
Software Engineering Department

ORT Braude College

Capstone Project Phase B – 61999

Taxi station management system.

Advisor: Zeev Barzily

Project code: 24-1-D-15

Student 1: Shorok Heib

ID: 315959429

E-mail 1: Shorok.heeb@e.braude.ac.il

Student 2: Emad Suliman

ID: 322386558

E-mail 2: emadsleman30@gmail.com

GIT Link: https://github.com/ShorokHeib/Taxi-station-management

GIT Link: https://github.com/emadsleman

# Content

**1. Introduction……4**

**2. Background and Related Work……... 5-7**

**3. *Project Review & Process Description* ……..…....7**

3.1 Our Solution ……..……..……7

3.2 Agile Development ..…….....……7

3.3 Web Development ……..……..….7-8

3.4 Project Goals & Unique Features…….……8-9

**4. Process…….……...9**

4.1 Problem Analysis ……..….....9

4.2 Requirements of Taxi Station ……..……...……...9

4.3 System Design and Features …….……9

4.4 Technology Selection……..…..9-10

4.5 Implementation and Testing…….……10

4.6 Deployment and User Training…….……10

4.7 Maintenance and Support…….……10

**5. Engineering Challenges & Solutions……...……..…….……10**

5.1 Integration of Diverse Systems ……..…..……11

5.2 Scalability and Performance……...…....…….11

5.3 Data Security and Privacy……..…..…11

5.4 User Interface Design and Usability…….……12

**6. Evaluation & Verification……...……..……12**

6.1 Unit Tests……...……13-14

6.2 Functional Tests…….……14-15

**7. Results & Conclusion**……16

**8. User Documentation**……17-26

**9. Maintenance Guide**…….27-29

**10. UML Diagrams**…….….30

10.1 Use Case Diagram…….……..……..30

10.2 Class Diagram……...……31

10.3 Activity Diagram……..……32

**11.Program Structure Description**……..33-35

**12. References**…….36

# *Abstract*

The Taxi Station Management System is a solution that optimizes the operation of taxi services, addressing the common challenge of managing taxi fleets and passenger demands. Through, meticulous planning, and the implementation of agile development methodologies, the system provides a user-friendly platform that enables taxi operators to streamline their operations, enhance driver allocation, and improve customer satisfaction.

Surveys and feedback from stakeholders informed the system′s development, ensuring alignment with the requirements of both taxi operators and passengers. The web interface offers centralized management capabilities, leveraging MongoDB for secure, scalable, and real-time data synchronization.

Engineering challenges were methodically tackled to deliver robust solutions, rigorously tested to ensure reliability and usability.

The Taxi Station Management System showcases the practicality and effectiveness of modernizing taxi operations. It serves as a versatile tool for managing fleets, ultimately empowering taxi services to deliver good transportation solutions.

# *1.Introduction*

In today′s rapidly evolving urban transportation landscape, the efficient management of taxi stations involves addressing numerous complexities, from coordinating vehicle availability to meeting passenger demands seamlessly. Existing systems provide basic fleet management tools but often lack the robust features needed to optimize operations and enhance customer service.

The system integrates cutting-edge technologies, with a frontend built on ReactJS with Redux for state management and a backend powered by Node.js with Express, ensuring scalability and real-time data synchronization crucial for managing a dynamic taxi fleet effectively. Secure access is facilitated through JWT authentication, guaranteeing data integrity and user privacy.

The project addresses the escalating demand for convenient and on-demand transportation services by streamlining day-to-day operations. It empowers managers to efficiently handle driver, monitor bookings, and analyze key performance metrics. Simultaneously, secretaries can effortlessly manage customer bookings, assign available taxis, and maintain customer information, enhancing operational efficiency and customer satisfaction.

By incorporating communication tools, reporting capabilities, and notifications, our system enhances user experience and supports informed decision-making. The focus on simplicity and functionality ensures that the system meets the unique requirements of small to medium-scale taxi stations, contributing to the evolution of urban transportation management systems.

This Taxi Station Management System represents a significant advancement in modernizing taxi operations, aiming to deliver transportation solutions, cultivate customer loyalty, and meet the evolving needs of today′s urban transportation ecosystem

## *2.Background and Related Work*

In Israel, there are taxi companies operating in the public transportation sector. Each city and region can work with various taxi companies, providing taxi services to customers in different cities and settlements.

These taxi companies in Israel operate different vehicles, and sometimes they utilize computerized systems or applications aimed at improving service and providing convenience to customers. In recent years, some taxi companies in Israel have launched applications that allow customers to read and order taxis through their mobile phones.

A taxi stand is an advertising space where taxis pick up passengers. Taxi drivers wait at the stand for potential customers. Taxi station management systems can assist in coordinating and improving processes at the stand, specializing in tracking vehicles and managing payments.

Using taxi services allows us to travel from one place quickly and efficiently to another. Unlike buses or trains, taxis do not have a predefined route and can take us from any location to any chosen destination at any time of the day.

This service has many advantages, enabling people who are not interested in using private cars to enjoy all the benefits of private transportation without a significant financial commitment.

The high availability of taxis adds to the efficiency of the service – during most hours of the day, one can easily find a taxi on the street without planning in advance or ordering a taxi home. However, if you want to pre-order a taxi or need one at an unconventional hour, a small fee and a phone call to the taxi station will allow you to book a taxi in advance.

Taxi station management systems include various tools and processes aimed at managing and improving the daily operations of the taxi station. The system's functions and roles can include:

1. **Trip Management:** Efficiently locating, coordinating, and managing trips, including planning, and organizing trips for drivers.
2. **Driver Management:** Managing drivers, including personal data reception, driver's licenses, activity management, and scheduling for trips.
3. **Scheduling and Shift Adjustments:** Consulting and adjusting shift schedules for drivers.
4. **Payments and Invoices Management:** Generating invoices, managing refunds, and financial updates.
5. **Information and Identification Systems:** Using computerized systems to record and manage information such as vehicle locations, bookings.

A taxi station's secretary role can assist in various tasks, including passenger registration, ticket sales and customer service. Secretaries can receive customer details, open orders, coordinate with drivers for trips, establish contact with customers, and match orders with available drivers.

The role of a taxi station manager includes managing and operating the station, managing the employee team, and being responsible for all operational and organizational aspects. This may involve planning and managing trips, financial management, customer service, human resource management, and overall strategic decision-making to promote the taxi station and improve service.

Taxi stations may use various future for trip management and driver matching, such as:

1. Identifying the suitable driver for a trip based on parameters availability and expected travel time.
2. Determining the order in which drivers will serve, improving their availability, and reducing customer waiting times.
3. Allocating and coordinating trips in advance, enhancing service availability, and reducing waiting times.

The use of algorithms can be tailored to the specific needs of the taxi station and the local market conditions.

# *3. Project Review & Process Description*

## 3.1. Our Solution:

Our project focuses on developing a robust Taxi Station Management System designed to streamline operations and enhance efficiency in urban transportation. The system caters specifically to the needs of taxi station managers and secretarial staff by providing a comprehensive web-based platform. This platform integrates cutting-edge technologies such as ReactJS with Redux for frontend development and Node.js with Express for backend functionalities. By leveraging these technologies, we ensure scalability, real-time data synchronization, and secure access through JWT authentication.

## 3.2. Agile Development:

We adopted Agile development methodologies throughout our project lifecycle to ensure flexibility, responsiveness to change, and continuous improvement. By breaking down the development process into iterative sprints, we maintained close collaboration between our development team, stakeholders, and end-users. This approach enabled us to prioritize features based on feedback, swiftly address challenges, and deliver incremental updates. Daily stand-up meetings, regular sprint reviews, and retrospectives were integral to maintaining transparency, optimizing workflows, and ensuring alignment with project goals.

## 3.3. Web Development:

In the realm of web development, our focus on creating a scalable and user-centric platform was paramount. The frontend development using ReactJS and Redux provided a robust foundation for managing complex state interactions and ensuring a seamless user experience. Concurrently,

the backend powered by Node.js and Express enabled efficient data processing, API integrations, and backend operations management. JWT authentication was implemented to secure user access and protect sensitive data, adhering to industry best practices.

Throughout the web development process, we emphasized usability, performance optimization, and responsive design to cater to diverse user needs and device types. Rigorous testing procedures, including unit testing, integration testing, and user acceptance testing, were conducted to validate functionality, ensure reliability, and enhance overall system stability.

### 3.4. Project Goals & Unique Features:

1. **Efficient Operations Management:** Develop a comprehensive information system that assists the taxi station in efficiently managing all aspects of its operations, including employee scheduling, customer bookings, and vehicle maintenance.
2. **Employee Management**: Enable the station manager to effectively recruit, manage, and schedule employees (drivers and administrative staff) based on their qualifications, availability, and job roles.
3. **Driver and Vehicle Tracking:** Implement real-time tracking of drivers and vehicles to optimize dispatching, monitor vehicle locations, and integrate leading vehicles into future trip planning.
4. **Maintenance and Service Management:** Coordinate vehicle maintenance and repairs by integrating with the responsible workshop, ensuring prompt handling of reported issues, and tracking the status of vehicle repairs.
5. **Financial Management:** Automate salary calculations for drivers based on fixed salaries and commissions from fares, and manage customer invoicing and payment processing to ensure accurate financial transactions.

In summary, the project aims to create a comprehensive management system that improves efficiency, enhances employee and customer management, ensures proper vehicle maintenance, and streamlines financial processes for the taxi station.

## *4. Process:*

To ensure efficient management of our taxi station, we've established a structured workflow that guides us through every stage of development. This approach is designed to meet the diverse needs of stakeholders and deliver a robust management solution.

### 4.1. Problem Analysis:

Our process begins with a thorough analysis of the challenges faced by taxi station management. We identify key issues, assess their impact, and strategize effective solutions. This phase is crucial for understanding operational gaps and setting clear objectives for system development.

### 4.2. Requirements of Taxi Station:

By talking with managers, drivers, and staff, and collecting relevant information, we understand the specific needs of the taxi station. This helps us customize the system to meet their requirements and improve efficiency.

### 4.3. System Design and Features:

Based on the identified needs, we design the system and decide on essential features. This includes driver scheduling, vehicle tracking, customer management, and maintenance scheduling. The goal is to create a complete platform that makes operations run smoothly.

### 4.4. Technology Selection:

We carefully select technologies that support the robustness and scalability of our system. This includes leveraging modern development

tools like Java and Python for backend operations, along with cloud solutions such as AWS or Azure for data storage and management.

### 4.5. Implementation and Testing:

During the implementation phase, we develop and integrate the system components while adhering to industry standards and best practices. Rigorous testing ensures that the system performs reliably under various operational scenarios and meets user expectations for usability and performance.

### 4.6. Deployment and User Training:

Upon successful testing, we deploy the system across our taxi station network. Comprehensive training programs are conducted to familiarize stakeholders with the system's functionalities and ensure seamless adoption.

### 4.7. Maintenance and Support:

After the system is deployed, we offer continuous maintenance and support to fix any problems and update the system. This proactive approach ensures the system runs smoothly and meets changing operational needs.

By following this structured development process, we aim to deliver a robust and user-centric Taxi Station Management System that enhances operational efficiency, improves service delivery, and meets the dynamic needs of our stakeholders.

## *5. Engineering Challenges & Solutions:*

The Taxi Station Management System is a solution that optimizes the coordination and operation of taxi services, addressing the common challenge of efficiently managing taxi fleets and passengers demands. Through comprehensive research, meticulous planning, and the implementation of agile development methodologies, the system

provides a user-friendly platform that enables taxi operators to streamline their operations, enhance driver allocation, and improve customer satisfaction. Engineering challenges were methodically tackled to deliver robust solutions, rigorously tested to ensure reliability and usability. Here are the key challenges along with proposed solutions:

### 5.1. Integration of Diverse Systems:

**Challenge**: Integrating various subsystems such as vehicle tracking, customer management, and scheduling into a cohesive platform can be complex due to differing technologies and data formats.

**Solution:** Implement a modular architecture using microservices. Each subsystem operates independently but communicates via well-defined APIs, ensuring seamless integration and flexibility to update or replace components as needed.

### 5.2. Scalability and Performance:

**Challenge:** Managing scalability to handle increasing volumes of transactions, vehicles, and users without compromising system performance or reliability.

**Solution:** Design the system with scalability in mind, leveraging cloud infrastructure for elastic scaling. Implement load balancing techniques and horizontal scaling of application servers and databases. Perform thorough performance testing to optimize system response times and resource utilization.

### 5.3. Data Security and Privacy:

**Challenge:** Safeguarding sensitive information such as customer data, transaction records, and operational details from unauthorized access and breaches.

**Solution:** Implement robust security measures including data encryption, access controls, and regular security audits.

### 5.4. User Interface Design and Usability:

**Challenge:** Designing an intuitive user interface that accommodates diverse user roles (e.g., managers, secretaries) and ensures ease of use across different devices.

**Solution:** Conduct user-centered design (UCD) workshops and usability testing to gather feedback and iteratively improve the interface. Implement responsive design principles to ensure usability on mobile and desktop devices. Provide customizable dashboards and role-based access controls to tailor the user experience.

By solving these engineering challenges proactively, the Taxi Station Management System modernizes taxi operations, manages fleets, optimizes routes, ensures punctual service, and helps taxi services provide better transportation and build customer loyalty.

## 6. *Evaluation & Verification:*

In order to check that our system is working properly, we will perform two types of tests:

• Unit Testing – A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system.

• Functional Testing – Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements. Each function is compared to the corresponding requirement to ascertain whether its output is consistent with the end user's expectations.

## *6.1. Unit test:*

| No | Test Subject | Expected result |
|---|---|---|
| **Login form** | | |
| 1 | Enter an empty username or password. | Error displays a message: "Please fill in the fields". |
| 2 | Log in with a 'frozen' status username and password | Display message: "User is Frozen". |
| 3 | Enter a valid username and password | Display message:" Login succeed". |
| 4 | Enter the wrong username or password. | Error displays a message: "Invalid details, try again" We don't want to give information about which field is invalid. |
| **Add new driver** | | |
| 5 | Enter empty details (one or more) | Error displays a message: "Please fill in the fields". |
| 6 | Enter an existing id. | Display a message: "This driver is already in the system". |
| 7 | Enter an existing email address | Display a message: "This email address is already in use". |
| 8 | Enter an existing phone number | Display a message: "This phone number is already in use". |
| 9 | Enter an existing username | Display a message: "This username is already in use". |
| 10 | Enter valid details | Display a message: "driver registration succeeds, he can now log in with the username and password". |
| **Add new secretary** | | |
| 11 | Enter an existing id. | Display a message: "This driver is already in the system". |
| 12 | Enter new email address | Display a message: "This email address not in use". |
| 13 | Enter an existing username | Display a message: "This username is already in use". |
| 14 | Enter valid details | Display a message: "driver registration succeeds, he can now log in with the username and password". |
| 15 | Enter empty details (one or more) | Error displays a message: "Please fill in the fields". |
| 16 | Enter an existing phone number | Display a message: "This phone number is already in use". |
| **Add new customer** | | |
| 17 | Enter an existing id. | Display a message: "This driver is already in the system". |
| 18 | Enter an existing phone number | Display a message: "This phone number is already in use". |

| | | |
|---|---|---|
| 19 | Enter an existing email address | Display a message: "This email address is already in use". |
| 20 | Enter valid details | Display a message: "customer registration succeeds, he can now log in with the username and password". |
| **Open new order** | | |
| 21 | Enter an existing name and location | Error displays a message: "This order is already existing" |
| 22 | Enter empty details (one or more) | Display message: "Please fill in the fields". |
| 23 | Enter valid details | Display a message: "Registration succeed, please wait for approvement". |

## *6.2 Functional Tests:*

| No# | Test Case | Steps | Expected result |
|---|---|---|---|
| **Login form** | | | |
| 1 | Navigation from the login screen to the dashboard of the manager screen after successful login. | Open the system, enter valid login credentials, and click on the 'login' button. | Move to the manager form. |
| 2 | Login with a valid secretary username and password | | Move to the secretary form. |
| **Add new driver** | | | |
| 3 | Navigation from the screen dashboard of the manager screen to the list of driver screen. | Open the system, click on the 'requitement of the drivers' button. | The system should navigate to the 'list of drivers' screens |
| 4 | Navigation from the screen list of drivers screen to add new driver screen. | Open the system, click on the 'Add new+' button. | The system should navigate to the 'Add new driver' screen. |
| **Add new customer** | | | |
| 5 | Navigation from the screen dashboard of the manager screen to the list of customer screen. | Open the system, click on the 'requitement of the customer' button. | The system should navigate to the 'list of customers' screens |
| 6 | Navigation from the screen list of customer screen | Open the system, click on the 'Add new+' button. | The system should navigate to the 'Add new customer' screen. |

| | | | |
|---|---|---|---|
| | to add new customer screen. | | |
| **Monthly updates** | | | |
| 7 | A driver has X orders for the last month. | | The driver will receive a bill that includes the amount of payment he has to pay “price per order * X”. |
| 8 | number of customer orders has decrease. | | The customer status will be changed from ‘active’ to ‘inactive’ |
| 9 | The cab has broken down | | The status of the cab has changed (active/inactive) |
| 10 | A driver has retired. | | The driver will remove from the list. |
| **Add new secretary** | | | |
| 11 | Navigation from the screen dashboard of the manager screen to the list of secretary screen | Open the system, click on the ‘requitement of secretary’ button. | The system should navigate to the ‘list of secretaries’ screens |
| 12 | Navigation from the screen list of drivers screen to add new driver screen. | Open the system, click on the ‘Add new+’ button. | The system should navigate to the ‘Add new secretary’ screen. |

## *7. Results & Conclusion*

The Taxi Station Management System (TSMS) has achieved great results and proven effective in improving taxi operations. With careful research and planning, we developed a platform that makes managing taxi fleets, coordinating drivers, and serving customers much easier. We used feedback from users and cutting-edge technologies like ReactJS and Node.js to build a system that is both user-friendly and scalable. We tackled engineering challenges with smart solutions and teamwork, which led to a strong and reliable system. By using visual diagrams and improving the design, we created an easy-to-use interface. We also conducted thorough testing to fix any issues and ensure high quality. In summary, the TSMS meets the needs of managing taxi services efficiently and offers a better experience for both operators and customers. This project highlights our dedication to improving urban transportation and enhancing taxi services.

## *8. User Documentation*

**General Description:**

The Taxi Station Management System (TSMS) is a tool designed to make running taxi stations easier. It helps with managing taxi fleets, organizing drivers, and handling customer requests all in one place. The system allows taxi stations to schedule drivers, track where vehicles are, manage bookings, and handle vehicle maintenance. It uses a web-based platform with technologies like ReactJS and Node.js, which makes it flexible and updates in real time. The TSMS includes features for managing trips, tracking drivers and vehicles, and handling finances. Aimed at taxi station managers and staff, it improves efficiency and service quality. The easy-to-use design and strong backend ensure smooth operation, making taxi management simpler and better for both operators and customers.

**User Interface:**

- Our Logo:

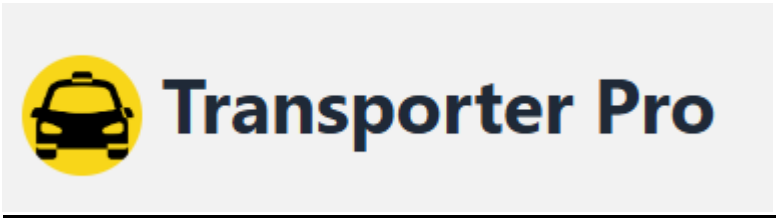



**Fig 1:** Application Logo

- Login Screen

  This is the first page that every user (station manager/secretary) sees when entering the website. Each user can join using their username and password, and open the relevant page. For users who have forgotten their password, they can click the button "Forgot Password" below to reset it.

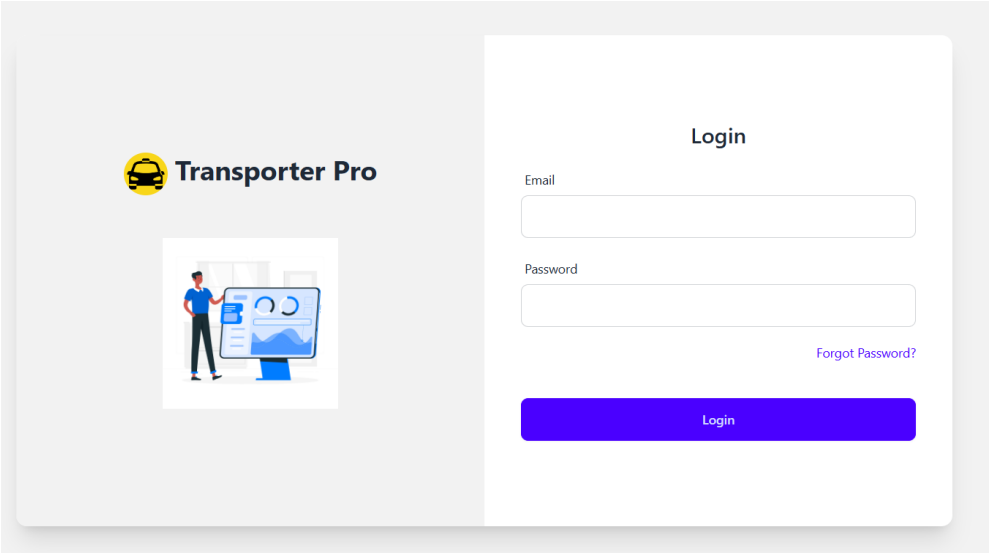
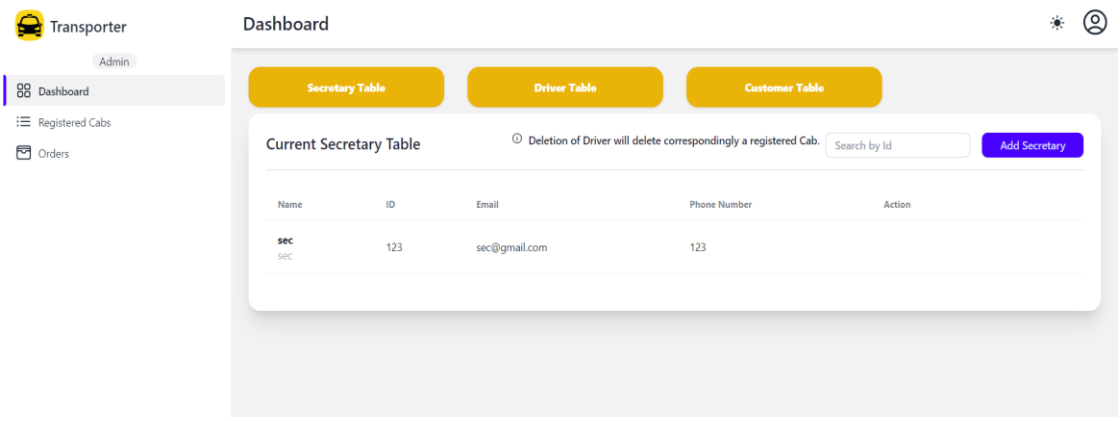



**Fig 2:** Login screen

- <u>administrator's dashboard</u>

  The administrator's dashboard has a page with three options: secretary table, driver table and customer table. When selecting " secretary table" a table displaying the secretaries and their details appears. The administrator also has the option to add a new secretary by clicking on the "Add secretary" button. Additionally, they can delete or update information by choosing one of the options in the "Action" section, namely "delete" or "Edit."
  in addition, administrator has the option to search in the table using the "Search Label".



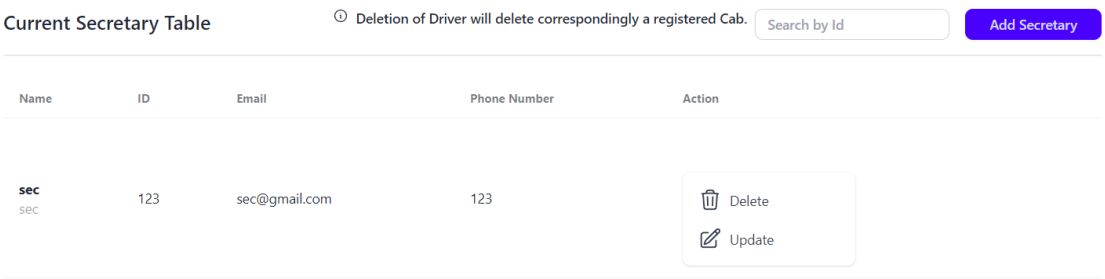



**Fig 3:** administrator's dashboard

- Add secretary

  This page appears after clicking the "Add secretary" button in fig 3. On this page, the administrator can fill in the details of the new secretary who wishes to join the company. Finally, they click the "save" button, and the secretary's information is automatically updated in the table located in fig 3.

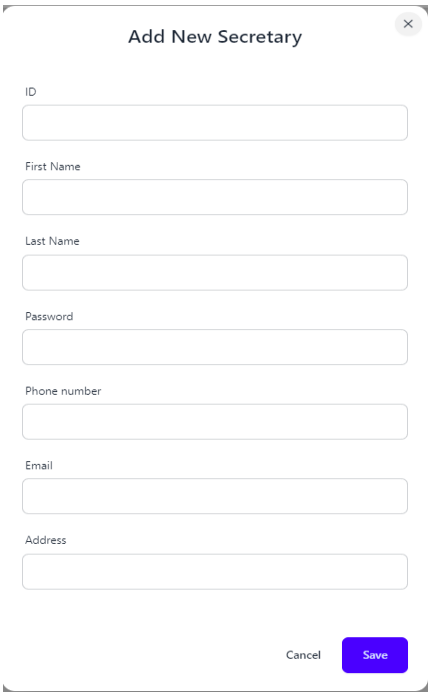



**Fig 4:** add secretary

- Registered Cabs

  This page appears for admin after he clicks on " Registered Cabs " on the left side of the page. A table displaying details of the cabs is shown.
  The administrator also has the option to add a new Cab by clicking on the "Add" button. Additionally, they can delete or update information by choosing one of the options in the "Action" section, namely "delete" or "Update".
  in addition administrator has the option to search in the table using the "Search Label".

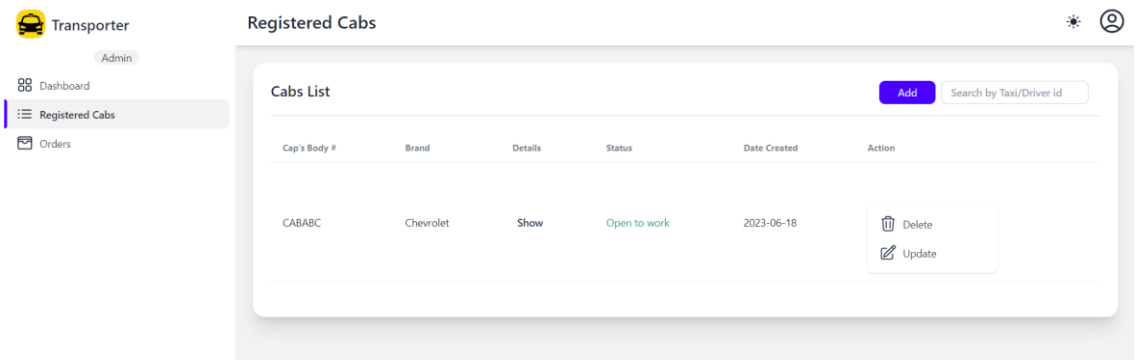



**Fig 5:** Registered Cabs

- Add New Cab

  This page appears after clicking the "Add" button in fig 5.
  On this page, the administrator can fill in the details of the new cab. The manager chooses the details of the cab and if he wants to save the details, they click on "Save", and the cab's information is automatically updated in the table located in fig 5, Otherwise, they can click on "Cancel".

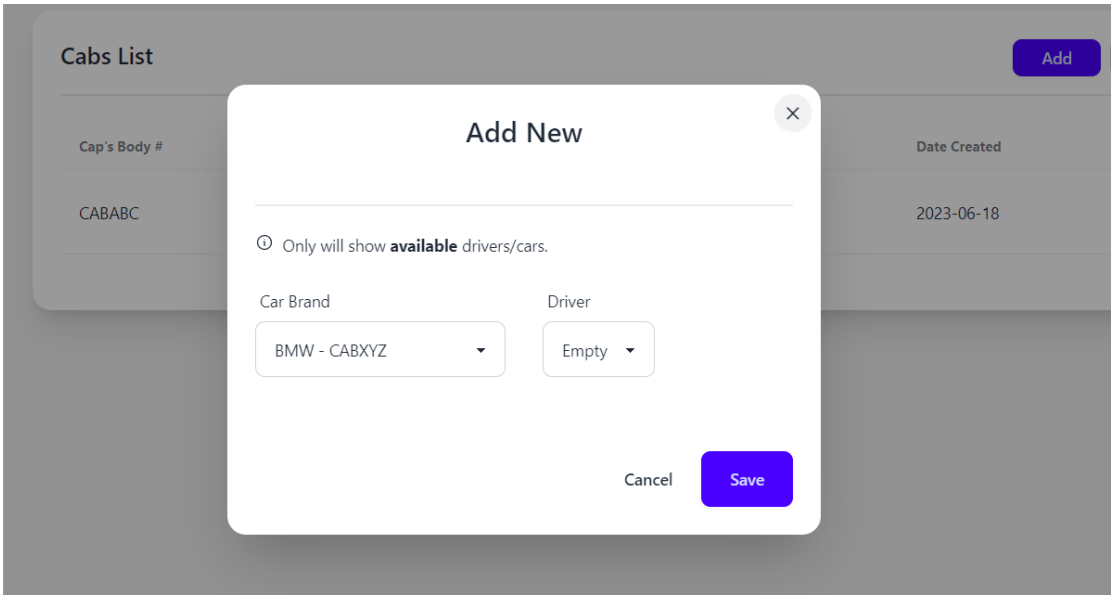
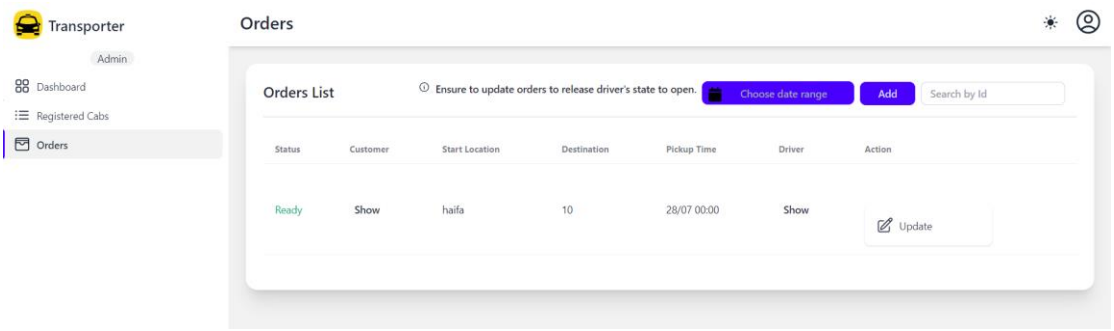



**Fig 6:** add Cabs

- Order Page

  This page appears for manger after he clicks on "Order" on the left side of the page. A table displaying details of the Order is shown. Additionally, they can update information.
  Status (Dropped off / Ready).
  The manager can view Driver's Details by clicking on "Show".
  in addition administrator has the option to search in the table using the "Search Label".



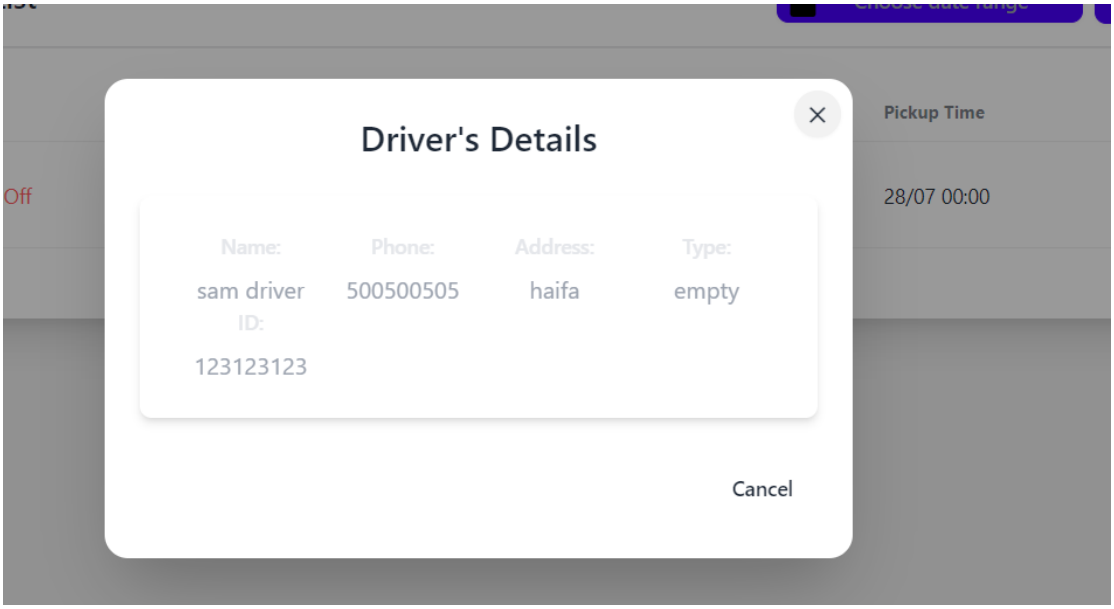
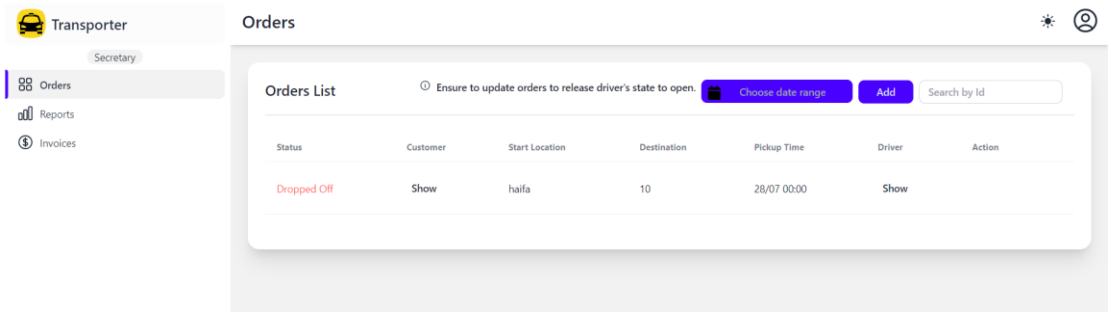



**Fig 7:** Order Page

- Secretary dashboard
  This page appears for secretary after she clicks on "Orders" on the left side of the page. A table displaying details of the orders is shown.
  The secretary also has the option to add a new order by clicking on the "Add" button.
  The secretary can view Driver's Details by clicking on "Show".
  in addition she has the option to search in the table using the "Search Label".



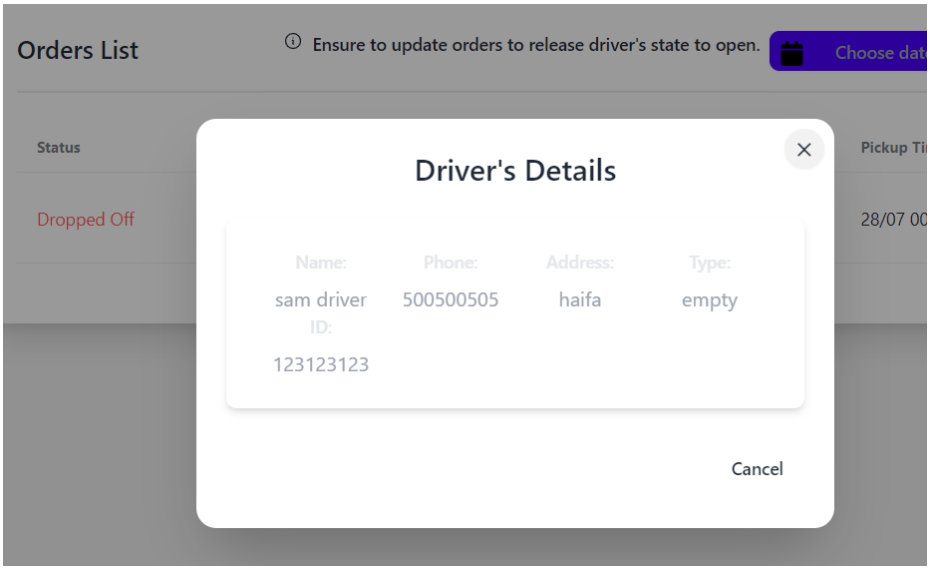



**Fig 8:** Orders Page

- Add new order
  This page appears after clicking the "Add" button in fig 8.
  On this page, the secretary can fill in the details of the new order. If she wants to save the details, she clicks on "Save", and the order's information is automatically updated in the table located in.

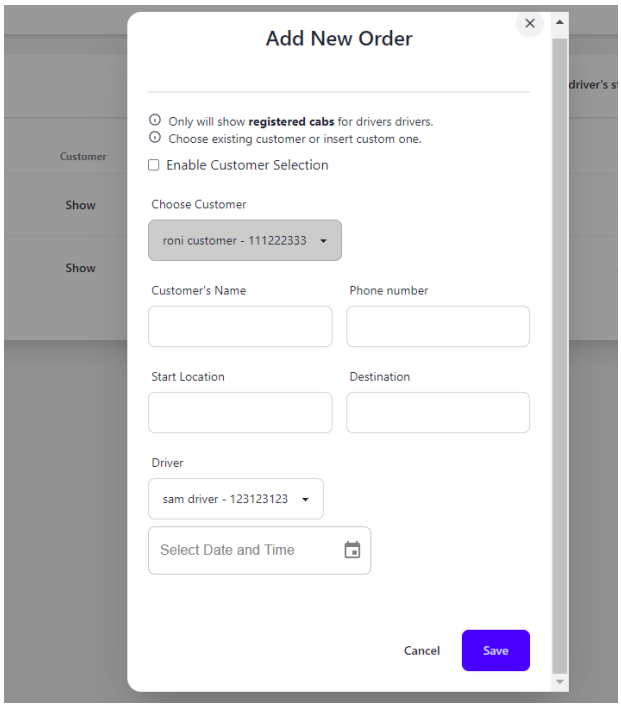



**Fig 9:** Add Order

- Reports
  This page appears for secretary after she clicks on "Reports" on the left side of the page. A table displaying details of the reports is shown.
  The secretary also has the option to add a new report by clicking on the "Add" button.
  in addition, she has the option to search in the table using the "Search Label".

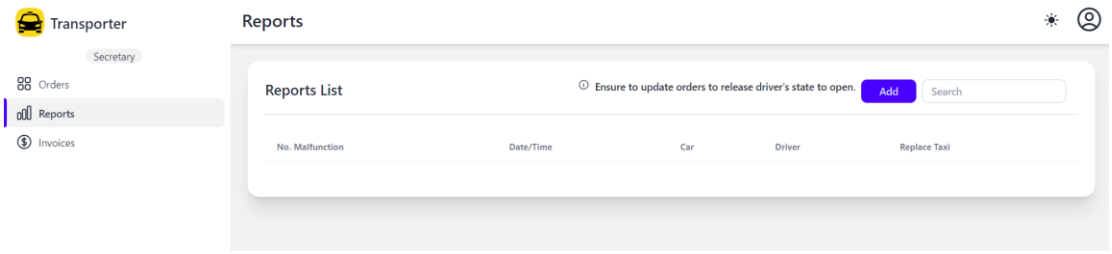



**Fig 10:** Reports

- Add new report
  This page appears after clicking the "Add" button in fig 10.
  On this page, the secretary can fill in the details of the new report.
  If she wants to save the details, she clicks on "Save", and the report's information is automatically updated in the table located in.

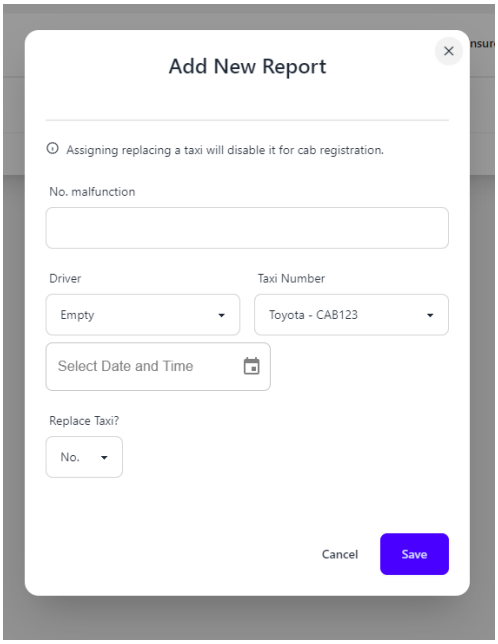



**Fig 11:** Add Reports

- Invoices
  This page appears for secretary after she clicks on "Invoices" on the left side of the page. A table displaying details of the Invoice is shown.
  The secretary also has the option to add a new report by clicking on the "Add" button.
  in addition, she has the option to search in the table using the "Search Label".

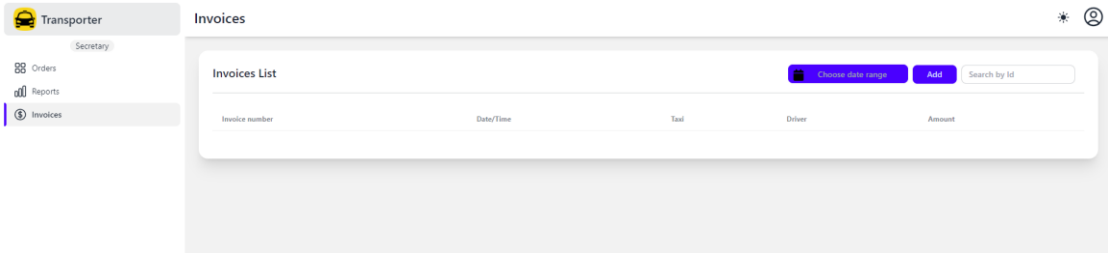



**Fig 12:** Invoices

- Add Invoices
  This page appears after clicking the "Add" button in fig 12.
  On this page, the secretary can fill in the details of the new invoice. If she wants to save the details, she clicks on "Save", and the invoice's information is automatically updated in the table located in.

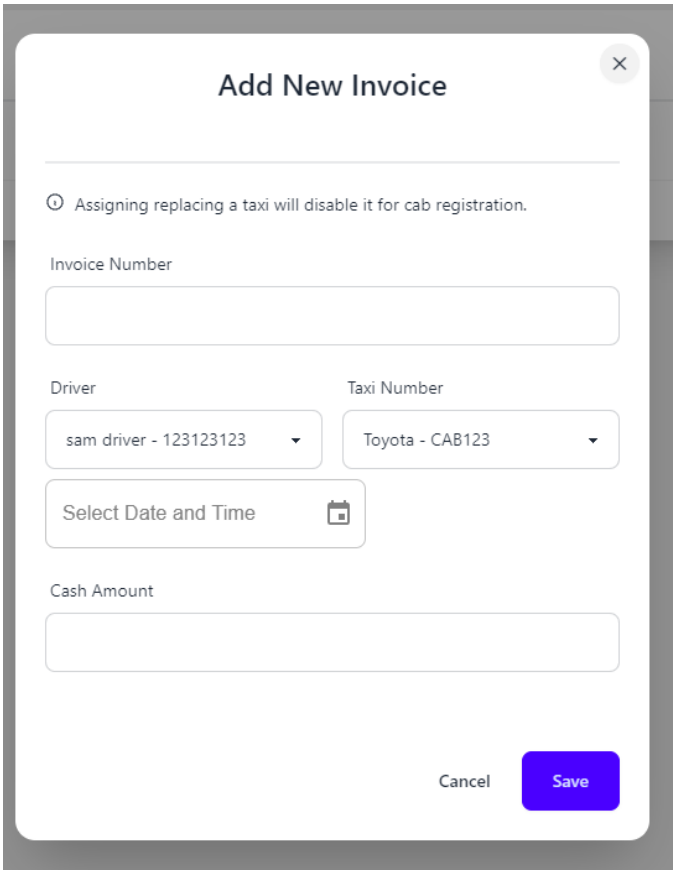
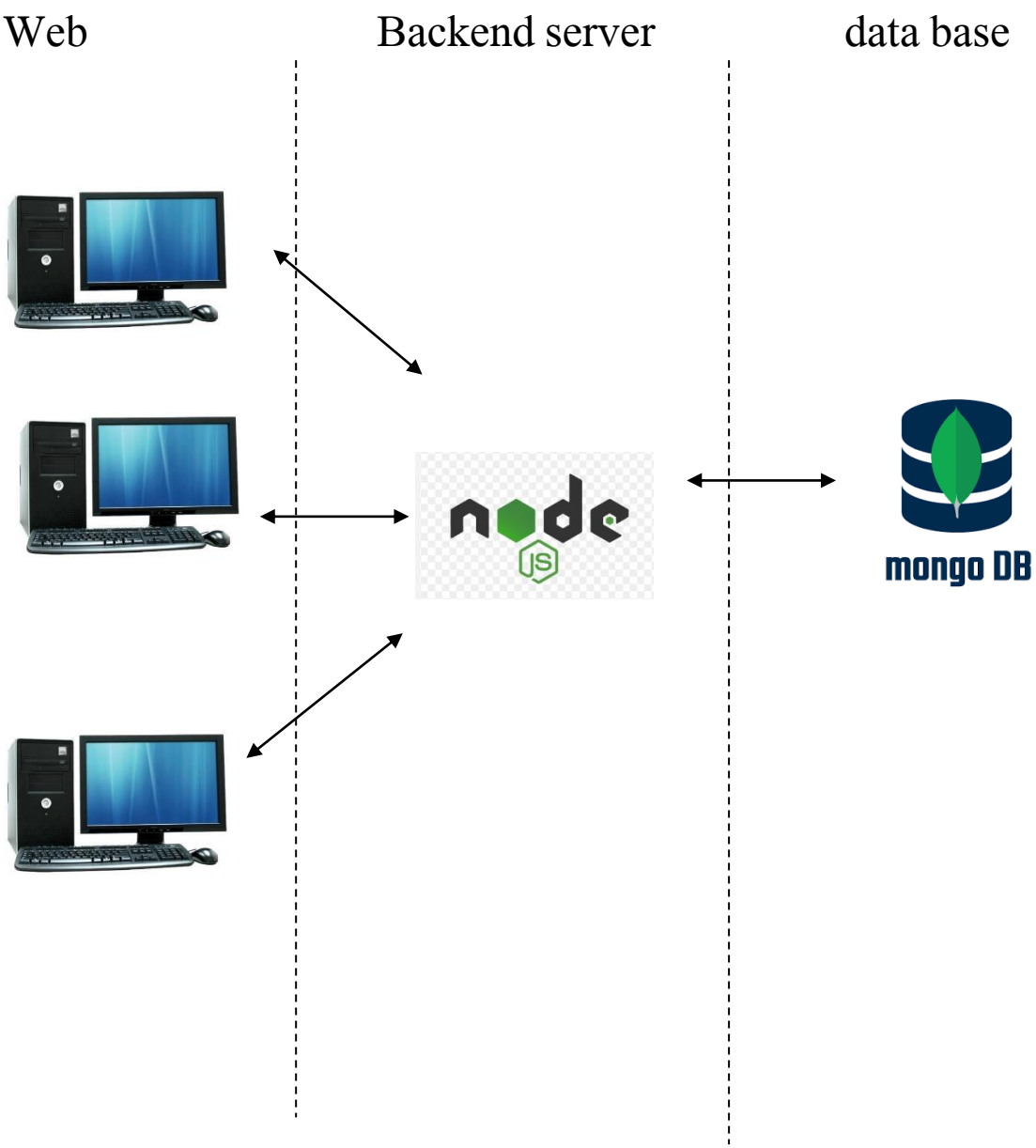



**Fig 13:** Add Invoices

# *9. Maintenance Guide*

Product:

Software Architecture Diagram:



**Architecture Overview**:

The architecture of the system follows a three-tier client-server model, consisting of a Web client, a Backend server utilizing Node.js, and a MongoDB database. This setup provides a scalable and efficient framework for web applications.

***Components of the Architecture***:

***Web Client***:

The Web client component encompasses the user interface (UI) and user interactions, operating on web browsers. It handles the presentation layer, enabling users to interact with the application. The Web client communicates with the Backend server to send requests, retrieve responses, and manage data. Multiple clients can connect simultaneously, ensuring scalable access to the application.

**Backend Server (Node.js):**

The Backend server, represented by the Node.js platform, functions as the application logic layer. It processes requests from Web clients, executes the business logic, and interfaces with the MongoDB database for data retrieval and storage operations. Node.js facilitates efficient handling of concurrent requests, ensuring that the application can manage high traffic and complex data manipulations.

**MongoDB Database**:

The MongoDB database provides the persistent storage layer for the application. It is responsible for storing and managing the application's data, including user information, application states, and transactional records. MongoDB offers flexible data modeling and efficient querying capabilities, enabling quick data access and manipulation.

**Communication:**

The Web client communicates with the Node.js Backend server through bidirectional arrows, illustrating the request-response cycles. This interaction allows for real-time updates and dynamic data exchange. The Node.js server, in turn, communicates with the MongoDB database through a bidirectional arrow, representing data read and write operations. This setup ensures that data is consistently stored and retrieved across the system.

This architectural design ensures a clear separation of concerns, with each component assuming specific responsibilities. The Web client manages user interfaces and interactions, the Node.js server handles application logic, and MongoDB provides data persistence. This separation enhances system maintainability, scalability, and development efficiency.

## *10. UML Diagrams* – this section is initiated in phase A of the project.

### 10.1 Use Case Diagram

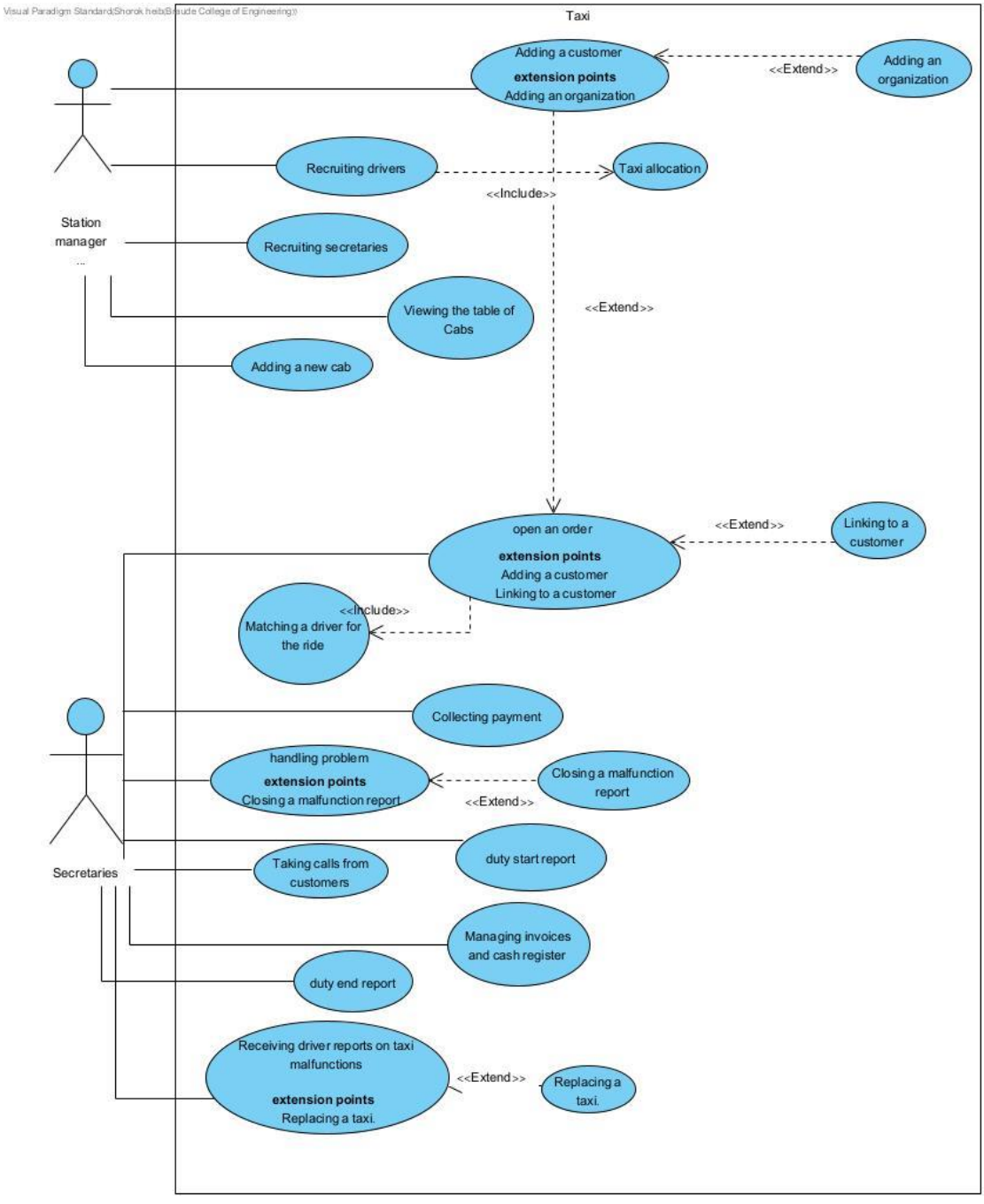



Fig1: Use Case Diagram

## 10.2 Class Diagram

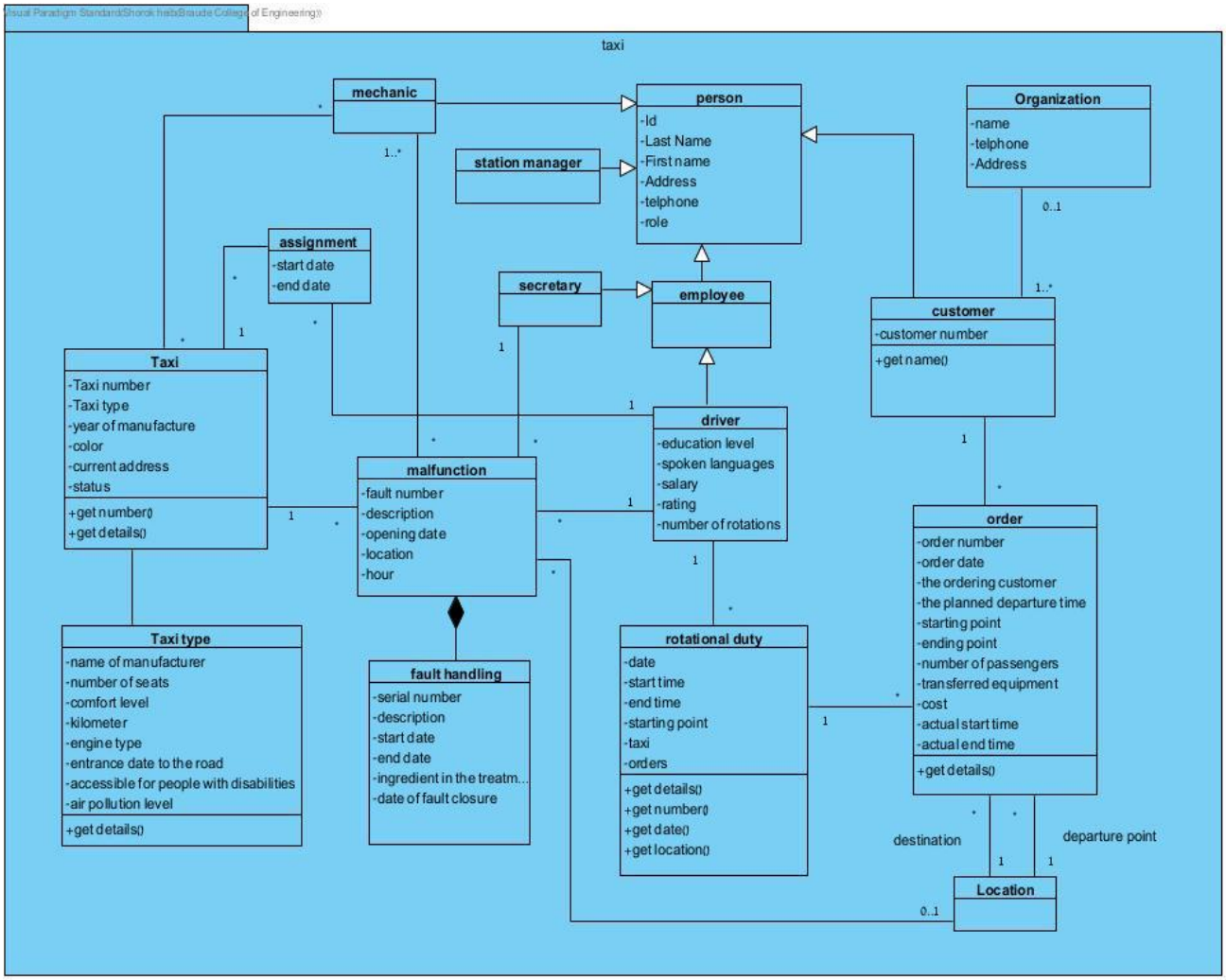



Fig2：Class Diagram

## 10.3 Activity Diagram

*Ordering a taxi through the reception desk.*

1. The secretary types the origin and destination.
2. The secretary asks for the customer's name.
3. If the customer is a member, she links the order to the customer.
4. If the customer is not a member, she types in the customer's name.
5. The secretary contacts the driver available during the order.
6. If the driver is not assigned to the task while placing the order, she checks how long it will take for him to reach the customer's location. If it takes a long time, she returns to step 5, and if it takes a short time, she assigns the order to the customer and notes the order process.
7. If the driver is not available for assignment, the secretary checks if there is another available driver with availability.
8. If there is another driver available during the order, the secretary proceeds to step 5.
9. If there is no additional driver available, the secretary informs the customer that there is no taxi available for their order.

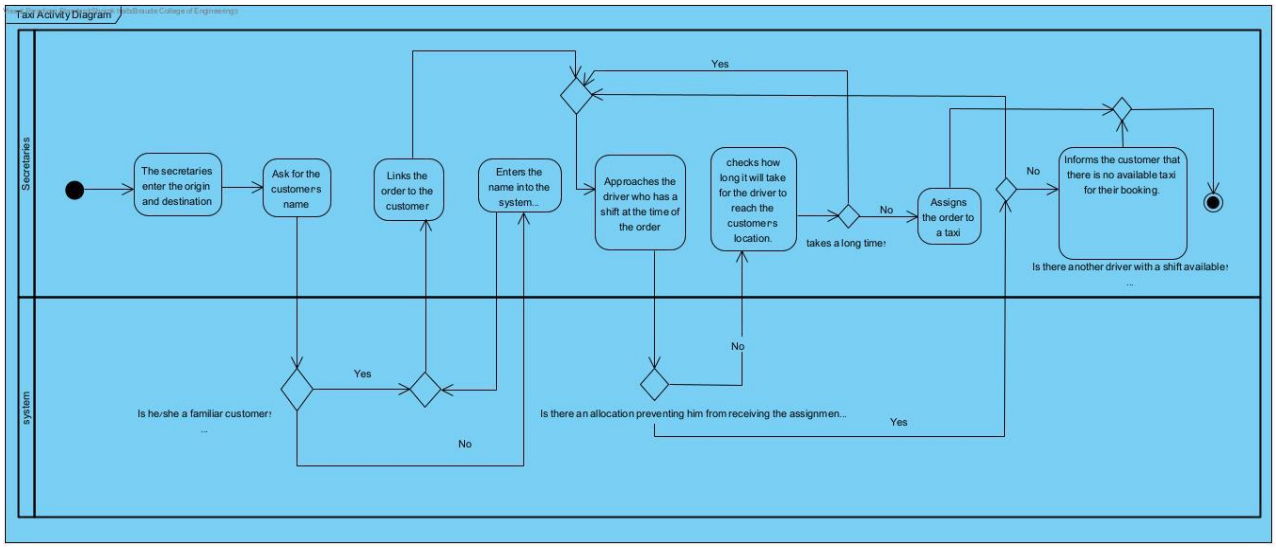



Fig3: Activity Diagram

## *11.Program Structure Description*

**MongoDB:**

In our system, we used MongoDB to store all the necessary information for managing the station (such as data on taxis, drivers, and trip history) in a structure that allows for quick searches and dynamic changes to the data. The code interacts with MongoDB through interfaces like Mongoose (a popular library in Node.js) to perform operations such as reading data, adding, updating, and deleting.

**DB Scheme:**

| Table Name | Value | Type |
|---|---|---|
| Cab | car | car id (Ref to Car table) |
| | driver | driver id (Ref to User table) |
| | taken | Boolean |
| | free | Boolean |

| Table Name | Value | Type |
|---|---|---|
| Car | brand | String |
| | dateCreated | String |
| | plateNum | Number |
| | vehicleModel | String |
| | cabBodyNum | String |
| | taken | Boolean |
| | deleted | Boolean |

| Table Name | Value | Type |
|---|---|---|
| invoice | amount | Number |
| | date | Date |
| | driver | mongoose.Schema.Types.ObjectId, ref: 'User' |
| | car | mongoose.Schema.Types.ObjectId, ref: 'Car' |
| | Num | Number |

| Table Name | Value | Type |
|---|---|---|
| Order | customer | mongoose.Schema.Types.ObjectId, ref: 'User' |
| | phone | Number |
| | endingLocation | String |
| | date | Date |
| | driver | mongoose.Schema.Types.ObjectId, ref: 'User' |
| | orderStatus | Number |
| | name | String |

| Table Name | Value | Type |
|---|---|---|
| Report | malNum | Number |
| | date | Date |
| | driver | mongoose.Schema.Types.ObjectId, ref: 'User' |
| | car | mongoose.Schema.Types.ObjectId, ref: 'Car' |
| | replace | Boolean |

| Table Name | Value | Type |
|---|---|---|
| Role | name | String |

| Table Name | Value | Type |
|---|---|---|
| User | firstName | String |
| | lastName | String |
| | email | String |
| | password | String |
| | address | String |
| | Phone | String |
| | role | String |
| | type | String |
| | status | Boolean |
| | idNum | Number |
| | taken | Boolean |
| | deleted | Boolean |
| | otp | Number |

# 12. References


- **React (Frontend Development):** https://reactjs.org/docs/getting-started.html
- **Node.js (Backend Development):** https://nodejs.org/en/docs/
- **MongoDB (Database):** https://docs.mongodb.com/manual/
- **Express.js (Web Application Framework for Node.js):** https://expressjs.com/en/starter/installing.html
- **JWT Authentication:** https://jwt.io/introduction
- **Agile Development Methodology:** https://agilemanifesto.org/
- **UML Diagrams:** https://www.uml.org/what-is-uml.htm
- **Software Architecture Patterns:** https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/
- **Web Application Security:** https://owasp.org/www-project-top-ten/
- **Responsive Web Design:** https://developers.google.com/web/fundamentals/design-and-ux/responsive
- **RESTful API Design:** https://restfulapi.net/