Module Name: Advanced Programming

Module Code: CIS6003

Name: Fathima Sumaiya Siyanudeen

CMU Student ID: ST20253341

ICBT Student ID: CL/BSCSD/24/83

Submitting Date: 19th September 2022

Introduction

GoCheeta is an online cab service system which allows transportation within a city. This is made to make people's day to day transportation activities easier, where it connects the driver and the customer. GoCheeta has several branches within Sri Lanka. They are in Galle, Kandy, Nugegoda, Gampaha, Kurunegala and Jaffna.

(TASK A) Requirement Specification

Technologies Used

Client: HTML, CSS, JavaScript

Service: Java

Database: MySQL

Web Service: REST Service

Functional requirement

The expected performance of the software is captured by the functional requirements. The software's necessary performance of these behaviors could be described as offerings, activities, or functionalities (Malan and Bredemeyer, 2001).

From customer's perspective:

- ➔ Register: The customer should be able to register into the system by themselves by giving the required details.

- ➔ Login: The customer should be able to login into the system by themselves with their username and password.
- ➔ Search Vehicle Type: The customer should be able to search for a vehicle type based on their preference or capacity.
- ➔ View Available Vehicles of that type chosen: The customer should be able to see available vehicles of that chosen vehicle type and select to book.
- ➔ Select city: The customer should be able to select the city from where they want to do the ride.
- ➔ Select source/destination: The customer should be able to select the source and destination within a specific city.
- ➔ View Price and Booking details: The customer should be able to view the calculated price and the booking details for their ride.
- ➔ Book a Vehicle: The customer should be able to select a vehicle type and book a ride according to their preference and budget.
- ➔ View All Bookings/Past Rides: The customer should be able to view their previous bookings and its details.
- ➔ Give Feedback: The customer should be able to give feedback about their ride experience.
- ➔ View Profile: The customer should be able to view their profile details anytime.
- ➔ Edit Profile: The customer should be able to update their profile details anytime.

From admin's perspective:

- ➔ Register: The admin should be able to register into the system by themselves by giving the required details.
- ➔ Login: The admin should be able to login into the system by themselves with their username and password.

- ➔ Add Driver: The admin should be able to add a driver into the system.
- ➔ View Driver: The admin should be able to view the details of the driver.
- ➔ Update Driver: The admin should be able to update driver details of a registered driver.
- ➔ Remove Driver: The admin should be able to remove a driver from the system.
- ➔ Add Vehicle: The admin should be able to add a vehicle into the system, so that the customer can see the available vehicles.
- ➔ View Vehicle: The admin should be able to view the details of the vehicle.
- ➔ Update Vehicle: The admin should be able to update vehicle details of a registered vehicle, so the customer and driver can see the updated details about the vehicle.
- ➔ Remove Vehicle: The admin should be able to remove a vehicle from the system.
- ➔ Add Vehicle Type: The admin should be able to add a vehicle type into the system.
- ➔ View Vehicle Type: The admin should be able to view the details of the vehicle type.
- ➔ Update Vehicle Type: The admin should be able to update vehicle type details of a registered vehicle type.
- ➔ Remove Vehicle Type: The admin should be able to remove a vehicle type from the system.
- ➔ Add City: The admin should be able to add a city into the system.
- ➔ View City: The admin should be able to view the details of the city.
- ➔ Update City: The admin should be able to update city details of a registered city.

- ➔ Remove City: The admin should be able to remove a city from the system.
- ➔ Add Street: The admin should be able to add a street into the system.
- ➔ View Street: The admin should be able to view the details of the street.
- ➔ Update Street: The admin should be able to update street details of a registered street.
- ➔ Remove Street: The admin should be able to remove a street from the system.
- ➔ View Customer Bookings: The admin should be able to view all the customer bookings for all branches along with customer, driver and vehicle details associated with the booking.
- ➔ View Total Sales Each Branch: The admin should be able to view the total sales report for each branch.
- ➔ View Total Sales All Branches: The admin should be able to view the total sales report for all branches.
- ➔ View New Bookings: The driver should be able to view new bookings made and requested by the customer.

From driver's perspective:

- ➔ Login: The driver should be able to login into the system by themselves with their username and password.
- ➔ Send details to admin to register: The driver should be able to fill up the details and send them to admin.
- ➔ Accept/ Decline a Booking: The driver should be able to either accept or decline the booking.
- ➔ End Ride: The driver should be able to end the ride when the ride is done.

- ➔ View Profile: The driver should be able to view their profile details anytime.
- ➔ Edit Profile: The driver should be able to update their profile details anytime.
- ➔ See past ride/activity: The driver should be able to see their past ride/activity details, along with the feedbacks they got from their customers for their rides.

Nonfunctional requirement specification 3

A specification of an activity that a software should be able to carry out despite taking physical limitations into account (Glinz, 2007).

- Usability: The website should swiftly guide users to the accomplishment of their primary objective with minimal distracting features or crowded interface. Users should be able to immediately learn the interface of the product during the first encounter. Users ought to be able to recollect their former interface and user experience. If someone has already made a booking on a website, making another one must be simpler.
- Security: To protect the system from unwanted access, the system offers a login and password. The user password must have more than eight characters. Only authorized firm users should be able to access the system. The subsystem should offer a high level of security and integrity for the data it holds.

- Performance: Every user instruction must receive a response from the system in no more than 10 seconds at the most. When processing user input, the system should operate quickly and respond in a short amount of time.

- Error handling: The likelihood of error should be greatly reduced, and it should be possible to recover from errors by providing the user with the proper error message. User input validation is really crucial.

- Ease of use: Given the degree of expertise the system's users possess, a straightforward yet high-quality user interface should be created to make it simple to use and necessitate less training.

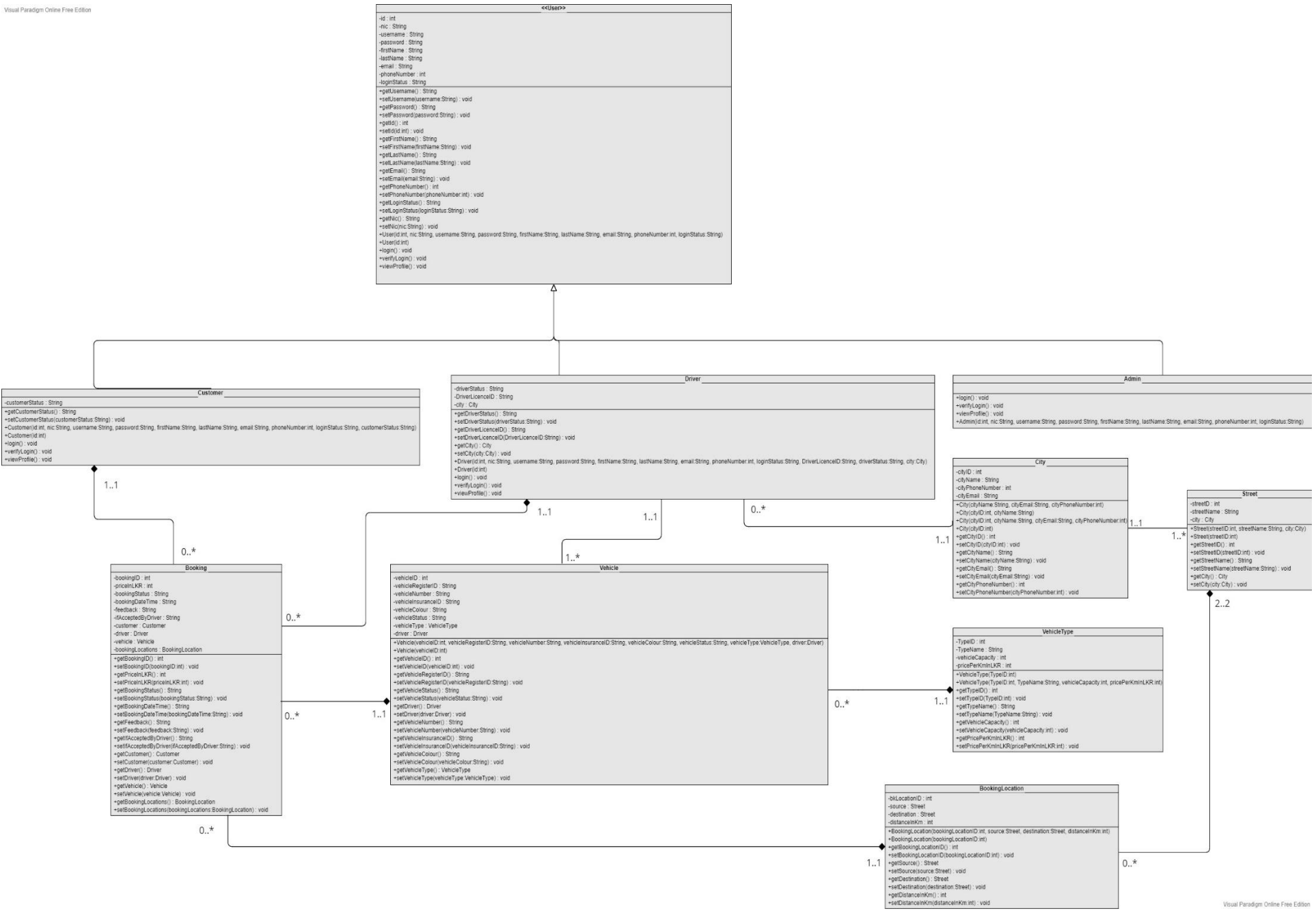
- Availability: The users can access the system all times.

(TASK B) UML DIAGRAMS

UML is a genuinely expansive notation that offers a wide variety of drawings, and for every design it offers a substantial number of components to address each modeller's potential needs for any work (Reggio, Leotta, Ricca and Clerissi, 2013).

- **Class diagram**

  The information about the topic of interest is modelled using UML class diagrams, which show the connections between items sorted into different classes (Berardi, Calvanese and De Giacomo, 2005).
  Link to my Class Diagram: https://postimg.cc/TKyDcRT6



- The Customer, Driver and the Admin classes, have similar attributes. Therefore, a User Abstract class is created with all the common attributes, Customer, Driver and Admin classes inherit all the attributes and methods from User class. Therefore, they possess an inheritance relationship.
- As mentioned in the requirement specification, the Driver is registered in a specific branch/city. Therefore, the Driver class has an attribute of City class, to get the data of the city that the driver is registered in. They possess a One-to-Many relationship. A city can have many drivers registered, if it was an early registered city, or a city can have no drivers, if it was a newly registered city. A driver should have a registered city.
- As mentioned in the requirement specification, the Customers, should be able to choose from categories of vehicles. Therefore, to have a data of the types of vehicles available, the VehicleType class is created along with the necessary attributes.
- As mentioned in the requirement specification, a Vehicle should be associated with a Driver. Therefore, the Vehicle class has an attribute of Driver class, as the data of the driver associated, can be retrieved. And the Vehicle class has an attribute of VehicleType class, as they possess a vehicle type. Vehicle and VehicleType classes possess a One-to-Many composition relationship. As a vehicle cannot exist without a vehicle type. A vehicle should have a type. A vehicle type can have, many vehicles of that type if it is an already registered vehicle type, or can have no vehicle if it is a newly registered type. The Vehicle and Driver class possess a One-to-Many relationship. A vehicle can be associated with only one driver. A driver can have, atleast one vehicle only then he can make a ride, or can have many vehicles associated with him, if he was an already registered driver.
- As mentioned in the requirement specification, the Customer can make a ride by choosing streets to make a ride within a city. Therefore, Street class was created along with the necessary attributes. And since they have a city, the Street class has an attribute of City class, to retrieve all the data of that city the street is located in. The Street and City classes possess a One-to-Many relationship, as a street should

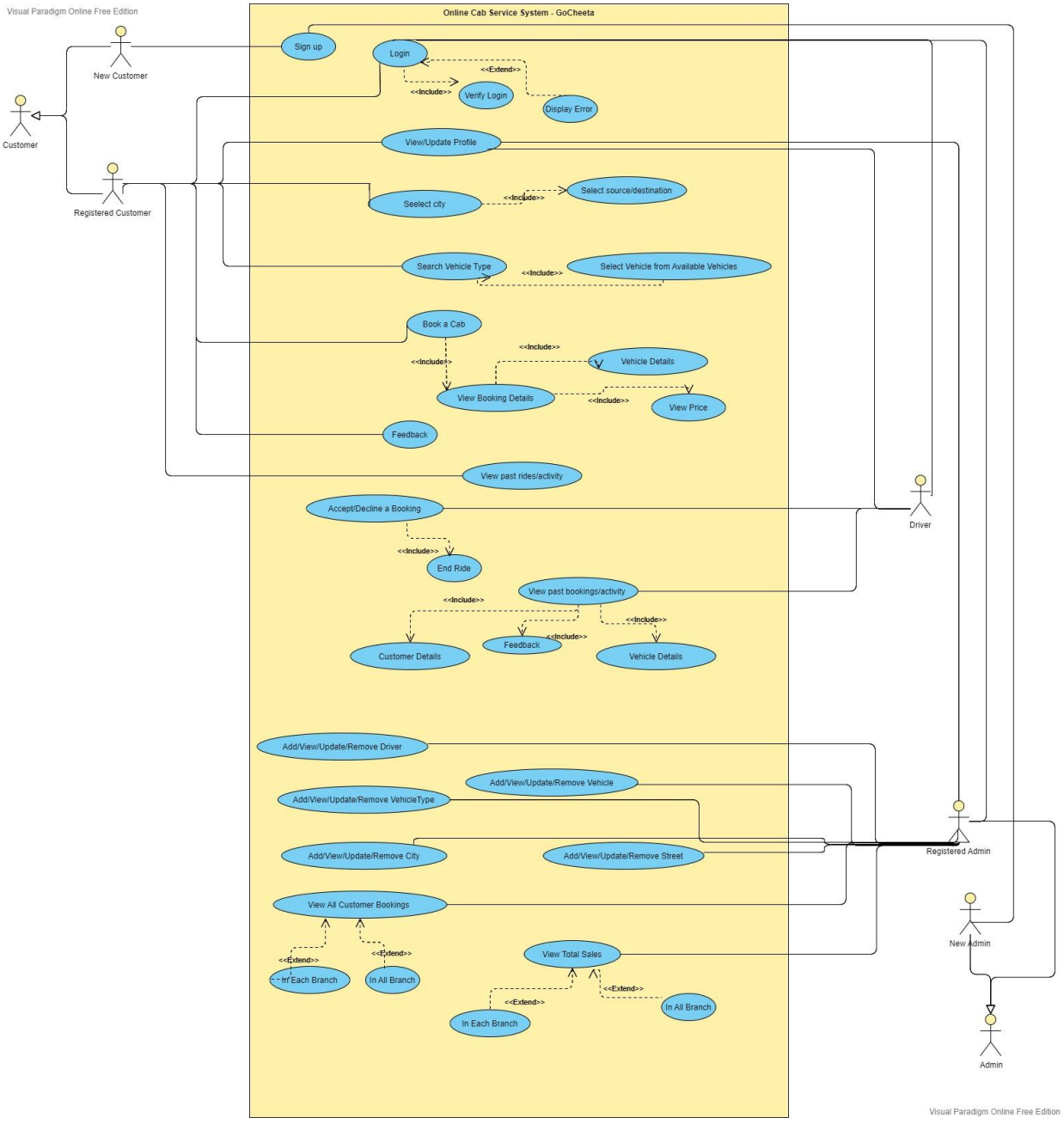
have a city. A city can have, atleast one street, if it is a newly registered city, or can many streets if it was already registered city.

- The BookingLocation class was created to store the distance of the possible source and destination within a city. Since the source and destination are streets, the BookingLocation class two Street class as attributes, for both source and destination, to retrieve further details of that streets. The BookingLocation and the Street class possess a Two-to-Many relationship, as two streets are anyway needed for a ride. A street can be involved in many rides if it is already registered street, or involve in atleast one ride if it is newly registered street.
- A Booking classes is created to store all the booking details of a ride. Since a booking involves a customer, driver, vehicle, source and destination, the Booking class has attributes of classes Customer, Driver, Vehicle and BookingLocation classes. It possesses a One-to-Many composition relationship with all as it cannot exist without them.

- **Use case diagram**

  Actors and use cases are used in the use case model. These ideas just serve as a tool for describing what the system should do and what should be done by players who exist outside the system (Wegmann and Genilloud, 2000). The include relationship denotes that when a base use case happens, the included use case happens along with it. The extend relationship denotes that when a base use case happens and when some conditions are met, the extended use case happens.

  Linl to my Use Case Diagram: https://postimg.cc/TyMpqgkV



Customer's perspective

- The use case diagram contains elements like actors, use case to represent the functionalities. The Customer is specialised into Registered Customer, where they can do all the use cases shown in the diagram above, and New Customer, where he can Sign up and proceed with other use cases as Registered Customer.
- A Registered Customer can login on their own, when the login happens the verification for Login happens along therefore, it is denoted with included

relationship. If the Login was invalid, it displays error message which is denoted as extended relationship.

- The Customer can view and update their profile.
- The Customer can select the city along with the source and destination from that selected city therefore it is denoted with include relationship.
- The Customer can search for a vehicle type, and select an available vehicle of that type. Therefore, they have an include relationship.
- The Customer can Book a cab, where they can view the full booking details along with the vehicle and pricing details. So they have an include relationship.
- The Customer can also give feedback for a ride regarding the driver or service or their experience.
- The Customer can also view their past booking/activity and their complete details.

Driver's perspective

- A Driver can login on their own, when the login happens the verification for Login happens along therefore, it is denoted with included relationship. If the Login was invalid, it displays error message which is denoted as extended relationship.
- The Driver can Accept or Decline Booking/Ride according to his preference. And when the ride is complete, they can End the ride as well, therefore, they have an included relationship.
- The Driver can view and update their profile.
- The Driver can view the past ride/activity which contains all the details of the customer and the vehicle along with the feedback. Therefore, they have an included relationship.

Admin's Perspective

- The Admin is specialised into Registered Admin, where they can do all the use cases shown in the diagram above, and New Admin, where he can Sign up and proceed with other use cases as Registered Admin.

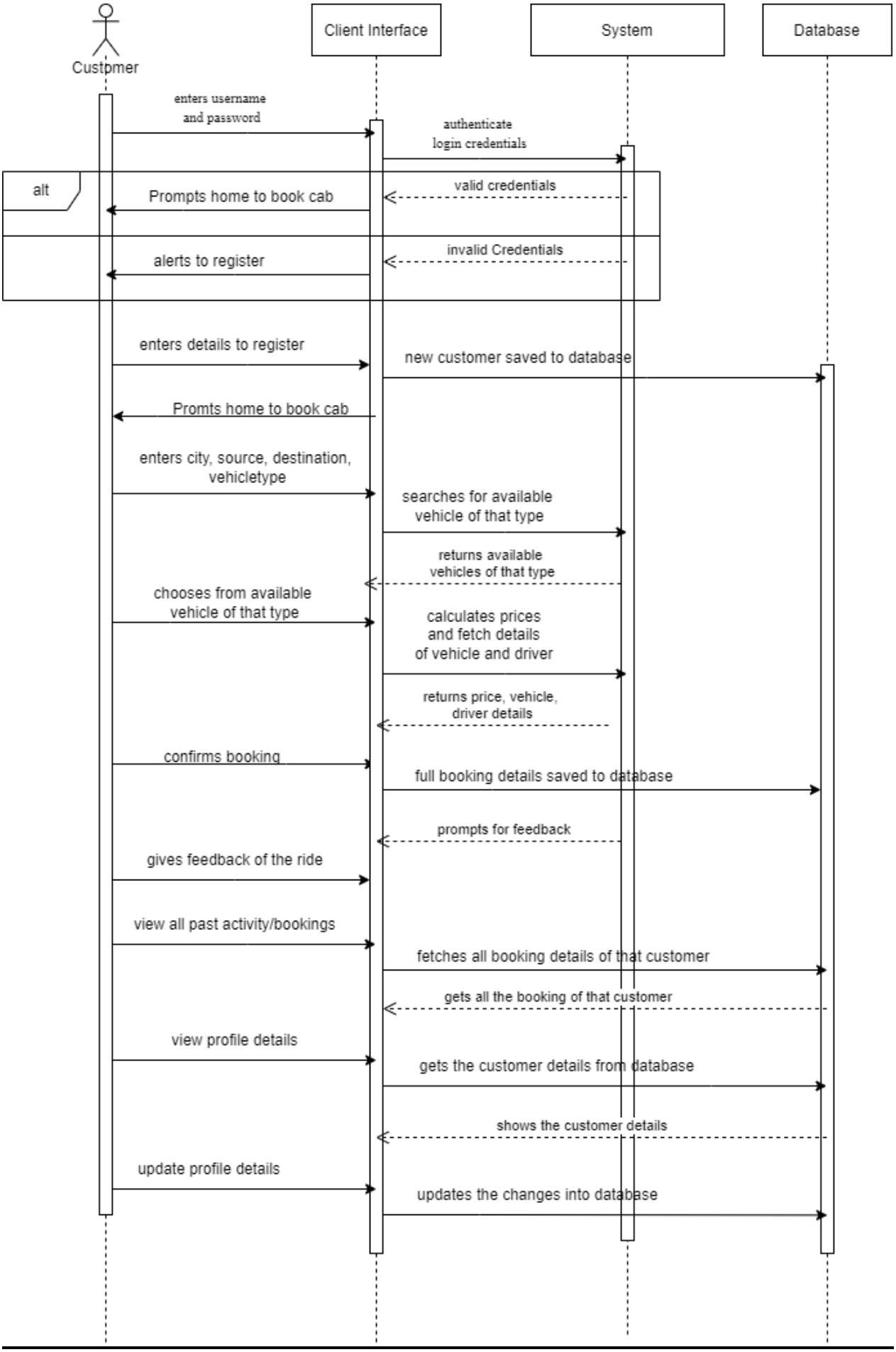
- A Registered Admin can login on their own, when the login happens the verification for Login happens along therefore, it is denoted with included relationship. If the Login was invalid, it displays error message which is denoted as extended relationship.
- The Admin can view and update their profile
- The Admin can add, view, update and delete a driver.
- The Admin can add, view, update and delete a vehicle.
- The Admin can add, view, update and delete a vehicle type.
- The Admin can add, view, update and delete a city.
- The Admin can add, view, update and delete a street.
- The Admin can view the customer bookings in each branch as well as all branches when they need. Therefore, they have an extended relationship.
- The Admin can view the total sales in each branch as well as all branches when they need. Therefore, they have an extended relationship.

- **Sequence diagram**

  Key UML components for representing the functioning of systems include sequence diagrams. For software development, documenting, understanding, and testing reasons, the drawings represent the process of control throughout object interaction (Sharp and Rountev, 2005).

  Customer's perspective

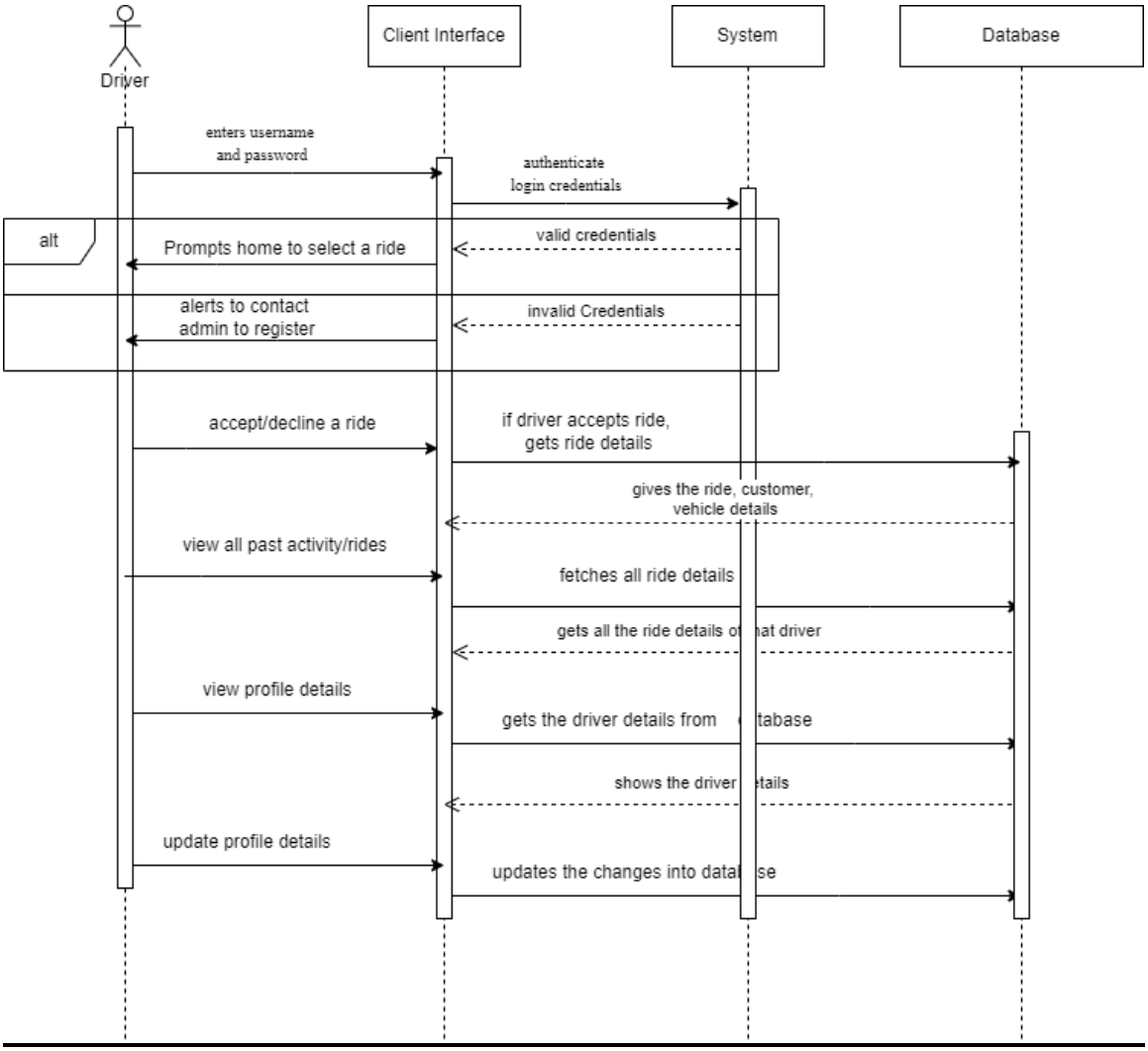
  Link to my Customer Sequence Diagram: https://postimg.cc/NLMfXdsh


Custpmer
Client Interface
System
Database
enters username and password
authenticate login credentials
alt
Prompts home to book cab
valid credentials
alerts to register
invalid Credentials
enters details to register
new customer saved to database
Promts home to book cab
enters city, source, destination, vehicletype
searches for available vehicle of that type
returns available vehicles of that type
chooses from available vehicle of that type
calculates prices and fetch details of vehicle and driver
returns price, vehicle, driver details
confirms booking
full booking details saved to database
prompts for feedback
gives feedback of the ride
view all past activity/bookings
fetches all booking details of that customer
gets all the booking of that customer
view profile details
gets the customer details from database
shows the customer details
update profile details
updates the changes into database

- The Customer, enters the username and password, on the interface, where the data is taken for authentication in the system. If the data is valid they will be prompted to the home page of the website to book a cab, if it is not valid, they will be alerted with an error message and to register first.
- If they enter details to register, the data is sent to the system, and to database to be stored. Then they are directed the home page as usual registered customer to book a cab.
- The customer can enter the city along with source, destination and vehicle type as they desire to place a booking. Then the system searches for an available vehicle of that type and shows the available vehicle of that type on interface.
- The customer can thereon choose from those available vehicles, the system calculates the price according the distance and prompts all details, the customer can then confirm the booking. Then the details of the booking is sent to database to be stored.
- Once the ride is complete, the customer can give feedback on their experience and service.
- The customer can view the past activity/booking and their full details. The system fetches all the booking details of that customer from database and shows on interface.
- The customer can view their profile details. The system fetches the customer details from database and shows on interface.
- The customer can update their profile details. The customer's altered data is taken into the system and stored in database.

Driver's perspective

Link to my Driver Sequence Diagram: https://postimg.cc/q6HhWty2

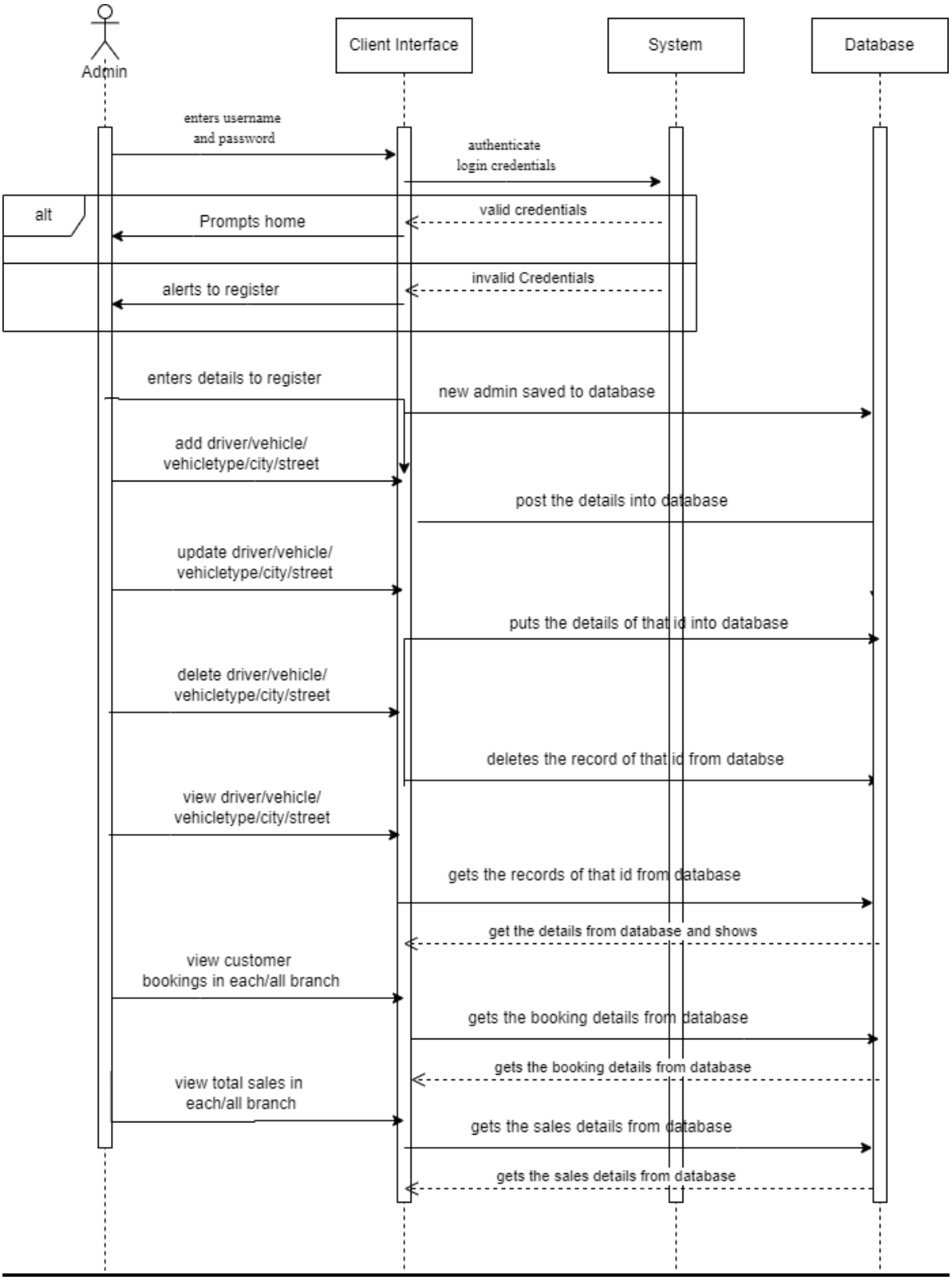


- The Driver, enters the username and password, on the interface, where the data is taken for authentication in the system. If the data is valid they will be prompted to the home page of the website, if it is not valid, they will be alerted with an error message and to contact admin to register first.
- The driver can accept or decline a booking according to his preferences. If the ride is accepted, the complete ride detail is shown.
- The driver can view all the activity. Then the system gets all the booking details of the customer, which has the vehicle and price details, along with the feedback the customer gave.

- The driver can view their profile details. The system fetches the driver details from database and shows on interface.
- The driver can update their profile details. The driver's altered data is taken into the system and stored in database.

Admin's perspective

Link to my Admin Sequence Diagram: https://postimg.cc/xJFQgS0w



- The admin, enters the username and password, on the interface, where the data is taken for authentication in the system. If the data is valid they will be prompted to the home page of the website, if it is not valid, they will be alerted with an error message and to register first.
- If they enter details to register, the data is sent to the system, and to database to be stored. Then they are directed the home page as usual registered admin.
- The admin can add driver, vehicle, vehicle type, city and street. The system posts all the details into the database.
- The admin update add driver, vehicle, vehicle type, city and street. The system puts all the details into the database.
- The admin delete add driver, vehicle, vehicle type, city and street. The system deletes all the details into the database.
- The admin can view driver, vehicle, vehicle type, city and street. The system gets all the details into the database.
- The admin can view all the customer bookings in each/all branches. The system gets all the data and shows on the interface.
- The admin can view all the total sales in each/all branches. The system gets all the data and shows on the interface.

(TASK C) DESIGN PATTERNS

Design patterns are attempts fixes for persistent issues. The majority of the literature shows object-oriented (OO) language implementations of design patterns. OO languages

are adequate for implementing the majority of patterns, but they lack the right structures to enable design pattern implementations manageable, local, and reusable (Aljasser, 2016).

Factory design pattern

When it is necessary to concentrate the collection of resources needed to manufacture, generate, or make an item, the factory pattern is the preferred design pattern. The primary purpose of the factory method is to separate, decouple, or specialize the creation of an object from its intended usage. The goal of the factory pattern is not to avoid object creation, but rather to stop object creation inside client code that already utilizes the object. The client code produced by using the factory design pattern is cleaner, clearer, and easier to comprehend than traditional code, which combines the generation of objects and their use in one file. Client code is limited to requests. And the advantage is, child class can select the kind of objects to produce using the factory method pattern. It encourages loose coupling by removing the requirement to integrate application-specific classes into the code. (Eskca and Bondugula, 2014).

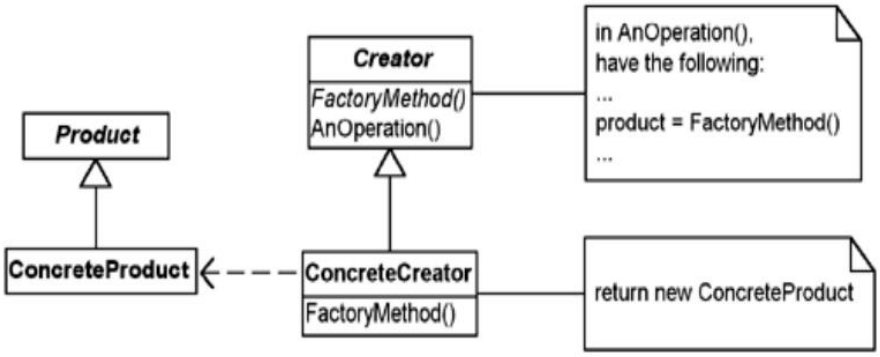


Fig 1: Above is a simplied diagram of the inside view of the factory pattern [Alan Shalloway, et al]

(Eskca and Bondugula, 2014)

Abstract factory design pattern

The abstract factory is the preferred design pattern when you need to generate a collection or group of related items without explicitly defining their particular classes. It is typically necessary to implement the abstract factory to make the

logic of selecting or switching between two or more closely related objects simpler when you have to make this decision. And the advantage is you may manage the classes of objects that an application generates by using the Abstract Factory design. Clients are isolated from implementation classes by a factory because it takes on the duty and controls the creation of product objects. Through their abstract interfaces, clients can control instances. (Eskca and Bondugula, 2014).

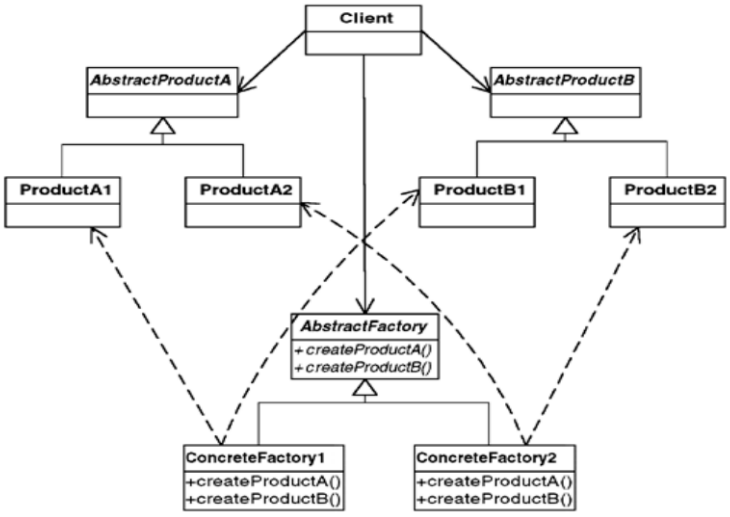
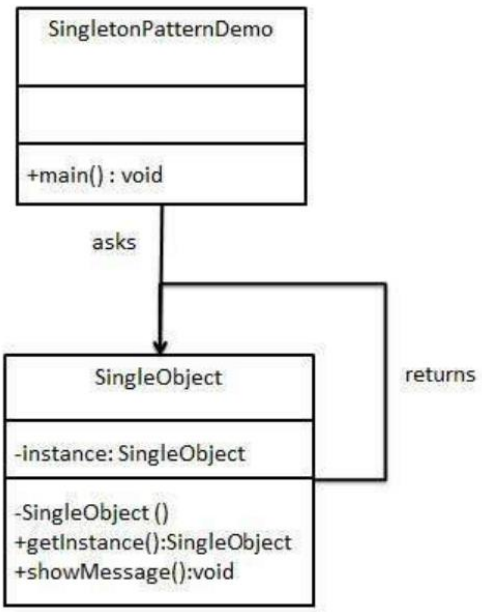


Fig 4: Standard, simplified view of the Abstract Factory pattern

(Eskca and Bondugula, 2014)

Singleton deign pattern

The purpose of the Singleton is to guarantee that a class has just one instance. A Singleton instance ought to be reusable, according to this. A Singleton instance may only be stored in a static (or global) context in order to be kept for future reuse. The benefit of using Singleton over global variables is that you may change your mind and handle any number of instances while being completely certain of the number of instances. (Stencel and Węgrzynowicz, 2008).



In my system, on the service, I have implemented, Singleton design pattern in my DBConnection.java in the dBUtils package,

```java
package project.onlinecabservice.service.dBUtils;

import java.sql.*;

/**
 *
 * @author Sumaiya
 */
public class DBConnection {

    //create an object of SingleObject
    private static DBConnection instance = new DBConnection();

    //make the constructor private so that this class cannot be instantiated
    private DBConnection(){}

    //Get the only object available
    public static DBConnection getInstance(){
       return instance;
    }

    //Databse Connection Variables
    static final String DATABASENAME = "cabservicedatabase";
    static final String URL = "jdbc:mysql://localhost:3306/" + DATABASENAME + "?autoReconnect=true&useSSL=false";
    static final String USERNAME = "root";
    static final String PASSWORD = "Root@123";

    public Statement dBConnectionInit() {
        Statement statement = null;
        try{
            // Open a connection
            Connection connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            System.out.println("Connected to Database");
            statement = connection.createStatement();

        } catch (SQLException e) {
            System.out.println(e);
        }
        return statement;
    }
}
```

and used the instance in where ever needed

```java
DBConnection dBInit = null;
Statement statement = null;
```

```java
y {
dBInit = DBConnection.getInstance();
statement = dBInit.dBConnectionInit();
```

and GSON.java in the service package

```
 * @author Sumaiya
 */
public class GSON {
    //create an object of SingleObject
    private static GSON instance = new GSON();

    //make the constructor private so that this class cannot be instantiated
    private GSON(){}

    //Get the only object available
    public static GSON getInstance(){
       return instance;
    }

    public Gson createGSON() {
        Gson gson = new GsonBuilder().create();

        return gson;
    }

}
```

and used the instance in where ever needed

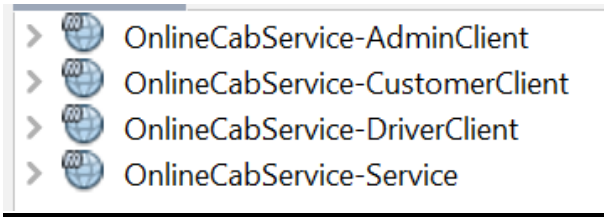
```
GSON gsonSingleton = null;
```

```
gsonSingleton = GSON.getInstance();
Gson gson = gsonSingleton.createGSON();
```

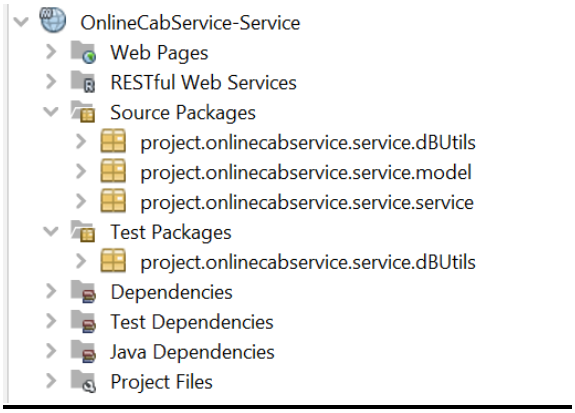
This reduces the complexity and keeps the code clean. We do not need to create the same object over and over again therefore, saves lots of memory.

(TASK D) SYSTEM

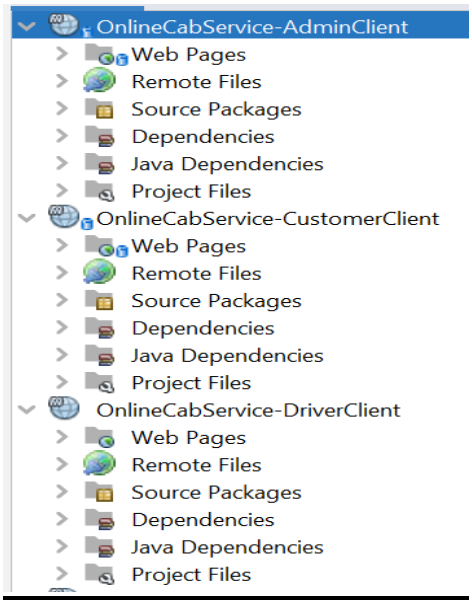
The GitHub Repository link to the entire coding of the system:
https://github.com/new-username-now/Online-Cab-Service-System



These are the four projects that builds this system, it has a service project which connects the MySQL database to the service side via Java DataBase Connectivity.
And uses REST services for web services. The Source Packages folder contains three packages. The model packages contain all the java model classes. The dBUtil package contains all the Get, Post, Put, Delete operations for all classes connected to database. The service package contains the service classes which implements REST service to do Get, Post, Put, Delete operations in client.

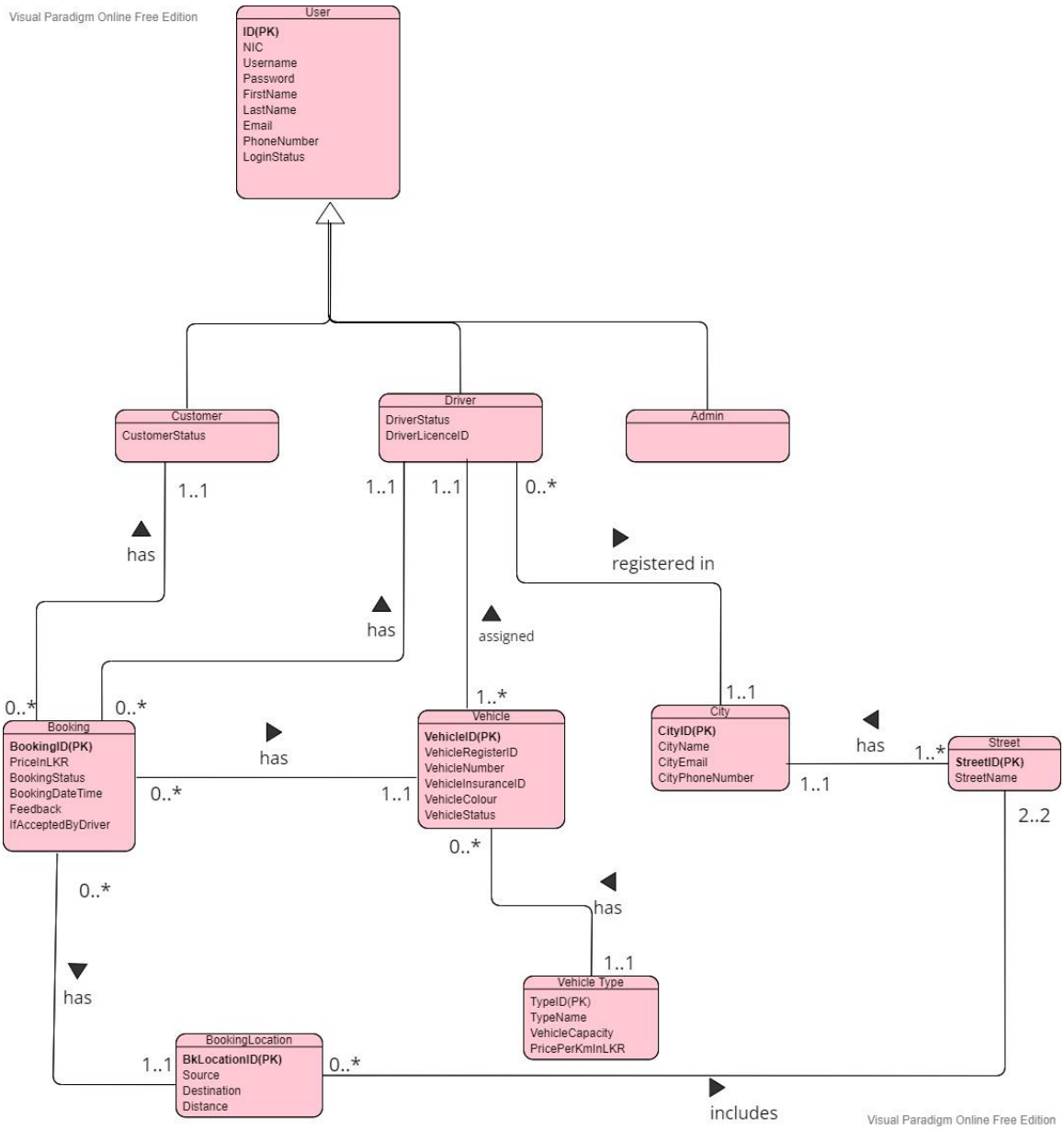


The below are the structure of the client projects. Where the Web Pages folder contains the HTML and JS files, relevant to their client, to build a user interface. The JS files fetches data from service side from database.



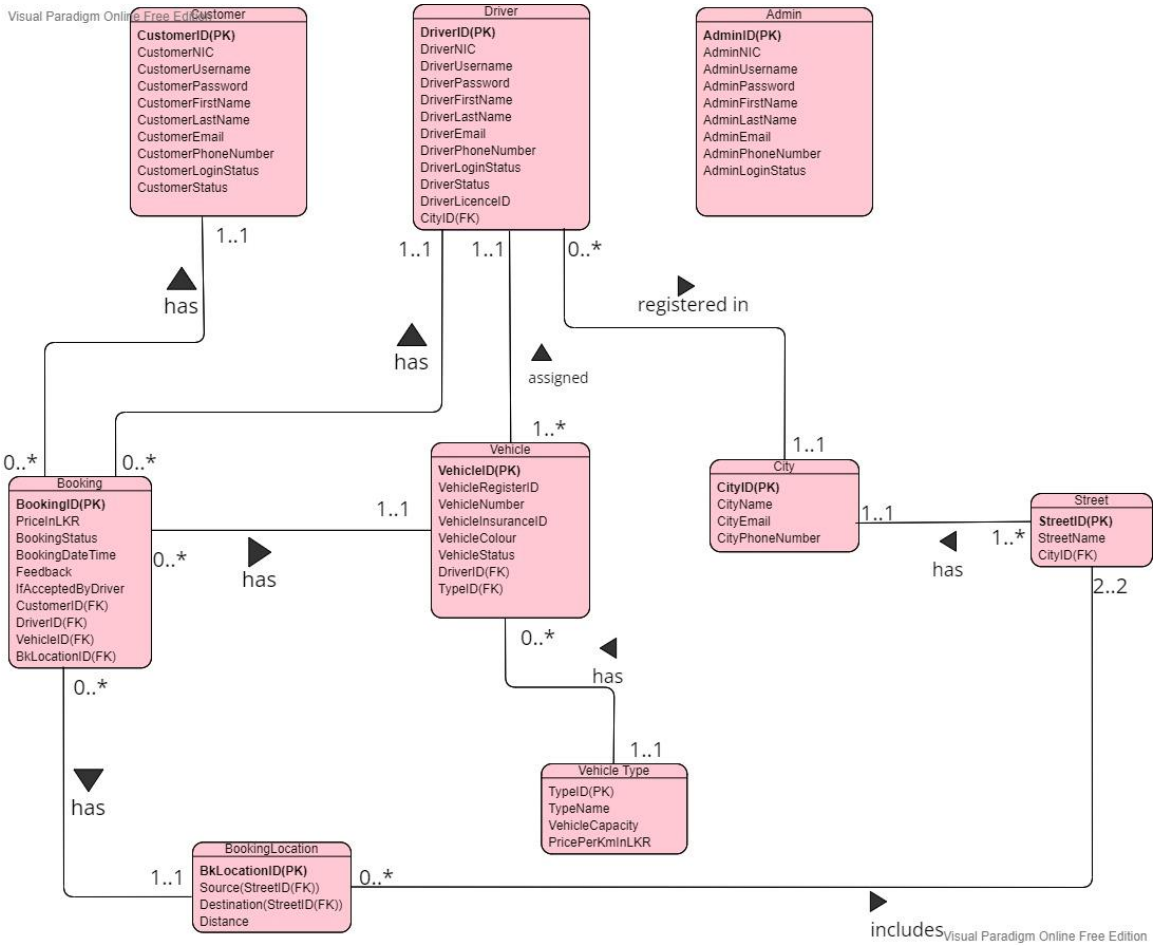
The Conceptual ER Diagram derived for creating the database.

Link to my Conceptual ER Diagram: https://postimg.cc/v1zSLLQk


Visual Paradigm Online Free Edition
User
ID(PK)
NIC
Username
Password
FirstName
LastName
Email
PhoneNumber
LoginStatus
Customer
CustomerStatus
Driver
DriverStatus
DriverLicenceID
Admin
1..1
1..1
1..1
0..*
has
has
assigned
registered in
0..*
0..*
1..*
1..1
Booking
BookingID(PK)
PriceInLKR
BookingStatus
BookingDateTime
Feedback
IfAcceptedByDriver
has
Vehicle
VehicleID(PK)
VehicleRegisterID
VehicleNumber
VehicleInsuranceID
VehicleColour
VehicleStatus
City
CityID(PK)
CityName
CityEmail
CityPhoneNumber
has
1..*
Street
StreetID(PK)
StreetName
0..*
1..1
1..1
2..2
0..*
0..*
has
1..1
Vehicle Type
TypeID(PK)
TypeName
VehicleCapacity
PricePerKmInLKR
has
BookingLocation
BkLocationID(PK)
Source
Destination
Distance
1..1
0..*
includes
Visual Paradigm Online Free Edition

The Logical ER Diagram derived for creating the database.

Link to my Logical ER Diagram: https://postimg.cc/f3BB0zCh



(TASK E) TESTING

Test Driven Development(TDD), is using unit test cases for an item as a starting point, a system is implemented in this approach. When test cases are coded and executed, more objects and functions become necessary (George and Williams, 2004).

Since test driven development is a process of development where the test cases are written with respect to the functionality requirements, I coded for the test cases for all methods in each classes existing as per the functionality requirements. Initially all my test cases failed as expected, as no methods were implemented properly. And as along, I refactored the methods as per the requirements were met, and the test cases passed expectedly one by one.

Test Plan

Test case class	Methods implemented	Expected	Result	Pass/Fail
City.java	None	All Fail	All Fail	Pass
Admin.java	None	All Fail	All Fail	Pass
VehicleType.java	None	All Fail	All Fail	Pass
Customer.java	None	All Fail	All Fail	Pass
Street.java	None	All Fail	All Fail	Pass
Driver.java	None	All Fail	All Fail	Pass
Vehicle.java	None	All Fail	All Fail	Pass
BookingLocation.java	None	All Fail	All Fail	Pass
Booking.java	None	All Fail	All Fail	Pass
City.java	getCity()	getCity pass, others fail	getCity pass, others fail	Pass
Admin.java	getAdmin()	getAdmin pass, others fail	getAdmin pass, others fail	Pass
VehicleType.java	getVehicleType()	getVehicleType pass, others fail	getVehicleType pass, others fail	Pass
Customer.java	getCustomer()	getCustomer pass, others fail	getCustomer pass, others fail	Pass
Street.java	getStreet()	getStreet pass, others fail	getStreet pass, others fail	Pass
Driver.java	getDriver()	getDriver pass, others fail	getDriver pass, others fail	Pass
Vehicle.java	getVehicle()	getVehicle pass, others fail	getVehicle pass, others fail	Pass
BookingLocation.java	getBookingLocation()	getBookingLocation pass, others fail	getBookingLocation pass, others fail	Pass
Booking.java	getBooking()	getBooking pass, others fail	getBooking pass, others fail	Pass
City.java	updateCity()	getCity, updateCity pass, others fail	getCity, updateCity pass, others fail	Pass

Admin.java	updateAdmin()	getAdmin, updateAdmin pass, others fail	getAdmin, updateAdmin pass, others fail	Pass
VehicleType.java	updateVehicleType( )	getVehicleType, updateVehicleType pass, others fail	getVehicleType, updateVehicleType pass, others fail	Pass
Customer.java	updateCustomer()	getCustomer, updateCustomer pass, others fail	getCustomer, updateCustomer pass, others fail	Pass
Street.java	updateStreet()	getStreet, updateStreet pass, others fail	getStreet, updateStreet pass, others fail	Pass
Driver.java	updateDriver()	getDriver, updateDriver pass, others fail	getDriver, updateDriver pass, others fail	Pass
Vehicle.java	updateVehicle()	getVehicle, updateVehicle pass, others fail	getVehicle, updateVehicle pass, others fail	Pass
BookingLocation.java	updateBookingLocation()	getBookingLocation, updateBookingLocation pass, others fail	getBookingLocation, updateBookingLocation pass, others fail	Pass
Booking.java	updateBooking()	getBooking, updateBooking pass, others fail	getBooking, updateBooking pass, others fail	Pass
City.java	addCity()	getCity, updateCity, addCity pass, others fail	getCity, updateCity, addCity pass, others fail	Pass
Admin.java	addAdmin()	getAdmin, updateAdmin,	getAdmin, updateAdmin,	Pass

		addAdmin pass, others fail	addAdmin pass, others fail	
VehicleType.java	addVehicleType()	getVehicleType, updateVehicleType, addVehicleType pass, others fail	getVehicleType, updateVehicleType, addVehicleType pass, others fail	Pass
Customer.java	addCustomer()	getCustomer, updateCustomer, addCustomer pass, others fail	getCustomer, updateCustomer, addCustomer pass, others fail	Pass
Street.java	addStreet()	getStreet, updateStreet, addStreet pass, others fail	getStreet, updateStreet, addStreet pass, others fail	Pass
Driver.java	addDriver()	getDriver, updateDriver, addDriver pass, others fail	getDriver, updateDriver, addDriver pass, others fail	Pass
Vehicle.java	addVehicle()	getVehicle, updateVehicle, addVehicle pass, others fail	getVehicle, updateVehicle, addVehicle pass, others fail	Pass
BookingLocation.java	addBookingLocation()	getBookingLocation, updateBookingLocation, addBookingLocation pass, others fail	getBookingLocation, updateBookingLocation, addBookingLocation pass, others fail	Pass
Booking.java	addBooking()	getBooking, updateBooking,	getBooking, updateBooking,	Pass

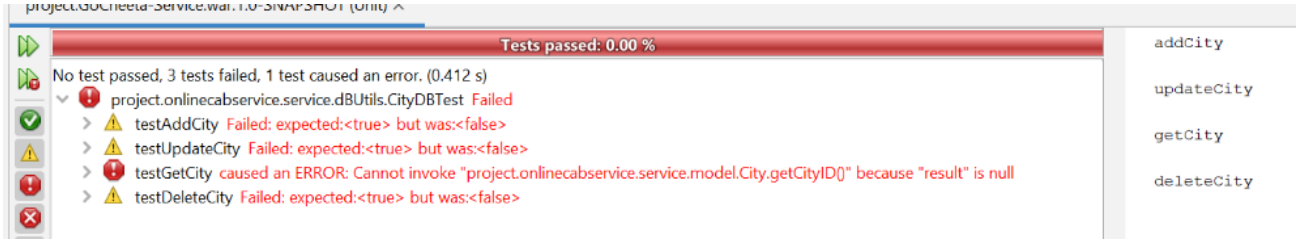
		addBooking pass, others fail	addBooking pass, others fail	
City.java	deleteCity()	All pass	All pass	Pass
Admin.java	deleteAdmin()	All pass	All pass	Pass
VehicleType.java	deleteVehicleType()	All pass	All pass	Pass
Customer.java	deleteCustomer()	All pass	All pass	Pass
Street.java	deleteStreet()	All pass	All pass	Pass
Driver.java	deleteDriver()	All pass	All pass	Pass
Vehicle.java	deleteVehicle()	All pass	All pass	Pass
BookingLocation.java	deleteBookingLocation()	All pass	All pass	Pass
Booking.java	deleteBooking()	All pass	All pass	Pass

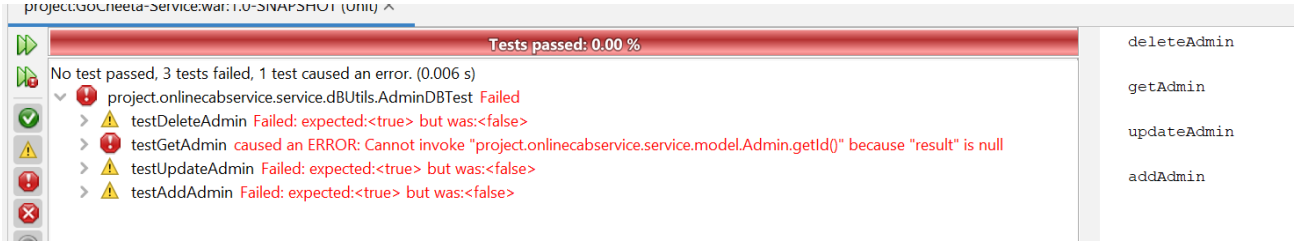
To derive test data for the system, I had populated my database accordingly to check with my test cases.

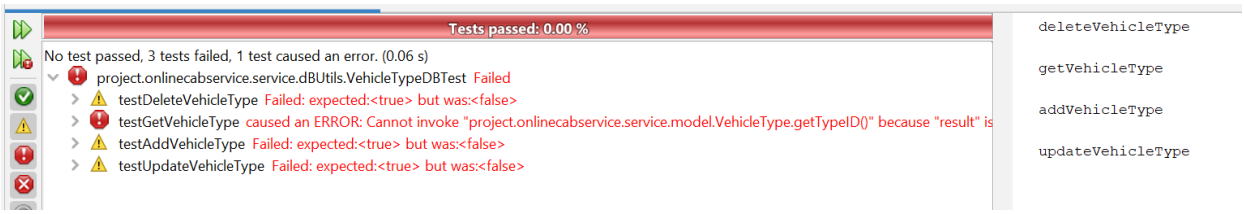
The test classes created are all available in the Git repository created for this project:

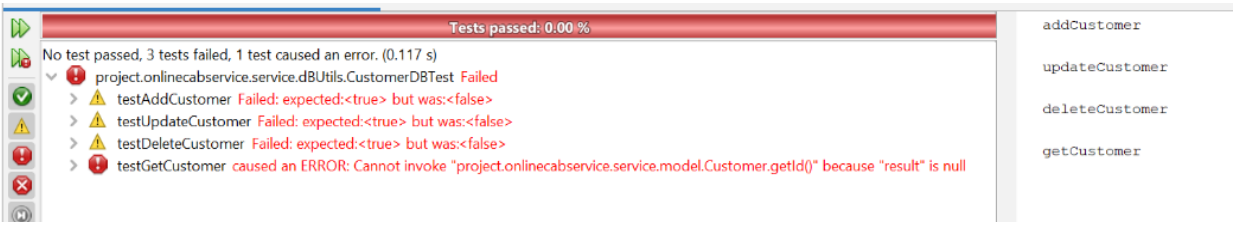
- (Git Repository URL) https://github.com/new-username-now/Online-Cab-Service-System.git
- (Git Repository URL for test classes) https://github.com/new-username-now/Online-Cab-Service-System/tree/main/OnlineCabService-Service/src/test/java/project/onlinecabservice/service/dBUtils

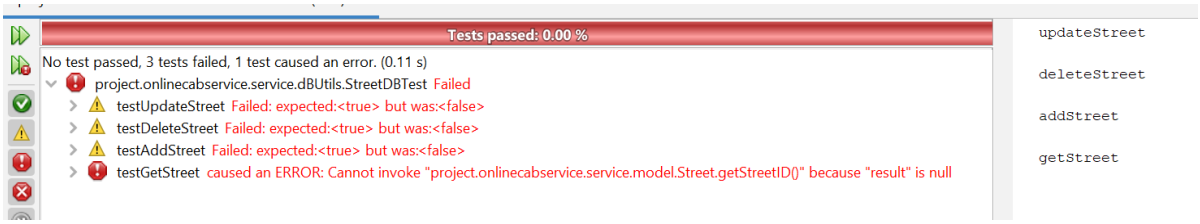
The screenshots below, shows the expected test results of the flow of test driven development. Initially all test cases of all classes failed as expected as no methods were implemented properly.

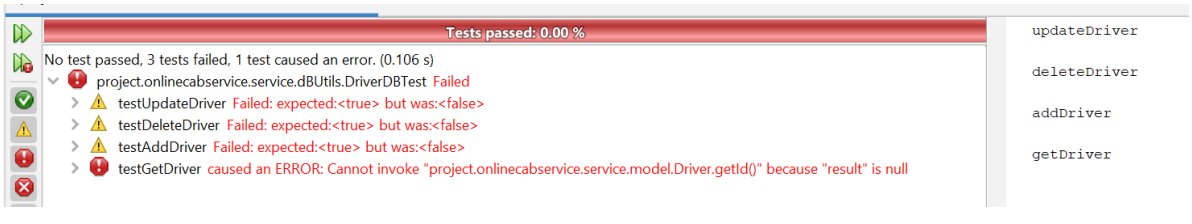

Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.412 s)
project.onlinecabservice.service.dBUtils.CityDBTest Failed
testAddCity Failed: expected:<true> but was:<false>
testUpdateCity Failed: expected:<true> but was:<false>
testGetCity caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.City.getCityID()" because "result" is null
testDeleteCity Failed: expected:<true> but was:<false>
addCity
updateCity
getCity
deleteCity

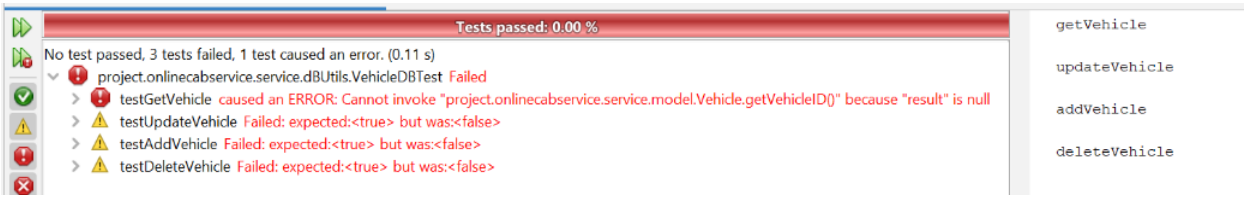

Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.006 s)
project.onlinecabservice.service.dBUtils.AdminDBTest Failed
testDeleteAdmin Failed: expected:<true> but was:<false>
testGetAdmin caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.Admin.getId()" because "result" is null
testUpdateAdmin Failed: expected:<true> but was:<false>
testAddAdmin Failed: expected:<true> but was:<false>
deleteAdmin
getAdmin
updateAdmin
addAdmin

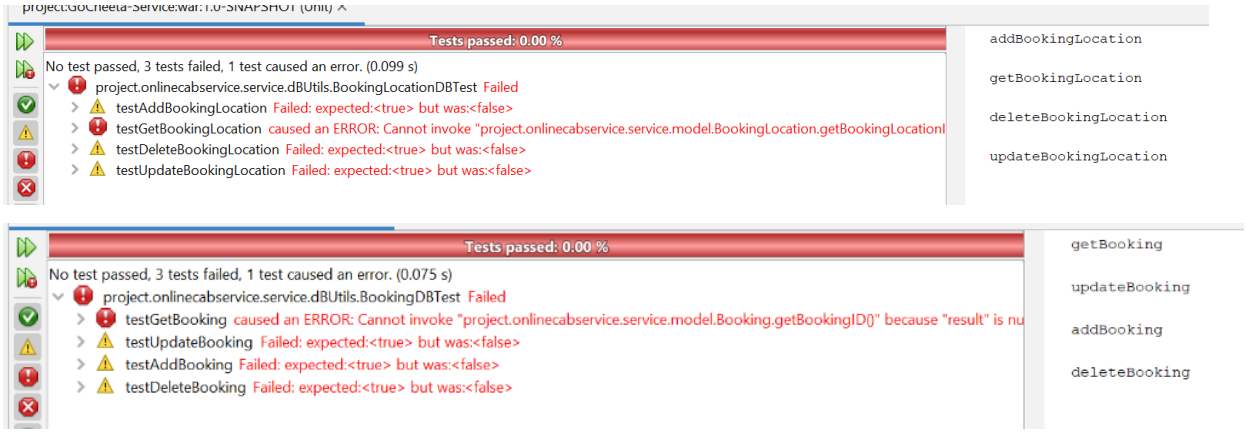

Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.06 s)
project.onlinecabservice.service.dBUtils.VehicleTypeDBTest Failed
testDeleteVehicleType Failed: expected:<true> but was:<false>
testGetVehicleType caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.VehicleType.getTypeID()" because "result" is
testAddVehicleType Failed: expected:<true> but was:<false>
testUpdateVehicleType Failed: expected:<true> but was:<false>
deleteVehicleType
getVehicleType
addVehicleType
updateVehicleType


Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.117 s)
project.onlinecabservice.service.dBUtils.CustomerDBTest Failed
testAddCustomer Failed: expected:<true> but was:<false>
testUpdateCustomer Failed: expected:<true> but was:<false>
testDeleteCustomer Failed: expected:<true> but was:<false>
testGetCustomer caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.Customer.getId()" because "result" is null
addCustomer
updateCustomer
deleteCustomer
getCustomer

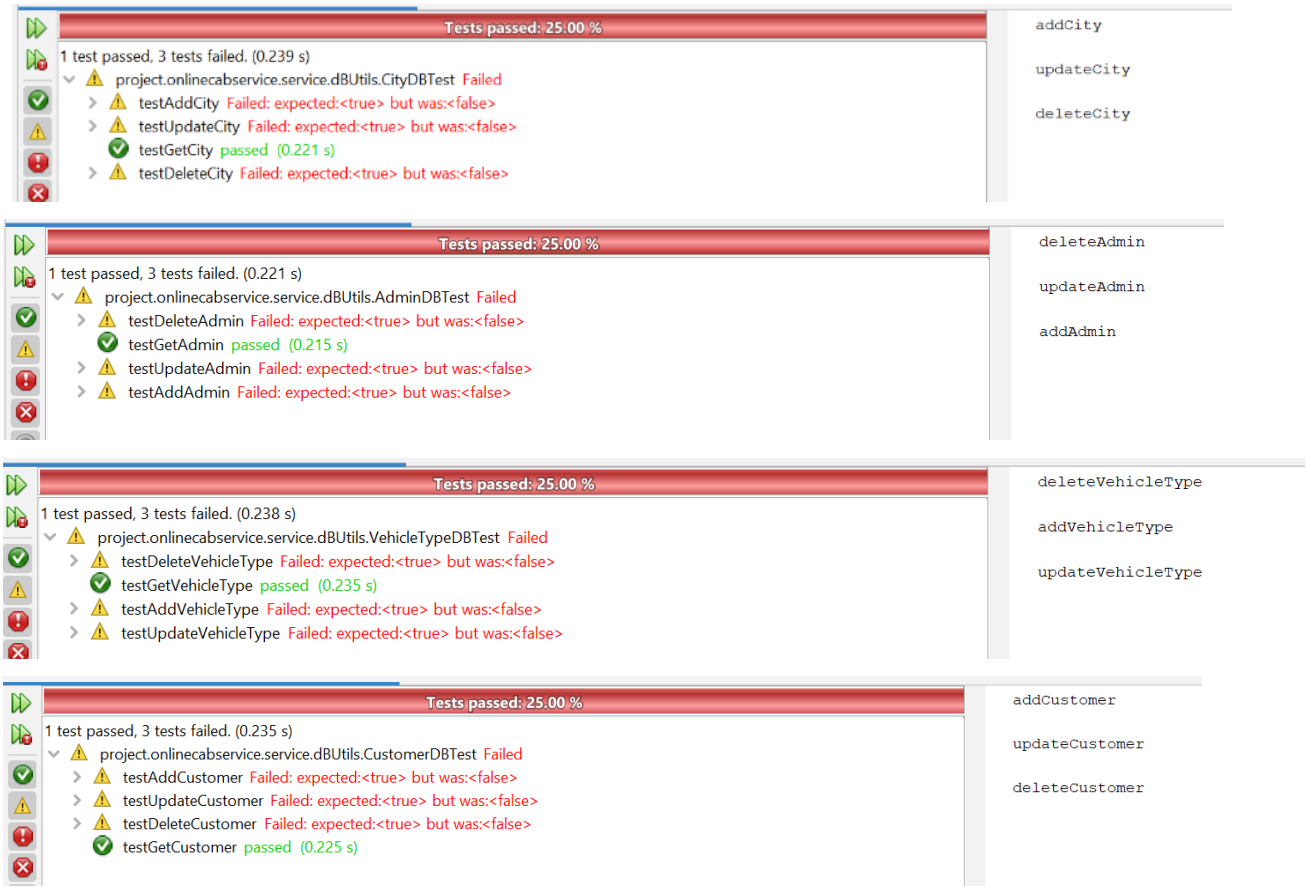

Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.11 s)
project.onlinecabservice.service.dBUtils.StreetDBTest Failed
testUpdateStreet Failed: expected:<true> but was:<false>
testDeleteStreet Failed: expected:<true> but was:<false>
testAddStreet Failed: expected:<true> but was:<false>
testGetStreet caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.Street.getStreetID()" because "result" is null
updateStreet
deleteStreet
addStreet
getStreet

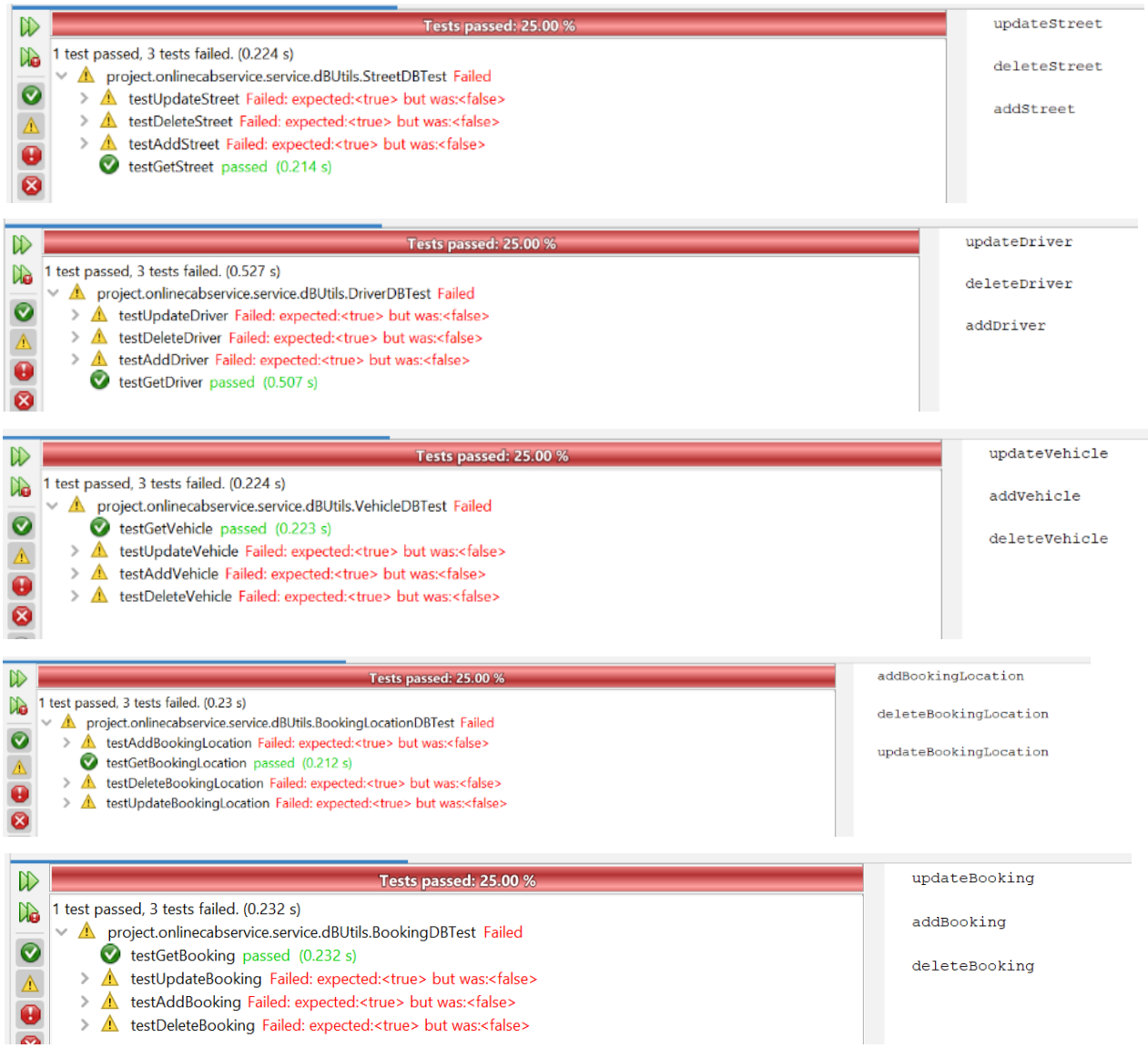

Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.106 s)
project.onlinecabservice.service.dBUtils.DriverDBTest Failed
testUpdateDriver Failed: expected:<true> but was:<false>
testDeleteDriver Failed: expected:<true> but was:<false>
testAddDriver Failed: expected:<true> but was:<false>
testGetDriver caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.Driver.getId()" because "result" is null
updateDriver
deleteDriver
addDriver
getDriver


Tests passed: 0.00 %
No test passed, 3 tests failed, 1 test caused an error. (0.11 s)
project.onlinecabservice.service.dBUtils.VehicleDBTest Failed
testGetVehicle caused an ERROR: Cannot invoke "project.onlinecabservice.service.model.Vehicle.getVehicleID()" because "result" is null
testUpdateVehicle Failed: expected:<true> but was:<false>
testAddVehicle Failed: expected:<true> but was:<false>
testDeleteVehicle Failed: expected:<true> but was:<false>
getVehicle
updateVehicle
addVehicle
deleteVehicle

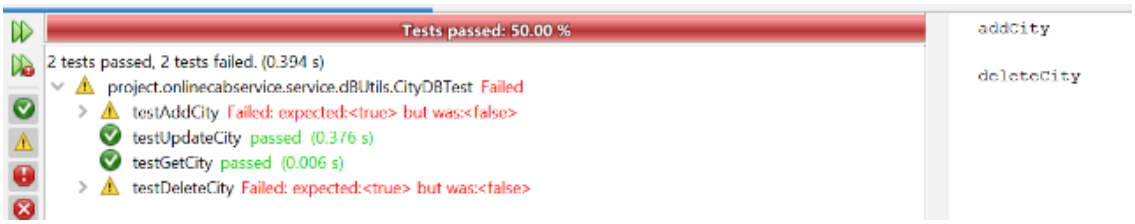


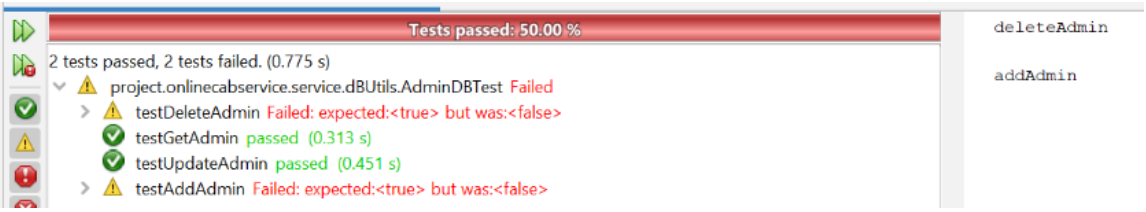
After refactoring, the get methods of all classes, the test cases for get methods in all classes passed as expected. Others failed as expected.

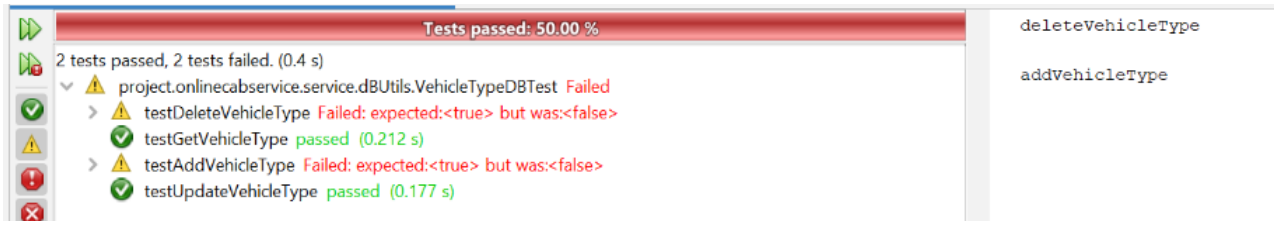


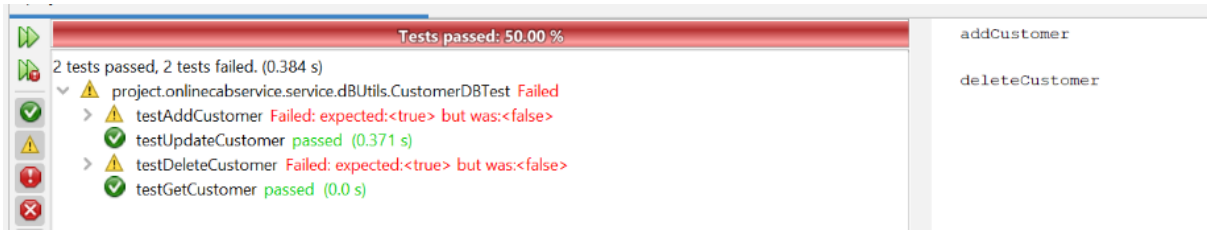


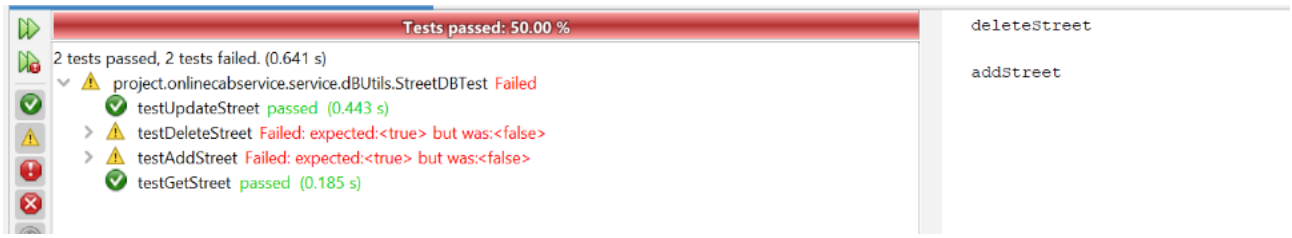
After implementing the update method, update as well as get test cases for all classes passed as expected. Others failed as expected.

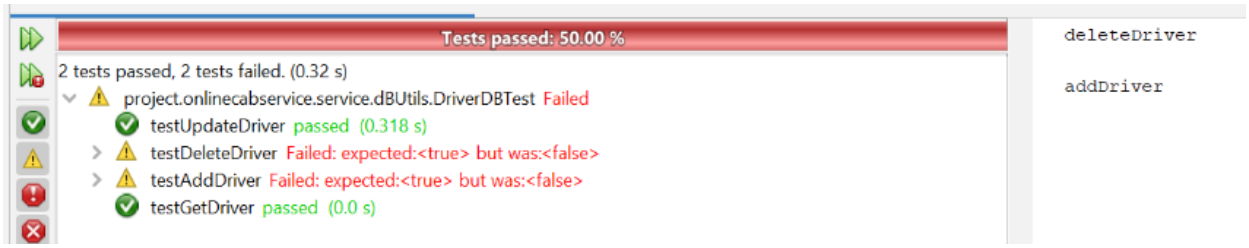

Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.394 s)
project.onlinecabservice.service.dBUtils.CityDBTest Failed
testAddCity Failed: expected:<true> but was:<false>
testUpdateCity passed (0.376 s)
testGetCity passed (0.006 s)
testDeleteCity Failed: expected:<true> but was:<false>
addCity
deleteCity

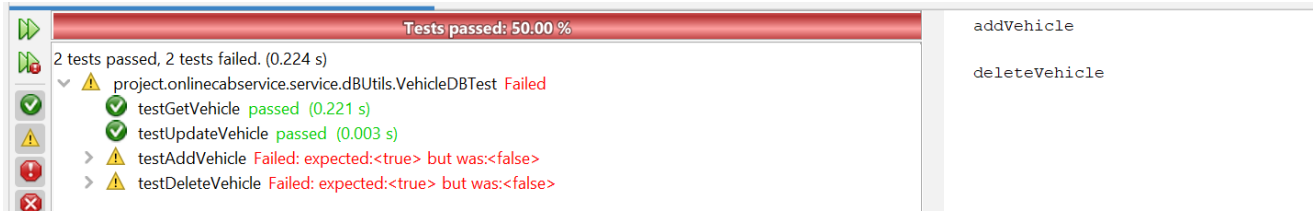

Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.775 s)
project.onlinecabservice.service.dBUtils.AdminDBTest Failed
testDeleteAdmin Failed: expected:<true> but was:<false>
testGetAdmin passed (0.313 s)
testUpdateAdmin passed (0.451 s)
testAddAdmin Failed: expected:<true> but was:<false>
deleteAdmin
addAdmin

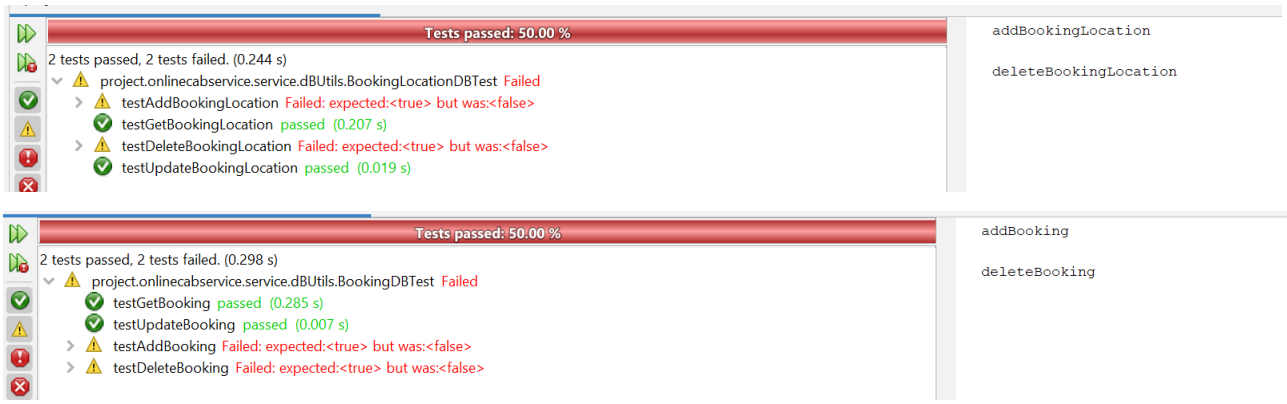

Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.4 s)
project.onlinecabservice.service.dBUtils.VehicleTypeDBTest Failed
testDeleteVehicleType Failed: expected:<true> but was:<false>
testGetVehicleType passed (0.212 s)
testAddVehicleType Failed: expected:<true> but was:<false>
testUpdateVehicleType passed (0.177 s)
deleteVehicleType
addVehicleType


Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.384 s)
project.onlinecabservice.service.dBUtils.CustomerDBTest Failed
testAddCustomer Failed: expected:<true> but was:<false>
testUpdateCustomer passed (0.371 s)
testDeleteCustomer Failed: expected:<true> but was:<false>
testGetCustomer passed (0.0 s)
addCustomer
deleteCustomer

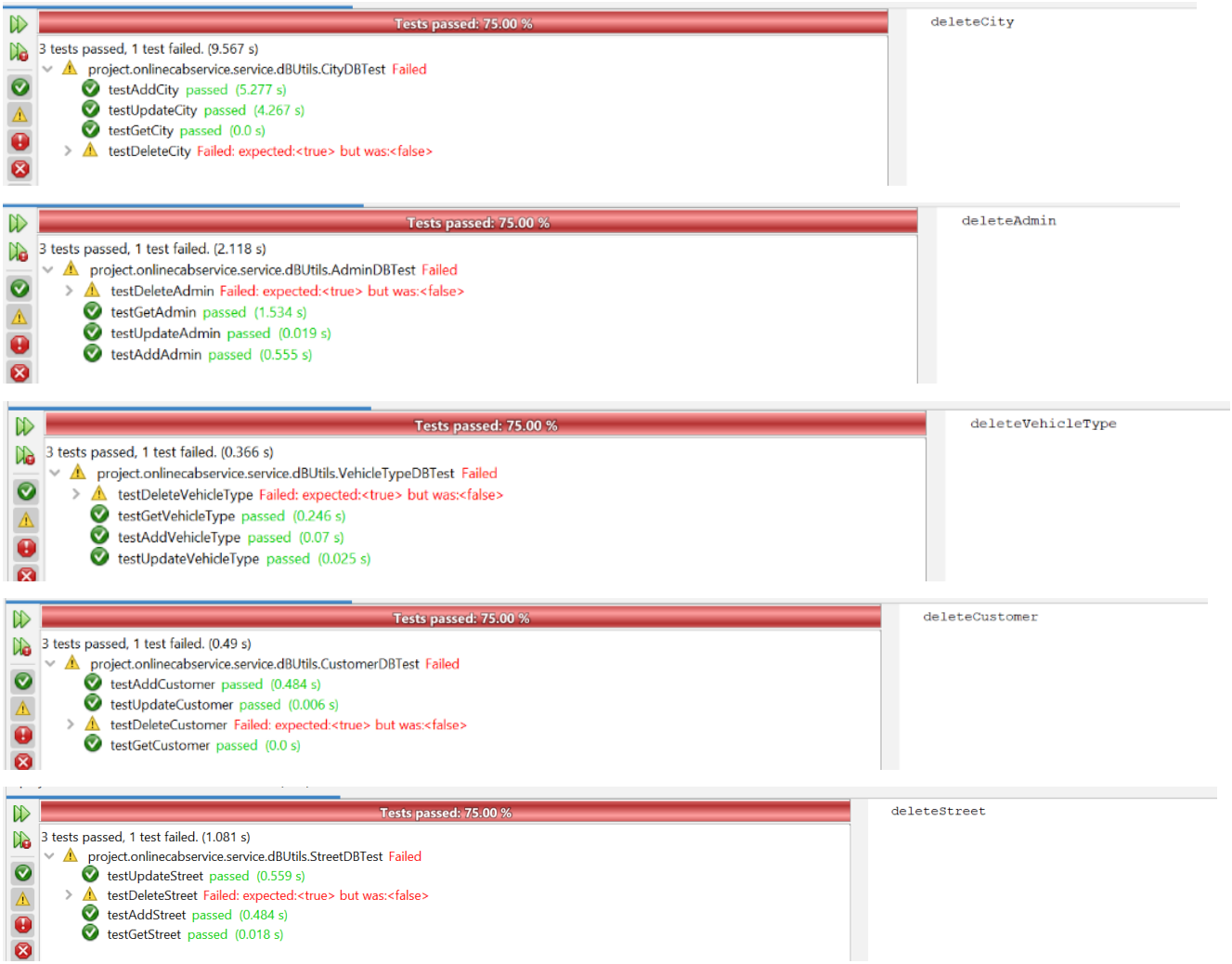

Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.641 s)
project.onlinecabservice.service.dBUtils.StreetDBTest Failed
testUpdateStreet passed (0.443 s)
testDeleteStreet Failed: expected:<true> but was:<false>
testAddStreet Failed: expected:<true> but was:<false>
testGetStreet passed (0.185 s)
deleteStreet
addStreet


Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.32 s)
project.onlinecabservice.service.dBUtils.DriverDBTest Failed
testUpdateDriver passed (0.318 s)
testDeleteDriver Failed: expected:<true> but was:<false>
testAddDriver Failed: expected:<true> but was:<false>
testGetDriver passed (0.0 s)
deleteDriver
addDriver

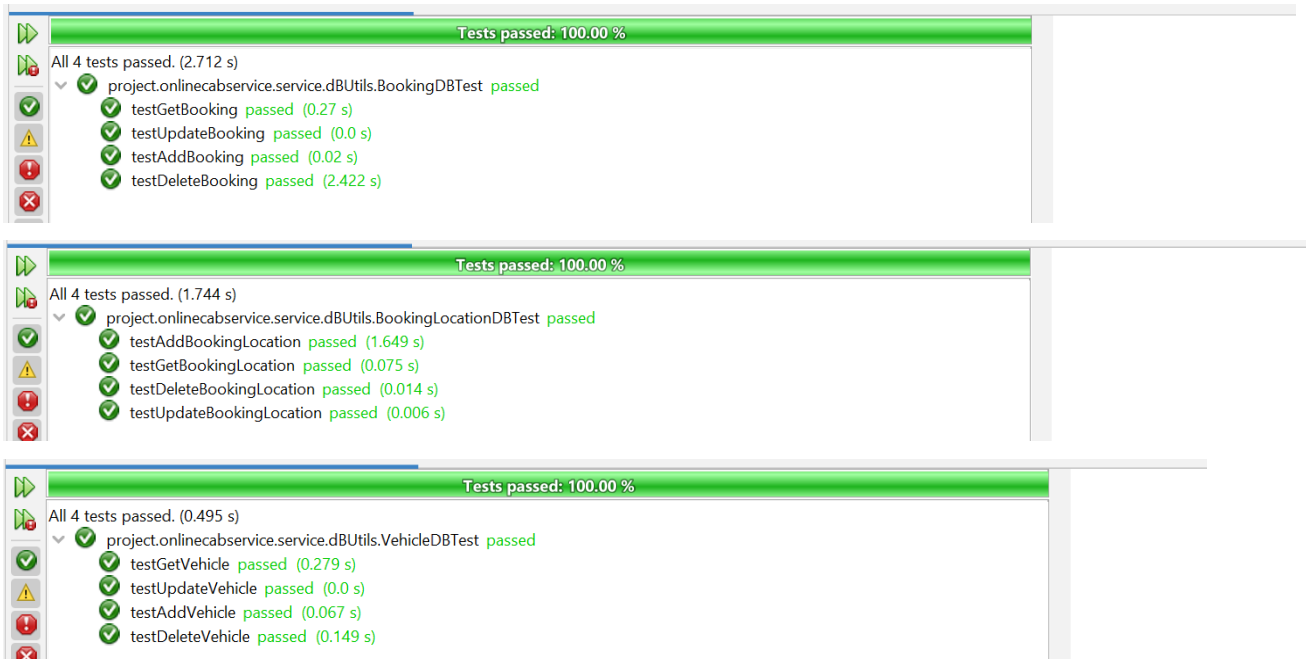

Tests passed: 50.00 %
2 tests passed, 2 tests failed. (0.224 s)
project.onlinecabservice.service.dBUtils.VehicleDBTest Failed
testGetVehicle passed (0.221 s)
testUpdateVehicle passed (0.003 s)
testAddVehicle Failed: expected:<true> but was:<false>
testDeleteVehicle Failed: expected:<true> but was:<false>
addVehicle
deleteVehicle

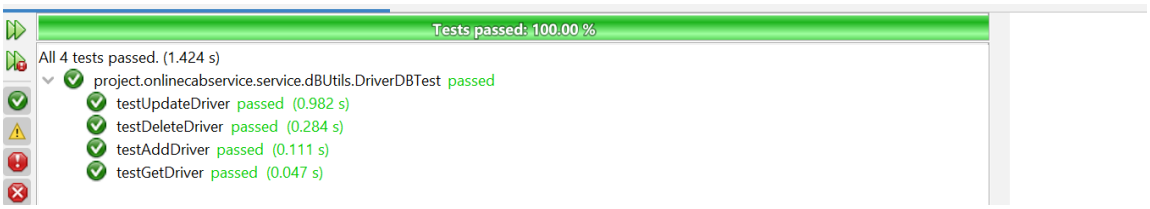


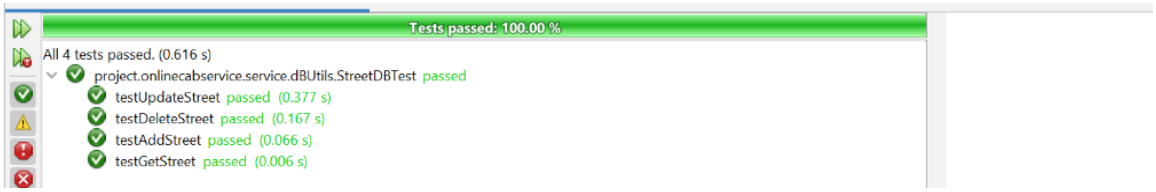
After implementing the add method, add, update as well as get test cases for all classes passed as expected. Others failed as expected.

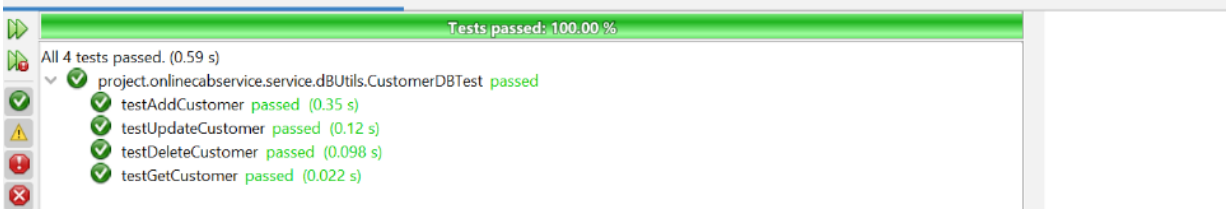


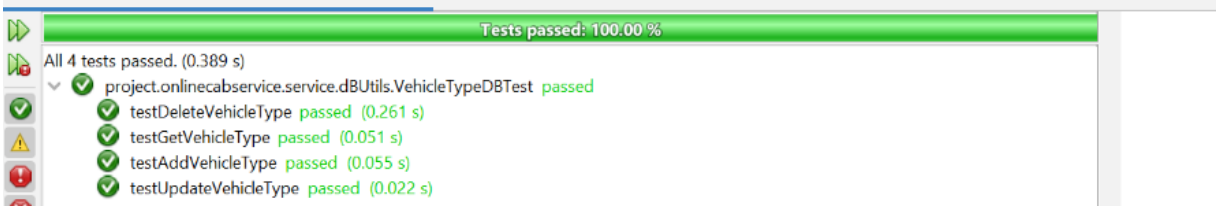
After implementing the delete method, delete, add, update as well as get test cases for all classes passed as expected.

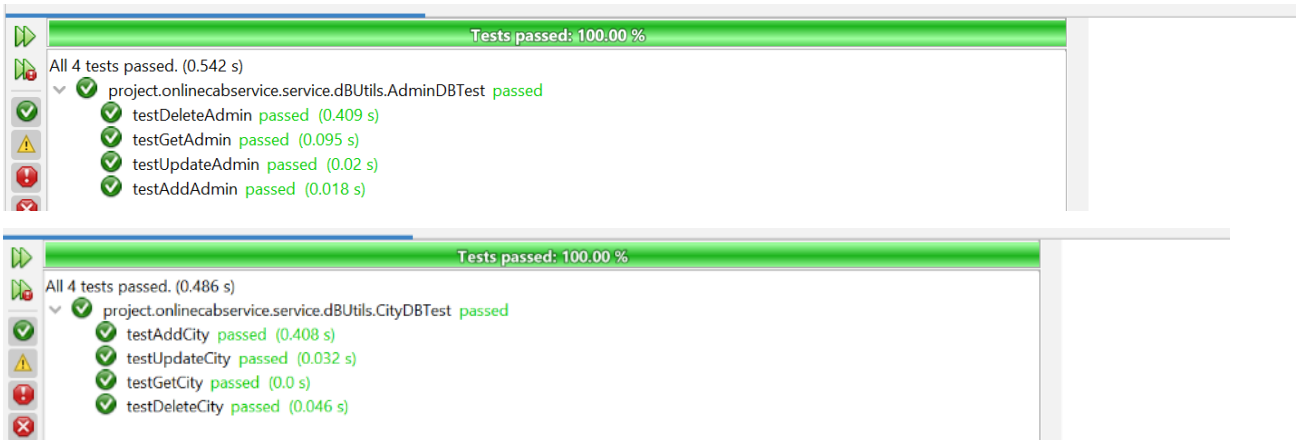



Tests passed: 100.00 %
All 4 tests passed. (1.424 s)
project.onlinecabservice.service.dBUtils.DriverDBTest passed
testUpdateDriver passed (0.982 s)
testDeleteDriver passed (0.284 s)
testAddDriver passed (0.111 s)
testGetDriver passed (0.047 s)


Tests passed: 100.00 %
All 4 tests passed. (0.616 s)
project.onlinecabservice.service.dBUtils.StreetDBTest passed
testUpdateStreet passed (0.377 s)
testDeleteStreet passed (0.167 s)
testAddStreet passed (0.066 s)
testGetStreet passed (0.006 s)


Tests passed: 100.00 %
All 4 tests passed. (0.59 s)
project.onlinecabservice.service.dBUtils.CustomerDBTest passed
testAddCustomer passed (0.35 s)
testUpdateCustomer passed (0.12 s)
testDeleteCustomer passed (0.098 s)
testGetCustomer passed (0.022 s)

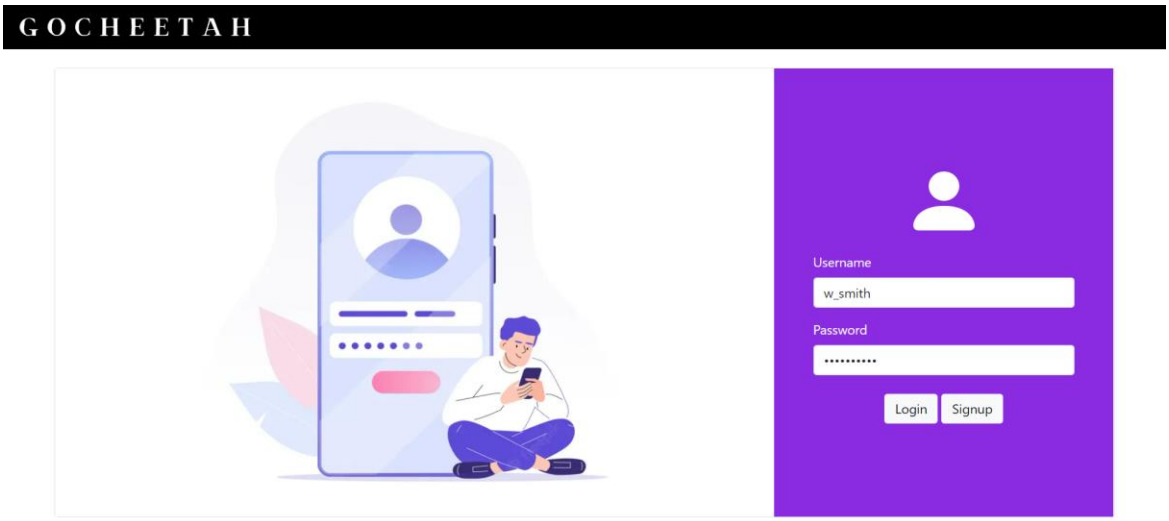

Tests passed: 100.00 %
All 4 tests passed. (0.389 s)
project.onlinecabservice.service.dBUtils.VehicleTypeDBTest passed
testDeleteVehicleType passed (0.261 s)
testGetVehicleType passed (0.051 s)
testAddVehicleType passed (0.055 s)
testUpdateVehicleType passed (0.022 s)



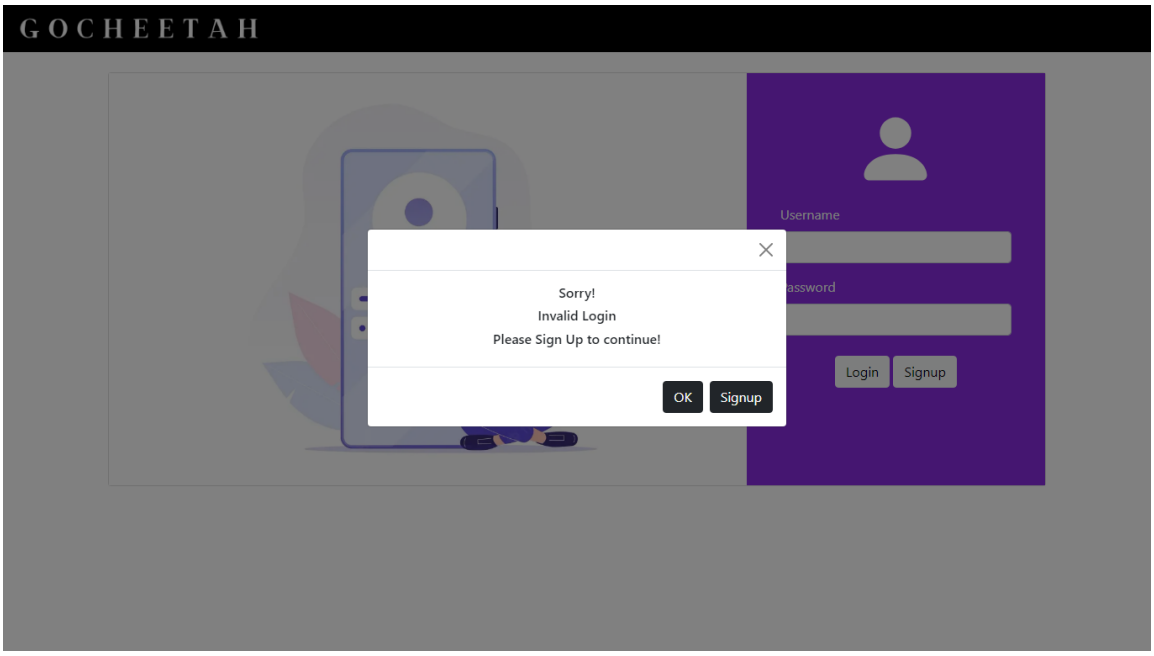
Lessons learned from implementing test driven development, was a guaranteed way, that the implemented methods were as per required. This makes sure that all requirements are met when implementing the methods. Therefore, we could not miss out on any requirement. Whereas, when the requirement changes or if any alterations need to be added in the service side, it was a hectic process as we have to change from scratch.

(TASK F) USER AND TECHNICAL DOCUMENTATION

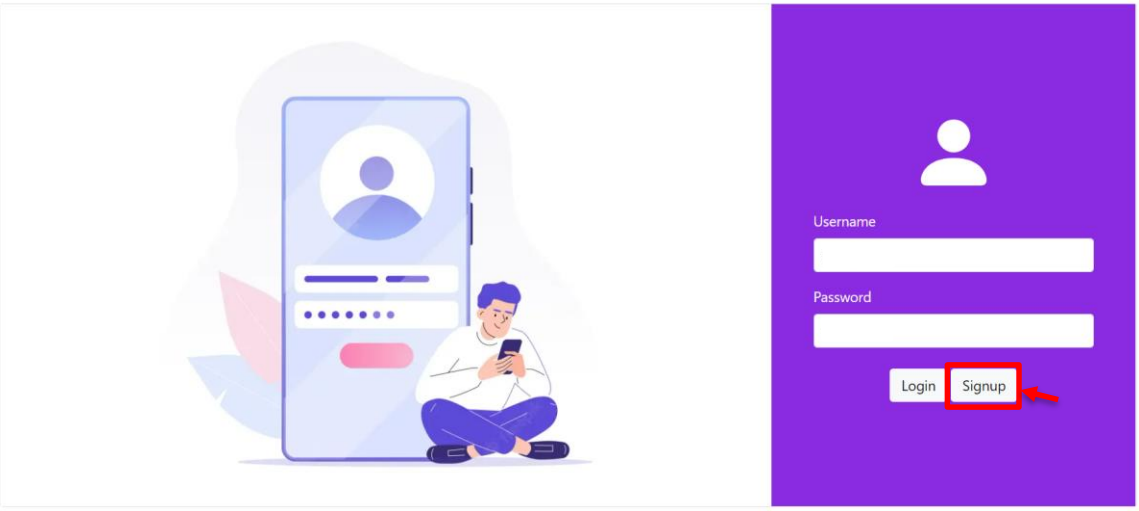
Customer Client



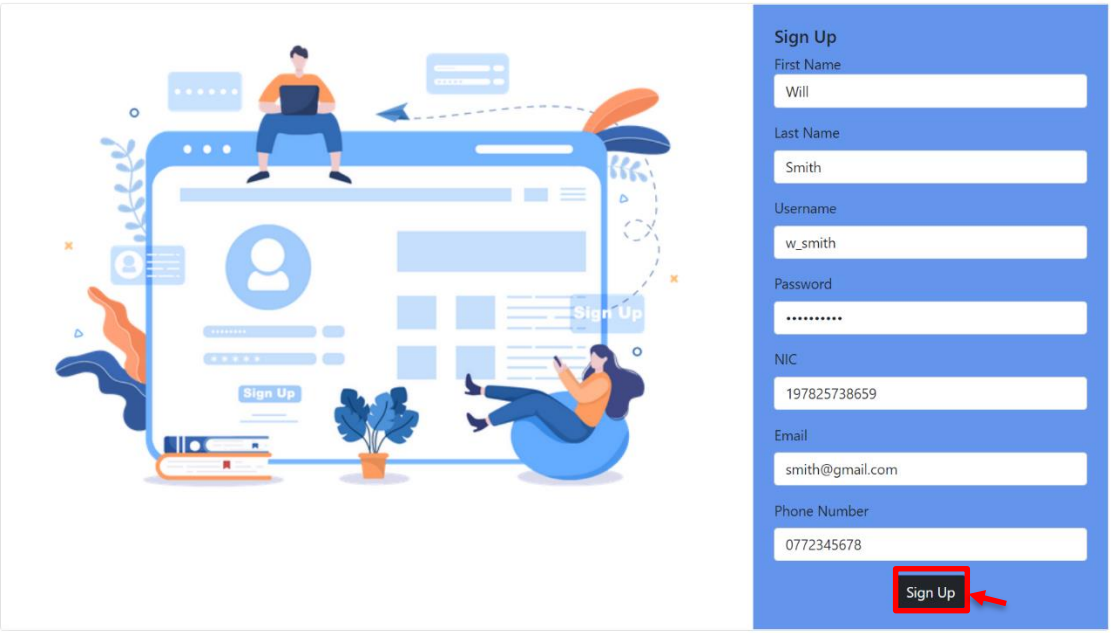
The customer can login to the website with their username and password.



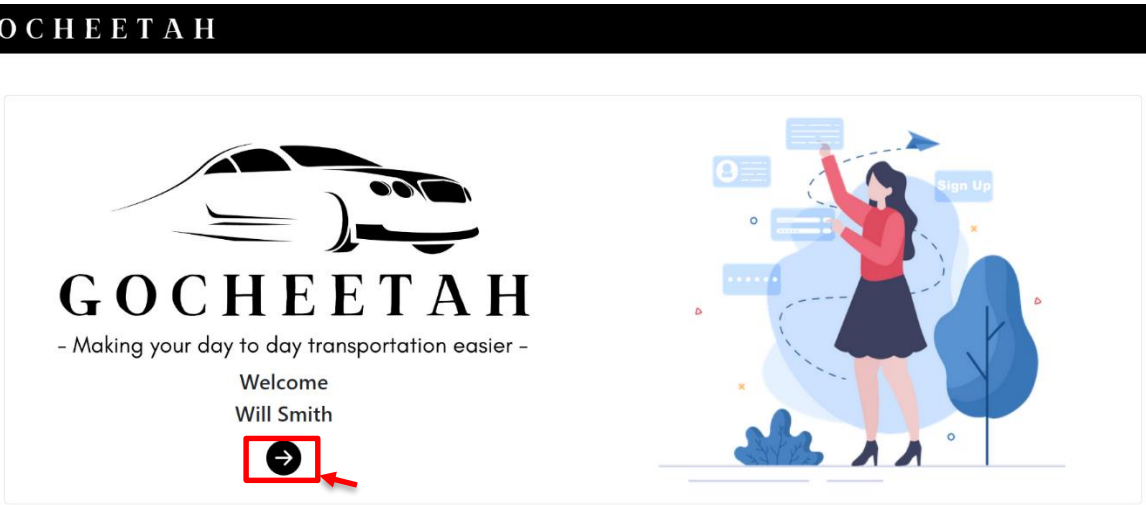
If the credential was wrong, they will be prompted with an alert error message, saying that it is an invalid login, they will have to sign up to use the website. They can either click Sigup in the pop up or click Signup as below shown image to signup.



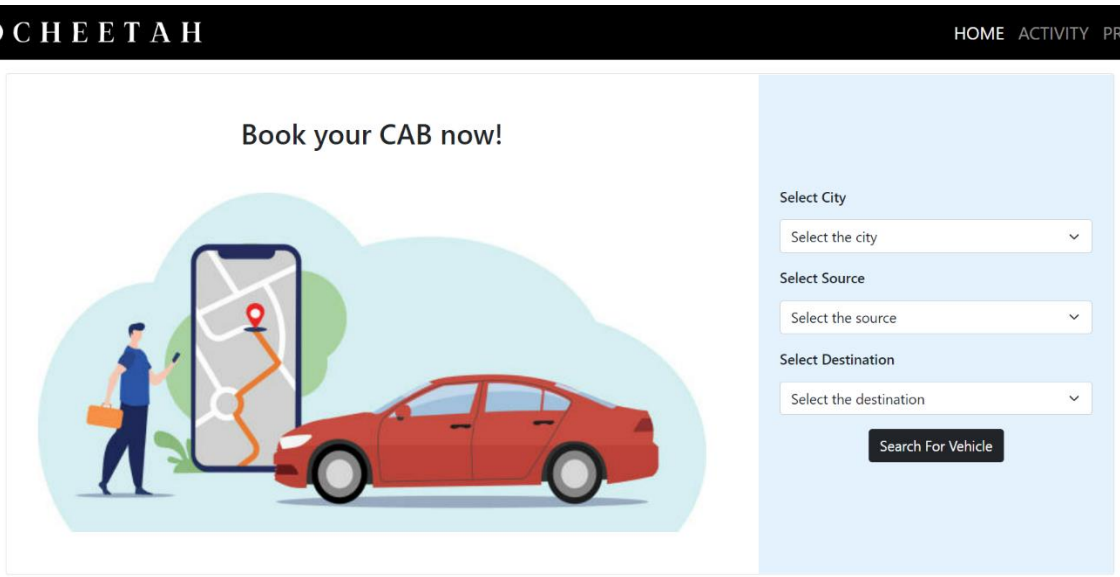
The customer can give their details in order to create an account. And they can use the username and password they have given to login to the website anytime.



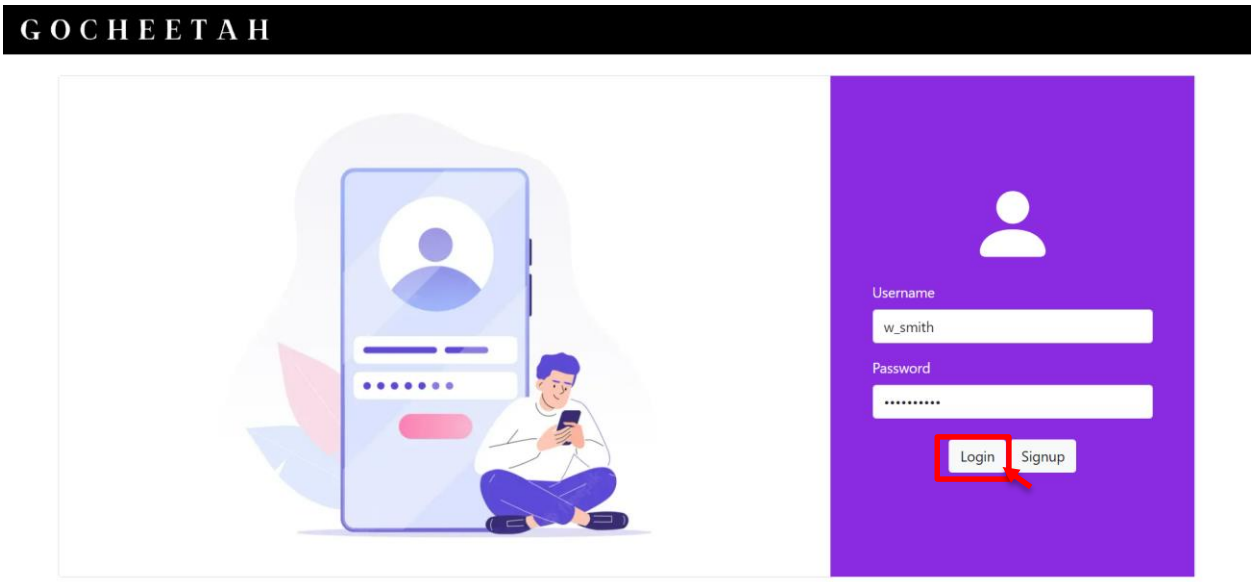
The newly registered customers will then be directed to a welcome page. Where the page will greet the customer with their name.



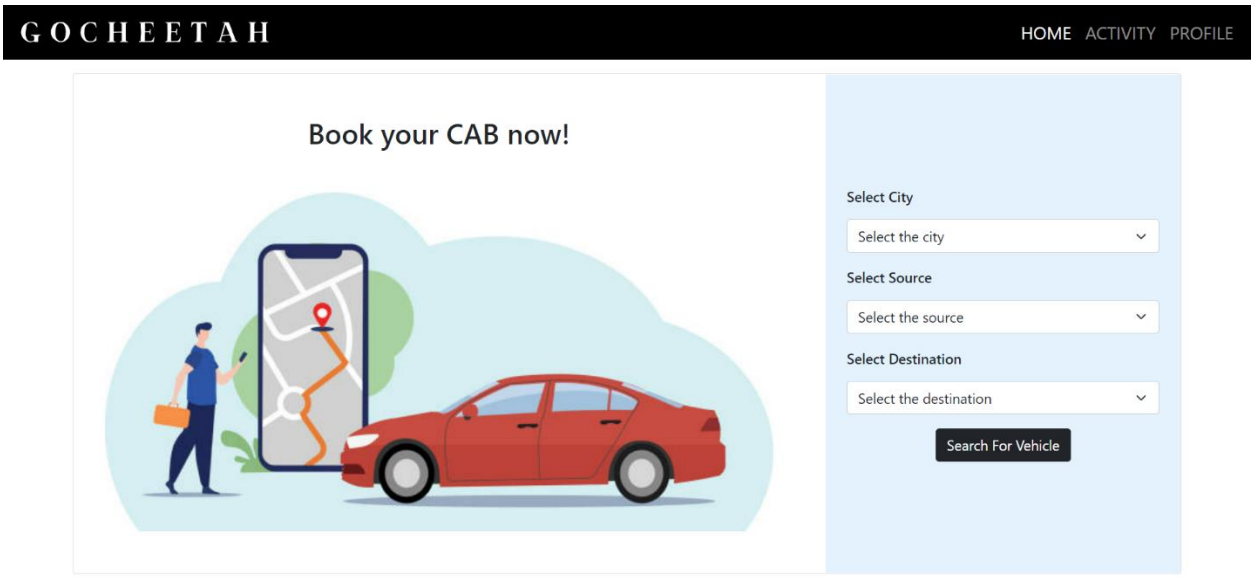
Once they click the arrow button shown above to begin, they will be directed to the home page where they can start to give details to book a cab.



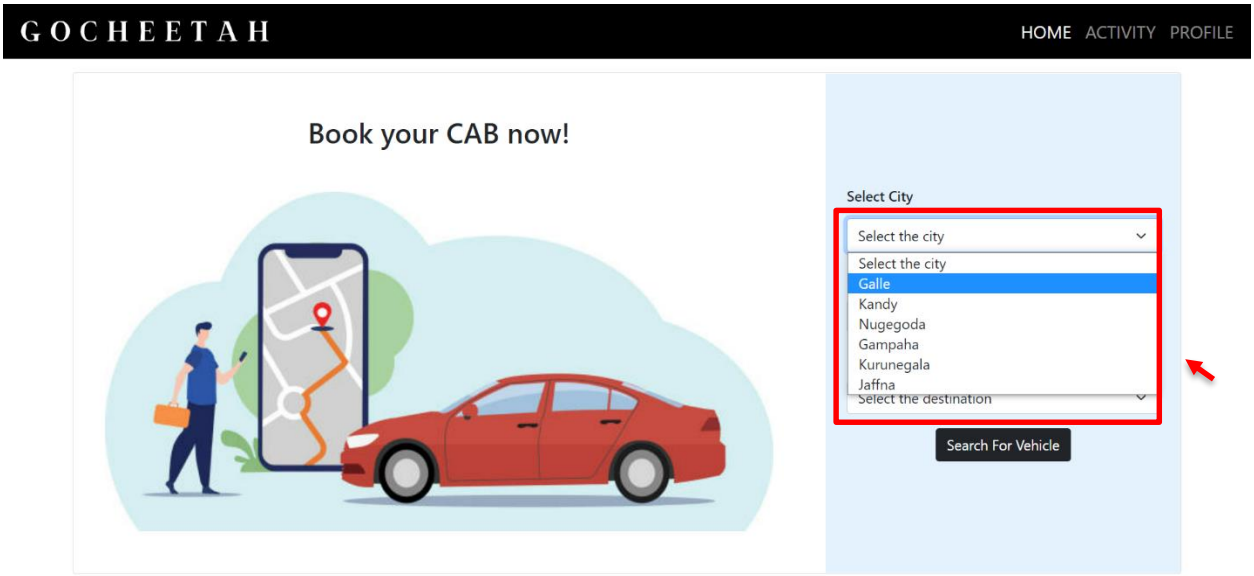
The registered customers can simply login to the system with valid credentials.



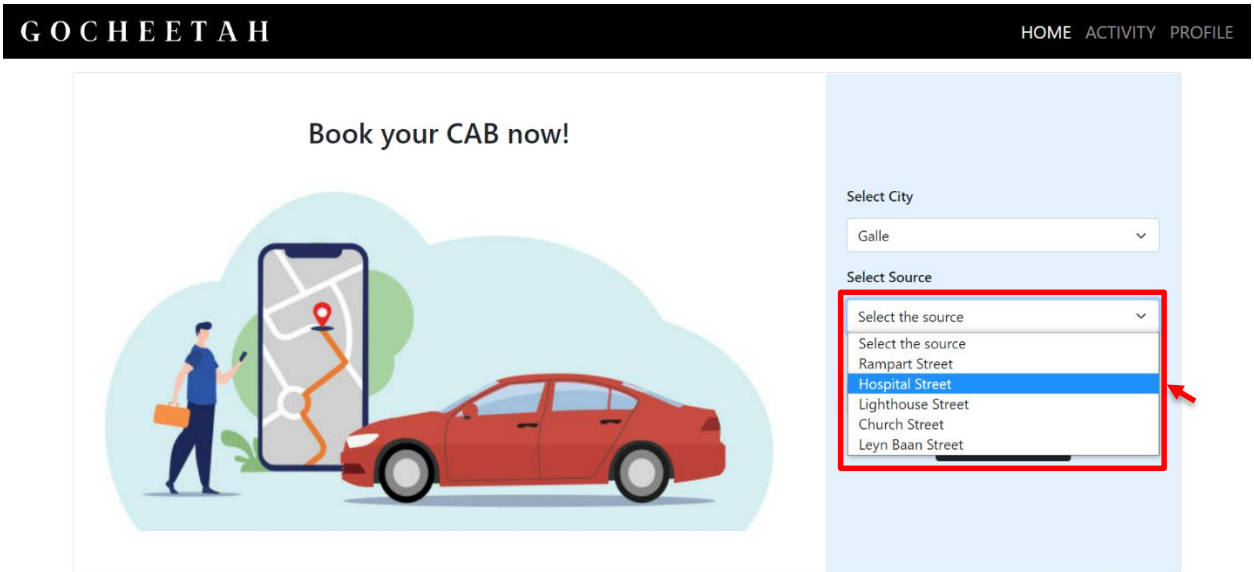
They as well will be directed to the home page, to start giving details to book a cab. The can select their preferred city, the source and destination within that city. They can select from drop down menu as shown below.



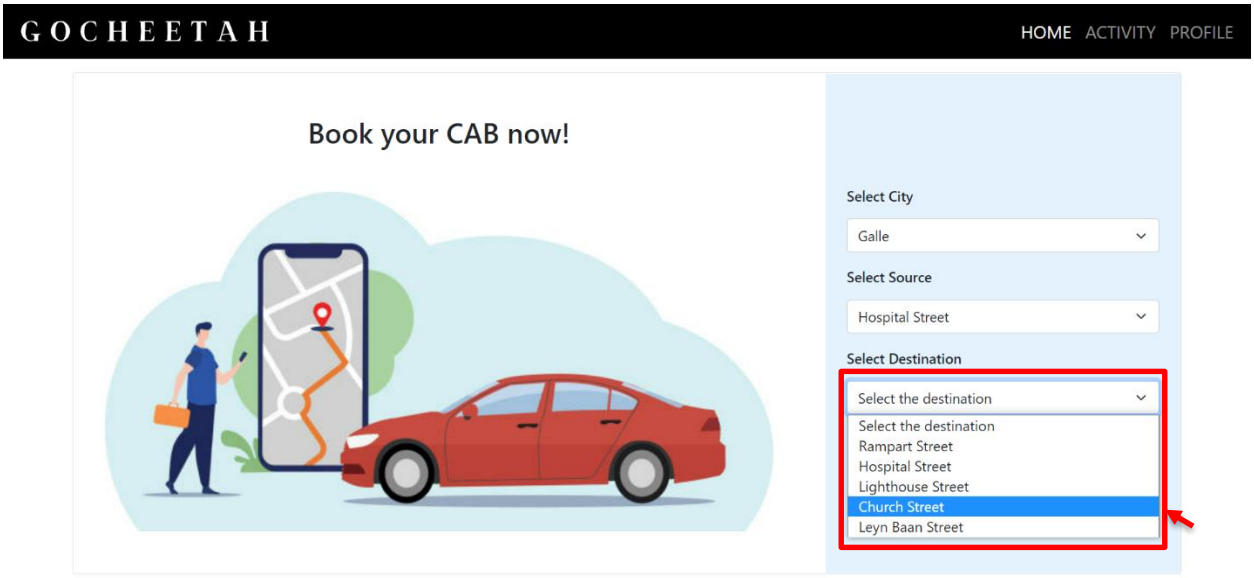
Selecting city from the drop down options. The drop down options will have all the registered cities that they can make a booking in. The customer can a ride only within a city, therefore, when they select a city, appropriate streets registered under that chosen city will be shown under source and destination drop down options.



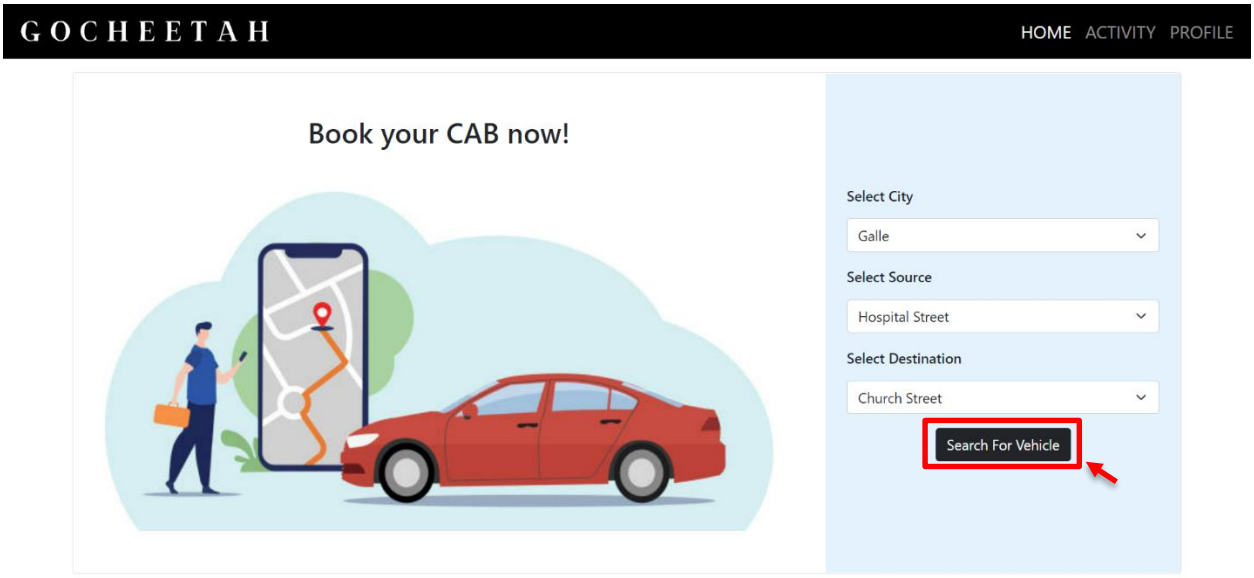
Selecting source street of the chosen from the drop down menu. The drop down options will prompt all the streets registered under that city where the customer can choose the destination.



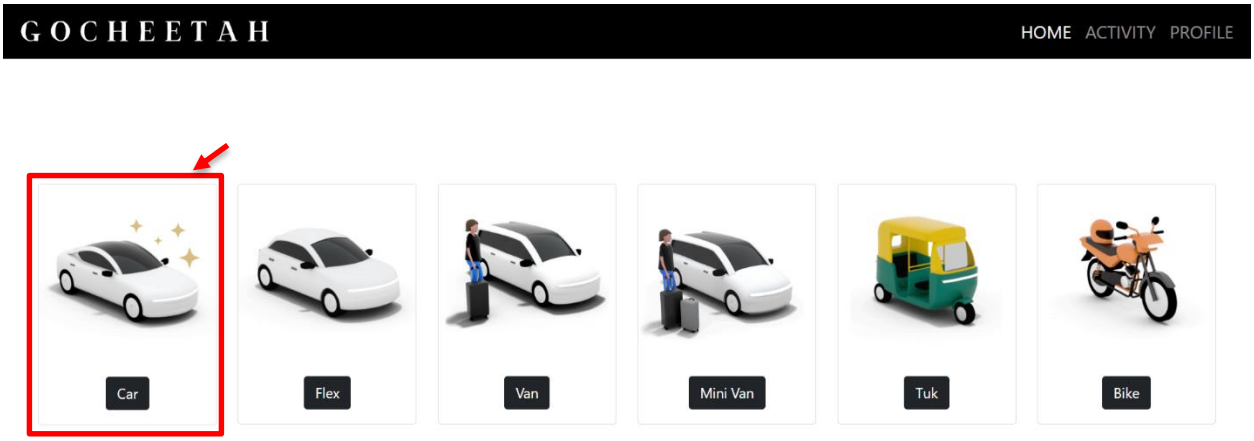
Selecting destination street of the chosen from the drop down menu. The drop down options will prompt all the registered streets under that city where the customer can choose the destination. The customer can choose source and destination as the same street. As they will be prompted with an error as no such ride can be done.



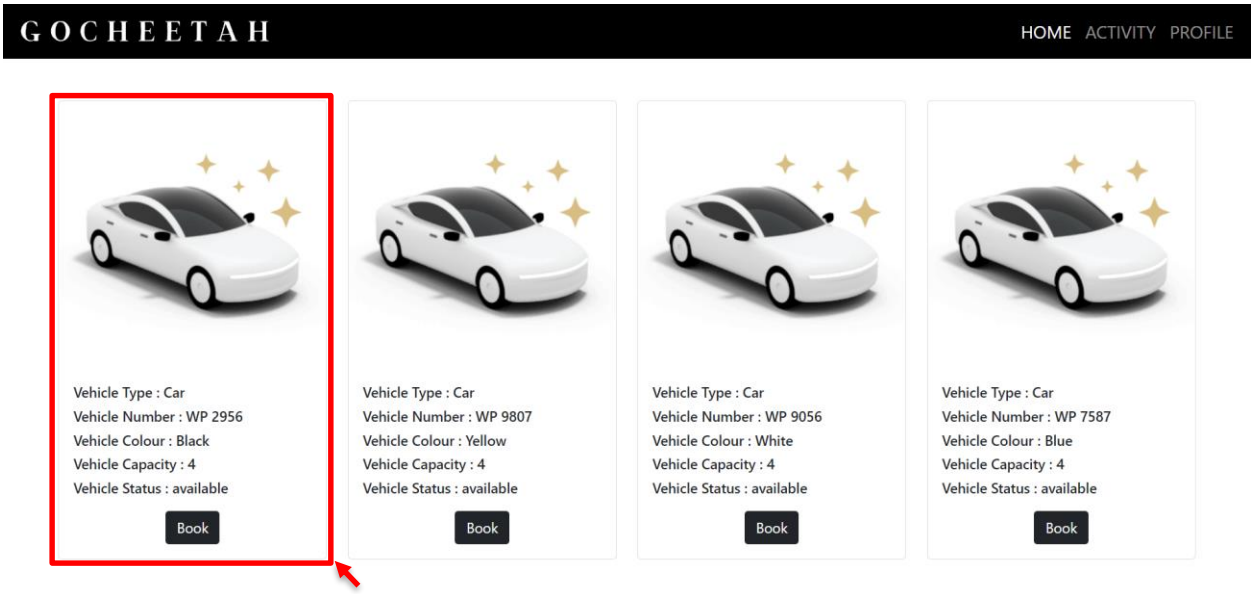
Once done, next, they can select the preferred vehicle where they will be directed to the next page to select the desired type of vehicle.



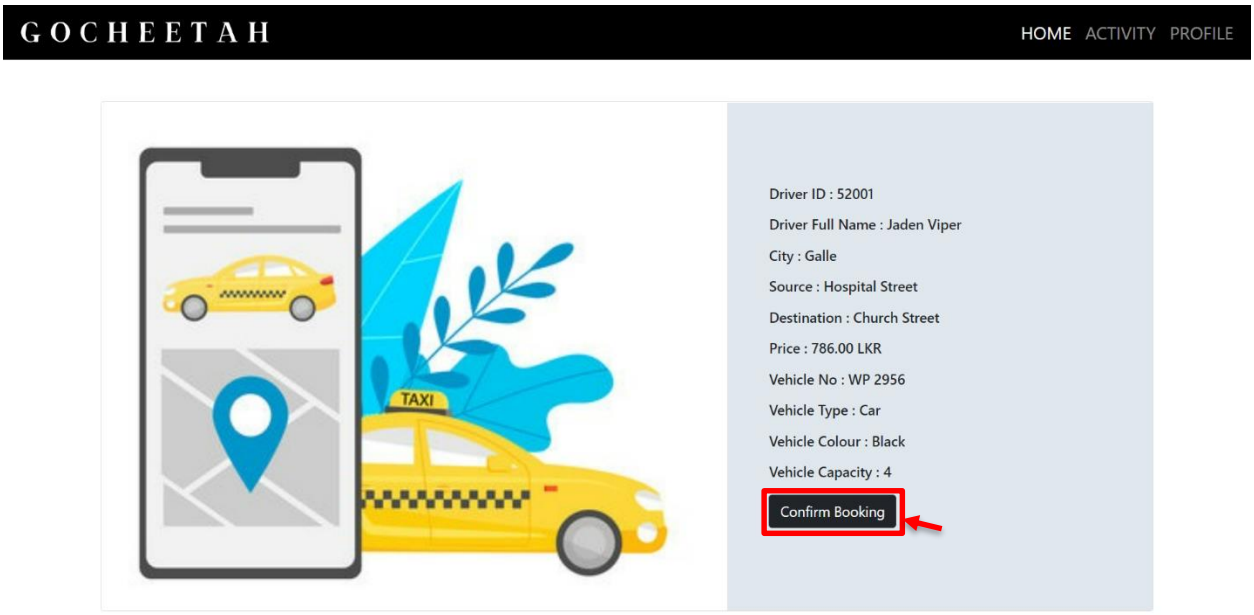
From here, they can choose the desired vehicle, according their needs. The page shown below shows the all the types vehicles registered in the system where the customer can make a ride with. Therefore according their needs and the number of people travelling the can choose a type of vehicle.



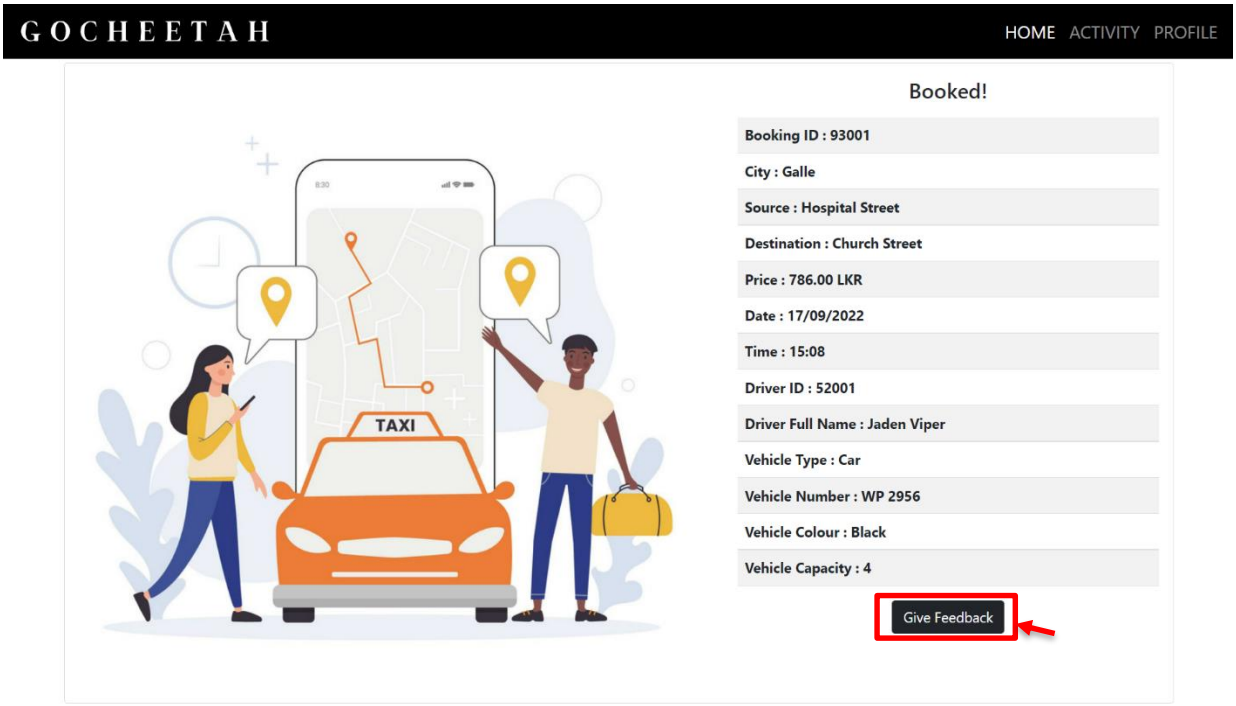
Once the customer chooses the type of vehicle, the available vehicle of that type will be prompted to choose from available vehicle. Here the available vehicle with all their details is prompted to the customer. Therefore, they are able to choose from the available vehicles according to their desire.

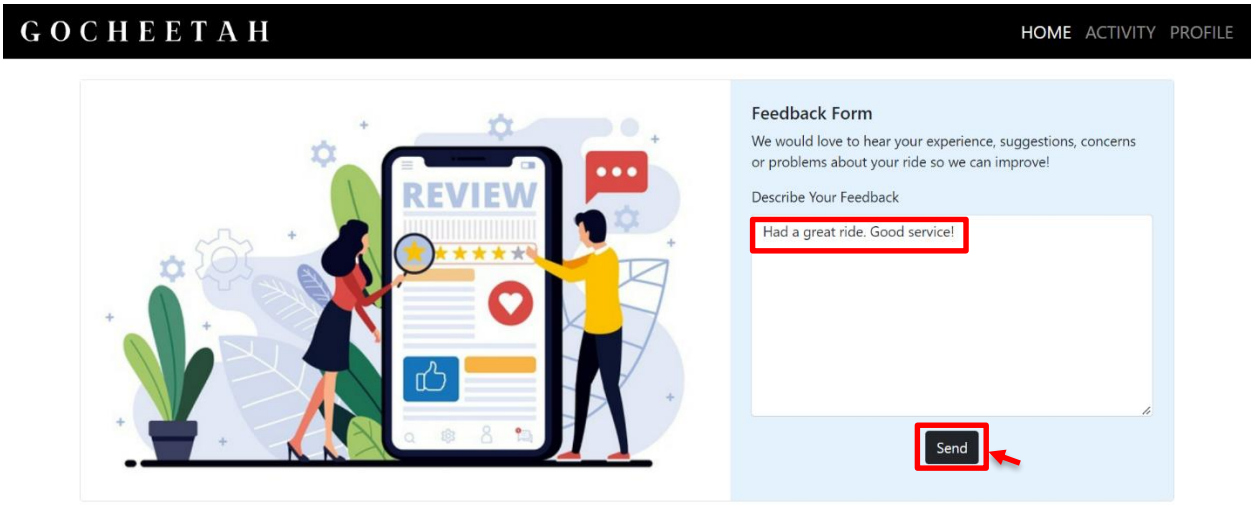


Once they choose the vehicle, the will be prompted with all the booking details given along with the price, to confirm the booking as per their need. The price is calculated as distance times the price per Km allocated for that type of vehicle.

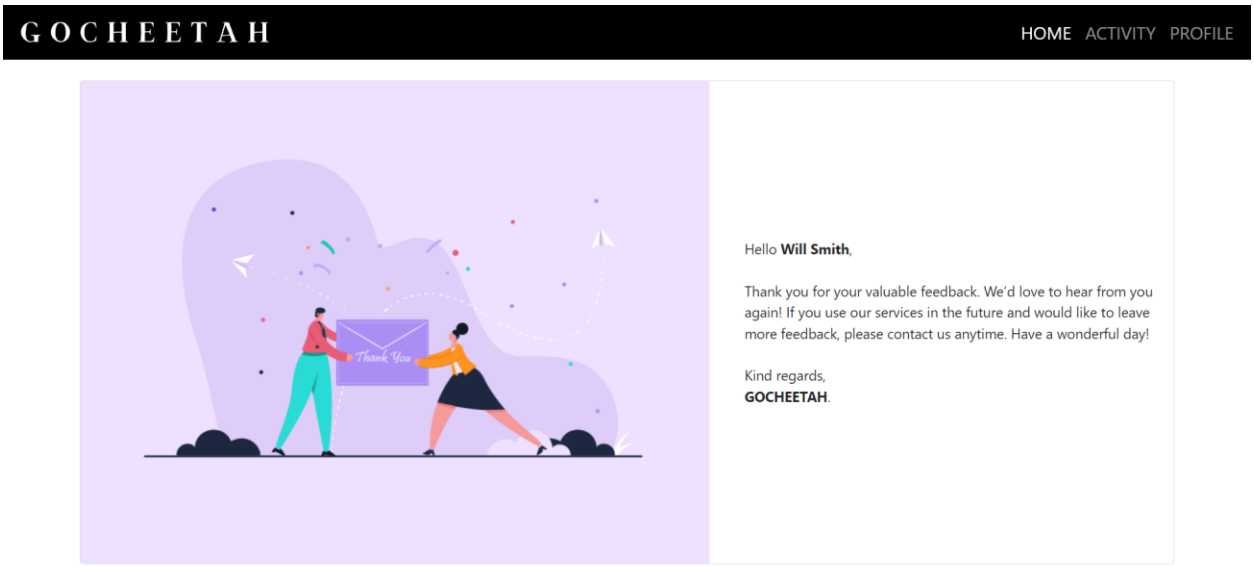


Then, when the customer is satisfied with the pricing, and once all details are checked, the booking can be placed and the customer is eligible to give a feedback on the ride once done.

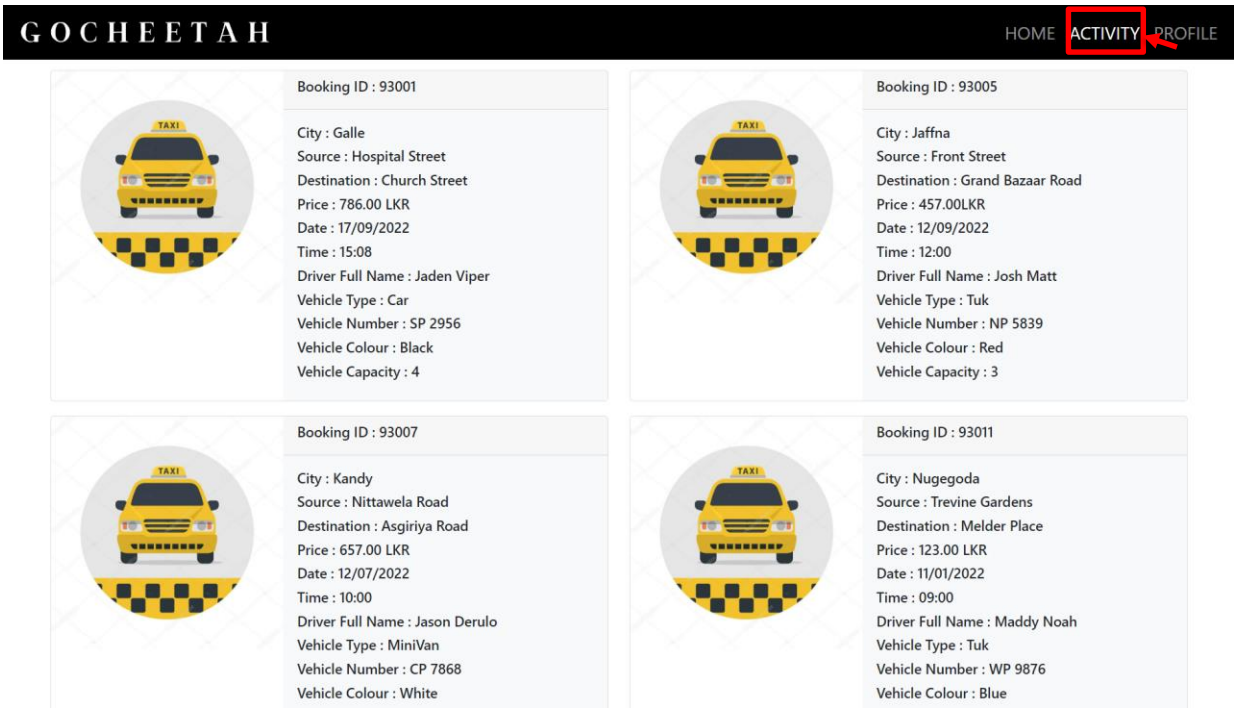

GOCHEETAH
HOME ACTIVITY PROFILE
TAXI
Booked!
Booking ID : 93001
City : Galle
Source : Hospital Street
Destination : Church Street
Price : 786.00 LKR
Date : 17/09/2022
Time : 15:08
Driver ID : 52001
Driver Full Name : Jaden Viper
Vehicle Type : Car
Vehicle Number : WP 2956
Vehicle Colour : Black
Vehicle Capacity : 4
Give Feedback


GOCHEETAH
HOME ACTIVITY PROFILE
REVIEW
Feedback Form
We would love to hear your experience, suggestions, concerns or problems about your ride so we can improve!
Describe Your Feedback
Had a great ride. Good service!
Send

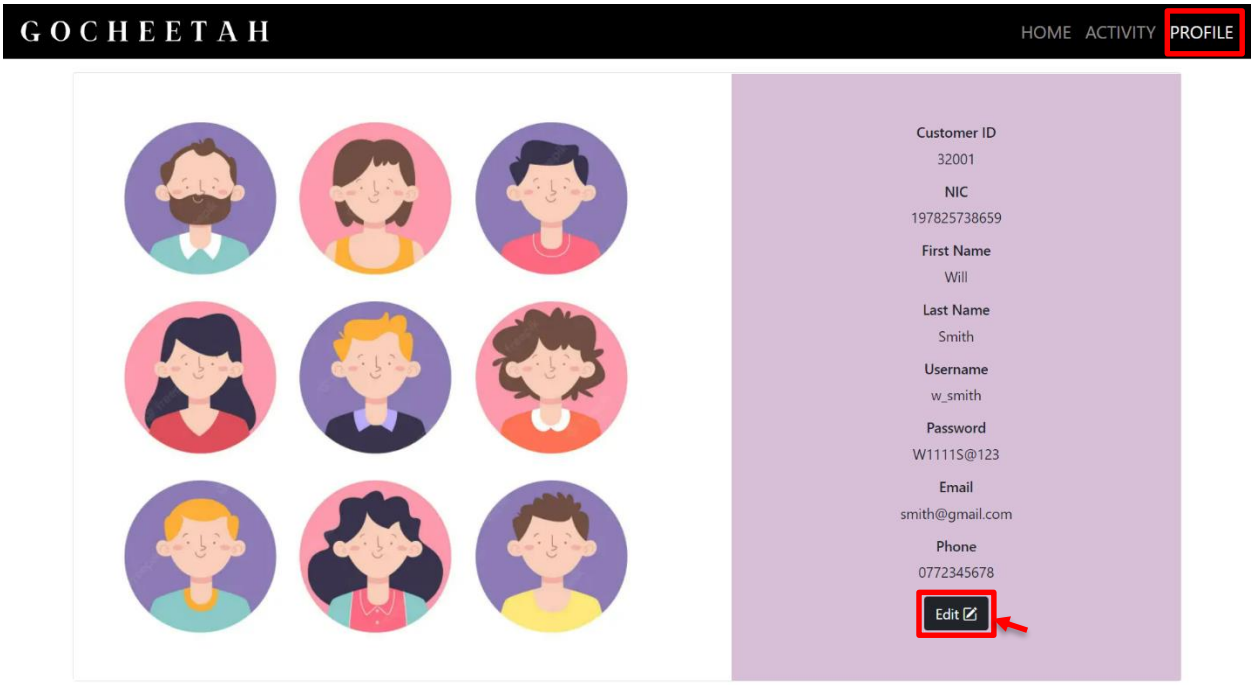
.....



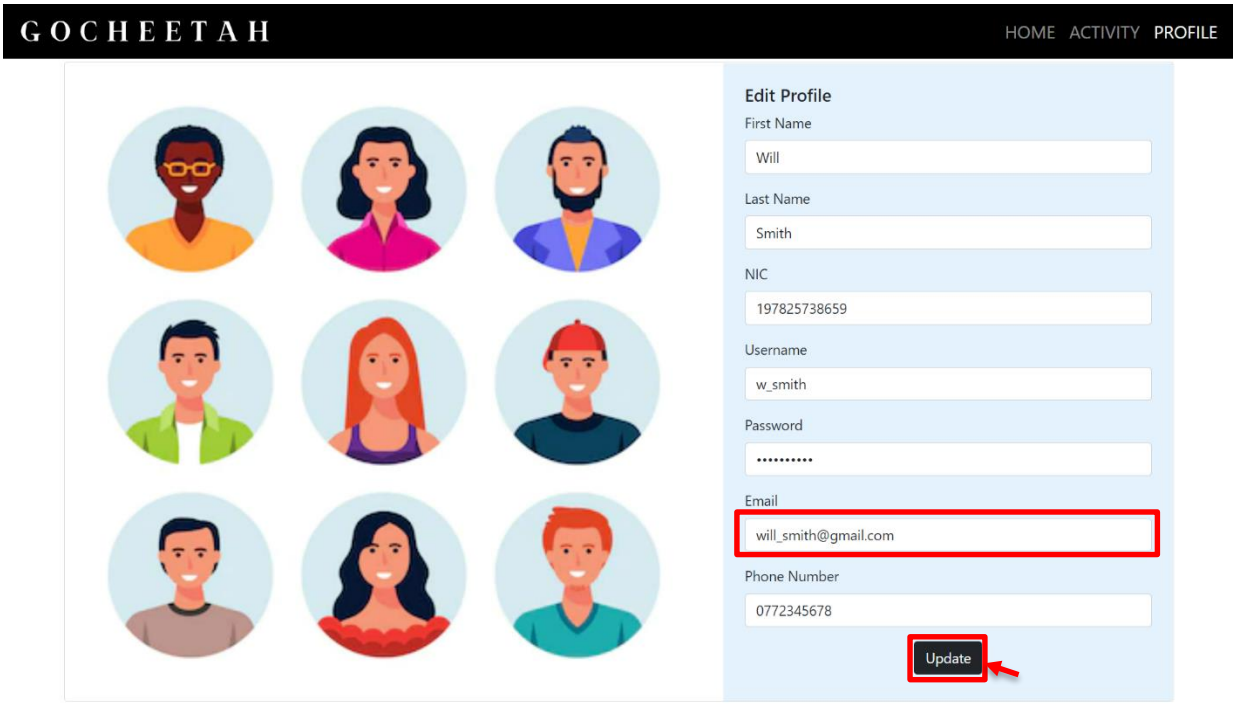
In the activity panel, they can view all their past booking/activity along with all the booking, vehicle, location, pricing and driver details. This will show a complete history of their past rides.



The customer can view their profile, and edit any changes anytime from the Profile panel.

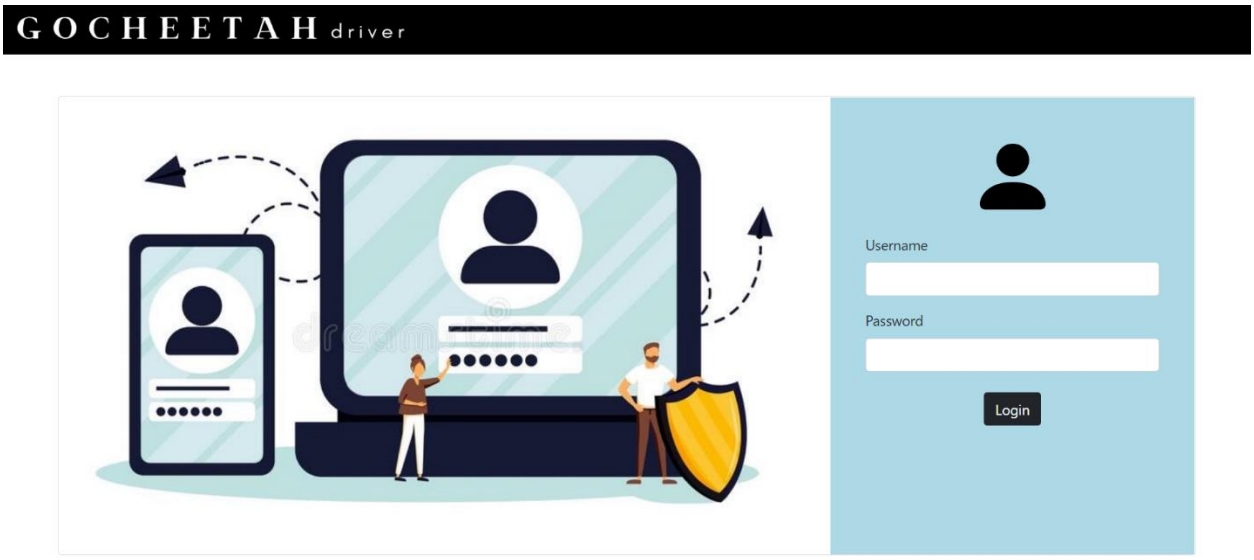


The can click on Edit as shown above, and they will be directed to a page as shown below. The fill the required data and click on Update as shown below.

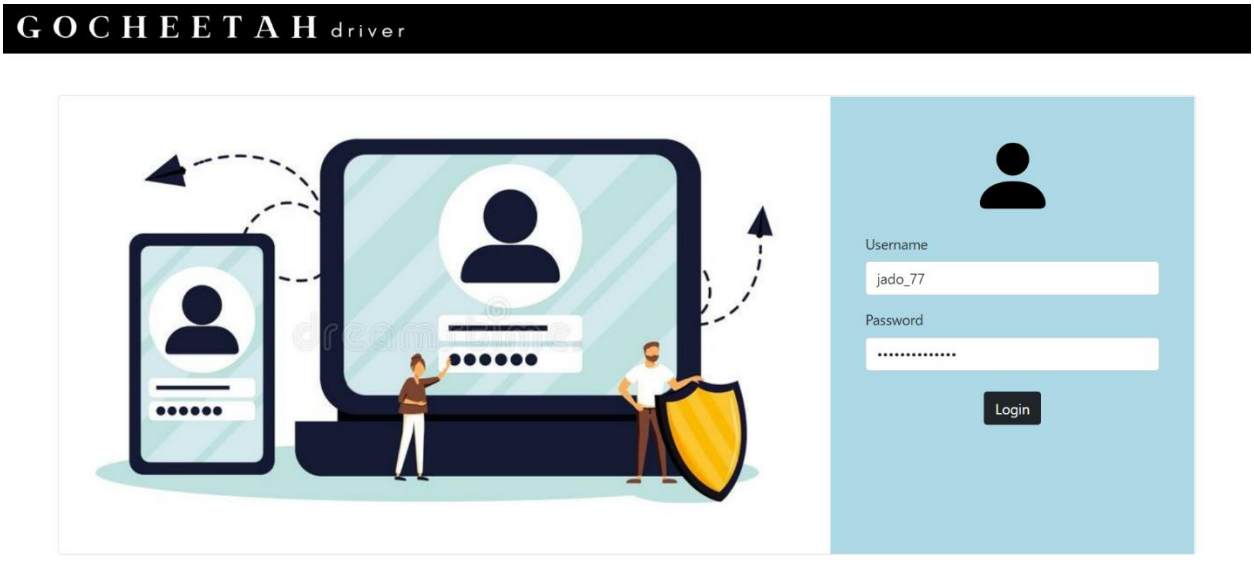


==========

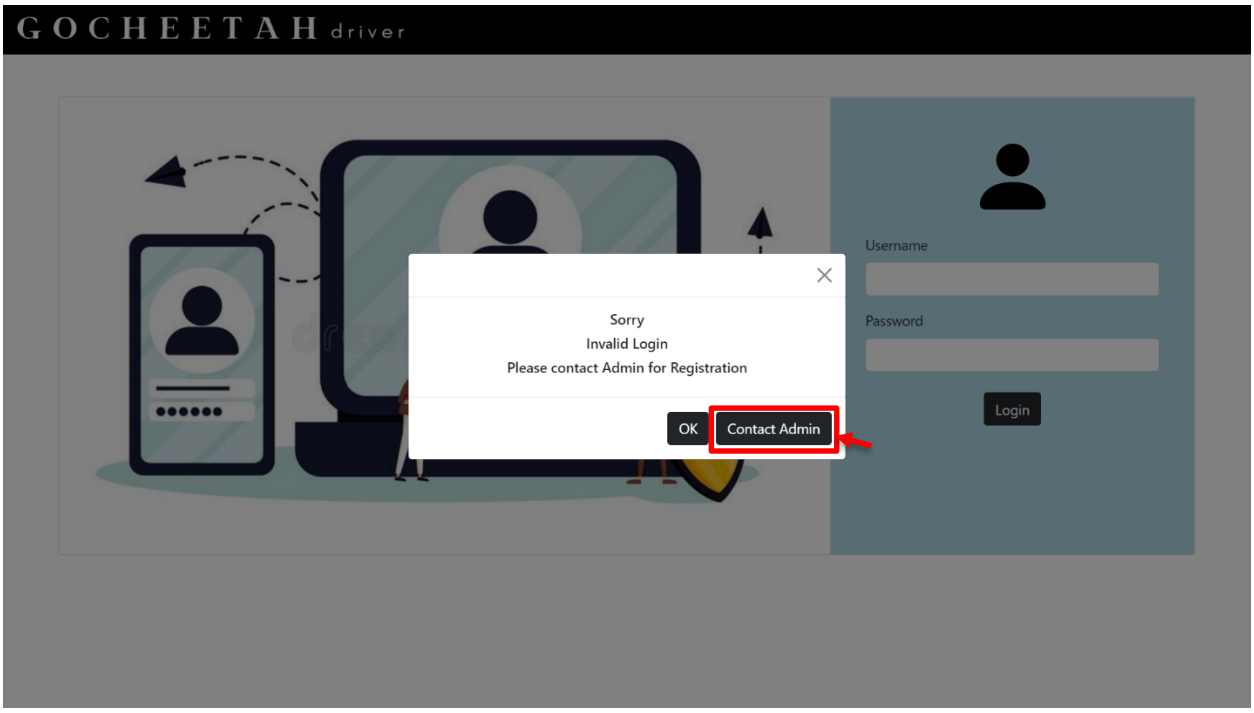
Driver Client



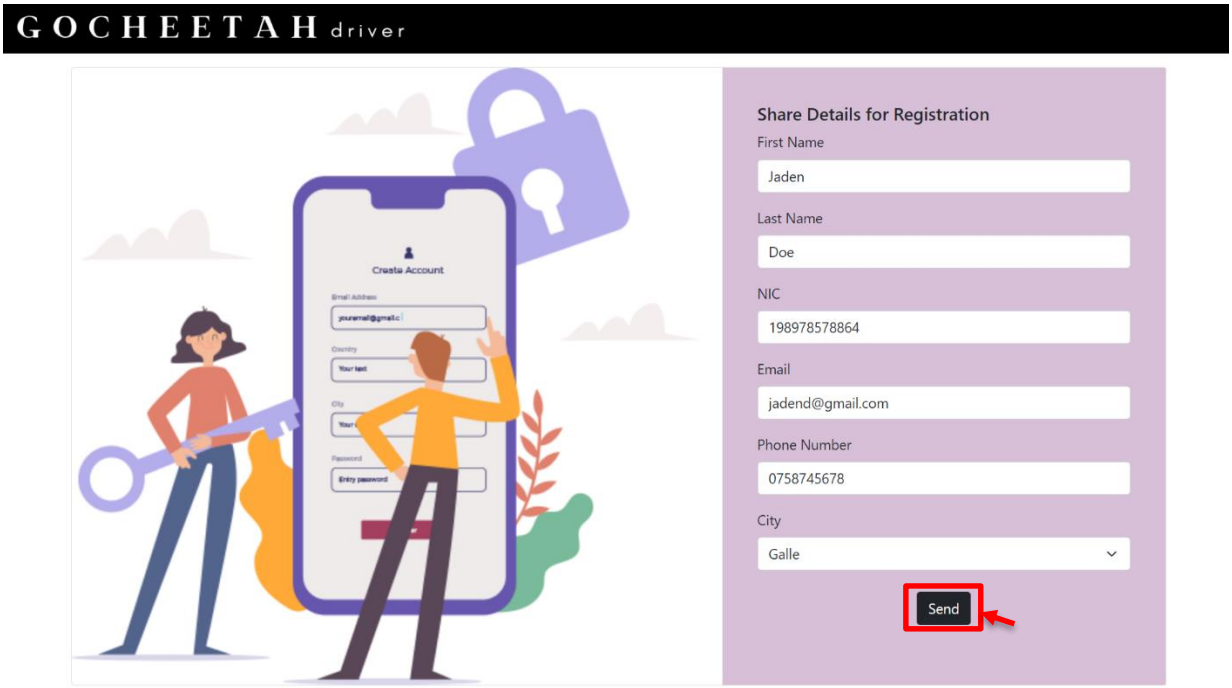
The driver can login to the website with their given valid username and password at anytime.



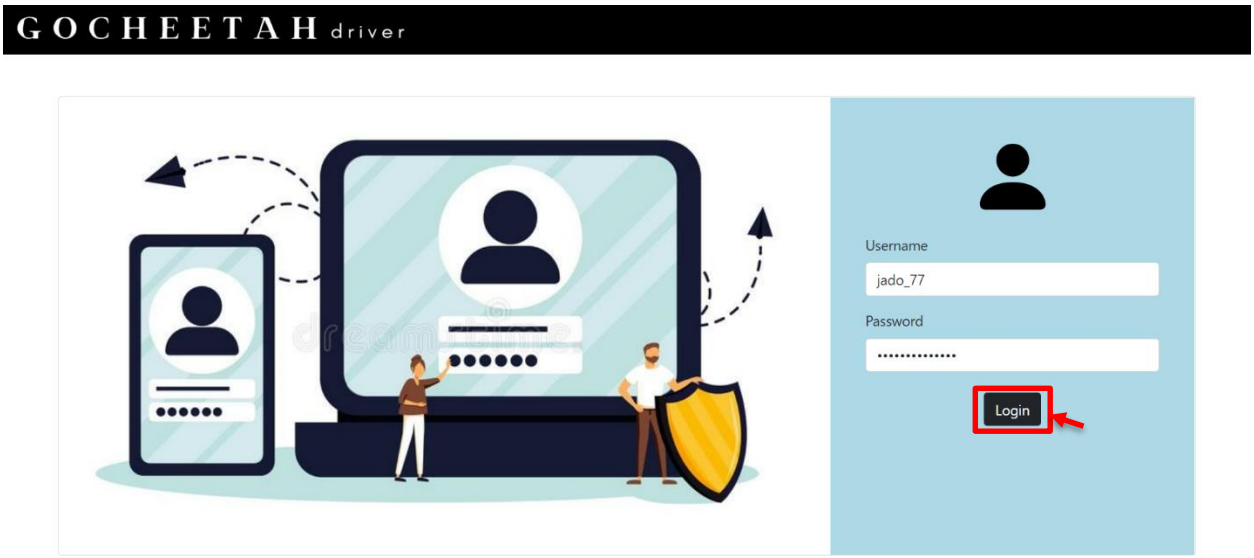
If the credentials were invalid, they will be prompted with an alert error message, saying that it is an invalid login, they will have to contact admin to sign up to use the website. They can click Contact Admin in the pop up as below shown image to contact admin.

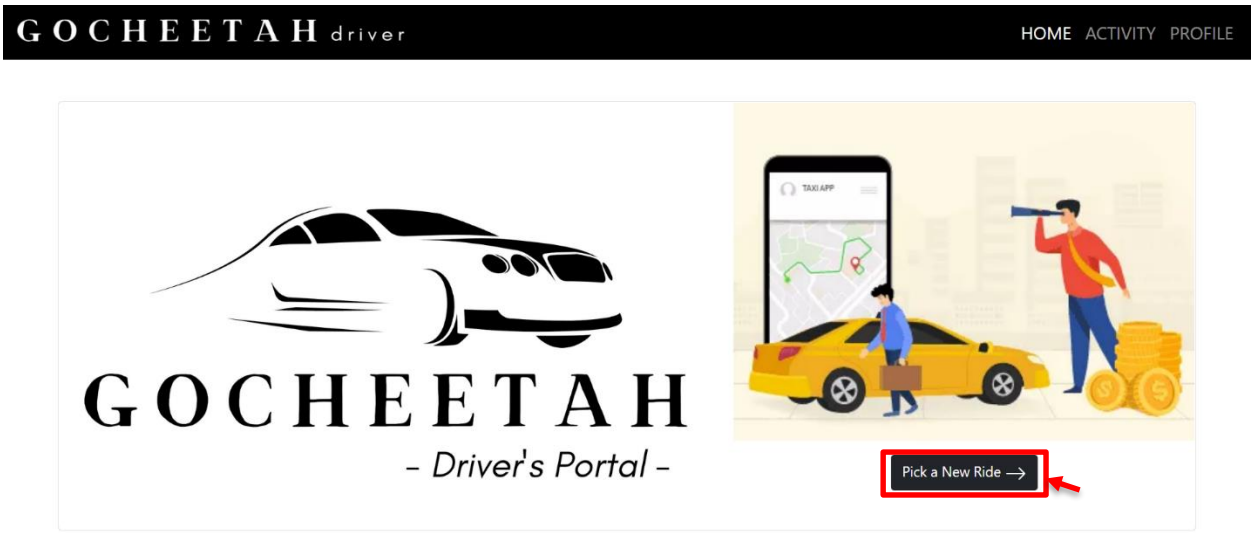


The driver can give their details and click Send to send all the details to admin.

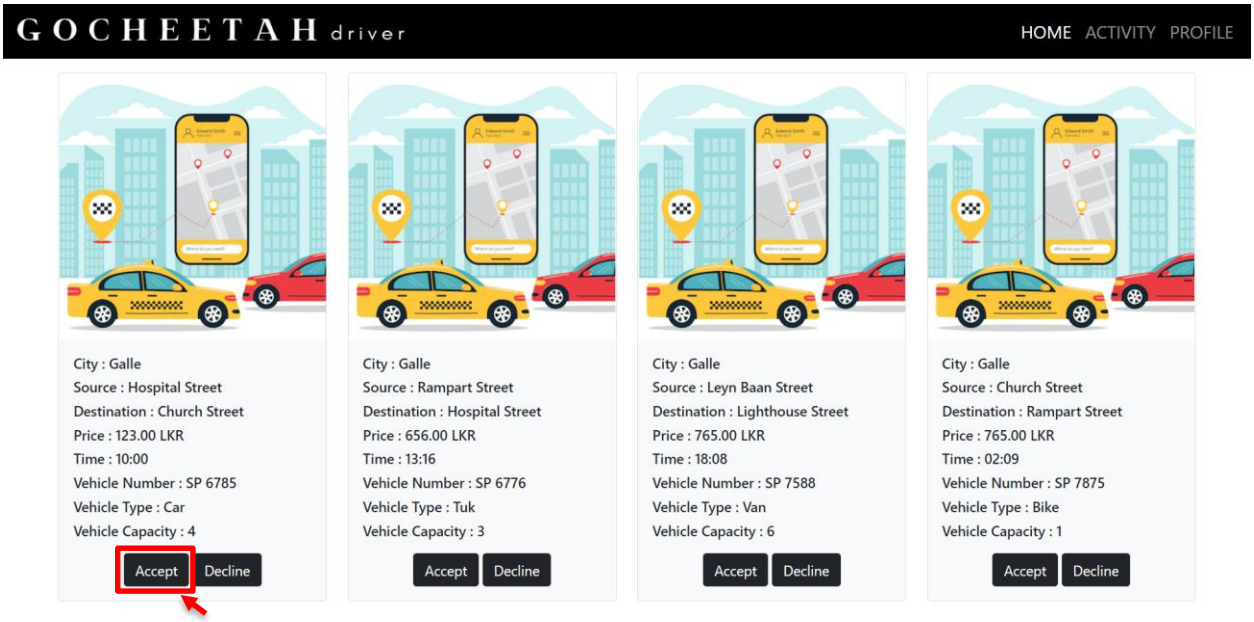


The registered drivers can login to the website with their given valid credentials and they will be directed to home page of driver's website.

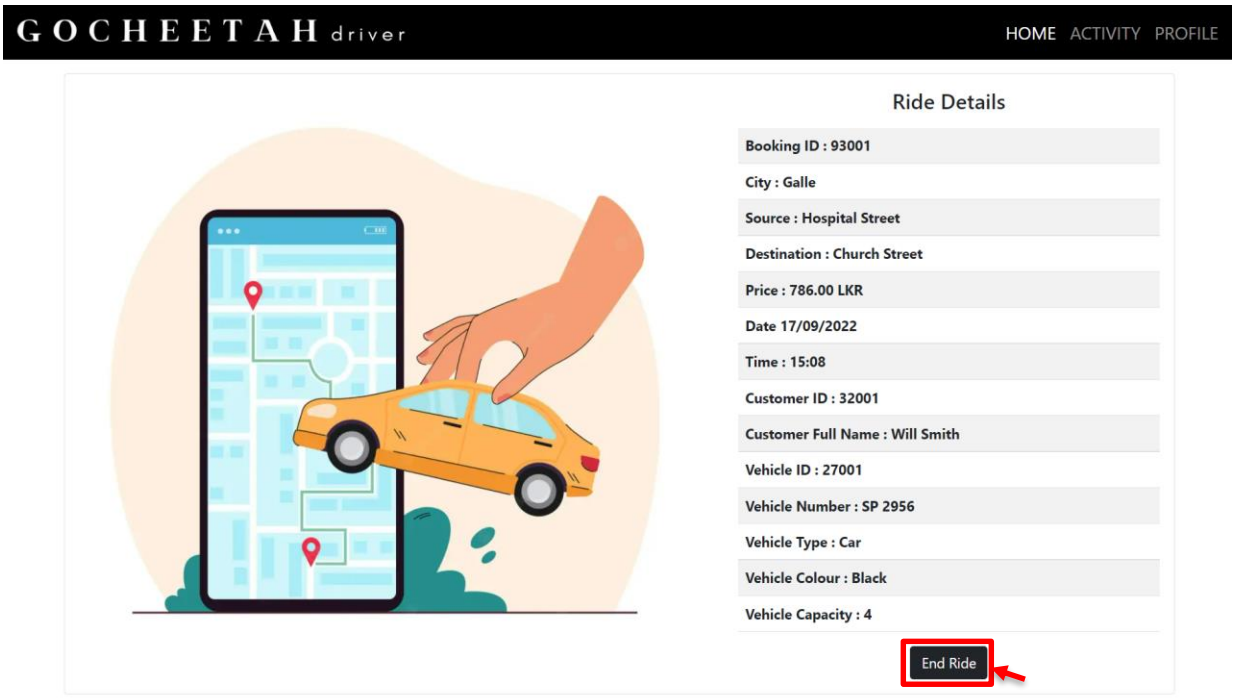


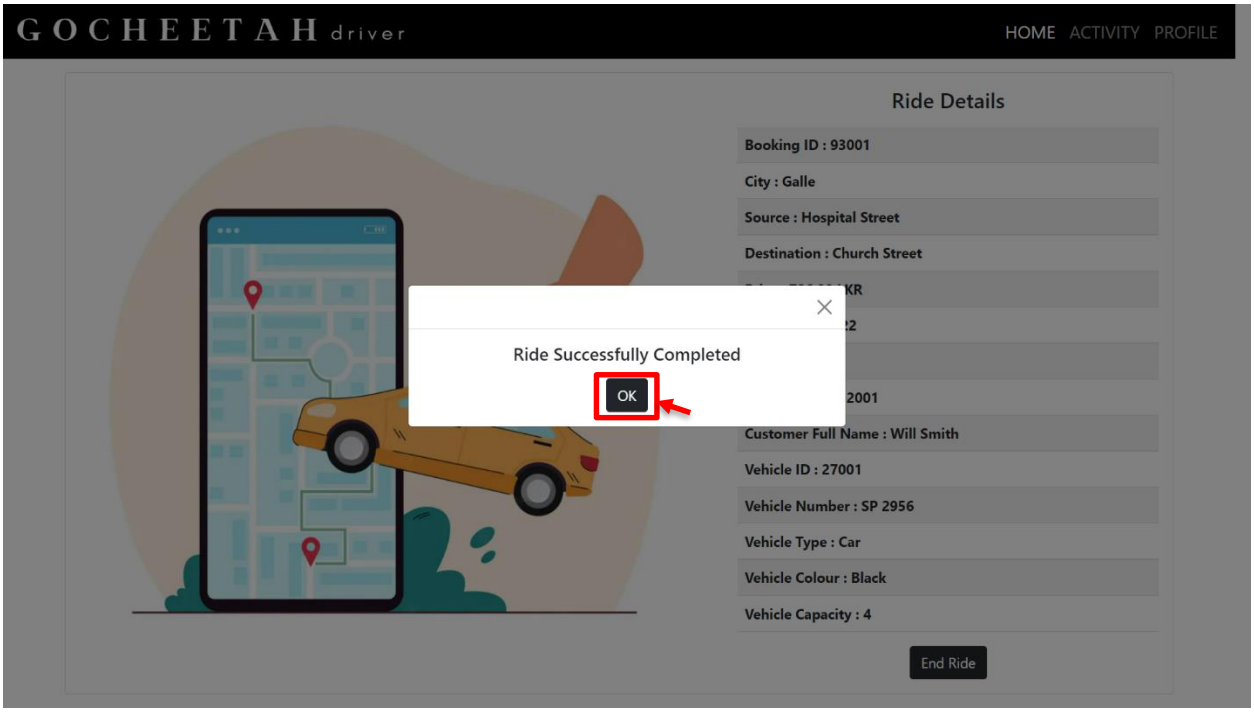


Here they can accept or decline a ride of their preference. The ride details are shown along with the city, source, destination, vehicle and pricing details. They can click on Accept or Decline as shown below.

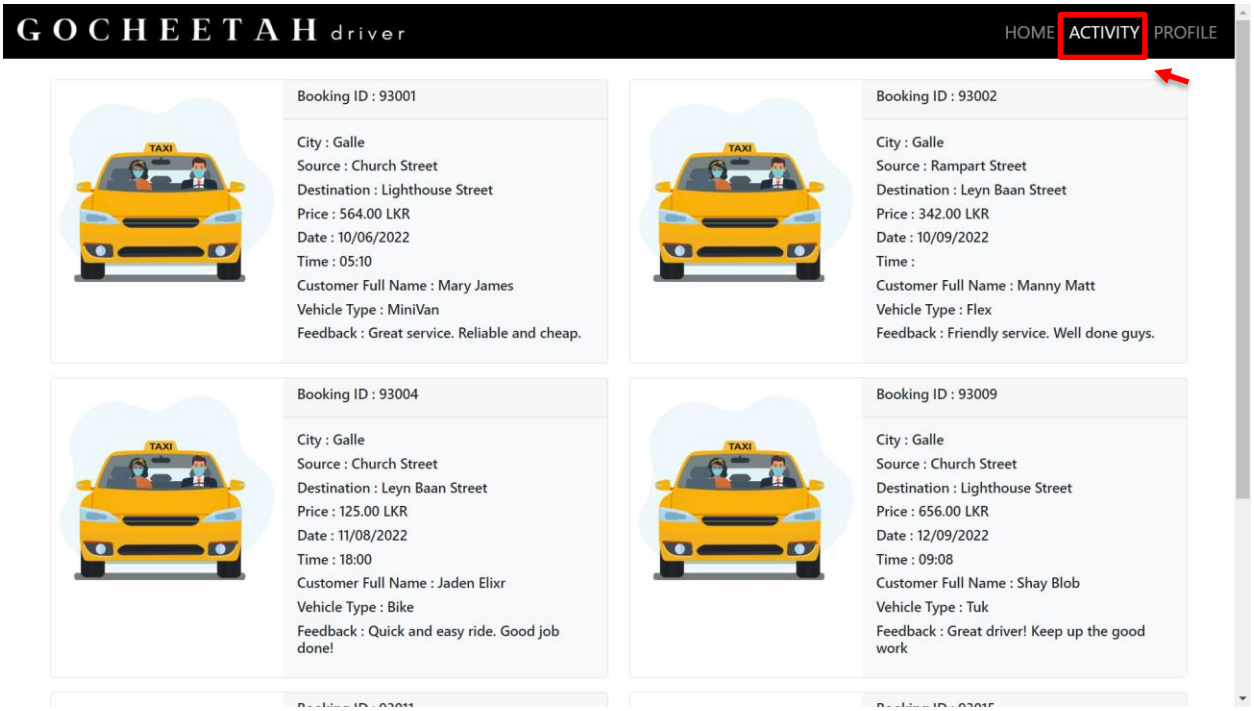


If they accept a ride, they will be directed to the page below, where it shows all details of the ride and allows the driver to end the ride once it is complete. They can click on End Ride as shown below it finish the ride.

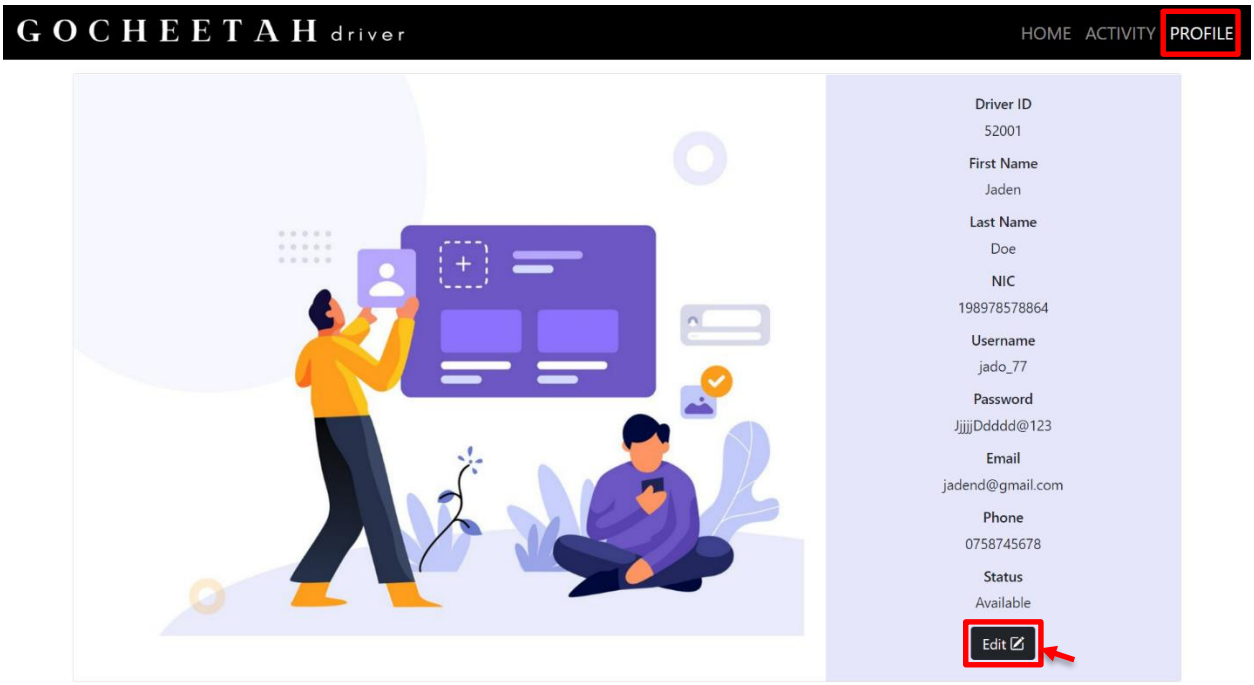




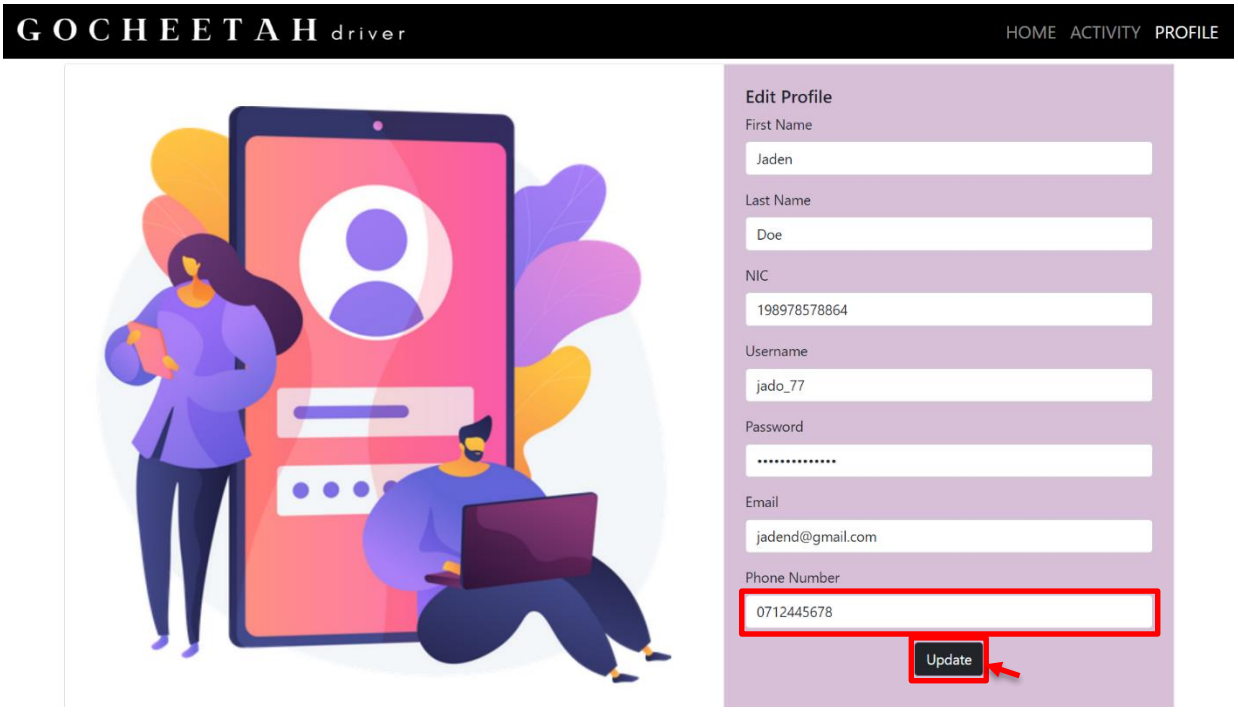
The driver can view all the past ride/activity along with the customer, vehicle, price and the location details and the feedback at any time in the Activity panel. This show a complete history of rides made by that driver with details.



The driver can view their profile in the Profile panel as well edit it anytime. By clicking on Edit as shown below, they will be directed to another to edit the details and update it.

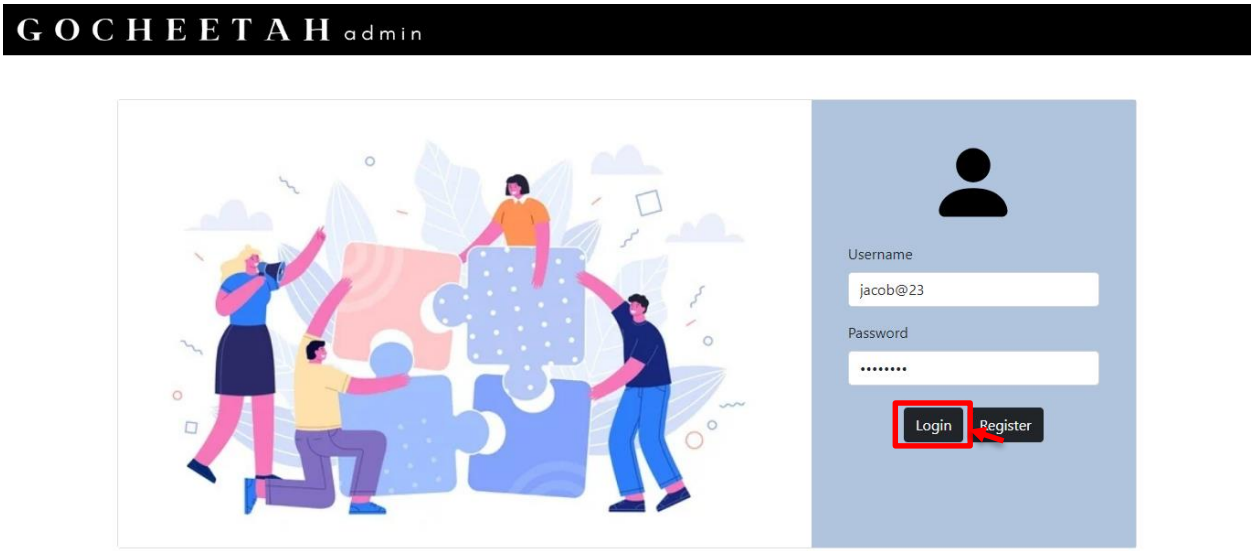


Here the driver can give all the required data and make the changes and can click on Update to update the altered details.

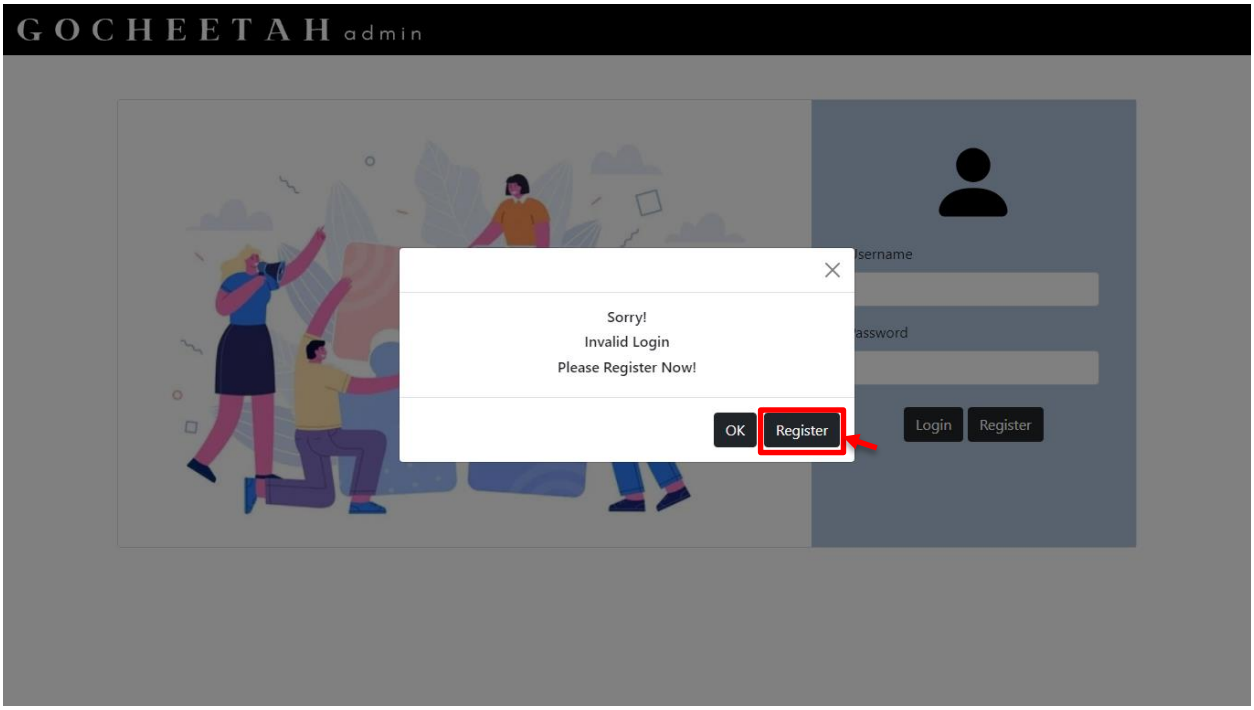


==========

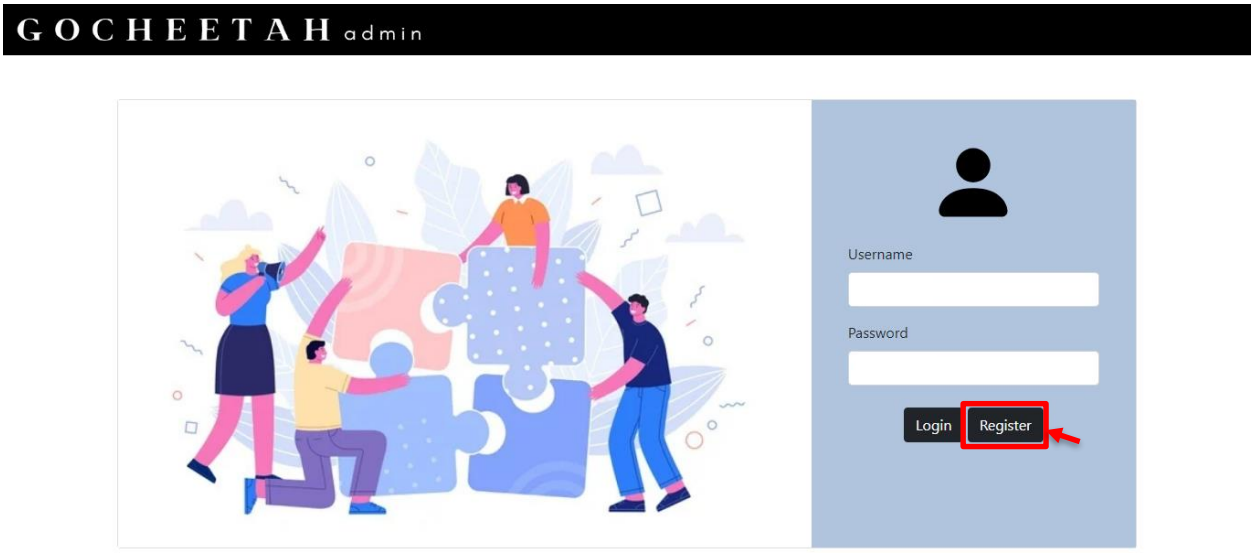
Admin

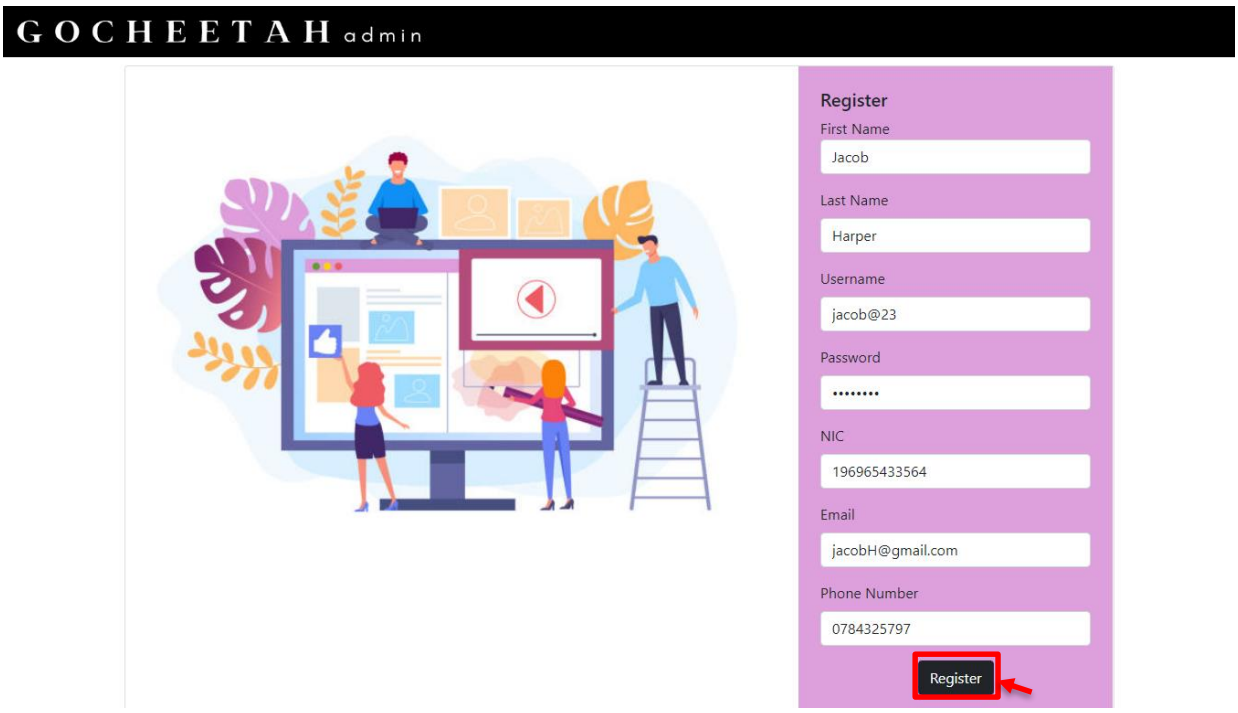


The admin can login to the website with their username and password at anytime.

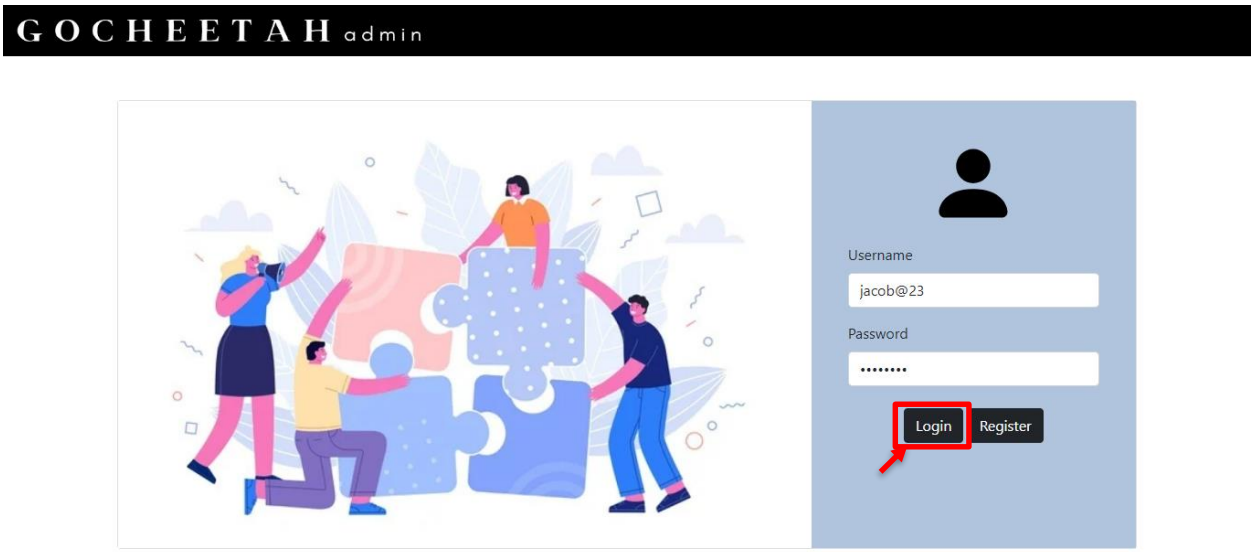


If the credential was wrong, they will be prompted with an alert error message, saying that it is an invalid login, they will have to sign up to use the website. They can either click Register in the pop up or click Register as below shown image to register.

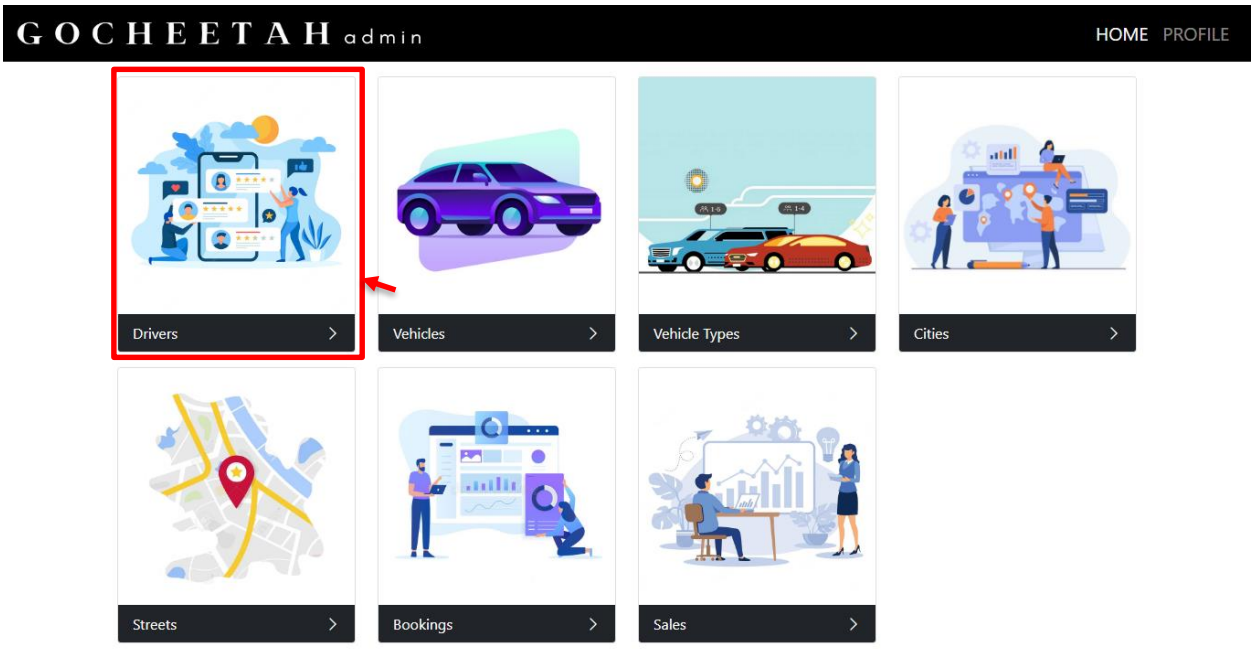




The new admin can give the required details and click Register to register into the system. And they can use the username and password they have given to login to the website anytime.

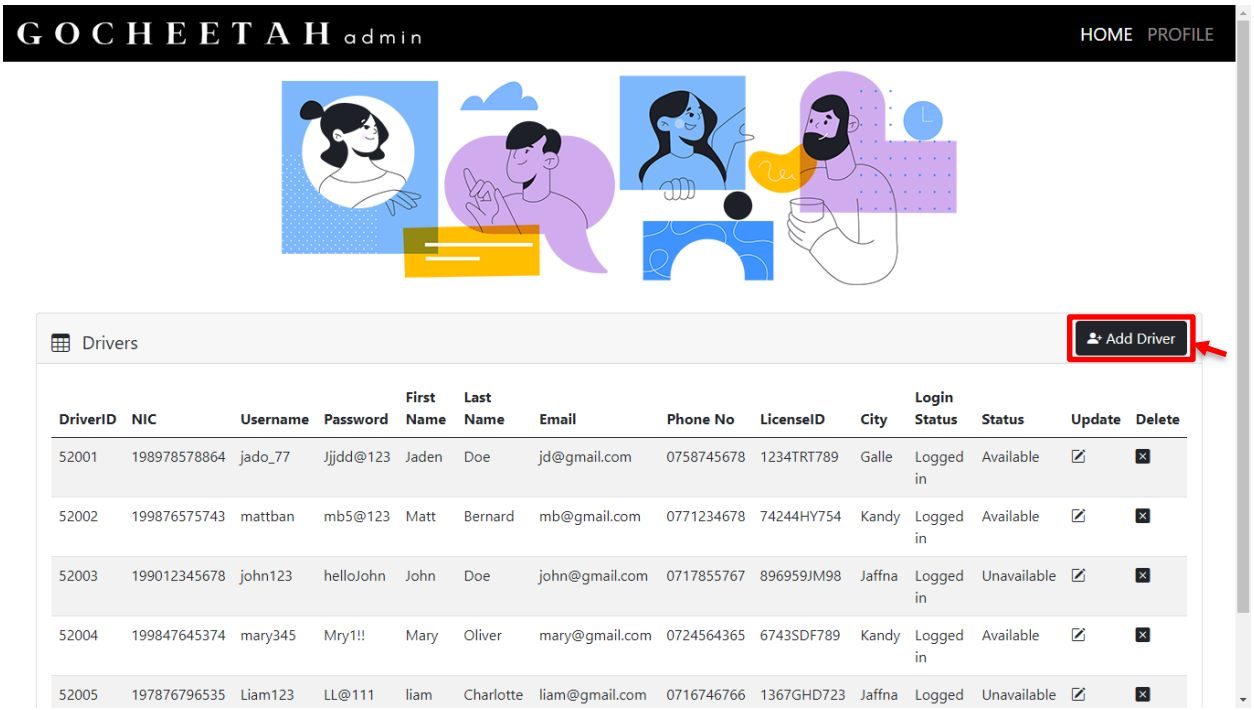


Then the admin can login with the valid username and password and they will be directed to the home page.

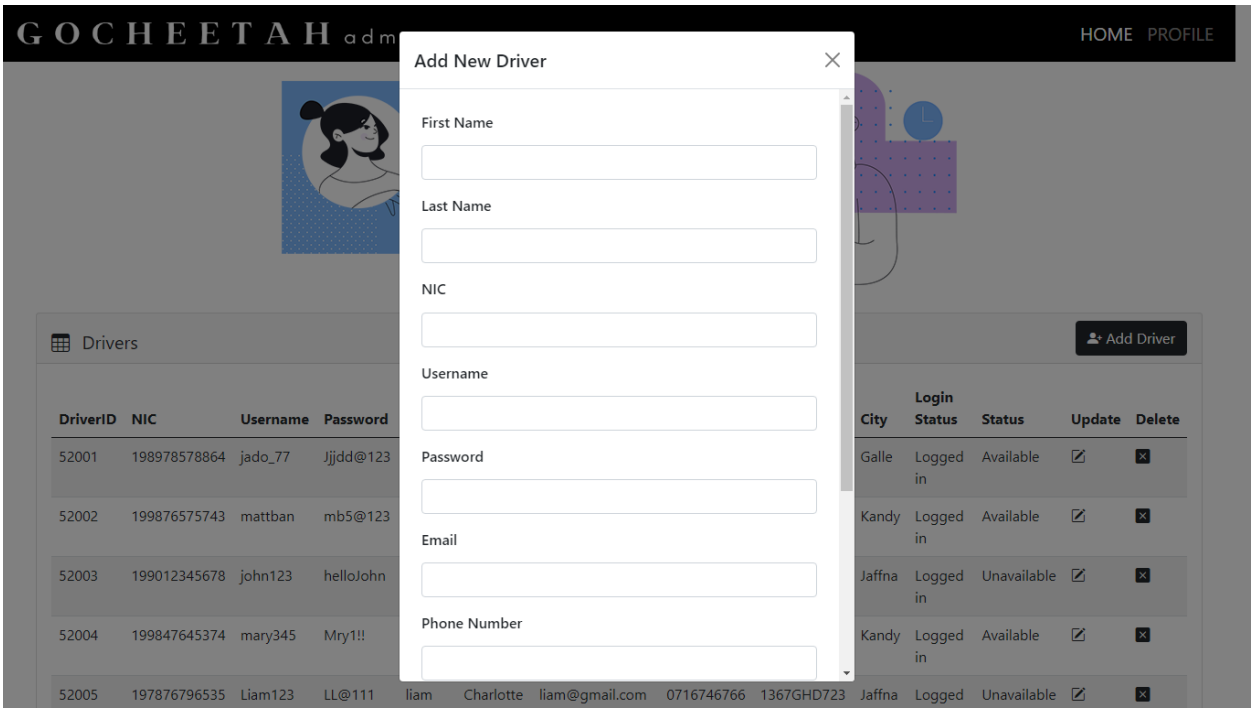


The admin can select the Driver, to view all drivers and if they have to Add, Update or Delete a driver. To delete, they can click on the cross icon beside the driver. As shown in

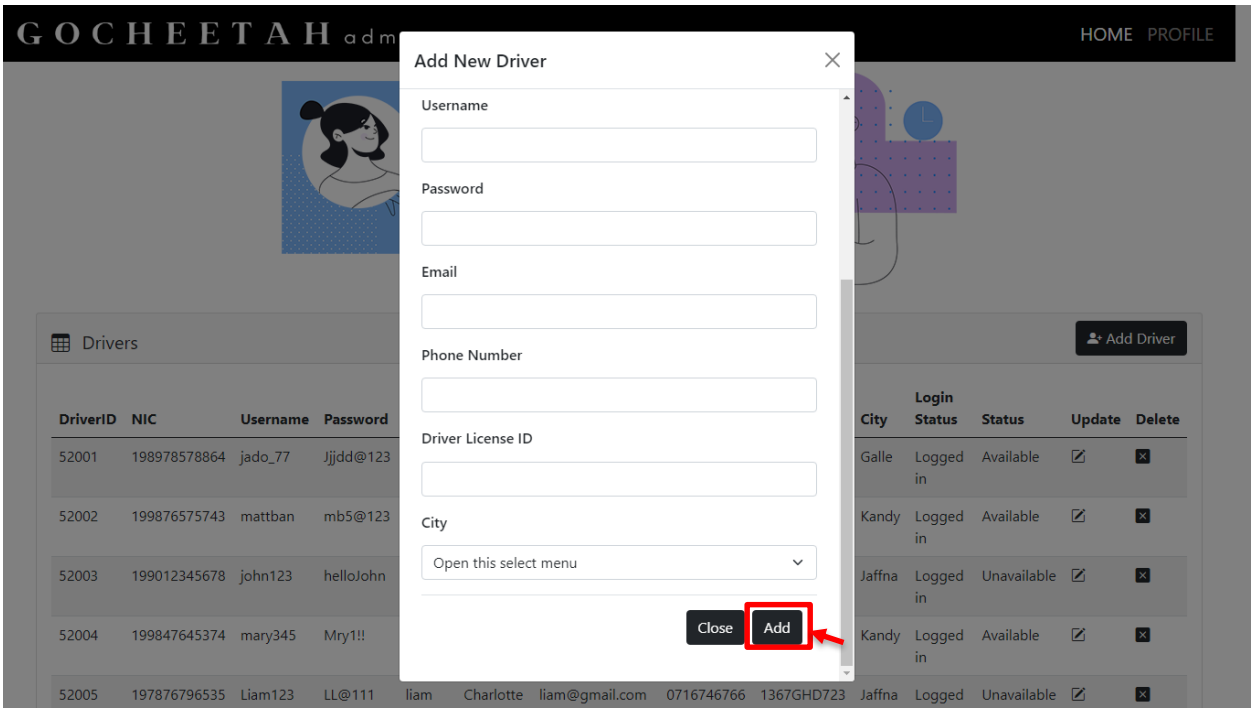
the image below it shows complete details of the registered drivers in the system and also facilitates to do other operations.



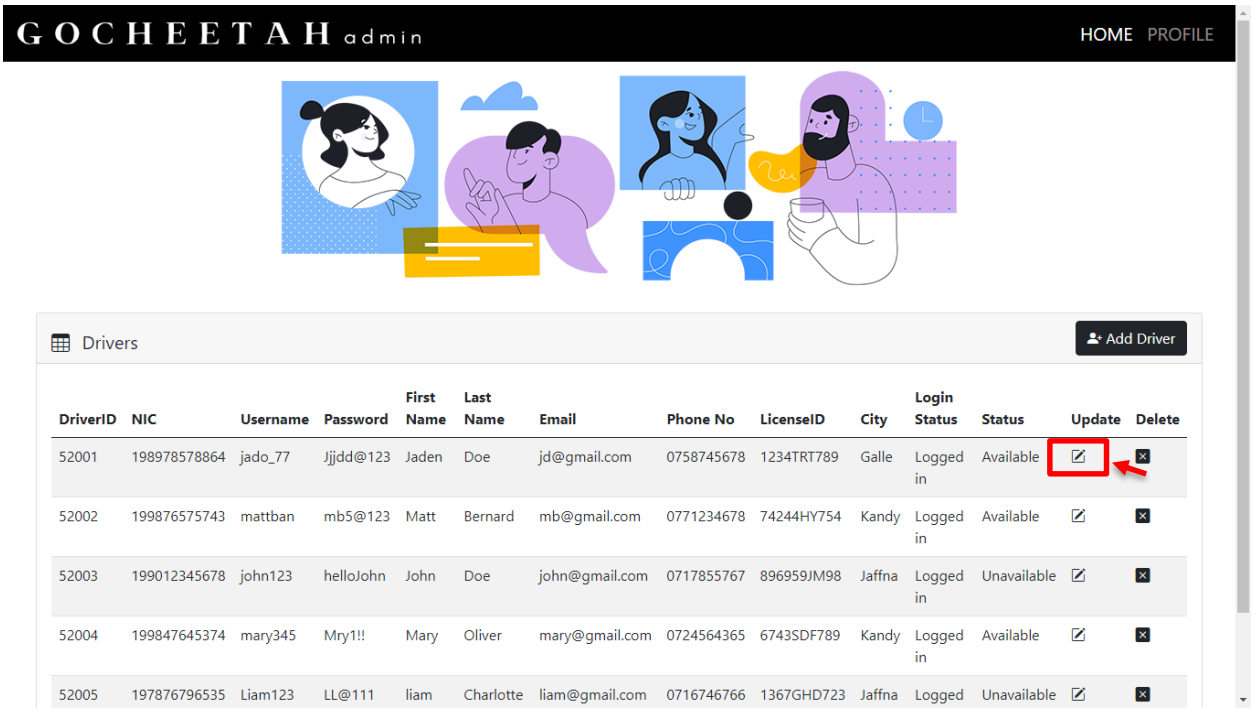
By clicking on Add Driver, they are able to add a new driver into the system.



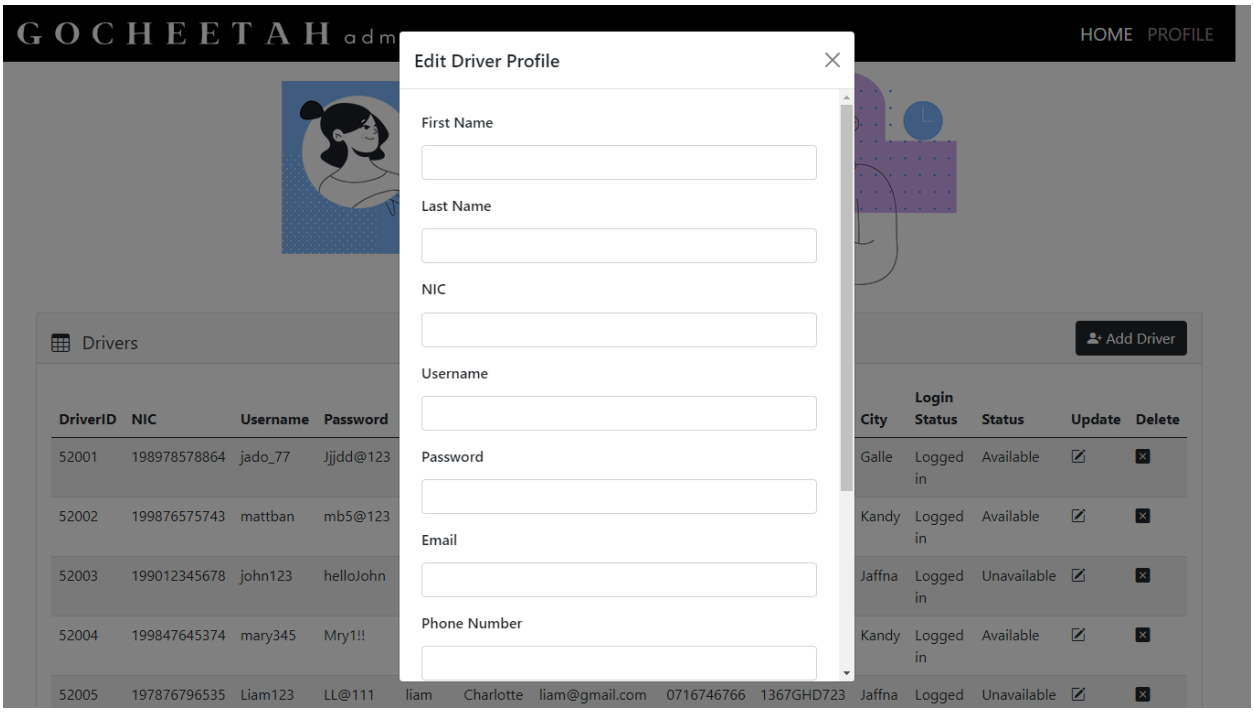
They need to provide all the required data for a new driver. And click Add as shown below.

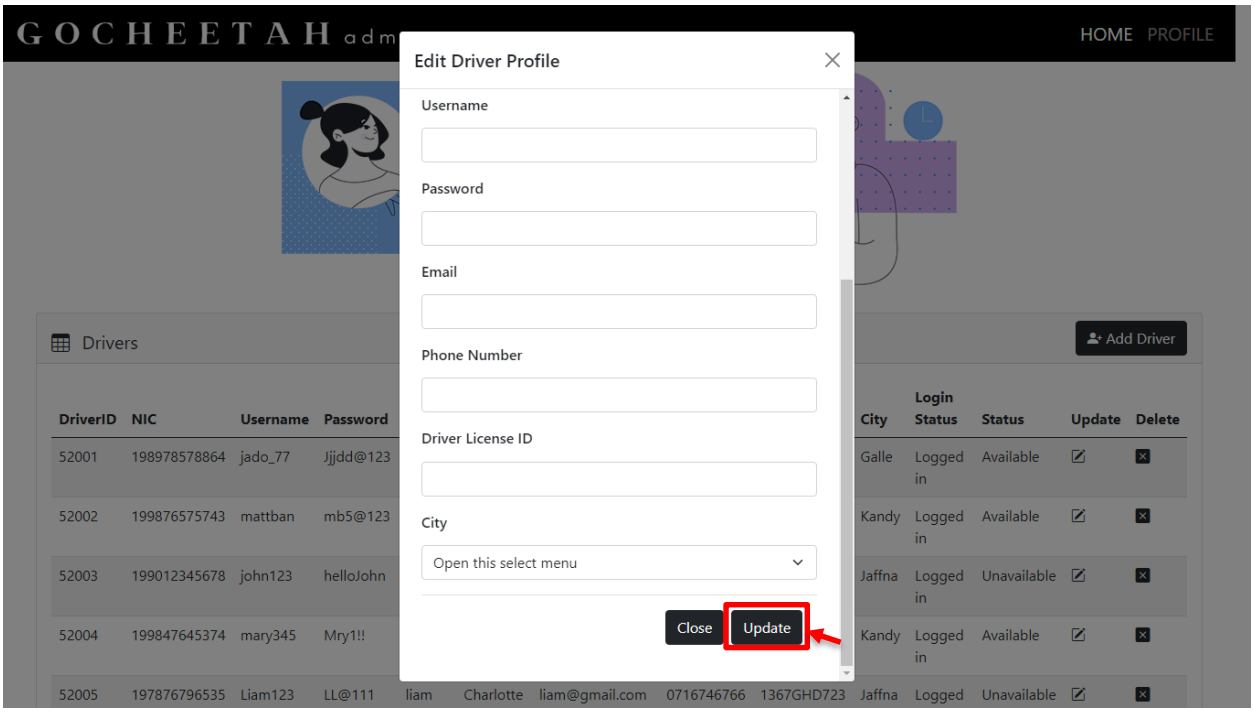


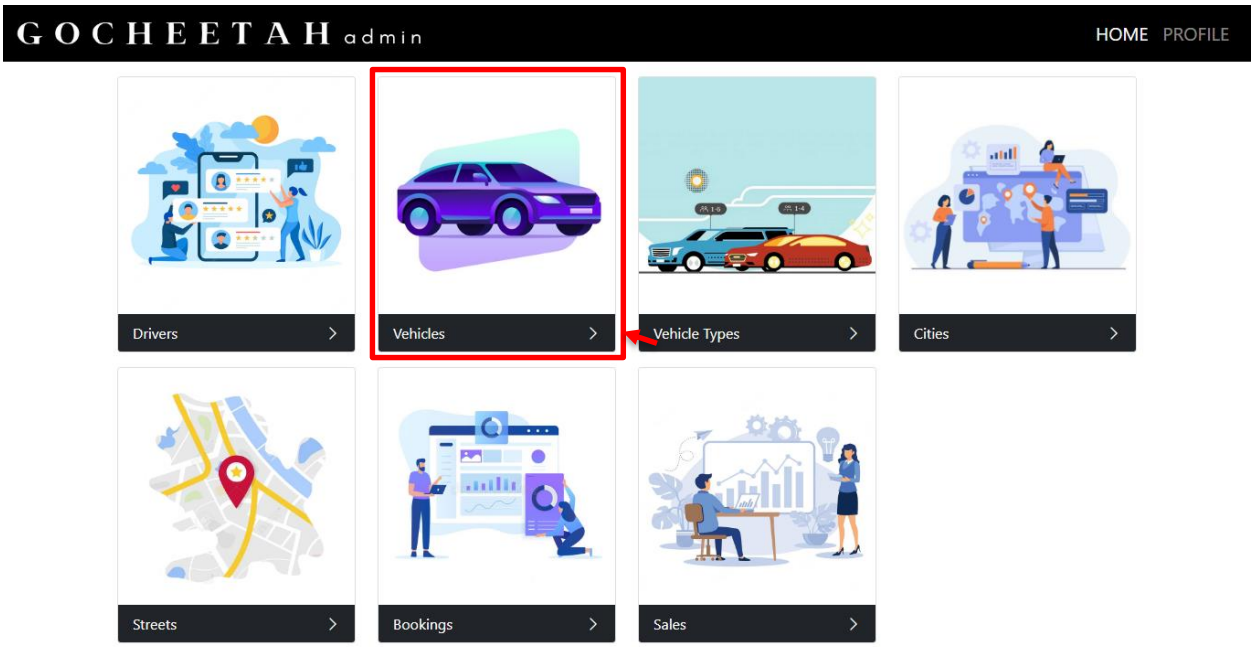
.....



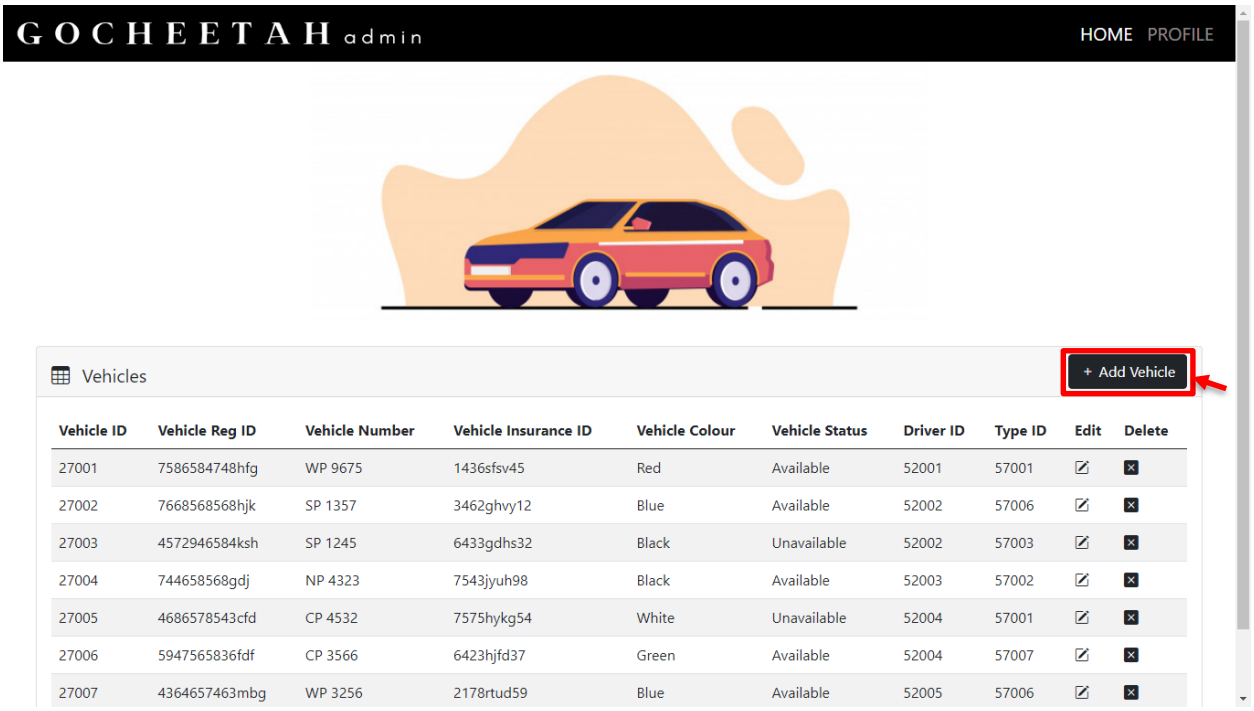
The admin can update an existing driver. And they fill up with required details as shown below and click Update, to update an existing driver.



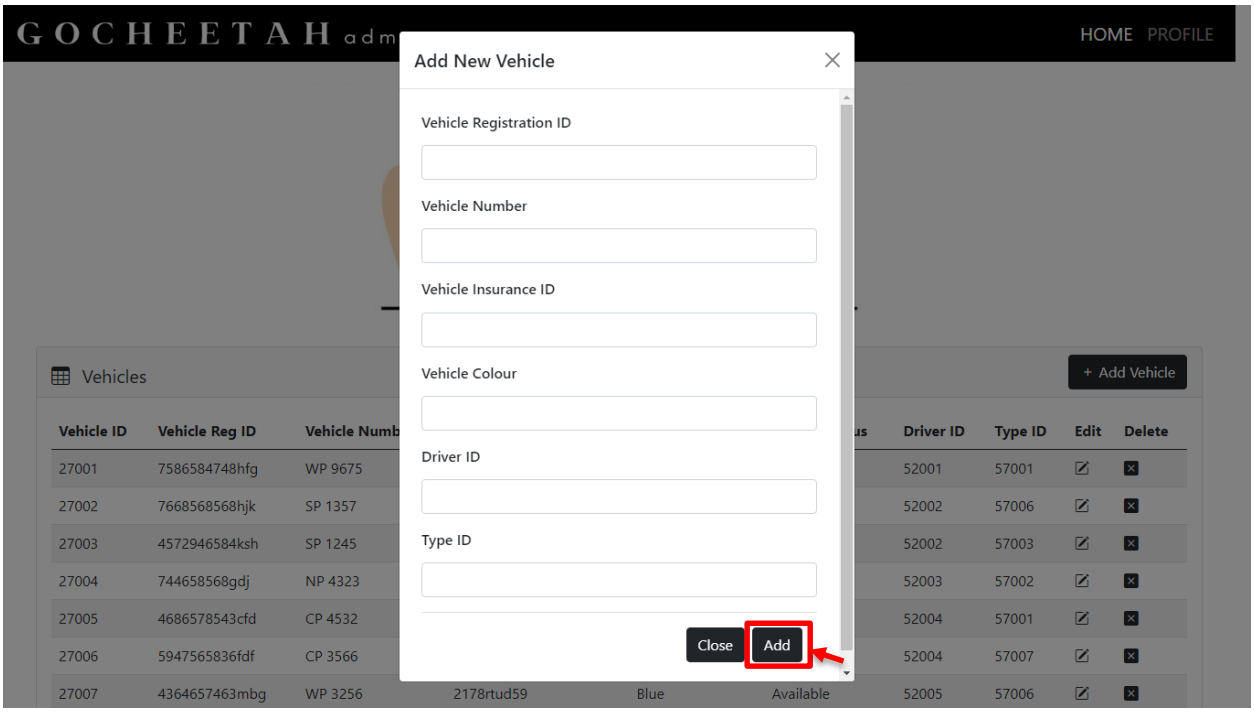


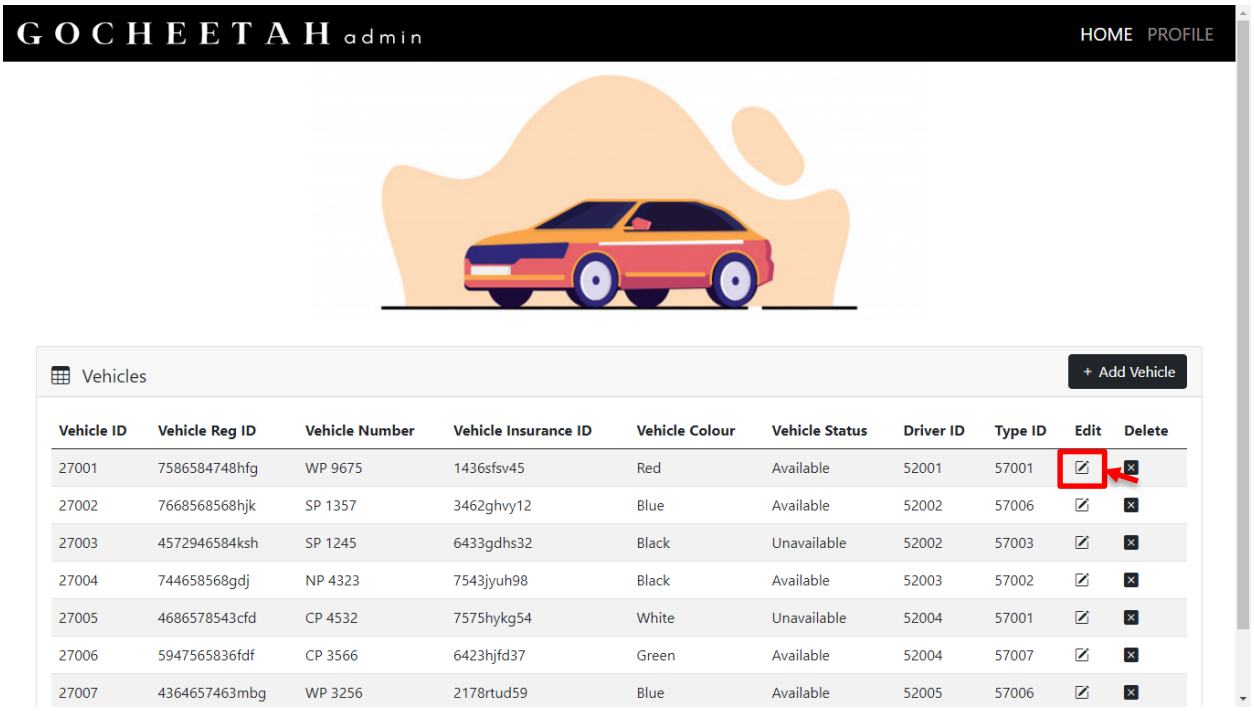


The admin can select the Vehicle, to view all vehicles and if they have to Add, Update or Delete a vehicle. To delete, they can click on the cross icon beside the vehicle. As shown in the image below it shows complete details of the registered vehicles in the system and also facilitates to do other operations.

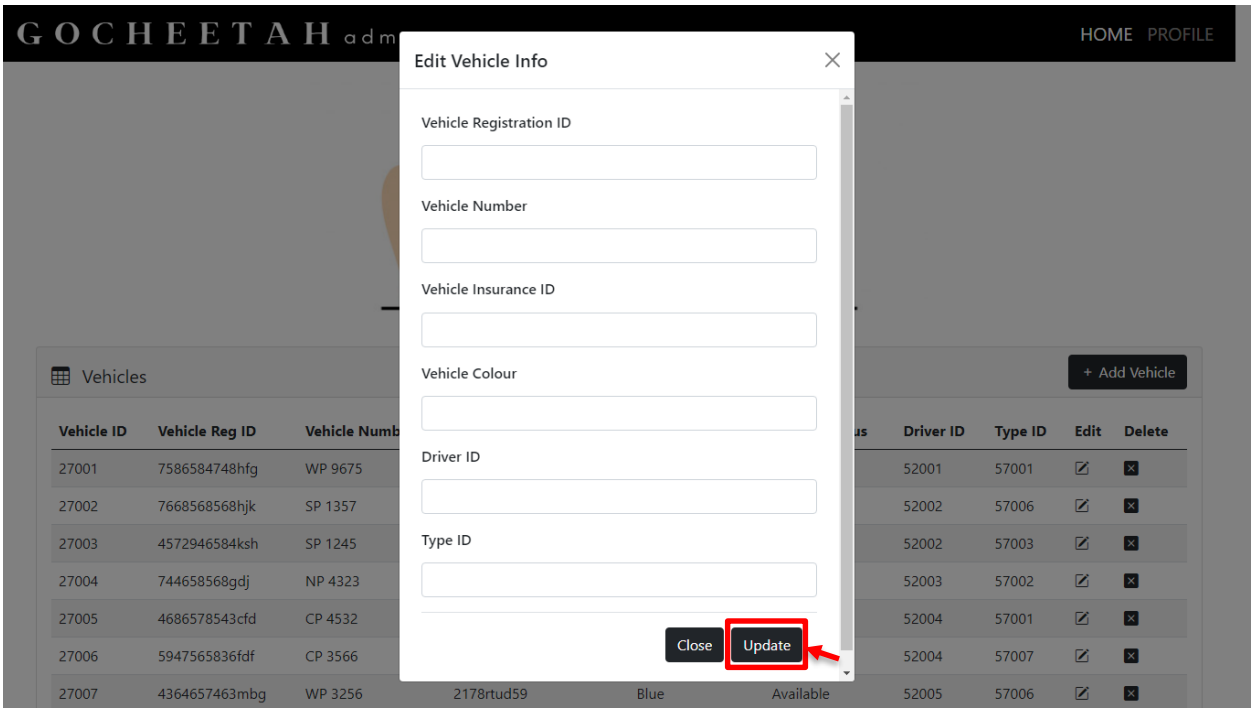


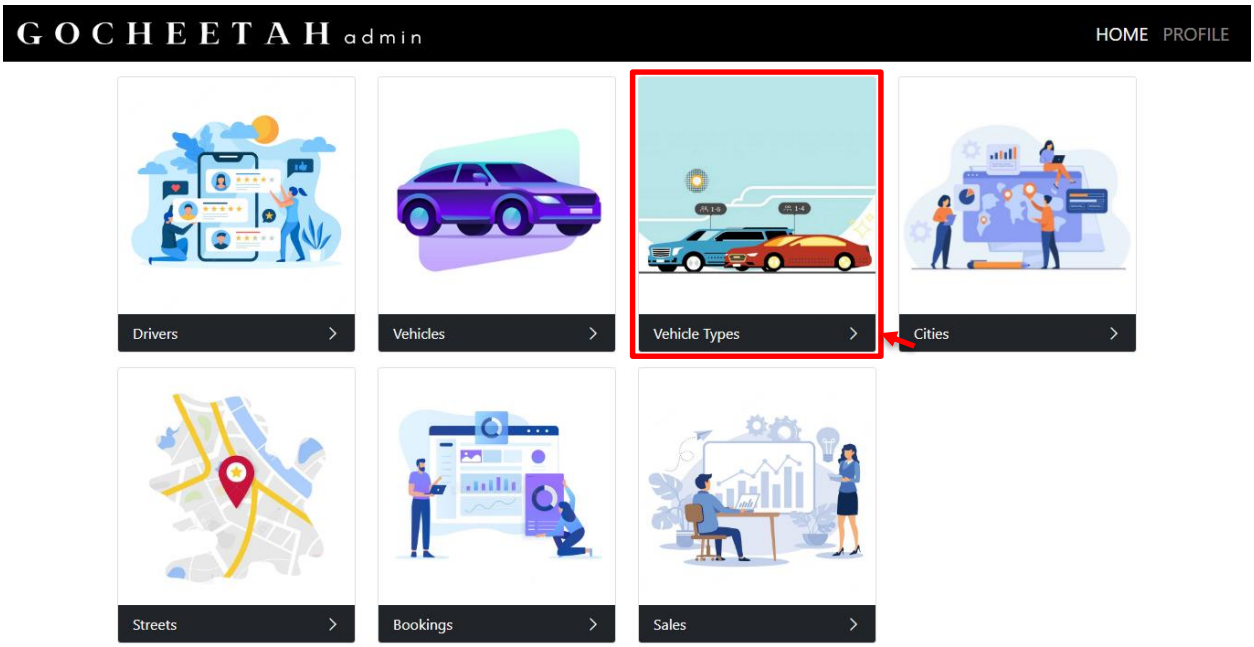
By clicking on Add Vehicle, they are able to add a new vehicle into the system. They need to provide all the required data for a new vehicle. And click Add as shown below.



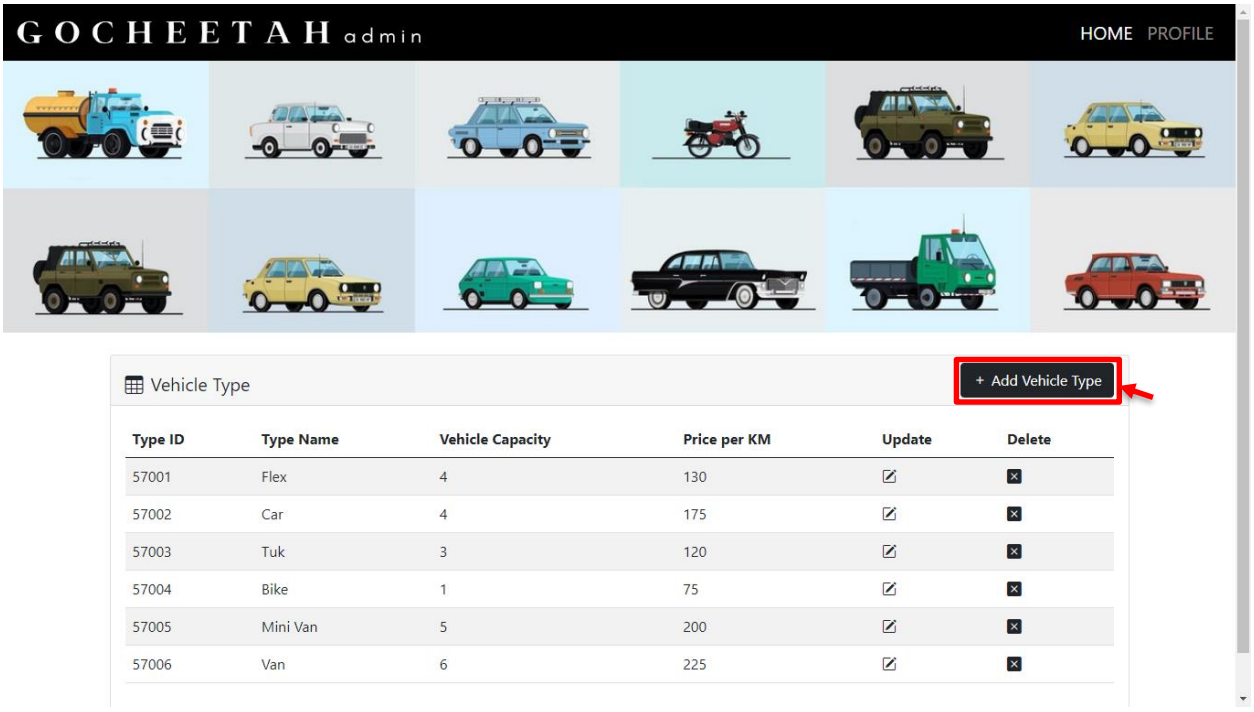


The admin can edit an existing vehicle. And they fill up with required details as shown below and click Update, to update an existing vehicle.

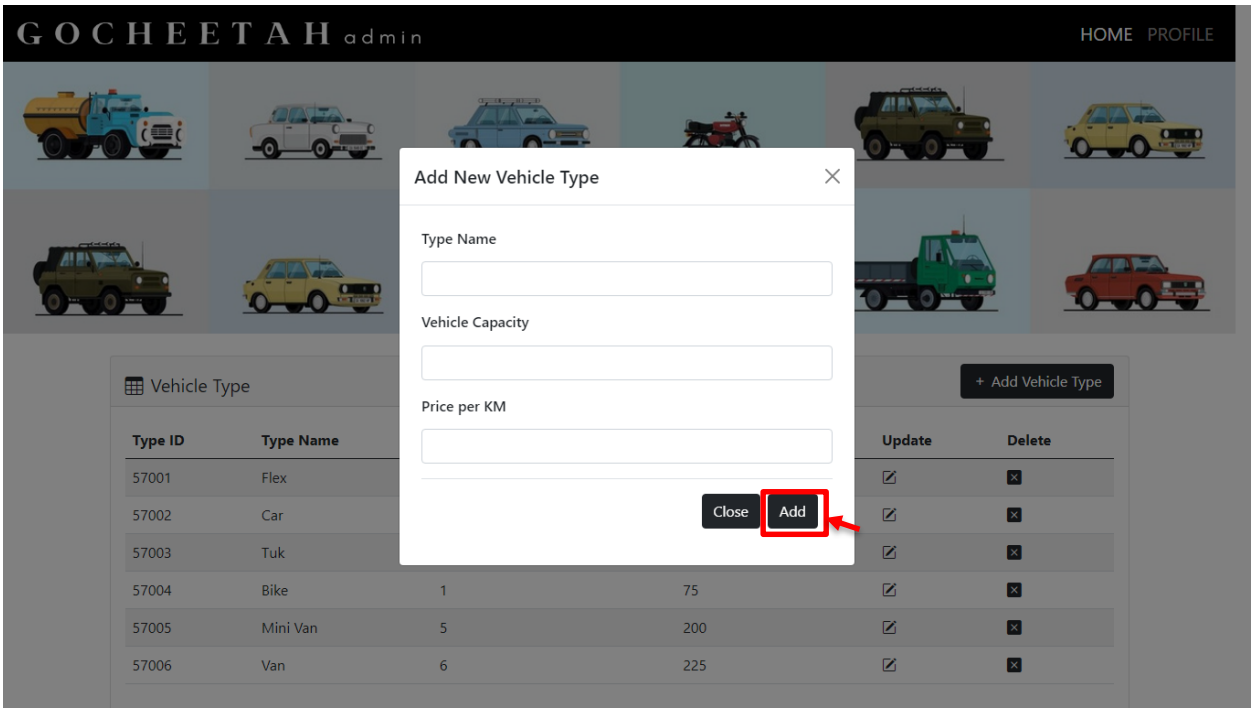




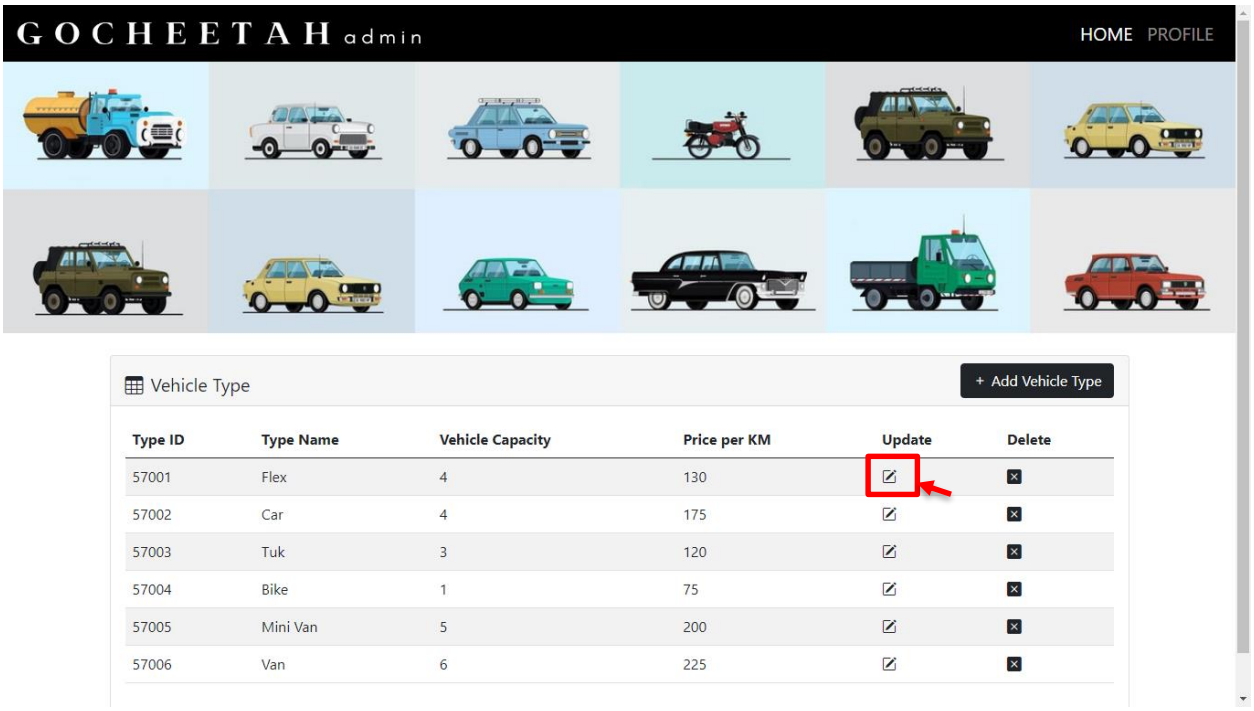
The admin can select the Vehicle Type, to view all vehicle types and if they have to Add, Update or Delete a vehicle type. To delete, they can click on the cross icon beside the vehicle type. As shown in the image below it shows complete details of the registered vehicle types in the system and also facilitates to do other operations.



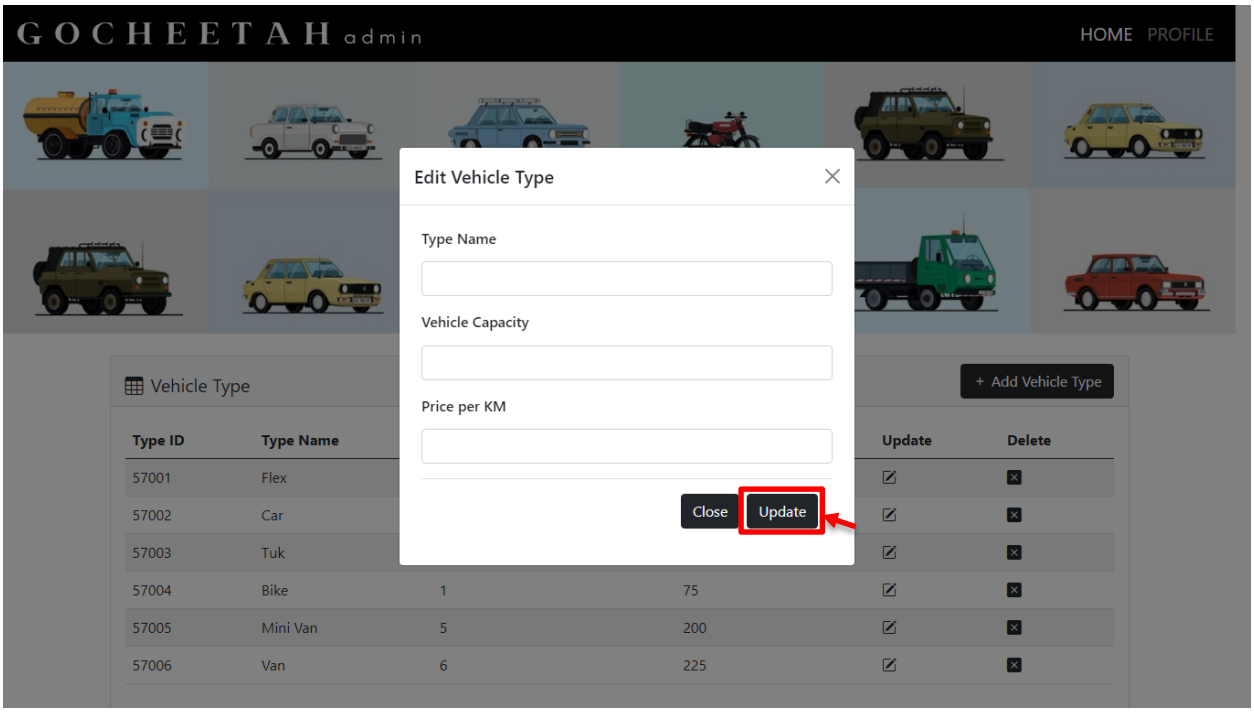
By clicking on Add Vehicle Type, they are able to add a new vehicle type into the system. They need to provide all the required data for a new vehicle type. And click Add as shown below.

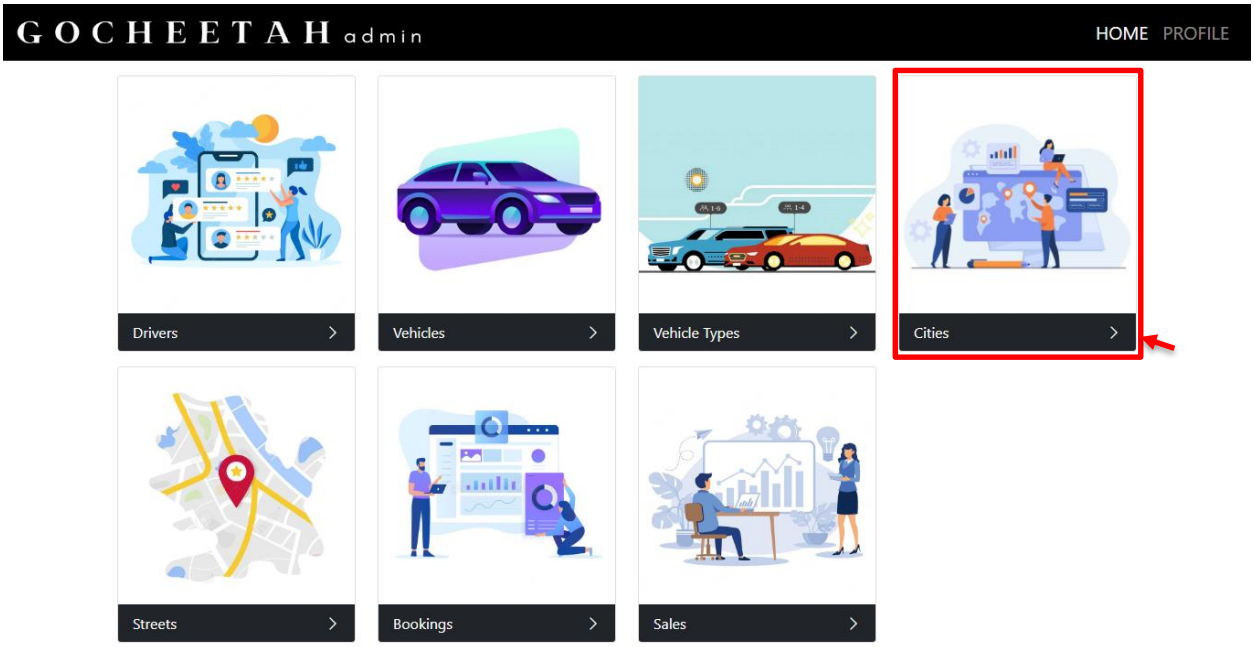


.....

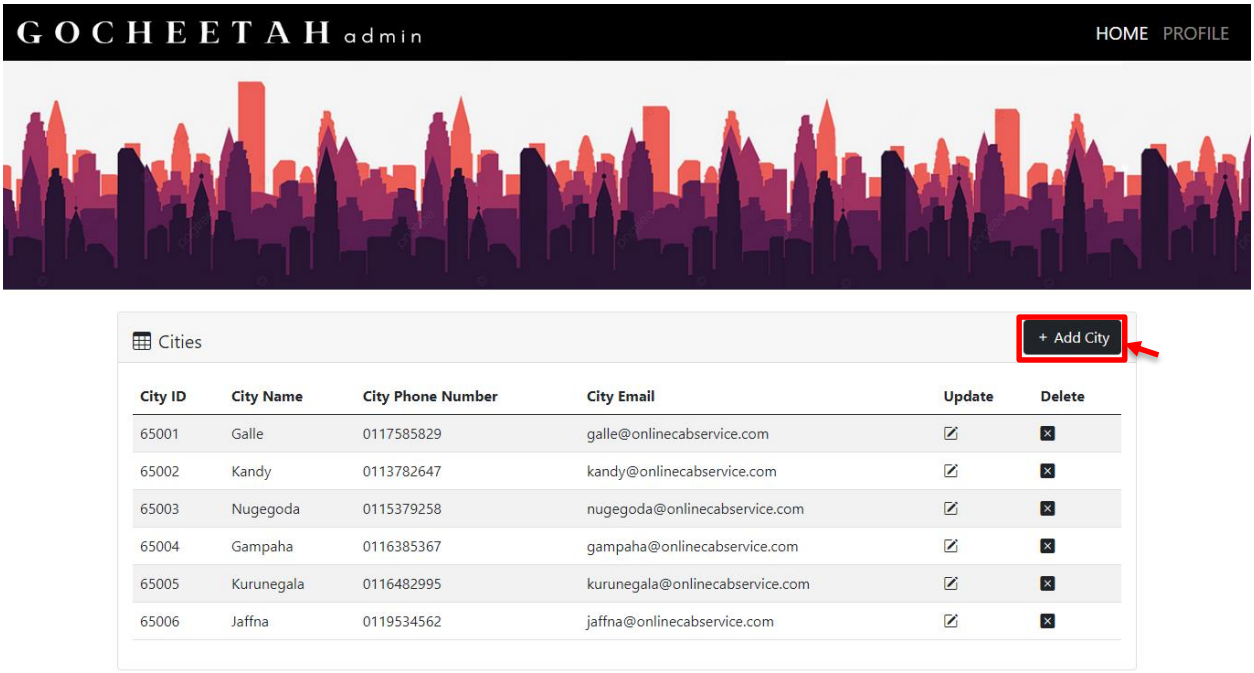


The admin can update an existing vehicle type. And they fill up with required details as shown below and click Update, to update an existing vehicle type.

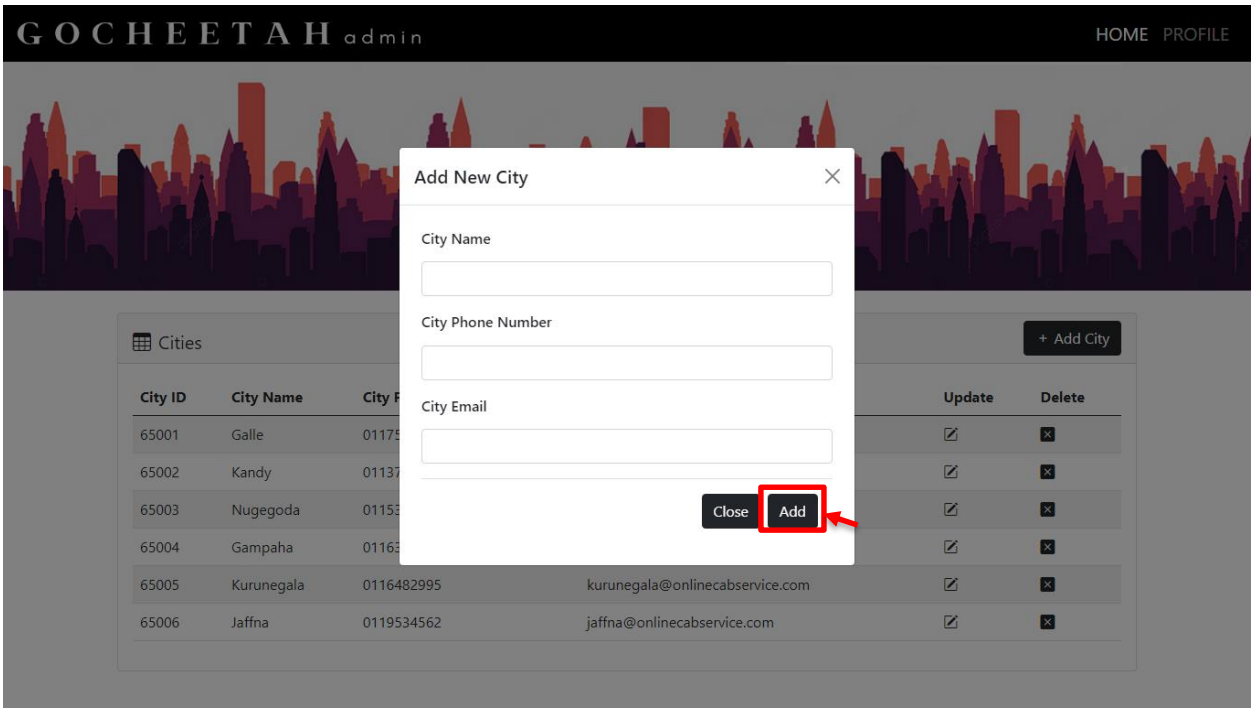




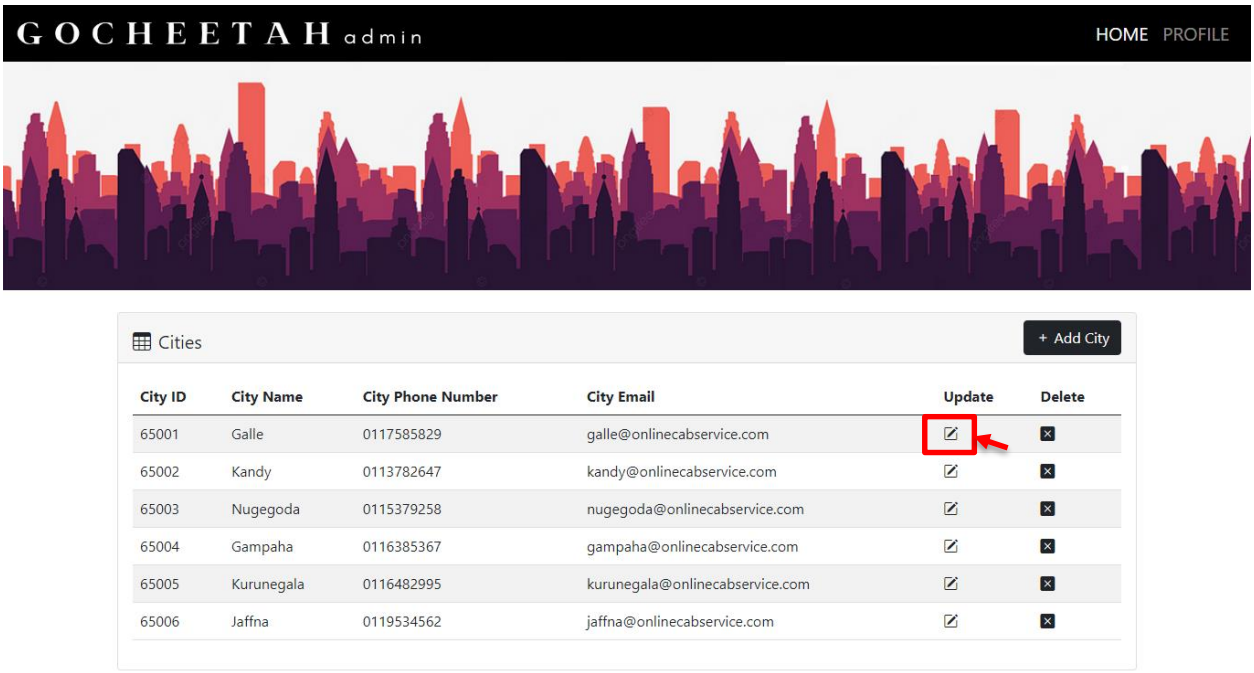
The admin can select the Cities, to view all cities and if they have to Add, Update or Delete a city. To delete, they can click on the cross icon beside the city. As shown in the image below it shows complete details of the registered cities in the system and also facilitates to do other operations.



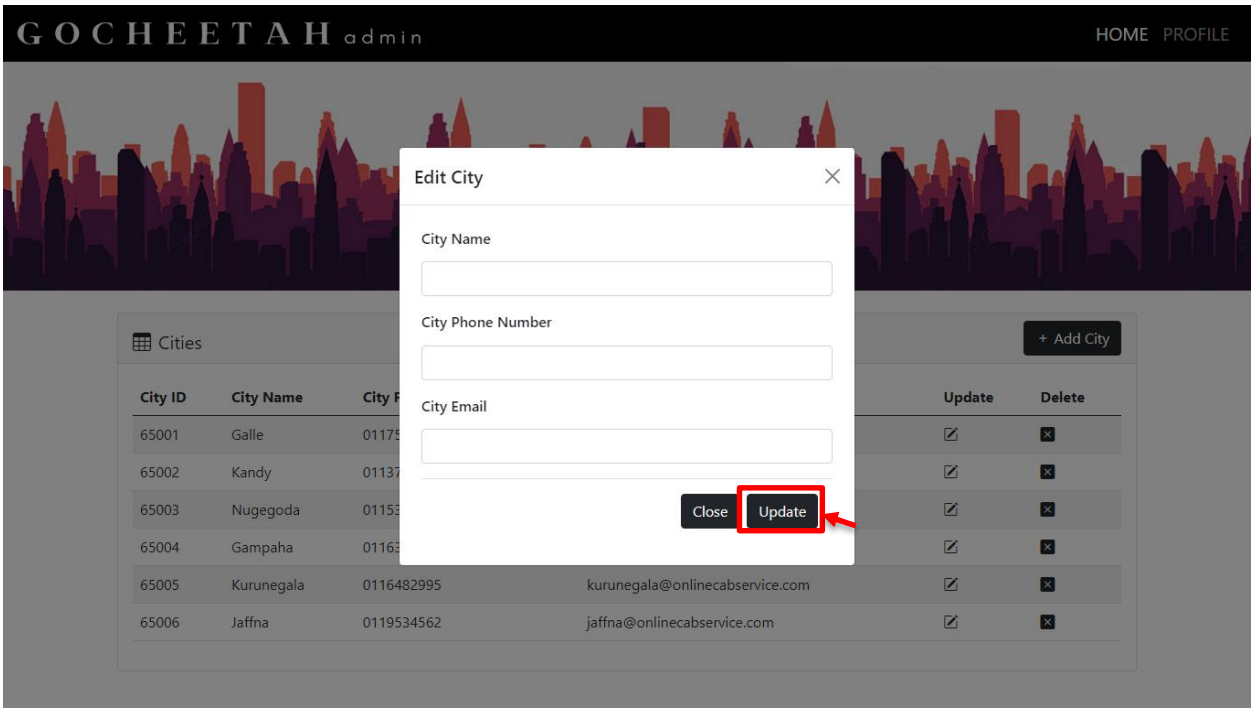
By clicking on Add City, they are able to add a new city into the system. They need to provide all the required data for a new city. And click Add as shown below.

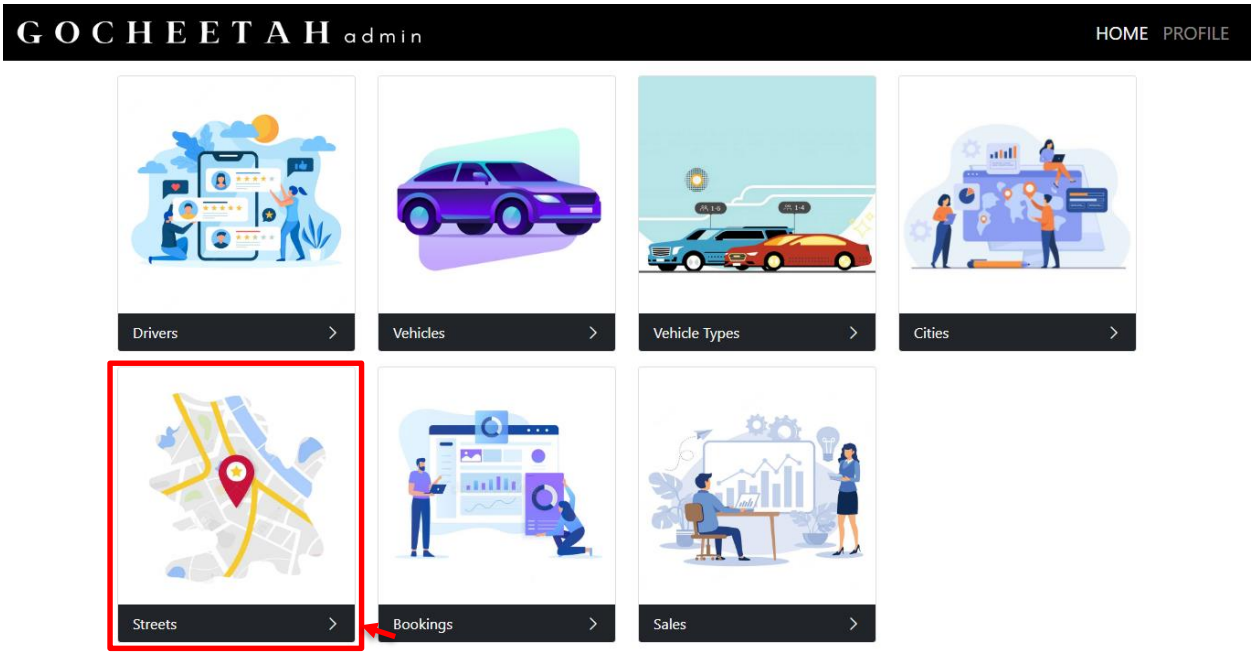


....

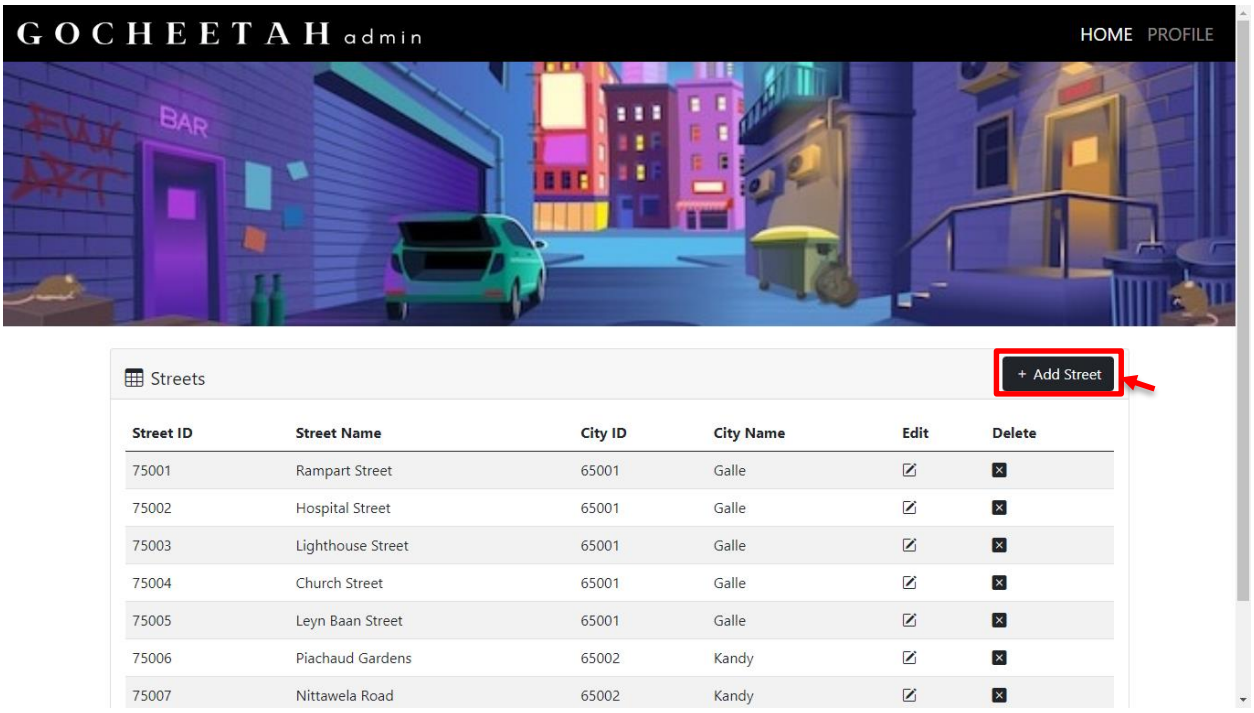


The admin can update an existing city. And they fill up with required details as shown below and click Update, to update an existing city.

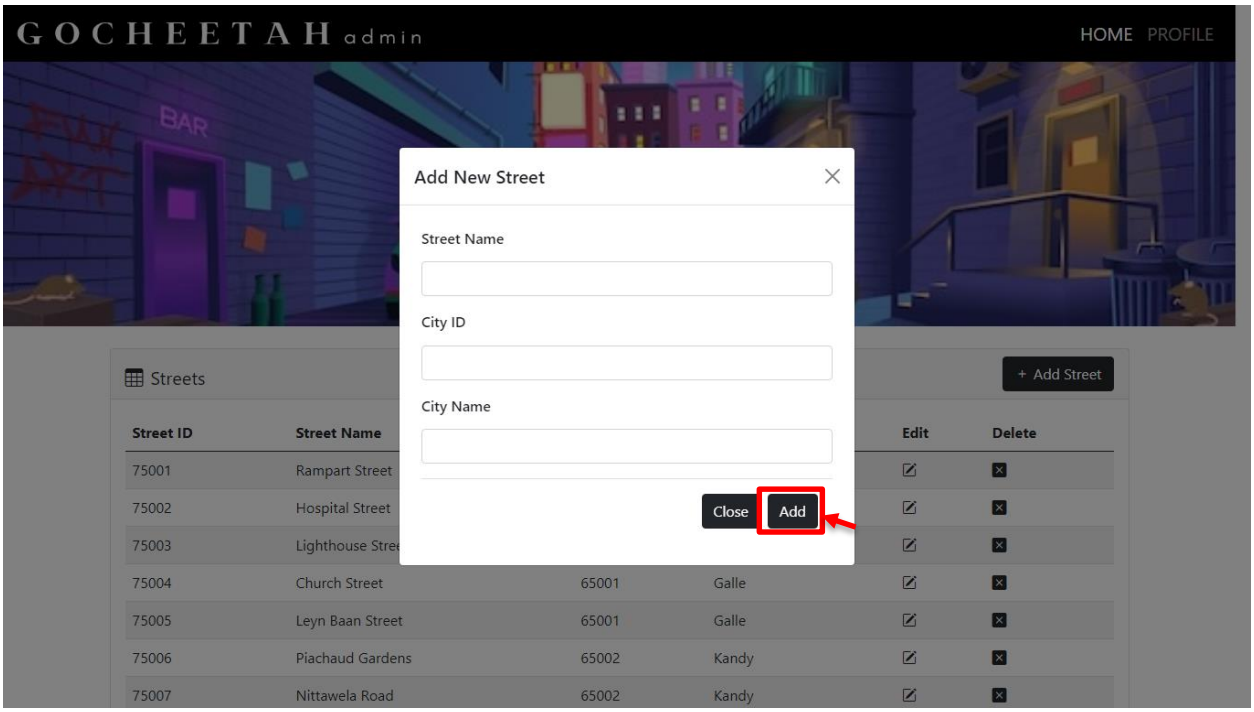




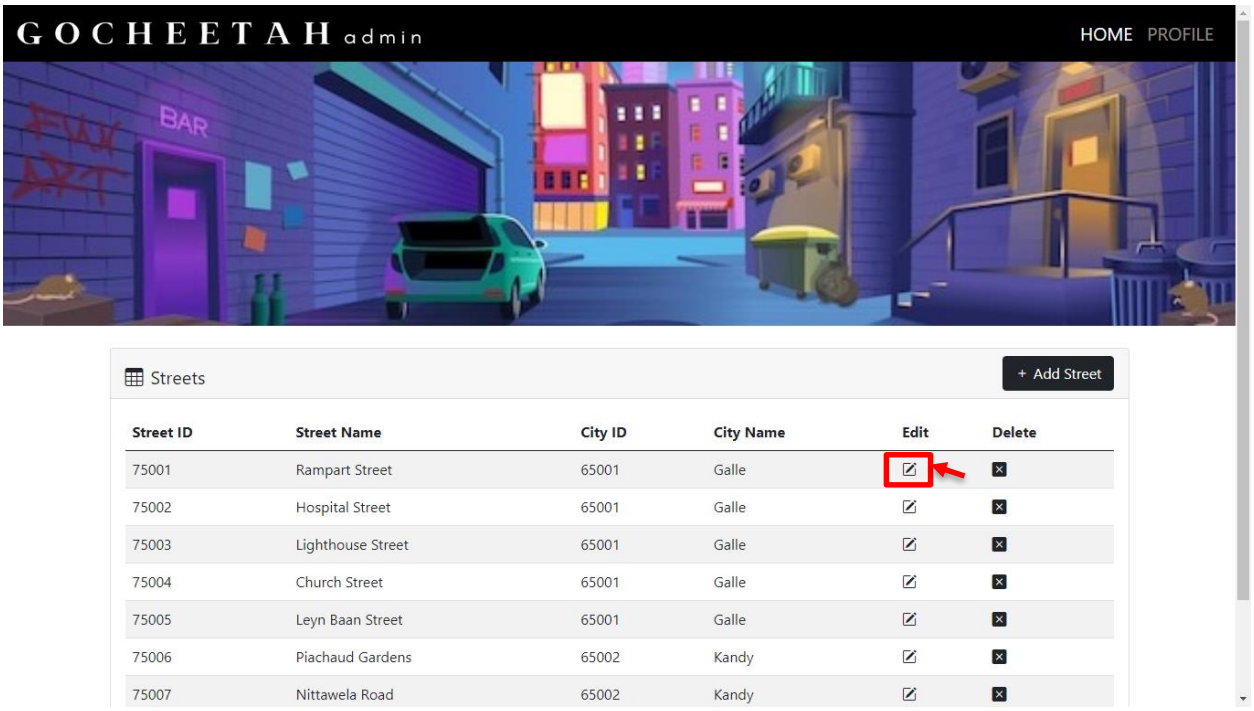
The admin can select the Street, to view all streets and if they have to Add, Update or Delete a street. To delete, they can click on the cross icon beside the street. As shown in the image below it shows complete details of the registered streets in the system and also facilitates to do other operations.



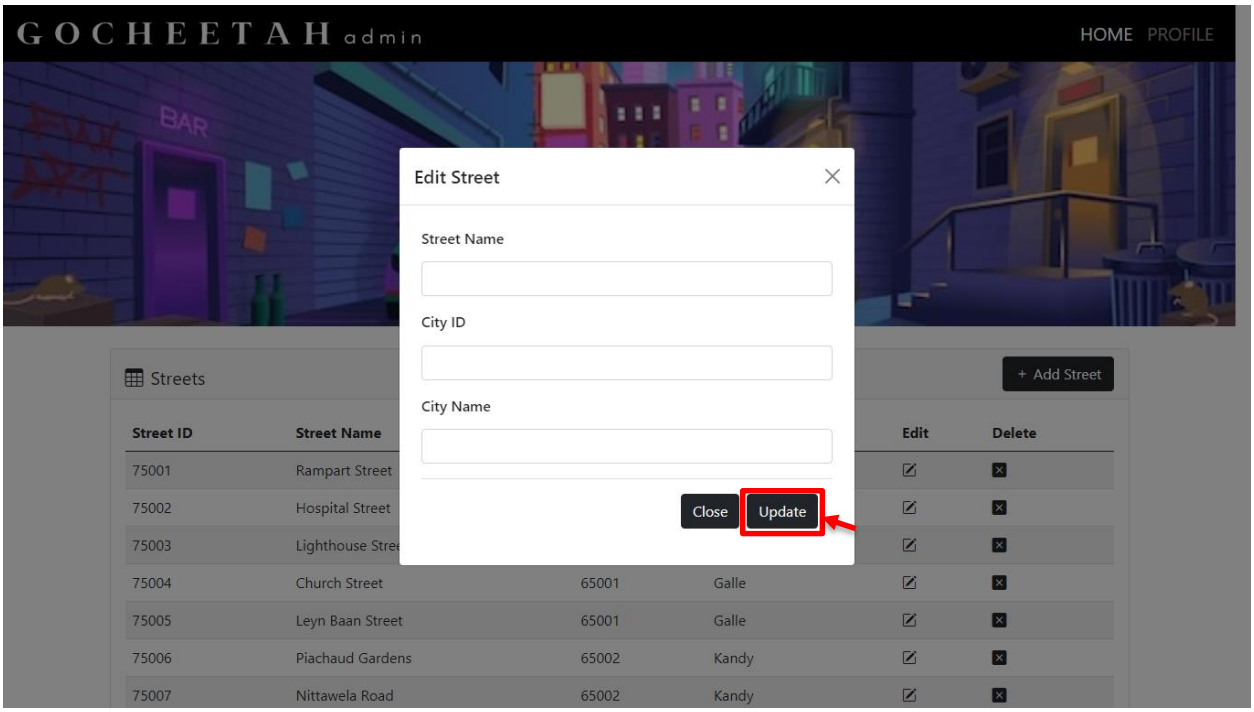
By clicking on Add Street, they are able to add a new street into the system. They need to provide all the required data for a new street. And click Add as shown below.

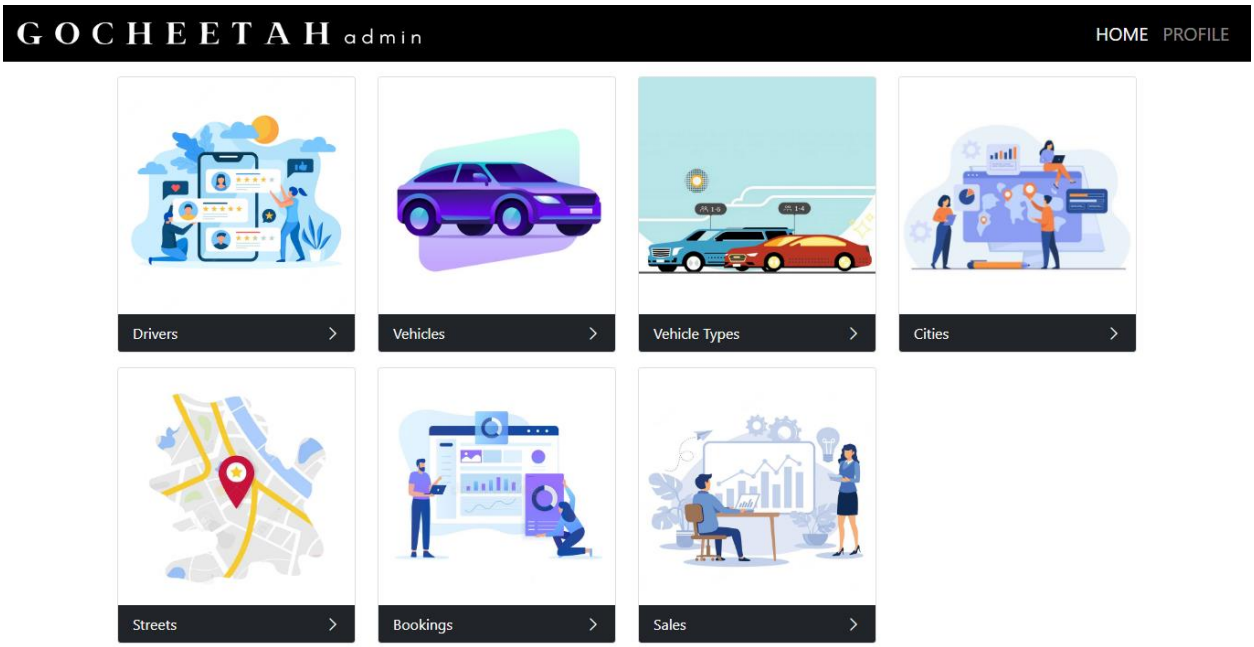


……

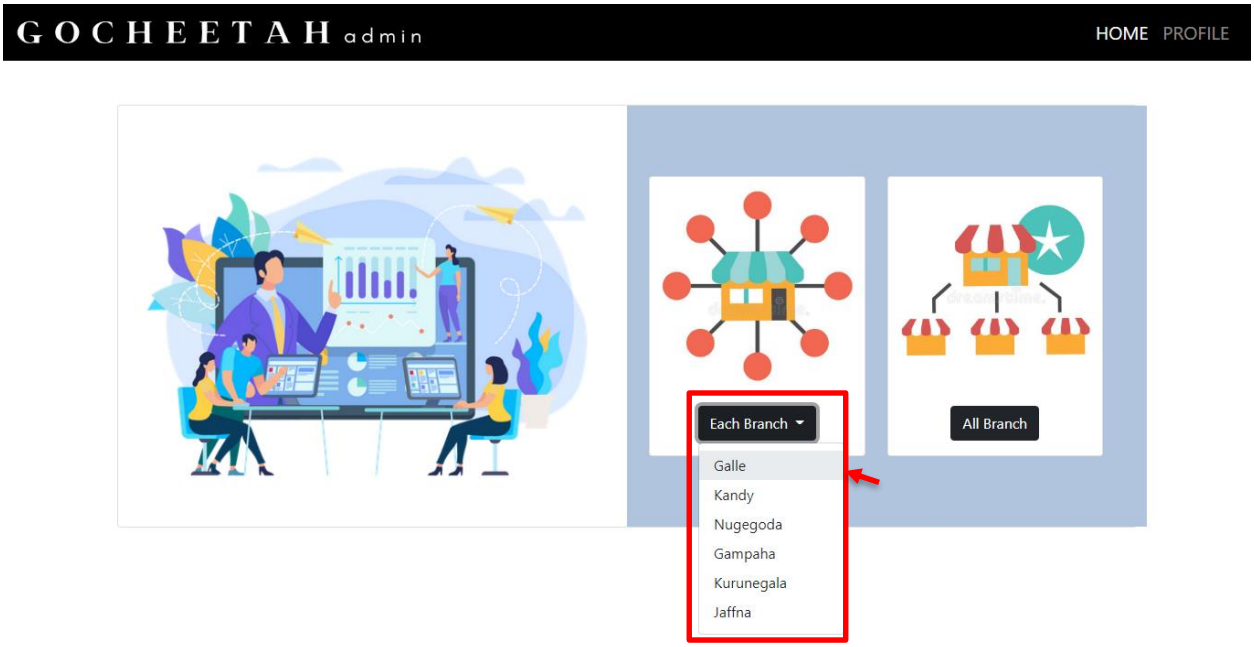


The admin can edit an existing street. And they fill up with required details as shown below and click Update, to update an existing street.

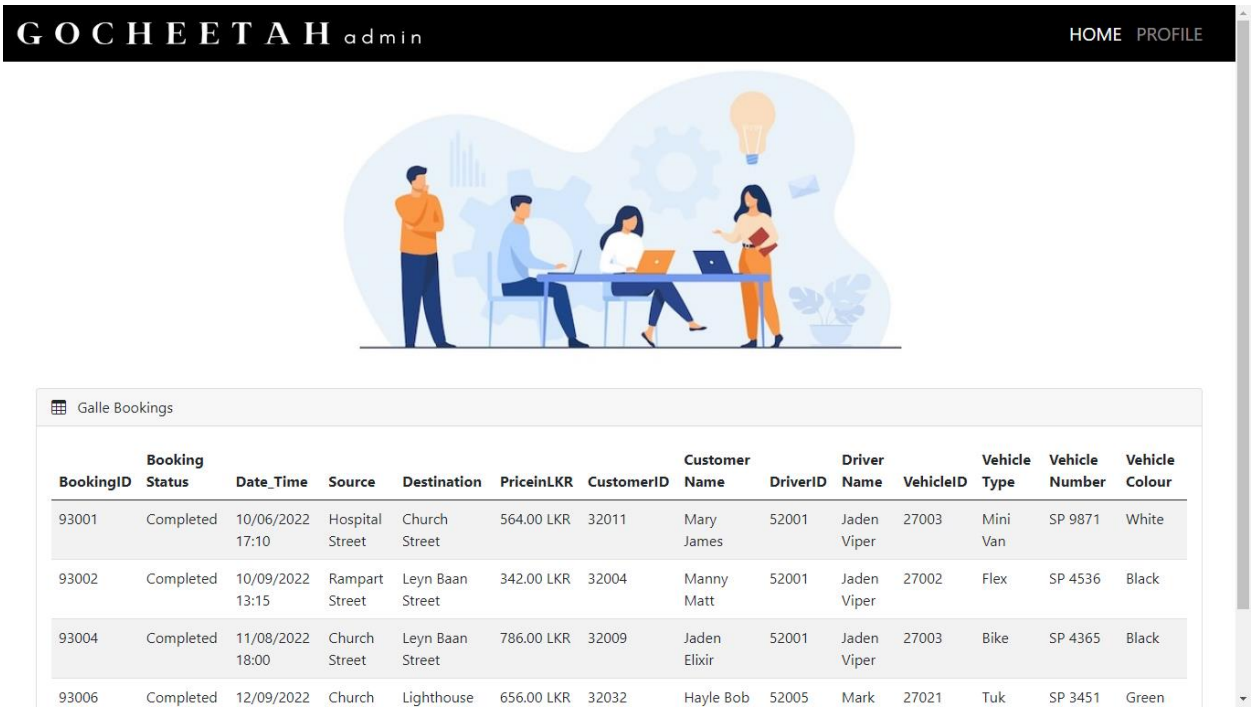




The admin can click on Booking, to view booking details in, all and each city/branch.

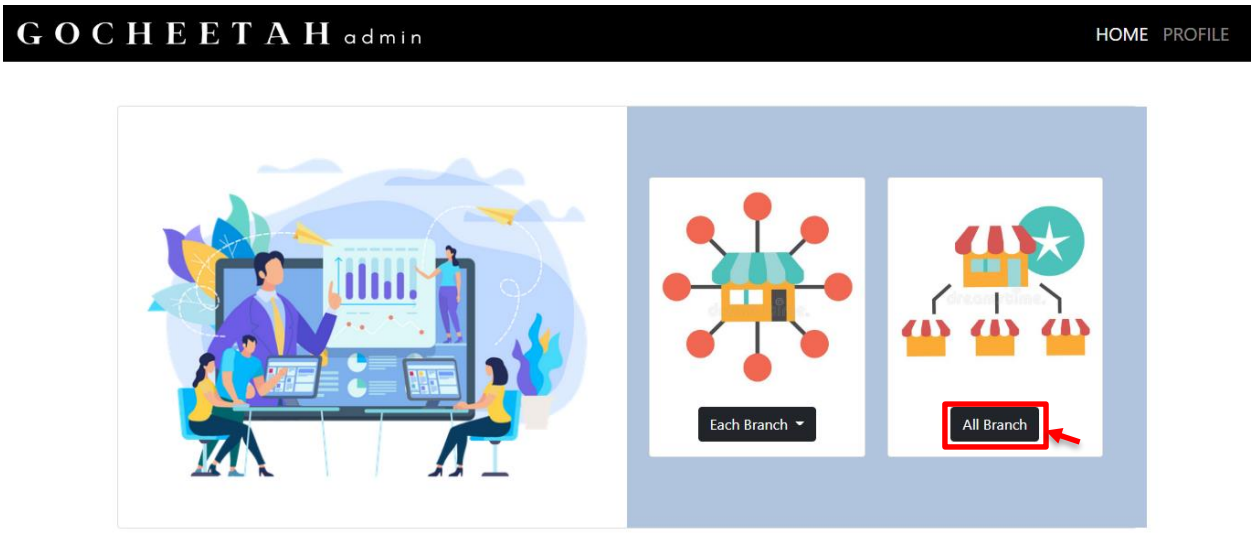


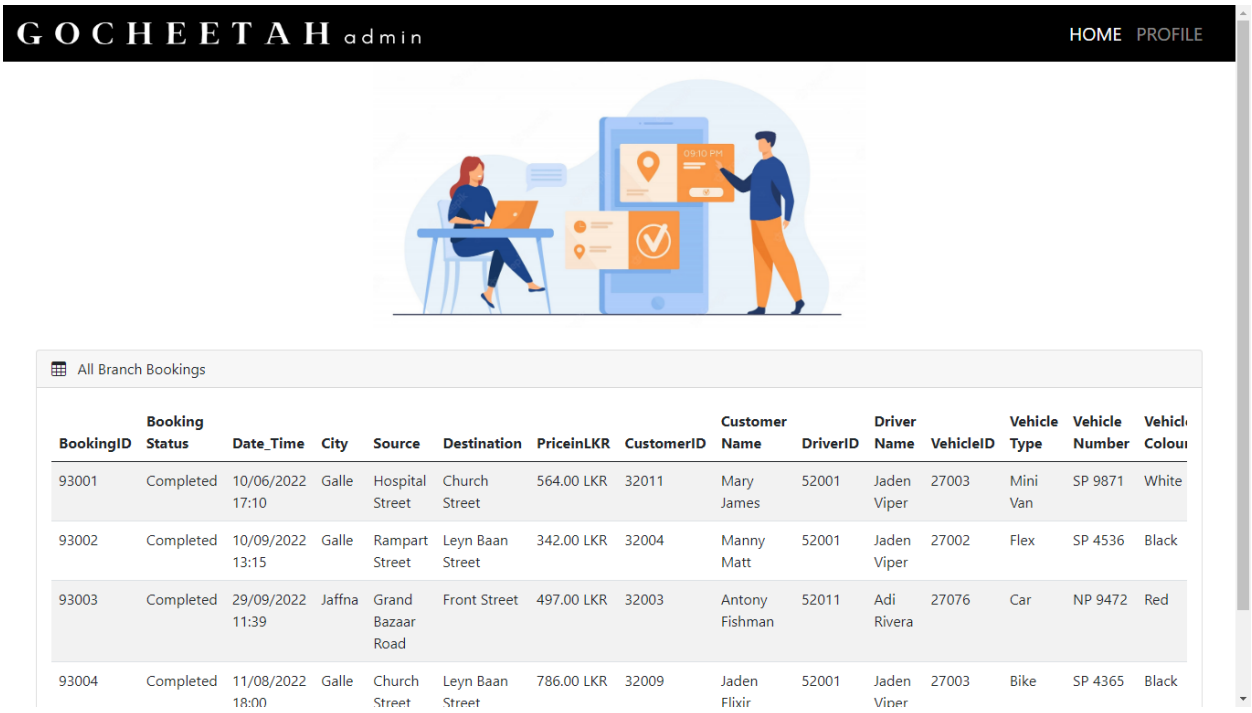
If the admin wants the see the booking in a specific city/branch, they can select the city/branch from the Each Branch drop down menu and view all the registered city/branch where they can choose from.



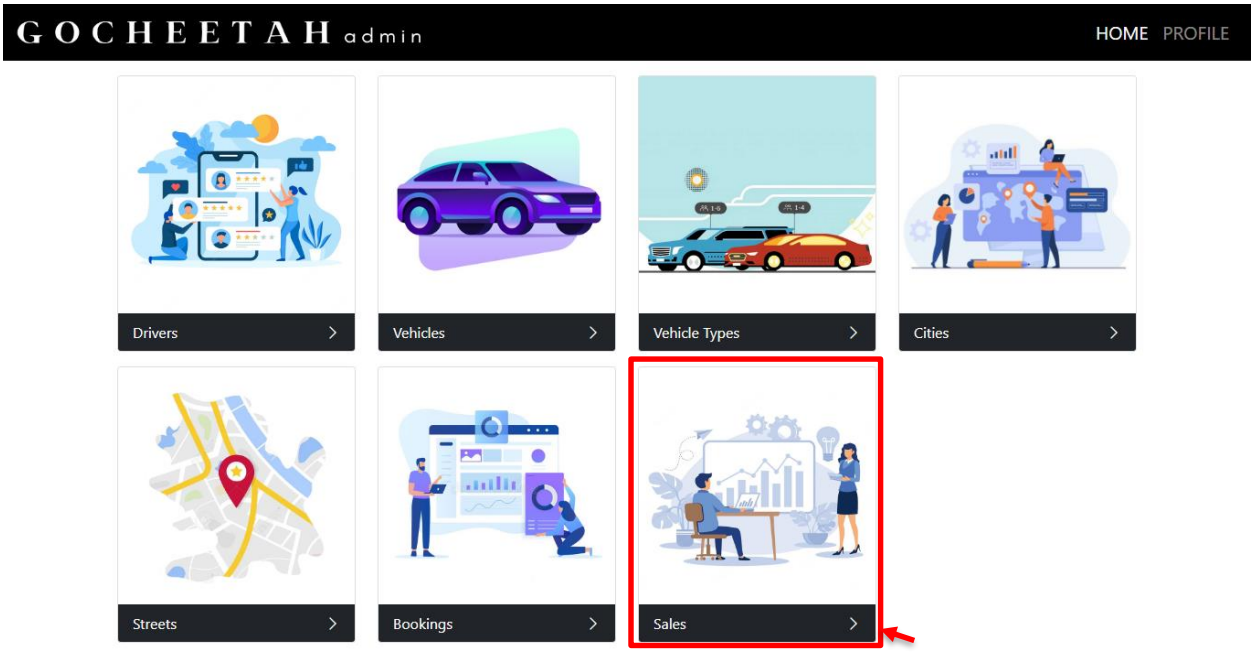
BookingID	Booking Status	Date_Time	Source	Destination	PriceinLKR	CustomerID	Customer Name	DriverID	Driver Name	VehicleID	Vehicle Type	Vehicle Number	Vehicle Colour
93001	Completed	10/06/2022 17:10	Hospital Street	Church Street	564.00 LKR	32011	Mary James	52001	Jaden Viper	27003	Mini Van	SP 9871	White
93002	Completed	10/09/2022 13:15	Rampart Street	Leyn Baan Street	342.00 LKR	32004	Manny Matt	52001	Jaden Viper	27002	Flex	SP 4536	Black
93004	Completed	11/08/2022 18:00	Church Street	Leyn Baan Street	786.00 LKR	32009	Jaden Elixir	52001	Jaden Viper	27003	Bike	SP 4365	Black
93006	Completed	12/09/2022	Church	Lighthouse	656.00 LKR	32032	Hayle Bob	52005	Mark	27021	Tuk	SP 3451	Green

If the admin wants the see the booking in all /branch, they can select All Branch and view the booking details for all the branches along with their booking, location, customer, vehicle, pricing and driver details.

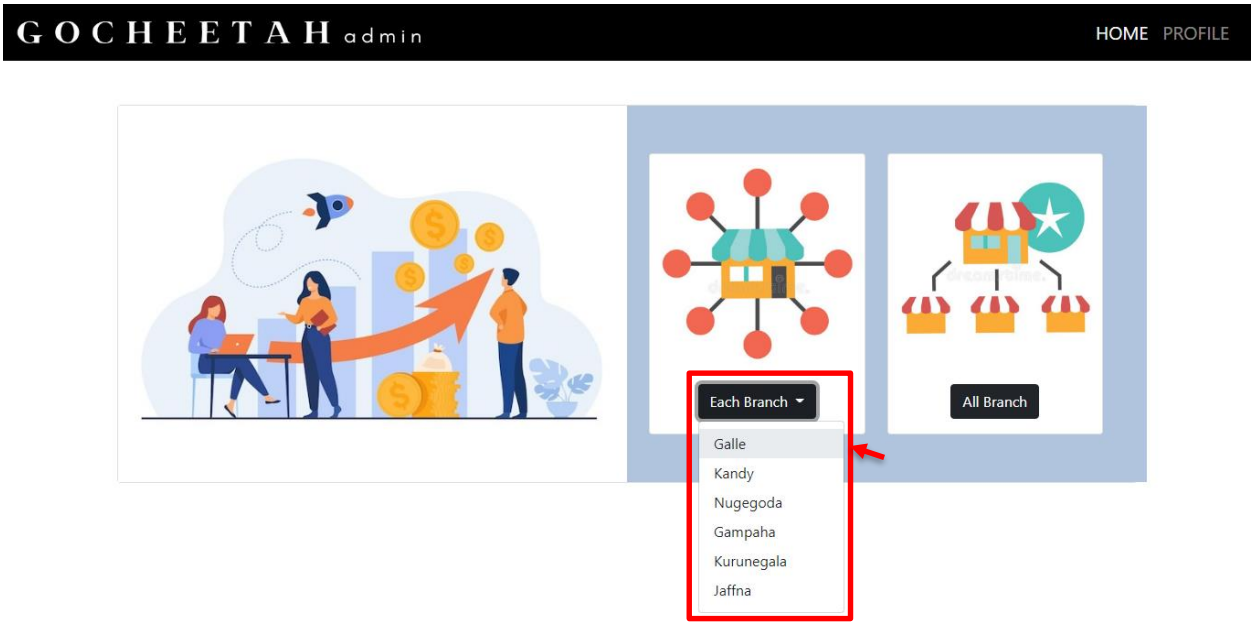




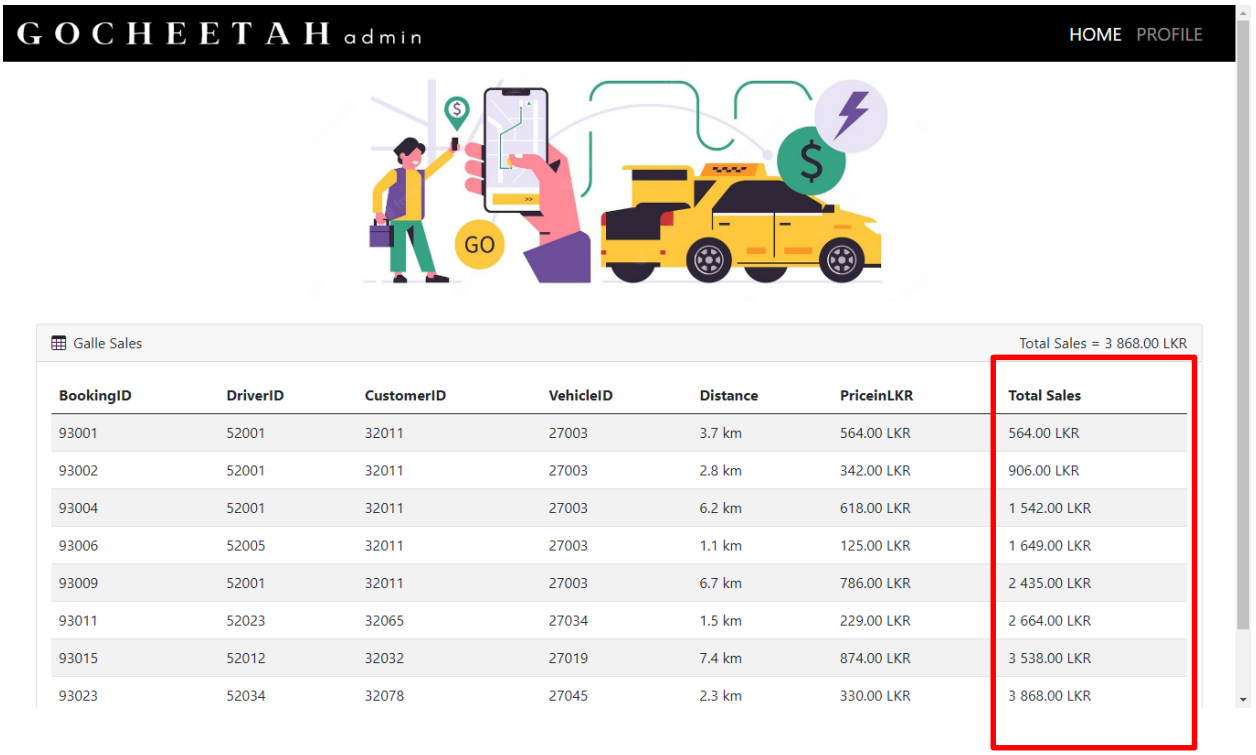
BookingID	Booking Status	Date_Time	City	Source	Destination	PriceinLKR	CustomerID	Customer Name	DriverID	Driver Name	VehicleID	Vehicle Type	Vehicle Number	Vehicl Colou
93001	Completed	10/06/2022 17:10	Galle	Hospital Street	Church Street	564.00 LKR	32011	Mary James	52001	Jaden Viper	27003	Mini Van	SP 9871	White
93002	Completed	10/09/2022 13:15	Galle	Rampart Street	Leyn Baan Street	342.00 LKR	32004	Manny Matt	52001	Jaden Viper	27002	Flex	SP 4536	Black
93003	Completed	29/09/2022 11:39	Jaffna	Grand Bazaar Road	Front Street	497.00 LKR	32003	Antony Fishman	52011	Adi Rivera	27076	Car	NP 9472	Red
93004	Completed	11/08/2022 18:00	Galle	Church Street	Leyn Baan Street	786.00 LKR	32009	Jaden Flixir	52001	Jaden Viper	27003	Bike	SP 4365	Black



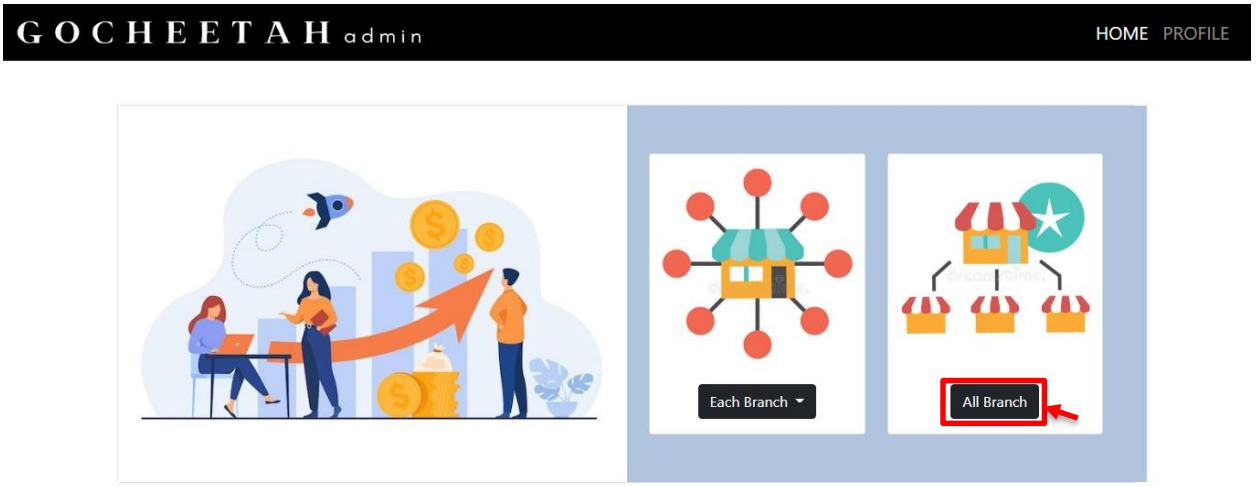
The admin can click on Sales, to view sales details in, all and each city/branch.



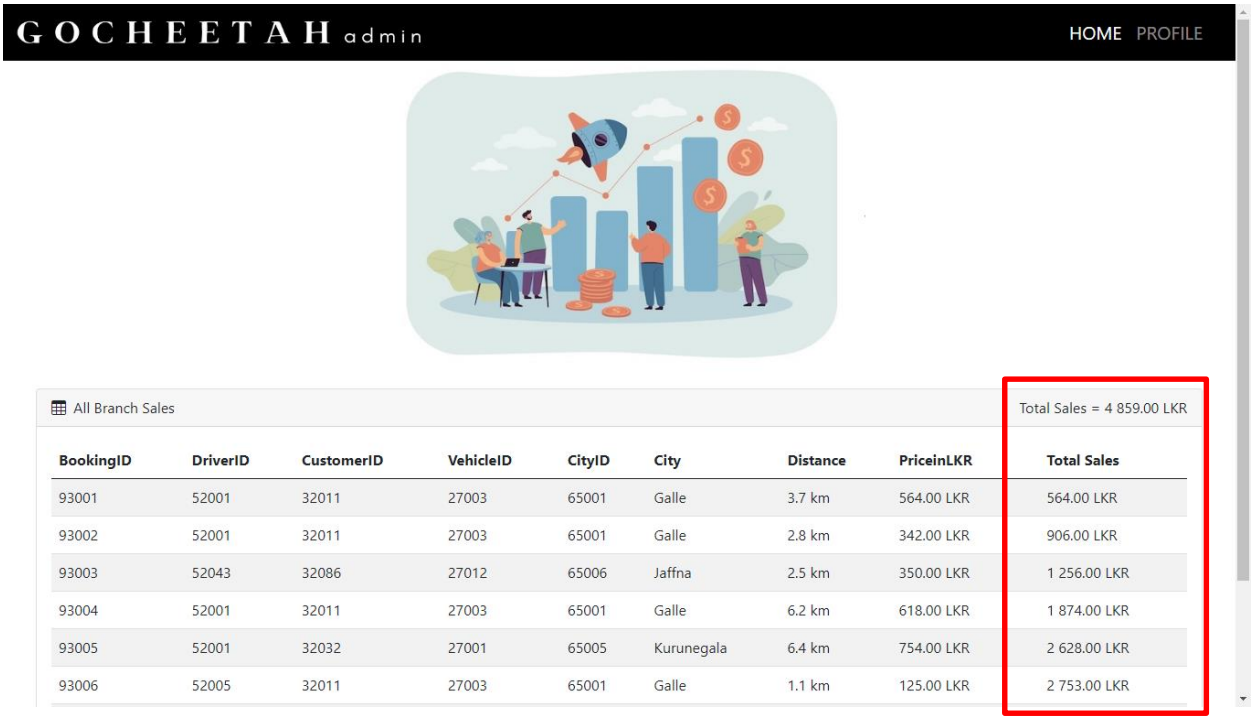
If the admin wants the see the sales in a specific city/branch, they can select the city/branch from the Each Branch drop down menu and view all the registered branch where they can choose from to view the complete sales details along with the booking details.



BookingID	DriverID	CustomerID	VehicleID	Distance	PriceinLKR	Total Sales
93001	52001	32011	27003	3.7 km	564.00 LKR	564.00 LKR
93002	52001	32011	27003	2.8 km	342.00 LKR	906.00 LKR
93004	52001	32011	27003	6.2 km	618.00 LKR	1 542.00 LKR
93006	52005	32011	27003	1.1 km	125.00 LKR	1 649.00 LKR
93009	52001	32011	27003	6.7 km	786.00 LKR	2 435.00 LKR
93011	52023	32065	27034	1.5 km	229.00 LKR	2 664.00 LKR
93015	52012	32032	27019	7.4 km	874.00 LKR	3 538.00 LKR
93023	52034	32078	27045	2.3 km	330.00 LKR	3 868.00 LKR

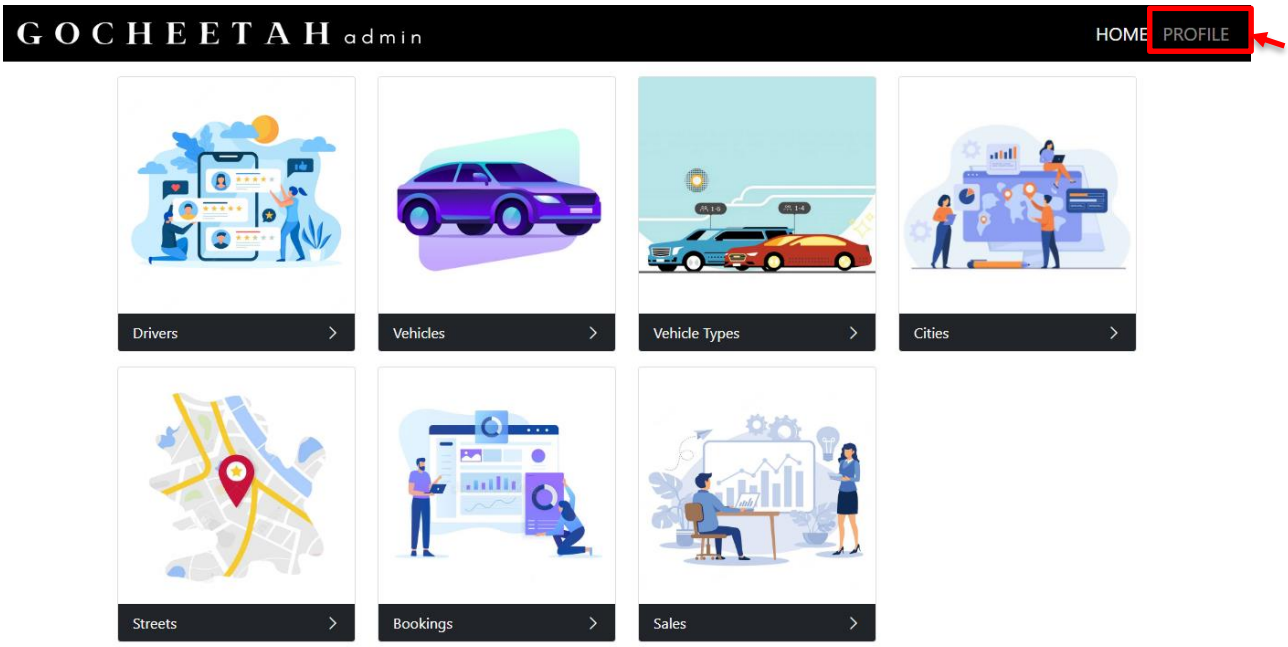


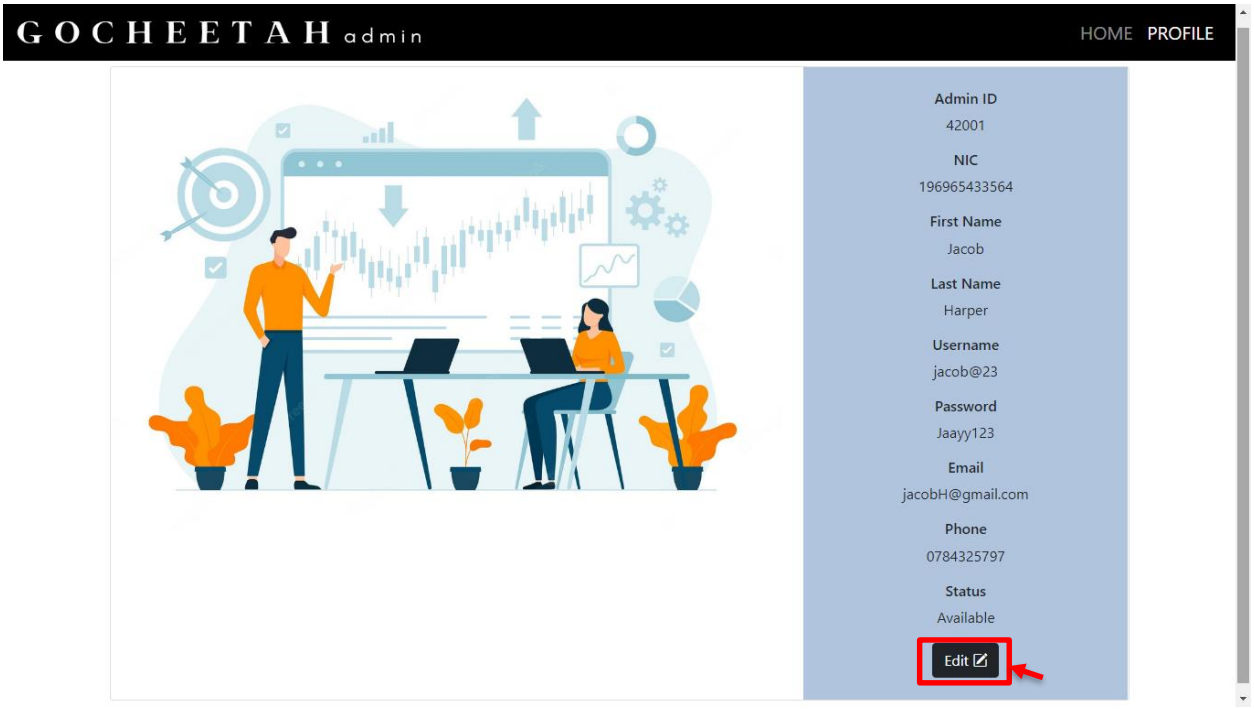
If the admin wants the see the sales in all city/branch, they can select All Branch and view the complete sales details along with the booking details.



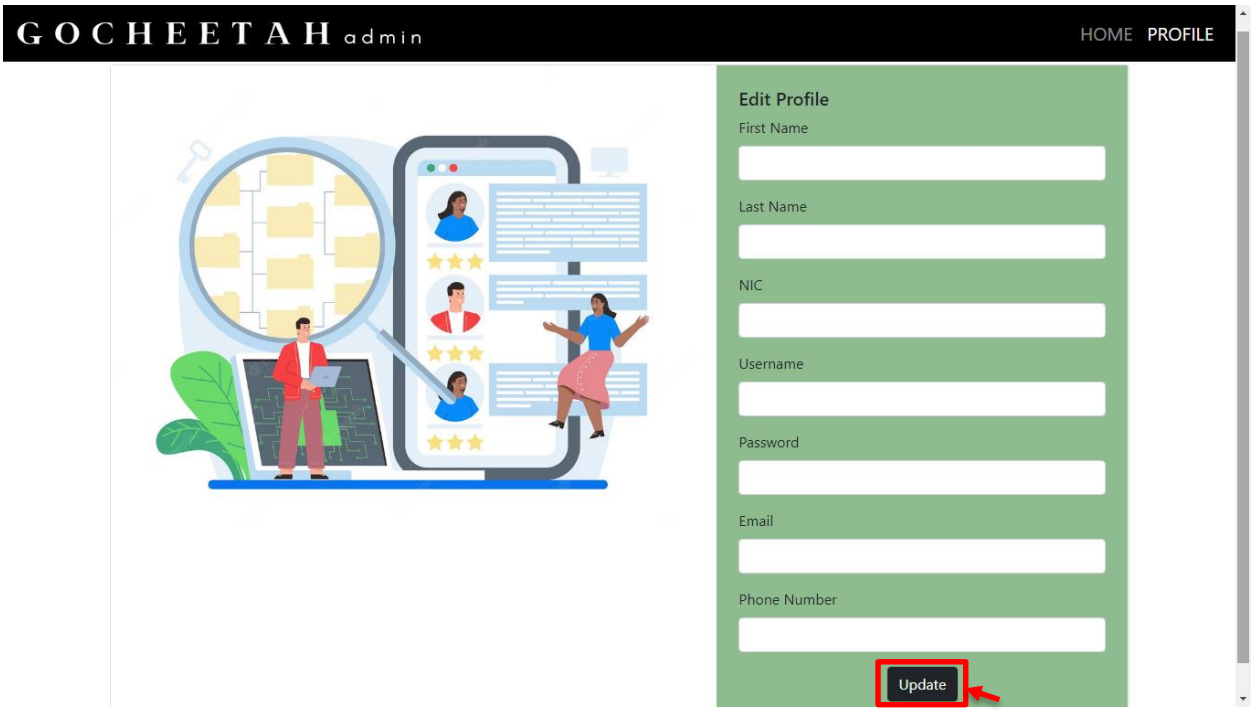
BookingID	DriverID	CustomerID	VehicleID	CityID	City	Distance	PriceinLKR	Total Sales
93001	52001	32011	27003	65001	Galle	3.7 km	564.00 LKR	564.00 LKR
93002	52001	32011	27003	65001	Galle	2.8 km	342.00 LKR	906.00 LKR
93003	52043	32086	27012	65006	Jaffna	2.5 km	350.00 LKR	1 256.00 LKR
93004	52001	32011	27003	65001	Galle	6.2 km	618.00 LKR	1 874.00 LKR
93005	52001	32032	27001	65005	Kurunegala	6.4 km	754.00 LKR	2 628.00 LKR
93006	52005	32011	27003	65001	Galle	1.1 km	125.00 LKR	2 753.00 LKR

The admin can view their profile details at anytime from the Profile panel.





They are also able to edit their profile details anytime by clicking on as shown above. They can give the required details with the changes they want to make and click on Update as shown below to update the changes.



(TASK G) GITHUB

The GitHub repository link:

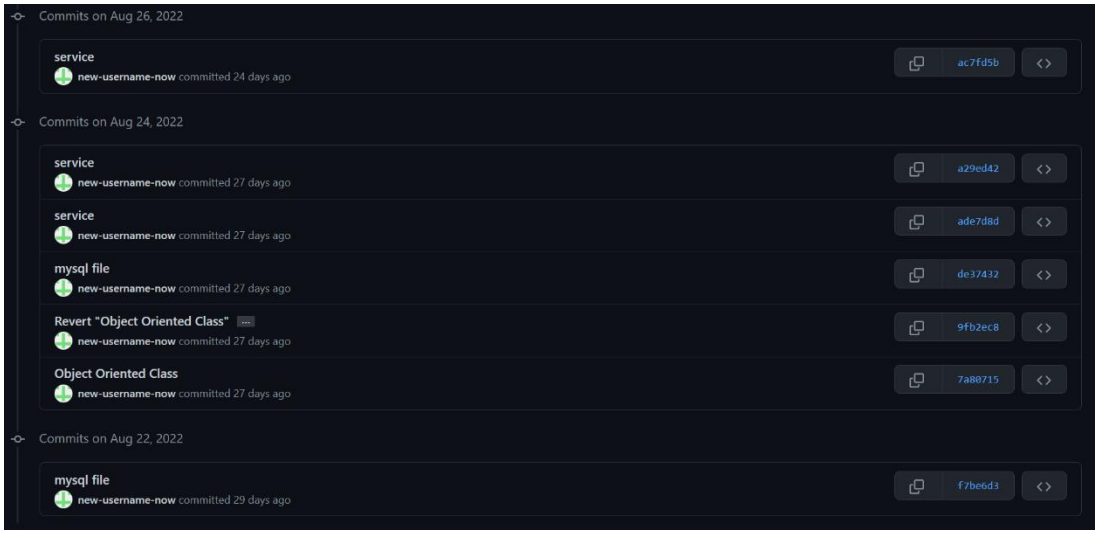
https://github.com/new-username-now/Online-Cab-Service-System.git

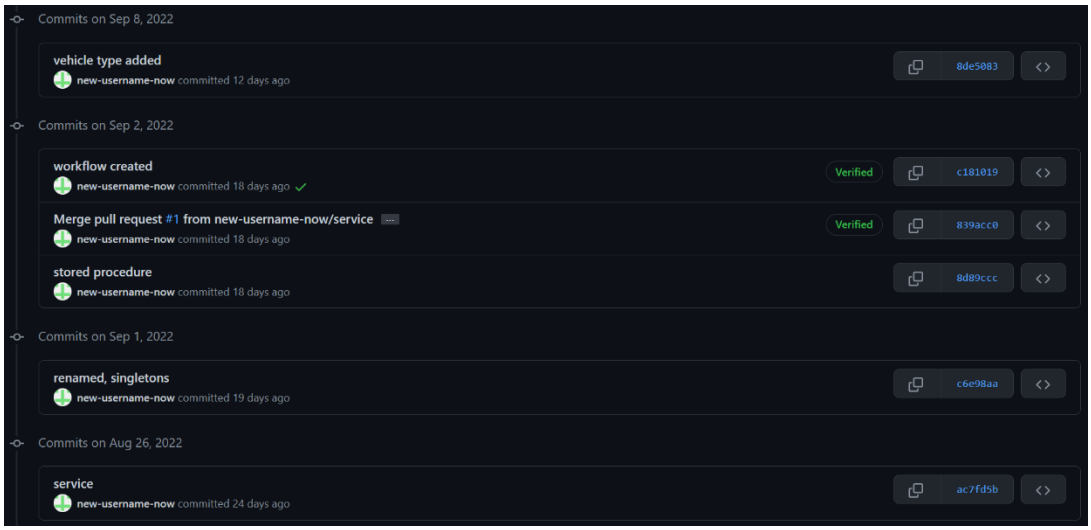
The GitHub is a highly useful platform, when it comes to software development. The different components like the service and the database were able to maintain properly with the git branch feature, once they were done, we could make pull request to main branch to keep the work neat and clean.

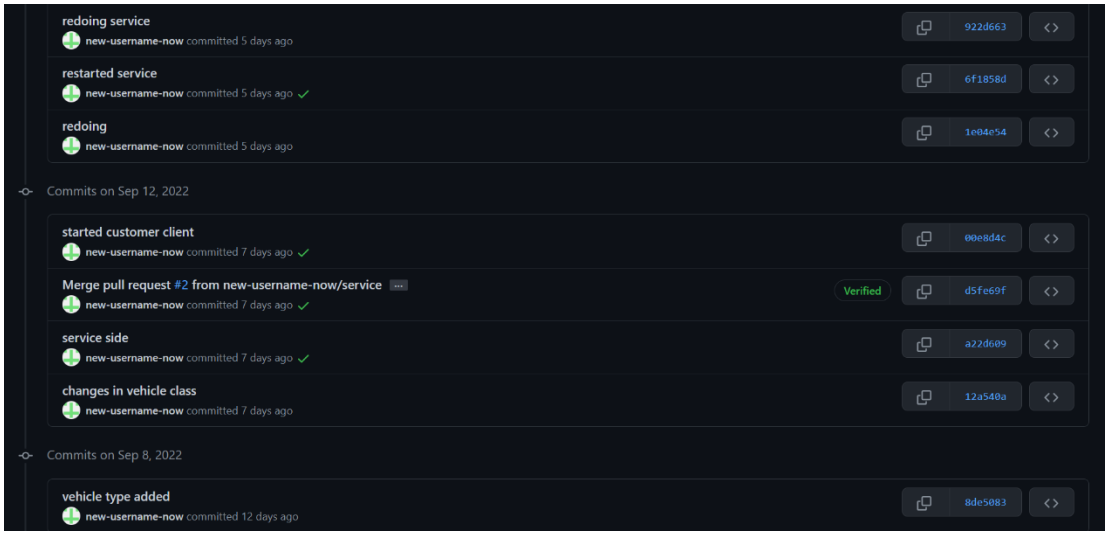
The biggest advantage of GitHub is the version control system. Where we can maintain different versions of the project as we build. It lets us compare and update versions anytime.

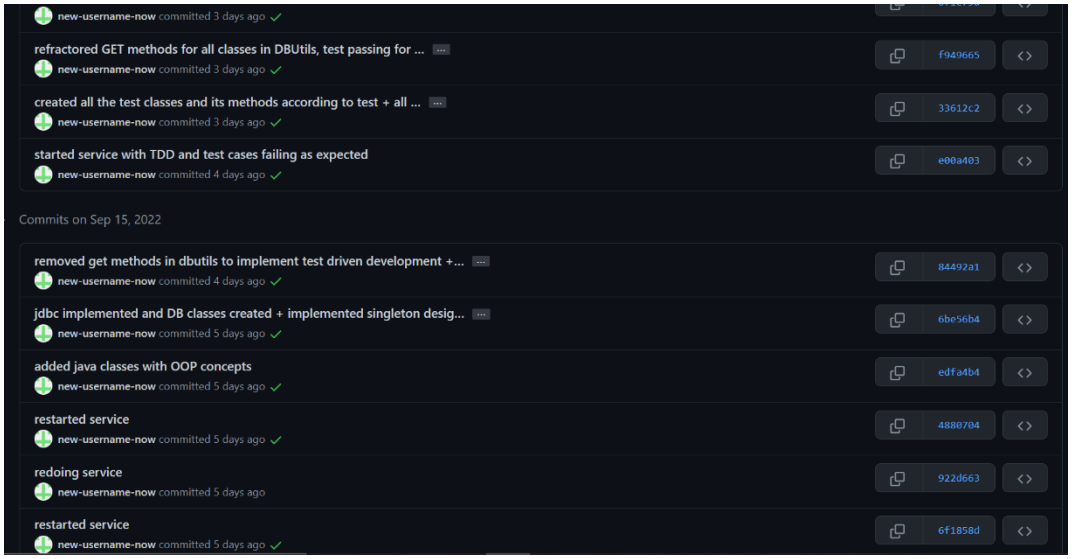
As I created the project I had created a GitHub repository for this project and uploaded. From then on as I made changes, I upload the changed version this way I maintain various different version of the project.

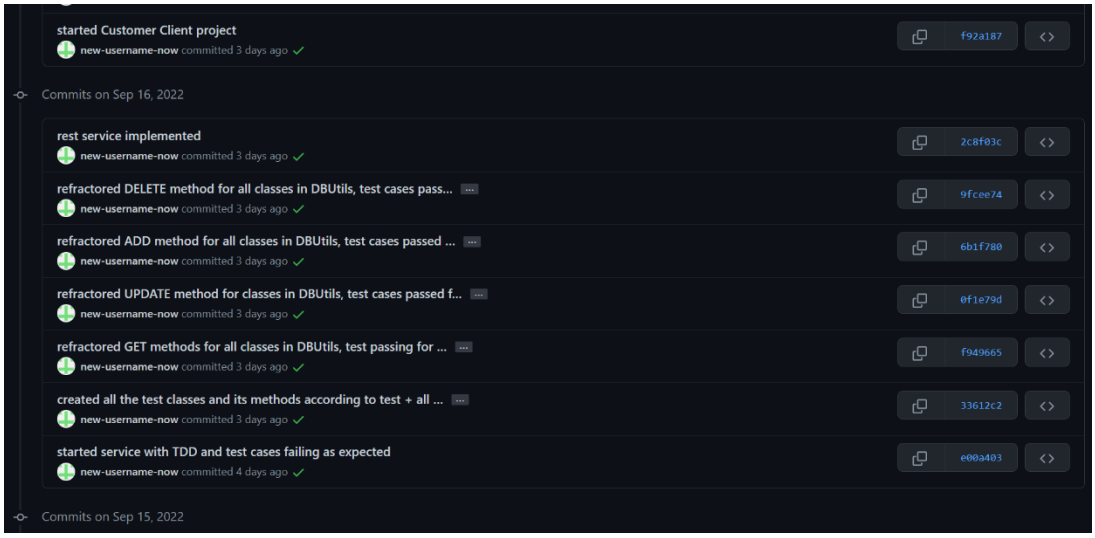
- When we use the command “git status”, we can check all the altered or added files.
- Then we can use the command “git add .” or “git add <filename>” to add the file in a our local repository.
- Then we commit the files to stage by using the command “git commit –m “<commit message>””.
- Then we can use the command “git push” to push the files to the remote repository.


Commits on Aug 26, 2022
service
new-username-now committed 24 days ago
ac7fd5b
Commits on Aug 24, 2022
service
new-username-now committed 27 days ago
a29ed42
service
new-username-now committed 27 days ago
ade7d8d
mysql file
new-username-now committed 27 days ago
de37432
Revert "Object Oriented Class"
new-username-now committed 27 days ago
9fb2ec8
Object Oriented Class
new-username-now committed 27 days ago
7a80715
Commits on Aug 22, 2022
mysql file
new-username-now committed 29 days ago
f7be6d3


Commits on Sep 8, 2022
vehicle type added
new-username-now committed 12 days ago
8de5083
Commits on Sep 2, 2022
workflow created
new-username-now committed 18 days ago
Verified
c181019
Merge pull request #1 from new-username-now/service
new-username-now committed 18 days ago
Verified
839acc0
stored procedure
new-username-now committed 18 days ago
8d89ccc
Commits on Sep 1, 2022
renamed, singletons
new-username-now committed 19 days ago
c6e98aa
Commits on Aug 26, 2022
service
new-username-now committed 24 days ago
ac7fd5b


redoing service
new-username-now committed 5 days ago
922d663
restarted service
new-username-now committed 5 days ago
6f1858d
redoing
new-username-now committed 5 days ago
1e04e54
Commits on Sep 12, 2022
started customer client
new-username-now committed 7 days ago
00e8d4c
Merge pull request #2 from new-username-now/service
new-username-now committed 7 days ago
Verified
d5fe69f
service side
new-username-now committed 7 days ago
a22d609
changes in vehicle class
new-username-now committed 7 days ago
12a540a
Commits on Sep 8, 2022
vehicle type added
new-username-now committed 12 days ago
8de5083


new-username-now committed 3 days ago
refractored GET methods for all classes in DBUtils, test passing for ...
new-username-now committed 3 days ago
f949665
created all the test classes and its methods according to test + all ...
new-username-now committed 3 days ago
33612c2
started service with TDD and test cases failing as expected
new-username-now committed 4 days ago
e00a403
Commits on Sep 15, 2022
removed get methods in dbutils to implement test driven development +...
new-username-now committed 4 days ago
84492a1
jdbc implemented and DB classes created + implemented singleton desig...
new-username-now committed 5 days ago
6be56b4
added java classes with OOP concepts
new-username-now committed 5 days ago
edfa4b4
restarted service
new-username-now committed 5 days ago
4880704
redoing service
new-username-now committed 5 days ago
922d663
restarted service
new-username-now committed 5 days ago
6f1858d


started Customer Client project
new-username-now committed 3 days ago
f92a187
Commits on Sep 16, 2022
rest service implemented
new-username-now committed 3 days ago
2c8f03c
refractored DELETE method for all classes in DBUtils, test cases pass...
new-username-now committed 3 days ago
9fcee74
refractored ADD method for all classes in DBUtils, test cases passed ...
new-username-now committed 3 days ago
6b1f780
refractored UPDATE method for classes in DBUtils, test cases passed f...
new-username-now committed 3 days ago
0f1e79d
refractored GET methods for all classes in DBUtils, test passing for ...
new-username-now committed 3 days ago
f949665
created all the test classes and its methods according to test + all ...
new-username-now committed 3 days ago
33612c2
started service with TDD and test cases failing as expected
new-username-now committed 4 days ago
e00a403
Commits on Sep 15, 2022


main
Commits on Sep 20, 2022
links to UML Diagrams
new-username-now committed 15 minutes ago
d671720
made changes to admin client, links to all the diagrams
new-username-now committed 16 minutes ago
fededf8
Commits on Sep 19, 2022
started Admin Client project
new-username-now committed 13 hours ago
c8ec270
Commits on Sep 17, 2022
started Customer Client project
new-username-now committed 3 days ago
4a8b33a
started Customer Client project
new-username-now committed 3 days ago
f92a187
Commits on Sep 16, 2022

Assumptions

- The customer/driver/admin can view and update their profile.
- Since admins can add drivers, the driver cannot register themselves, therefore they need to contact admin by sending their details to register.
- The driver can view past ride/activity with all the ride details.
- The driver can accept or decline a ride.
- The driver can end the ride once it is done.
- The admin can view/add/update/delete city.
- The admin can view/add/update/delete street.
- The admin can view customer bookings in each branch.

References

Aljasser, K., 2016. Implementing design patterns as parametric aspects using ParaAJ: the case of the singleton, observer, and decorator design patterns. *Computer Languages, Systems & Structures*, *45*, pp.1-15.

Berardi, D., Calvanese, D. and De Giacomo, G., 2005. Reasoning on UML class diagrams. *Artificial intelligence*, *168*(1-2), pp.70-118.

Eskca, E.B. and Bondugula, E., 2014. Simplifying the abstract factory and factory design patterns.

George, B. and Williams, L., 2004. A structured experiment of test-driven development. *Information and software Technology*, *46*(5), pp.337-342.

Glinz, M., 2007, October. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)* (pp. 21-26). IEEE.

Malan, R. and Bredemeyer, D., 2001. Functional requirements and use cases. *Bredemeyer Consulting*.

Reggio, G., Leotta, M., Ricca, F. and Clerissi, D., 2013, October. What are the used UML diagrams? A Preliminary Survey. In *EESSMod@ MoDELS* (pp. 3-12).

Sharp, R. and Rountev, A., 2005, September. Interactive exploration of UML sequence diagrams. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis* (pp. 1-6). IEEE.

Stencel, K. and Węgrzynowicz, P., 2008, November. Implementation variants of the singleton design pattern. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 396-406). Springer, Berlin, Heidelberg.

Wegmann, A. and Genilloud, G., 2000, October. The role of “Roles” in use case diagrams. In *International Conference on the Unified Modeling Language* (pp. 210-224). Springer, Berlin, Heidelberg.