { useState } from "react";
import axios from "axios";

const AddDriver = () => {
  const [userName, setUserName] =useState("");
  const [password, setPassword] = useState("");
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [carType, setCarType] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [address, setAddress] = useState("");
  const [city, setCity] = useState("");
```

```
  const DriverAdd = () => {
    axios
      .post("http://localhost:3001/AddDriver", {
        userName : userName,
        password: password,
        firstName: firstName,
        lastName: lastName,
        carType : carType,
        email: email,
        phone: phone,
        address: address,
        city: city,
      })
      .then(() => {
        console.log("success");
      });
  };

  return (
    <div>
      <div>
        <div class="min-h-screen p-6 bg-gray-100 flex items-center justify-center">
          <div class="container max-w-screen-lg mx-auto">
            <div>

              <div class="bg-white rounded shadow-lg p-4 px-4 md:p-8 mb-6">
                <div class="grid gap-4 gap-y-2 text-sm grid-cols-1 lg:grid-cols-3">
                  <div class="text-gray-600">
                    <p class="font-medium text-lg">ADD DRIVER INFORMATIONS</p>
                  </div>
```

source code 27 represent driver information form(own work).

- Package.json: package.json holds important information about the project. It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number and a list of dependencies required by the application.



```
{
  "name": "server-side",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mysql": "^2.18.1"
  }
}
```

source code 28 represent package.json file(own work).

5.4 Database

Databases are organized collections of structured information, or data, stored electronically by a computer system. A database management system (DBMS) usually controls a database. The data and the Database Management Systems, along with the associated applications, are called database systems. It is often shortened to just a database.

The prevalent database systems of today typically model data in tables organized into rows and columns. This structure enhances the efficiency of processing and querying data, making it simple to access, manage, modify, update, control, and organize data. Most databases employ Structured Query Language (SQL) for writing and querying data.

5.4.1 MySQL Reference Manual

The documentation for MySQL versions 8.0 through 8.0.34 and NDB Cluster, based on NDB versions 8.0 through 8.0.33-ndb-8.0.33, is available in the MySQL Reference Manual. It might also include documentation for yet-to-be-released MySQL versions' features. Please consult the MySQL 8.0 Release Notes to learn which releases have been made.

The handbook discusses MySQL 8.0 features that, depending on the license agreement, might only be present in a special edition of MySQL 8.0. If you have any questions about the features offered by your edition of MySQL 8.0, please consult your licensing agreement for MySQL 8.0 or contact an Oracle sales representative (Oracle, 2023).

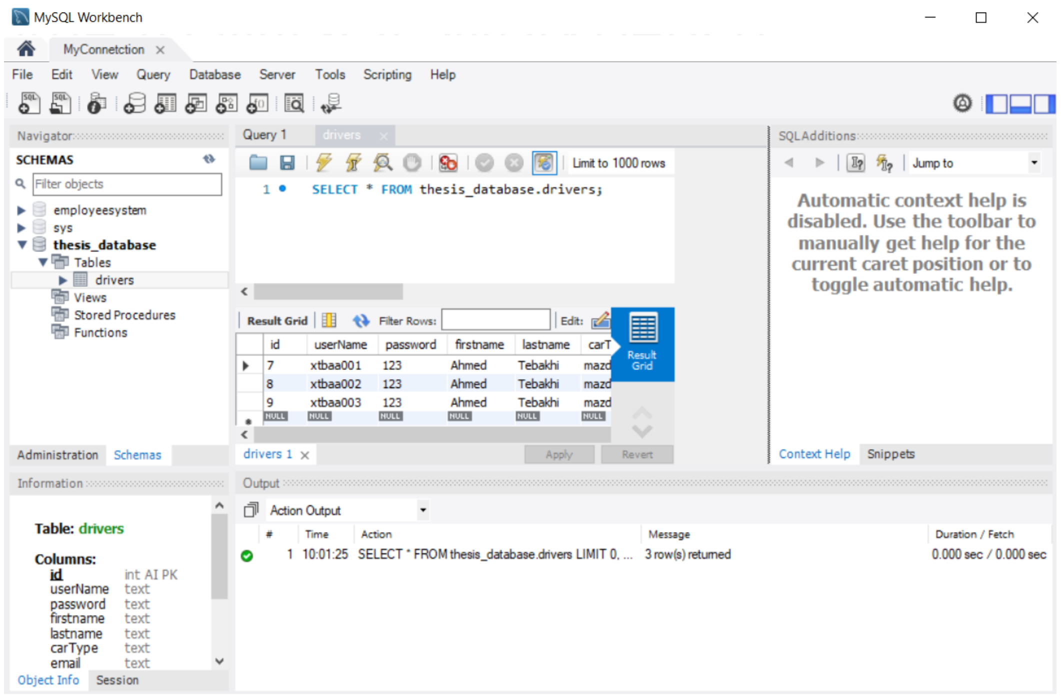
5.4.2 MySQL Workbench

DBAs, database architects, and developers can use MySQL Workbench as a unified visual tool. Besides data modeling and SQL development, MySQL Workbench offers comprehensive administrative tools for server configuration, user management, and backup. Besides Windows and Linux, MySQL Workbench is also available for Mac OS X.

- An administrator, developer, or data architect can design, model, create, manage, and generate databases visually using MySQL Workbench. Additionally, it provides tools for performing complex change management and documentation tasks that typically require much time and effort for a data modeler.

- You can create, execute, and optimize SQL queries visually using MySQL Workbench. Color syntax highlighting, auto-completion, snippet reuse, and execution history are all included in the SQL Editor. MySQL Fabric connections can be managed easily via the Database Connections Panel. Object Browsers provide instant access to database schema and objects.

- You can quickly gain better database visibility with MySQL Workbench, a visual console for managing MySQL environments. Developers use tools to configure servers, administer users, perform backup and recovery, inspect audit data, and view database health.



```
SELECT * FROM thesis_database.drivers;
```

id	userName	password	firstname	lastname	carT
7	xtbaa001	123	Ahmed	Tebakhi	mazd
8	xtbaa002	123	Ahmed	Tebakhi	mazd
9	xtbaa003	123	Ahmed	Tebakhi	mazd
NULL	NULL	NULL	NULL	NULL	NULL

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: drivers

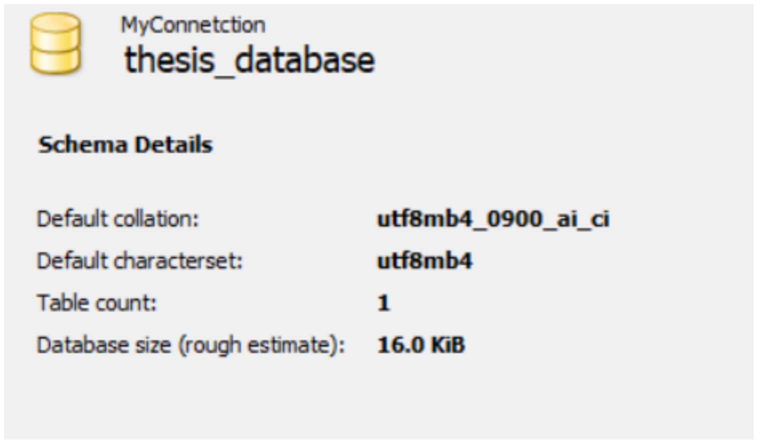
Columns:

id	int AI PK
userName	text
password	text
firstname	text
lastname	text
carType	text
email	text

```
#  Time      Action                                        Message             Duration / Fetch
1  10:01:25  SELECT * FROM thesis_database.drivers LIMIT 0, ...  3 row(s) returned   0.000 sec / 0.000 sec
```

Figure 26 shows MySQL Workbench software(own work).

Using MySQL Workbench, We created a database called thesis_database that contains all tables and data we need to store and make operations on it.



MyConnetction
thesis_database

Schema Details

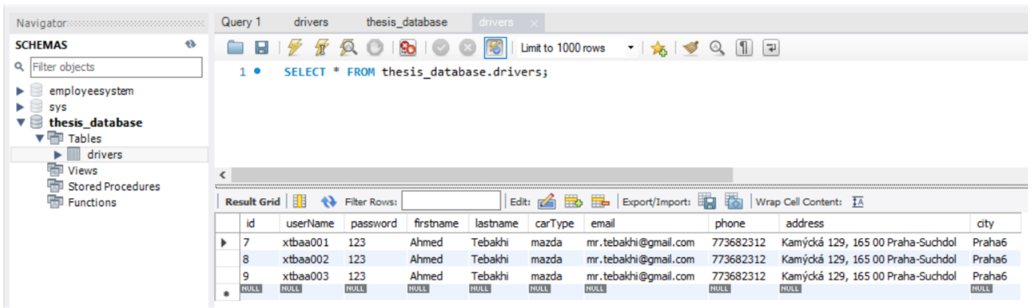
Default collation:	utf8mb4_0900_ai_ci
Default characterset:	utf8mb4
Table count:	1
Database size (rough estimate):	16.0 KiB

Figure 27 creation database called thesis_database(own work).

- Drivers table: After creating the database, we added a specific table to store driver information from the client side. This information includes username, password, first name, last name, car type, email, phone, address, and city.

```
CREATE TABLE `drivers` (
  `id` int NOT NULL AUTO_INCREMENT,
  `userName` text NOT NULL,
  `password` text aNOT NULL,
  `firstname` text NOT NULL,
  `lastname` text NOT NULL,
  `carType` text NOT NULL,
  `email` text NOT NULL,
  `phone` int NOT NULL,
  `address` text NOT NULL,
  `city` text NOT NULL,
  `usertype` char(20) DEFAULT 'driver',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

source code 29 represent creation table called Driver(own work).



```
SELECT * FROM thesis_database.drivers;
```

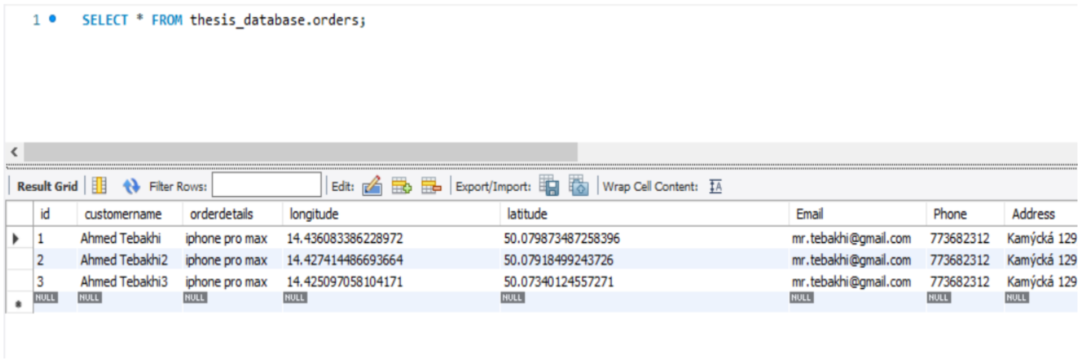
id	userName	password	firstname	lastname	carType	email	phone	address	city
7	xtbaa001	123	Ahmed	Tebakhi	mazda	mr.tebakhi@gmail.com	773682312	Kamýcká 129, 165 00 Praha-Suchdol	Praha6
8	xtbaa002	123	Ahmed	Tebakhi	mazda	mr.tebakhi@gmail.com	773682312	Kamýcká 129, 165 00 Praha-Suchdol	Praha6
9	xtbaa003	123	Ahmed	Tebakhi	mazda	mr.tebakhi@gmail.com	773682312	Kamýcká 129, 165 00 Praha-Suchdol	Praha6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 28 display data in the driver table(own work).

- Order table: After creating the database, we added a specific table to store orders information from the client side. This information includes customer name, order details, longitude, latitude, car type, email, phone, address, and city.

```
CREATE TABLE `orders` (
  `id` int NOT NULL AUTO_INCREMENT,
  `customername` text NOT NULL,
  `orderdetails` text NOT NULL,
  `longitude` double NOT NULL,
  `latitude` double NOT NULL,
  `Email` text NOT NULL,
  `Phone` text NOT NULL,
  `Address` text NOT NULL,
  `city` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

source code 30 represent creation table called Order(own work).



```
SELECT * FROM thesis_database.orders;
```

	id	customername	orderdetails	longitude	latitude	Email	Phone	Address
▶	1	Ahmed Tebakhi	iphone pro max	14.436083386228972	50.079873487258396	mr.tebakhi@gmail.com	773682312	Kamýcká 129
	2	Ahmed Tebakhi2	iphone pro max	14.427414486693664	50.07918499243726	mr.tebakhi@gmail.com	773682312	Kamýcká 129
	3	Ahmed Tebakhi3	iphone pro max	14.425097058104171	50.07340124557271	mr.tebakhi@gmail.com	773682312	Kamýcká 129
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 29 display data in the order table(own work).

- administrations table: After creating the database, we added a specific table to store user of admin type information. This information includes admin name, email, and password.

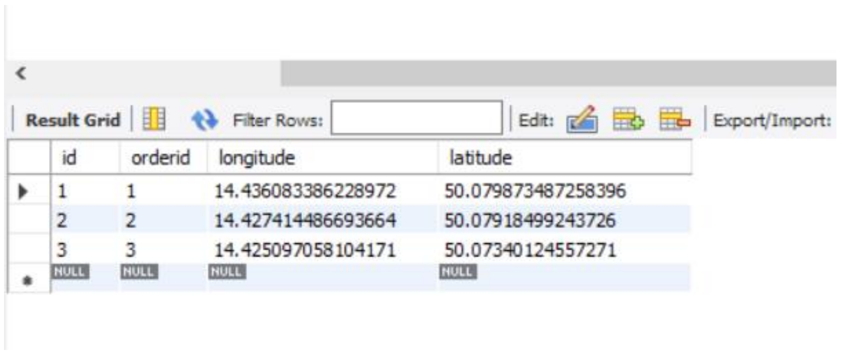
```
CREATE TABLE `admins` (
  `id` int NOT NULL AUTO_INCREMENT,
  `adminname` text NOT NULL,
  `email` text NOT NULL,
  `password` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

source code 31 represent creation table called admins(own work).

- Locations table: After creating the database, we added a location table. In this table, we defined one keys: foreign called order id. Moreover, one primary key is called the id. In this table, we make a relationship between orders, and location tables. This table gets the longitude and latitude from orders table. The value must exist in the other table for order id. Otherwise, it is not going to be able to add data.

```
CREATE TABLE `locations` (
  `id` int NOT NULL AUTO_INCREMENT,
  `orderid` int NOT NULL,
  `longitude` double NOT NULL,
  `latitude` double NOT NULL,
  PRIMARY KEY (`id`),
  KEY `id_idx` (`orderid`,`longitude`,`latitude`),
  CONSTRAINT `id` FOREIGN KEY (`orderid`) REFERENCES `orders` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

source code 32 represent creation table called locations(own work).



id	orderid	longitude	latitude
1	1	14.436083386228972	50.079873487258396
2	2	14.427414486693664	50.07918499243726
3	3	14.425097058104171	50.07340124557271
NULL	NULL	NULL	NULL

Figure 30 display data of locations table(own work).

- Routing table: After creating the database, we added a location table. In this table, we defined three keys: foreign, called “oid“ for order id, “did“ for driver id and “ lid“ for location id. Moreover, one primary key is called the id. In this table, we make a relationship between orders, drivers, and location tables. The value must exist in the other table for order id, driver id, and location id. Otherwise, it will not be able to add data. by this table; we can access all other columns in the other table; For example, if we want to get the driver name, we can use a foreign key called “did “.

```
CREATE TABLE `routing` (
  `id` int NOT NULL AUTO_INCREMENT,
  `locations` int NOT NULL,
  `driver` int NOT NULL,
  `order` int NOT NULL,
  `deliverytime` double DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `id_idx` (`driver`),
  KEY `id_idx1` (`order`),
  KEY `id_idx2` (`locations`),
  CONSTRAINT `did` FOREIGN KEY (`driver`) REFERENCES `drivers` (`id`),
  CONSTRAINT `lid` FOREIGN KEY (`locations`) REFERENCES `locations` (`id`),
  CONSTRAINT `oid` FOREIGN KEY (`order`) REFERENCES `orders` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

source code 33 represent creation table called Routing(own work).

6. Results and Discussion

This application aims to solve the problem of finding the shortest path. Furthermore, create a route plan for the drivers of the local target company. This application dramatically helps the people responsible for managing employees, especially drivers, coordinate and manage orders that are expected to be completed soon, as we know that the issue of tracking orders and drivers' status and problems requires time and great effort.

That is why there is an application that saves those in charge of managing companies' time and effort by carrying out the process of planning a route for drivers automatically from the time of entering the application and also updating the data simultaneously with entering any new data into the system and re-planning a new route plan that is commensurate with the new inputs in the system, by kneading no The person in charge needs to re-plan the road plan over and over again whenever we get new inputs or orders for the system.

In the following pictures, you will explain how creating a new route plan for one of the drivers, adding new points, and watching how the system reroute-planner the road, automatically taking care of the presence of new points.

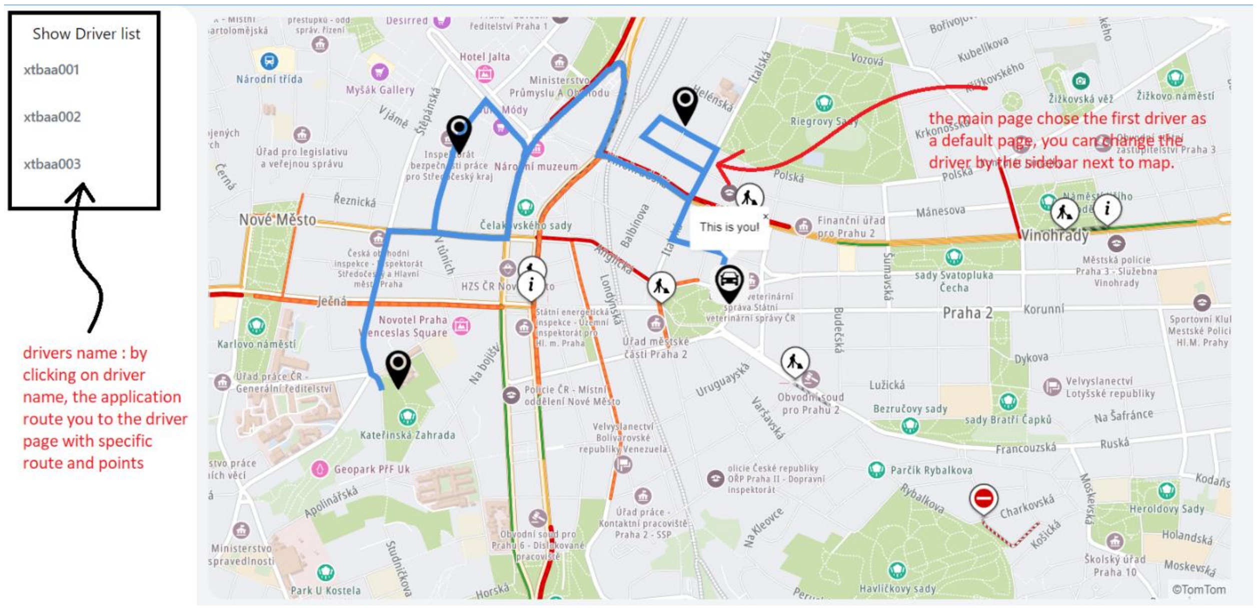
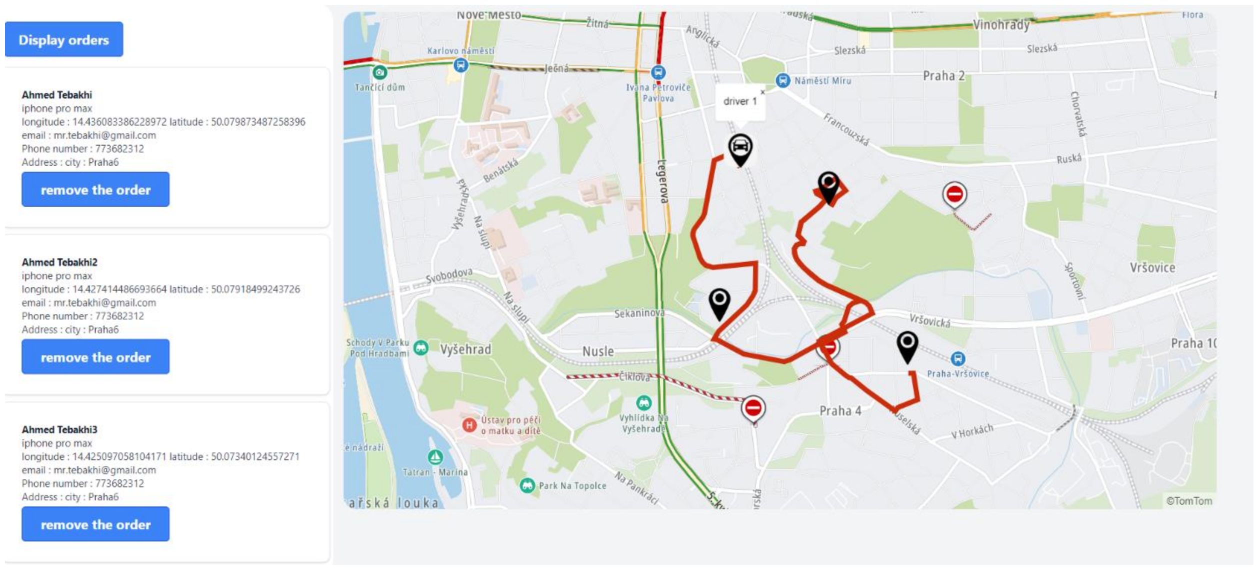

Show Driver list
xtbaa001
xtbaa002
xtbaa003
drivers name : by clicking on driver name, the application route you to the driver page with specific route and points
the main page chose the first driver as a default page, you can change the driver by the sidebar next to map.
This is you!
©TomTom

Figure 31 shows the route planner for a driver(own work).

On this page, the application displays to users of type admin the main page, where the user can see the map assigned to the first driver with points and the delivery plan, but to access more data, he needs to click on the driver's name from the sidebar located in the section on the left of the screen, and the user can through this page Navigating, monitoring and supervising drivers pages.

In the following figure, we dsipalyed what happend after the admin user clicked on one of the drivers.



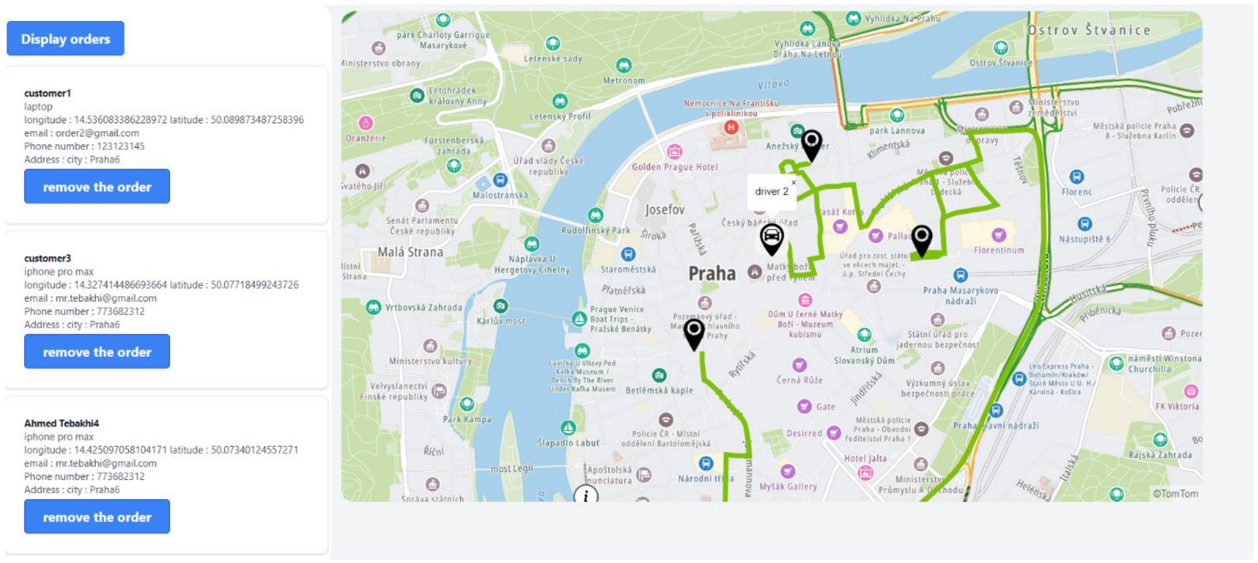
Ahmed Tebakhi
iphone pro max
longitude : 14.436083386228972 latitude : 50.079873487258396
email : mr.tebakhi@gmail.com
Phone number : 773682312
Address : city : Praha6

Ahmed Tebakhi2
iphone pro max
longitude : 14.427414486693664 latitude : 50.07918499243726
email : mr.tebakhi@gmail.com
Phone number : 773682312
Address : city : Praha6

Ahmed Tebakhi3
iphone pro max
longitude : 14.425097058104171 latitude : 50.07340124557271
email : mr.tebakhi@gmail.com
Phone number : 773682312
Address : city : Praha6

Figure 32 shows the route planner for a driver with orders information (own work).

As two sections appear, the admin reviews the delivery plan available to driver number one on this page. The first section is the map, containing all the points that need the delivery process. In the other section, there is a sidebar that contains information for all orders added to this driver, and this happens by extracting all the data stored in the database and shown on this page. This information remains present as long as the admin has not modified or deleted any data.



customer1
laptop
longitude : 14.536083386228972 latitude : 50.089873487258396
email : order2@gmail.com
Phone number : 123123145
Address : city : Praha6

customer3
iphone pro max
longitude : 14.327414486693664 latitude : 50.07718499243726
email : mr.tebakhi@gmail.com
Phone number : 773682312
Address : city : Praha6

Ahmed Tebakhi4
iphone pro max
longitude : 14.425097058104171 latitude : 50.07340124557271
email : mr.tebakhi@gmail.com
Phone number : 773682312
Address : city : Praha6

Figure 33 shows the route planner for another driver with orders information (own work).

As two sections appear, the admin reviews the delivery plan available to driver number one on this page. The first section is the map, containing all the points that need the delivery process. In the other section, there is a sidebar that contains information for all orders added to this driver, and this happens by extracting all the data stored in the database and shown on this page. This information remains present as long as the admin has not modified or deleted any data. After we noticed how to review the route plan for each driver and get all the information from the database, we will now see how the system enters one of the new orders.

The following figure shows a form for entering information or adding a new order. After the customer adds a new order, it will be stored in the database, and this data is saved below. How does the system add this information to the system? The system sends an HTTP request to take the coordinates of the new order and add them to the array called destination before entering them into calculating the shortest path and creating a route. After completing the equation that adds this point to the appropriate driver after a calculation process, it calculates all possible possibilities for all drivers and chooses the shortest time.

After the new points are added to the map, you can see the system re-routing automatically and show the route again.

There are advantages and disadvantaged to the systems:

- Advantages:
    - Real-time routing planner solution.
    - optimizing routes, reducing errors and delays, and improving customer satisfaction.
    - The system is fast and can handle up to 100 connection points.
    - the company can increase efficiency, reduce costs, and build a loyal customer base.
    - reducing the number of errors and delays in the delivery process.

- Disadvantaged:
    - Drivers are not allowed to adjust their routes or schedules as needed.
    - Drivers have no maximum locations can accept.
    - Administrator can not modify route planning manually.

To fix all these disadvantages, we need to add more functions to our application, for example, we can add a function that controls how many orders can be added to one driver during a day. We can add a function for drivers to adjust their routes or their status if they are able to drive or not. Also we can add fucntion to set a priority for very important orders.

7. Conclusion

In this chapter, we will explain to him how this application works. At first, when the user of the type of admin runs the application, the application asks the user to enter his data to ensure the validity and authenticity of this user to use this application. After verifying the validity of the user, the application route this user To the home page; after that, the user can create a route plan; how does that happen? In the beginning, we need to enter a few locations that need to make a route plan; this is done through a page dedicated to entering orders with an address and coordinates; after that, the application stores that data in tables dedicated to it in the database, and now we have the coordinates and locations that we need to create a route for it, the next step is to go to the page dedicated to making a routing plan, immediately after clicking on the main page, the program takes the user to the page through which a routing plan can be created, after moving to this page, the program automatically extracts the coordinates stored in the database and stores them in an array.

Then it will be used to create the shortest path routing plan for the driver. This process is considered the most difficult in terms of execution because of the time that the program needs to calculate the distance between one point and another and use complex algorithms such as Dijkstra to extract the shortest path. However, in light of technological development and the availability of many libraries that help develop and invent new tools that we can use in this field, this has become possible and possible.

Nevertheless, if we talk about it, building a complete web application specialized in the logistic management system is complex. It requires specialized companies to build it because the logistics management system does not stop at creating a route plan and solving the problem of the shortest path routing but instead needs a complete system to deal with customers in terms of Providing them with all the necessary information such as live tracking the order and the duration of delivery, and also needs to provide all functions for drivers also by providing them with control over the orders assigned to them, as well as modifying personal information and getting notifications if a new order or a new task is added, and many other functions.

However, this thesis focused on how to solve short path problems for drivers and solve the problem of the shortest route, opening the way for adding all the characteristics that were previously mentioned in other thesis and doing another study on how to improve the performance of the work of managing logistics systems.

8. References

Advances in Health Sciences Education, v. n. p.-1. J. 2., n.d. s.l.:s.n.
Anon., 2018. *https://truongtx.me/2018/06/07/binary-heap-heapsort-summary-part-1.* [Online].
Anon., 2023. *https://stacktuts.com/how-to-send-basic-auth-with-axios-in-request.* [Online].
Anon., 2023. *https://stacktuts.com/how-to-send-basic-auth-with-axios-in-request.* [Online].
Applegate, D. L., Bixby, R. M., Chvátal, V. & Cook, W. J. (. T. T. S. P. I. 9.-0.-6.-1.-8., n.d. [Online].
Cook, W. J., n.d. *The Traveling Salesman Problem.* s.l.:s.n.
Cormen, T. H., 2001. *Introduction to Algorithms.* second ed. s.l.:s.n.
CyberWolves, 2021. *https://dev.to/cyberwolves/how-to-call-apis-in-react-redux-3f9k.* [Online].
Deshpande, H., 2021. *https://javascript.plainenglish.io/what-is-cors-cross-origin-resource-sharing-and-how-to-fix-the-cors-error-d10130e99dd0.* [Online].
Dijkstra, D., 2001. *https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/.* [Online].
Fevrin, N., n.d. *https://insights.workwave.com/industry/home-delivery/traveling-salesman-problem-for-deliveries/.* [Online].
Geoapify, 2022. *https://www.geoapify.com/openstreetmap-routing.* [Online].
geosolutionsgroup, 2020. *https://training.mapstore.geosolutionsgroup.com/development/fe-technologies/react-redux/react-redux.html.* [Online].
Inc., B. T., 2023. *https://www.linkedin.com/pulse/reactjs-vs-angularjs-comprehensive-comparison-bytes-technolab/.* [Online].
Kawabata, R. K. K., 2007. *Systems Analysis for Collaborative System by Use Case Diagram.* s.l.:s.n.
Lopez-Rojas, E. A., n.d. *https://www.researchgate.net/figure/Simplified-Class-Diagram-for-the-Mobile-Money-Simulation_fig3_262117316.* [Online].
Mancas, C., 2015. *Conceptual Data Modeling and Database Design.* s.l.:s.n.
Manson, S., 2017. *Mapping, Society, and Technology.* s.l.:University of Minnesota Libraries Publishing.
Matco, C., 2023. *https://techbehemoths.com/blog/must-have-skills-ux-designer-should-have.* [Online].
McConnell, S., 2004. *Code Complete.* s.l.:s.n.
Nico, 2020. *https://codewithnico.com/react-wait-axios-to-render/.* [Online].
Oracle, ©. 2., 2023. *https://dev.mysql.com/doc/refman/8.0/en/.* [Online].
Ovuoba, S., 2021. *https://dev.to/devwares/10-awesome-projects-built-with-tailwind-css-4mmf.* [Online].
Porcello, E., 2020. *Modern Patterns for Developing React Apps.* 2nd edition ed. s.l.:s.n.
Rotterdam, n.d. *https://en.wikipedia.org/wiki/Rotterdam.* [Online].
sauer-utley, n., 2020. *https://levelup.gitconnected.com/finding-the-shortest-path-in-javascript-dijkstras-algorithm-8d16451eea34.* [Online].
support, T. d., 2023. *https://developer.tomtom.com/maps-sdk-web-js/documentation.* [Online].
tailwindcss, 2021. *https://tailwindcss.com/.* [Online].
Technologies, N., 2022. *https://www.linkedin.com/pulse/mysql-most-popular-database-management-system-netsqure/.* [Online].
tomtom, n.d. *https://developer.tomtom.com/user/me/apps?category=1.* [Online].
Trail", A. N. P. I. M. f. T. P. S. i. t. H. S. A. E. o. t. "., n.d. [Online].

Van der Zijpp, N., 1996. *Dynamic origin-destination matrix estimation on motorway networks.* s.l.:s.n.

9. Table of figures

10. Table of source code

11. List of symbol and Abbreviation

CSS:	Cascading Style Sheet.
OOP:	Object Oriented Programming.
RDBMS:	Relational Database Management System.
SQL:	Structure Query Language.
MySQL:	My Structured Query Language
HTTP:	The Hypertext Transfer Protocol.
API:	Application Programming Interface.
TSP:	Travel Salesman Problem.
VSc:	Visual Studio Code.
UML:	Unified Modeling Language.
UI:	User interface.
UX:	User experience.
JSX:	JavaScript syntax extension.
XML:	Extensible Markup Language.
CORS:	Cross-Origin Resource Sharing.
NPM:	Node Package Manager.
GIT:	Global Information Tracker.