CIS020-1 – Introduction to Software Development

Assignment 2 –Individual Project – Case Study (Taxi Booking System)

University ID : 2147440 **Full Name : Santosh Tamang**

CIS020-1 – Introduction to Software Development

Assignment 2 –Individual Project – Case Study (Taxi Booking System)

University ID : 2147440 **Full Name : Santosh Tamang**

Table of Contents

CIS020-1 – Introduction to Software Development - 2021-2022

Assignment 2 – Individual Project – Case Study (Taxi Booking System)

Introduction/Overview

The S&S Taxi Service has been providing taxi services for a long time. They now want to provide the booking facilities through an online medium and a desktop application was requested to be created.

Currently, all the details of their customer, driver and staff are registered in hard copy format along with the trips they have booked. The customer must come to the office, send an e-mail or call the office to book a trip, or to cancel the current trip. Searching for data takes a long time as it was hard to search through all the data. It was hard for the customer to just book or cancel a trip.

For the easiness of the customer to book, cancel and view trips and the staff to search and fill data by converting data into softcopy version. A desktop application is purposed to solve all the existing problem and creating user-friendly environment. The application allows different users (i.e. customer, staff and drive) to perform different activity. The application allows customer to login, book a trip, update the trip, view the trip and cancel the trip. The application allows staff to login, confirm a trip booked by the customer and assign a suitable driver. The application allows driver to login and view their upcoming trips.

To fulfil the requirement of the given assignment, a desktop application was created to book, view, update and cancel trips via online medium, with database to store all data. PostgreSQL database was used to create the required RDBMS with all the required queries and data dictionary.

Task Description

CIS020-1 – Introduction to Software Development

Assignment 2 –Individual Project – Case Study (Taxi Booking System)

University ID : 2147440 **Full Name : Santosh Tamang**

The task requires the creation of a desktop application with a functioning database for a taxi company that is attempting to offer its services online. There are only three users i.e., customer, driver and staff.

The customer should be able to book a trip, i.e., provide pickup location and time as well as the number of passengers, destination location and payment method. They should be able to view their bookings and booking can be cancelled if needed. To be able to do the above-mentioned activity customer must log in first. If not registered, the customer must register providing their name, address, email and telephone number.

The driver should be able to view all the upcoming trips after logging in. They should be able to start and complete the trip.

The trip booking is confirmed and drivers are assigned by the taxi company administrative after logging in. The staff can see all trips booked.

Include the scenario, and Use Case Diagram from the Brief.

Include any assumptions you make about the system.

Assumption

- Company have their own vehicle so the driver are assigned a vehicle when they are registered.
- Drivers are registered by the company due to the security of the passenger issues.
- Customers can only book a trip as late as 15 days and as soon as tomorrow.

Project Plan/Schedule

Week No.	Tasks	Priority
1	Research in the related field	MUST

2	Requirement Analysis and GUI Prototype Design	MUST
3	Database Design	MUST
4	GUI Implementation	MUST
5	Database Implementation and Testing the Application	MUST
6	Finalizing Product	MUST
7	Optimizing Product	SHOULD
8	Submit Group Report, Project Code and Video Recording	MUST
9	Project Presentation	MUST

Overview of Functional, Technical (Non-Functional Requirements) and Usability Requirements

Functional Requirements

TBS = Taxi Booking System

Customer

1. A customer must register.
2. A customer must log in.
3. A customer should be able to select the pickup locations and the destination.
4. A customer should be able to view their trips.
5. A customer can cancel the booking.
6. A customer should be able to make multiple bookings using the same id.

Staff

1. A staff must login.

2. A staff should be able to view all the customers, drivers, vehicles and drivers.
3. A staff should be able assign a driver to a booking.
4. A staff should be able view driver schedule.
5. A staff should be able to see the status of a trip.

Driver

1. A driver must login.
2. Drivers should be able to view their upcoming trip.
3. They should be able to start a trip and end them.

Non-functional Requirements

1. The system should respond as fast as possible.
2. The system must be platform independent.
3. The system must be online 24/7.
4. Future amendments must be allowed in the system.

Usability Requirements

1. The system should be user-friendly.
2. There should be validation to enter data and the forms should be short and easy to fill.
3. Enough information about the booking should be provided to meet users' needs.
4. The navigation in the application should be easy the user should not need a guide to do the basics.

Design:

UML Diagram

Uses Case Diagrams

(TechTarget Contributor, 2023) "A use case diagram is a way to summarize details of a system and the users within that system."

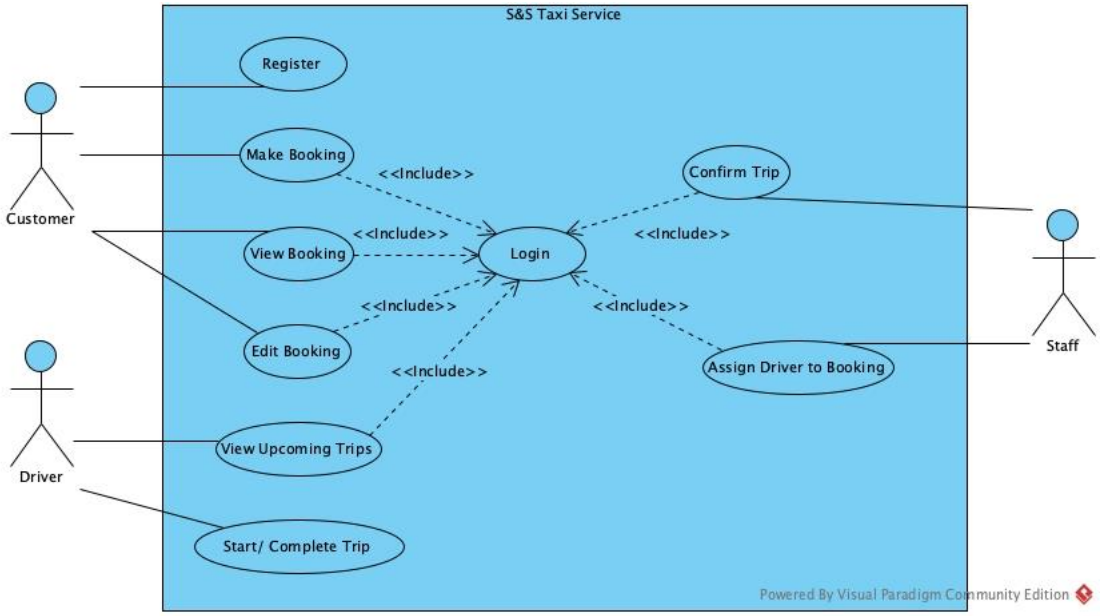


Fig.no.1.Sea Level Use Case

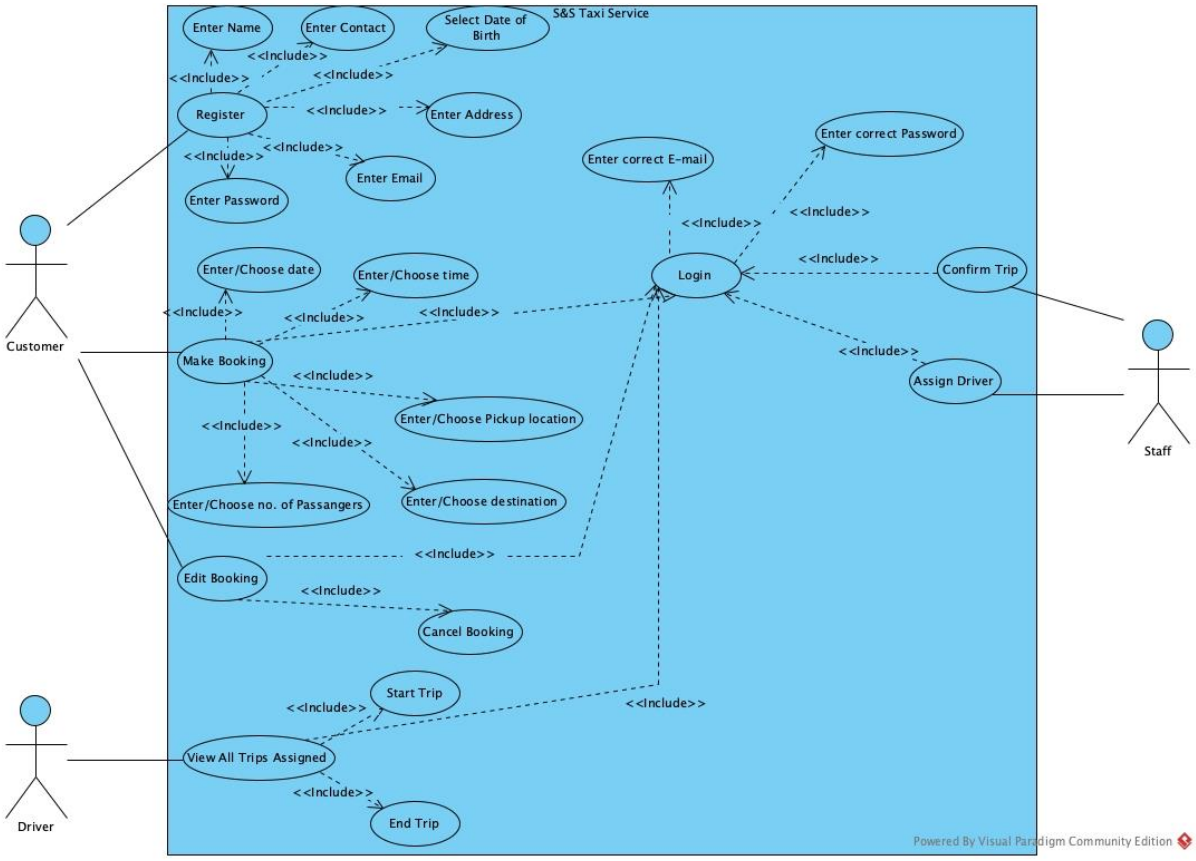


Fig.no.2.Fish Level Use Case

The use case shows how the software works on the surface level.

There are 3 types of users, i.e. customer, driver and staff. They all must login before performing any activities in the software.

To login customer must register with valid information. The registered customer can directly login and view the trips booked or book a trip then. They can even cancel or delete the trip.

A staff confirms a trip by assigning a driver, the driver must be empty, to the trip.

A driver can check for their upcoming trips, start a trip and end it. To start a trip, the date must of the same day or else they cannot.

Use Case Specifications / Description

Allocate Taxi

Only one taxi can be assigned to a single driver.

Make a Booking

After customer made a trip booking, the staff will assign a driver and make the booking confirmed.

Cancel Booking

A customer may cancel a trip any time before a diver is assigned to the trip.

Pickup Date

On the pickup date the driver will wait in the location entered by the Customer. They will wait for 5 extra minutes and call the customer for conformation. If they are not going to make the trip they will be fined

.

Assign Driver

After a driver is requested for a trip, the staff can see the request and assign one of the available driver to the trip.

Activity Diagram

(Tutorialspoint, 2022) "Activity diagram is basically a flowchart to represent the flow from one activity to another activity."

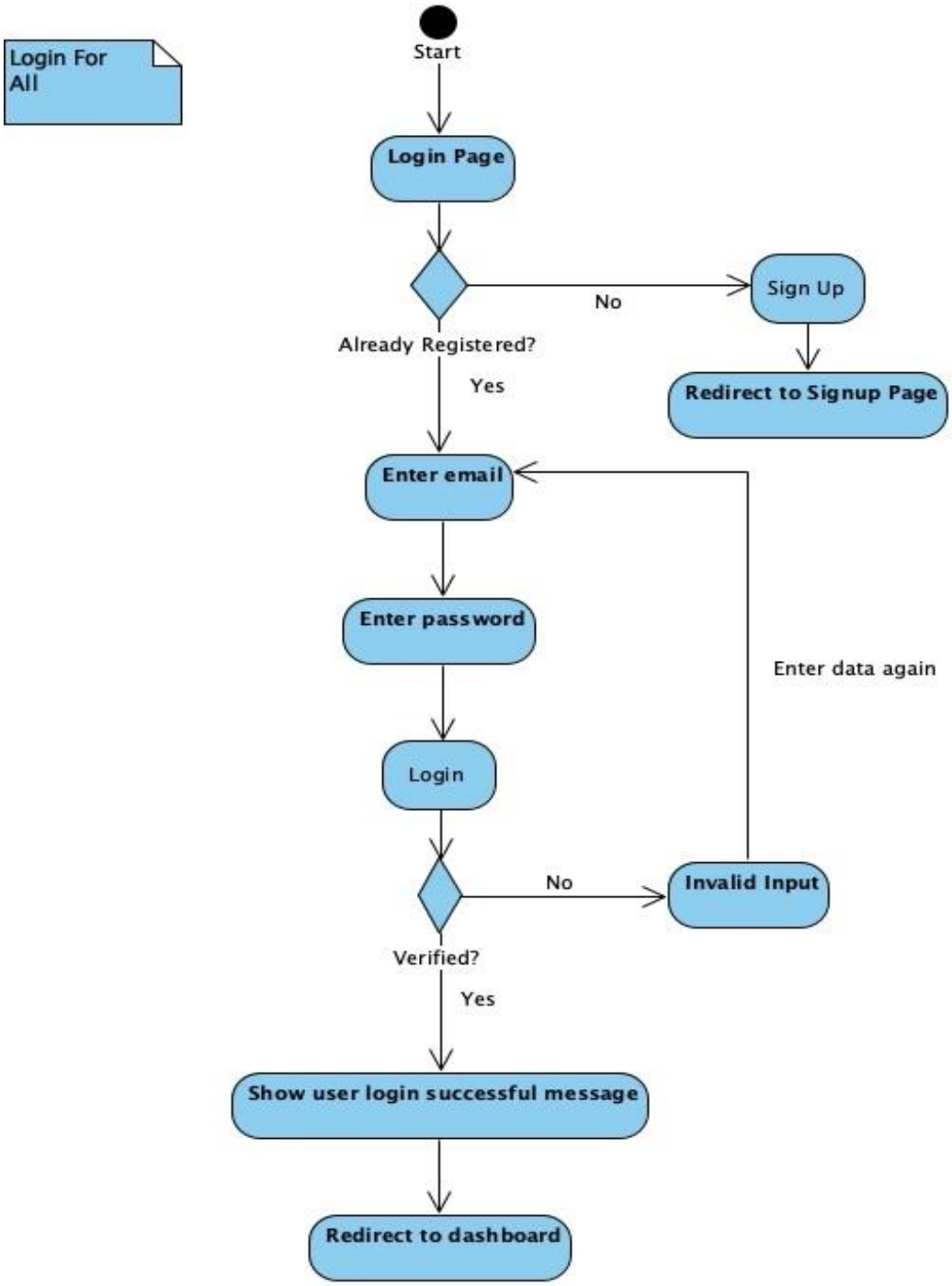


Fig.no.3. Activity Diagram For Logging

The given figure shows the flowchart of the logging process. After opening the software user are provided with two options, to login or register. A user having an account can directly login using this flowchart. For users trying to register pressing the signup button they are redirected to registration page. For the users having account, they are redirected to the dashboard.

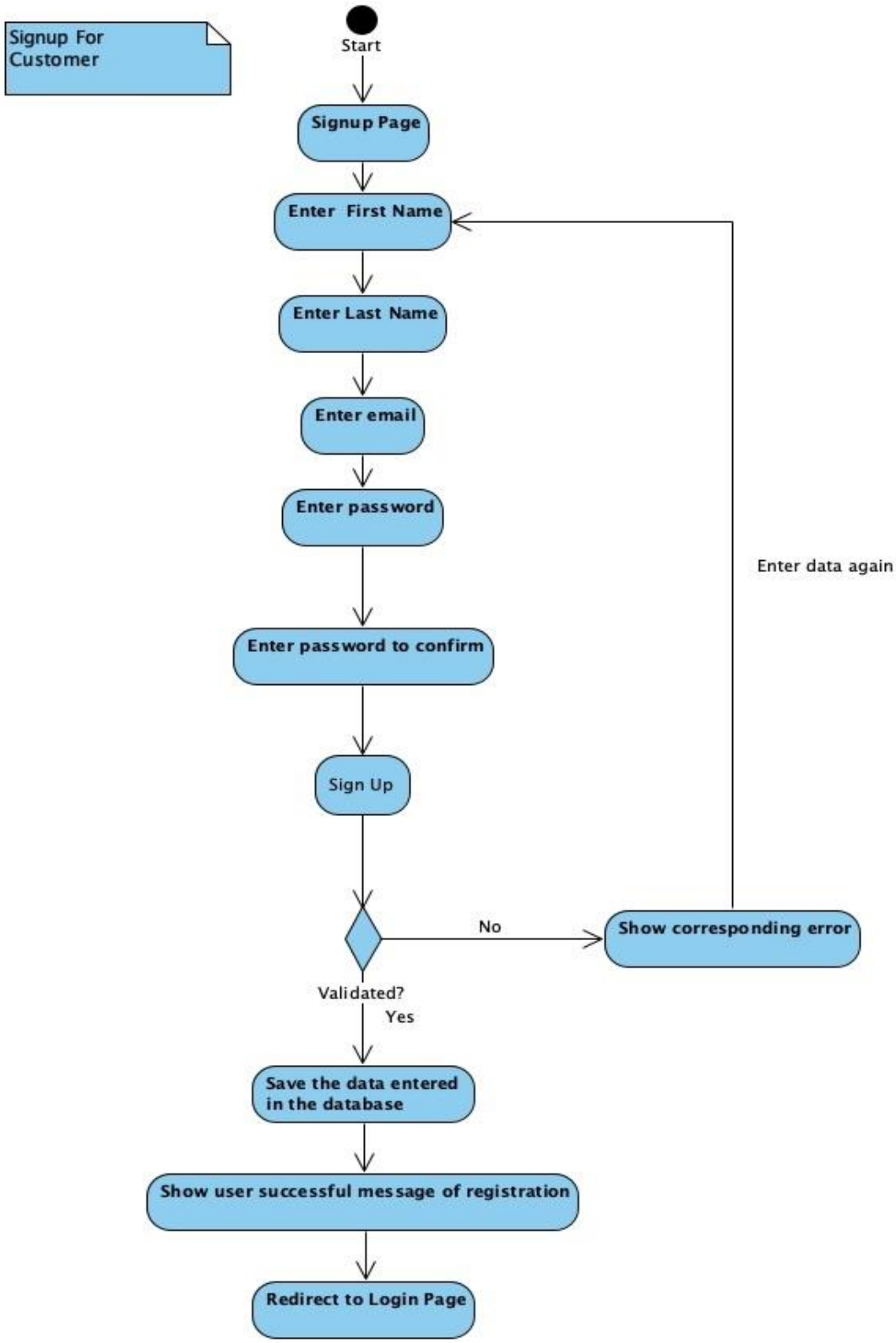


Fig.no.4. Activity Diagram For Registration

After reaching the registration page user can simply fill the form with valid data, if any data is considered incorrect user have to refill the form. After the success in the creation of the account users are redirected to login where they are supposed to login using the data they had just entered.

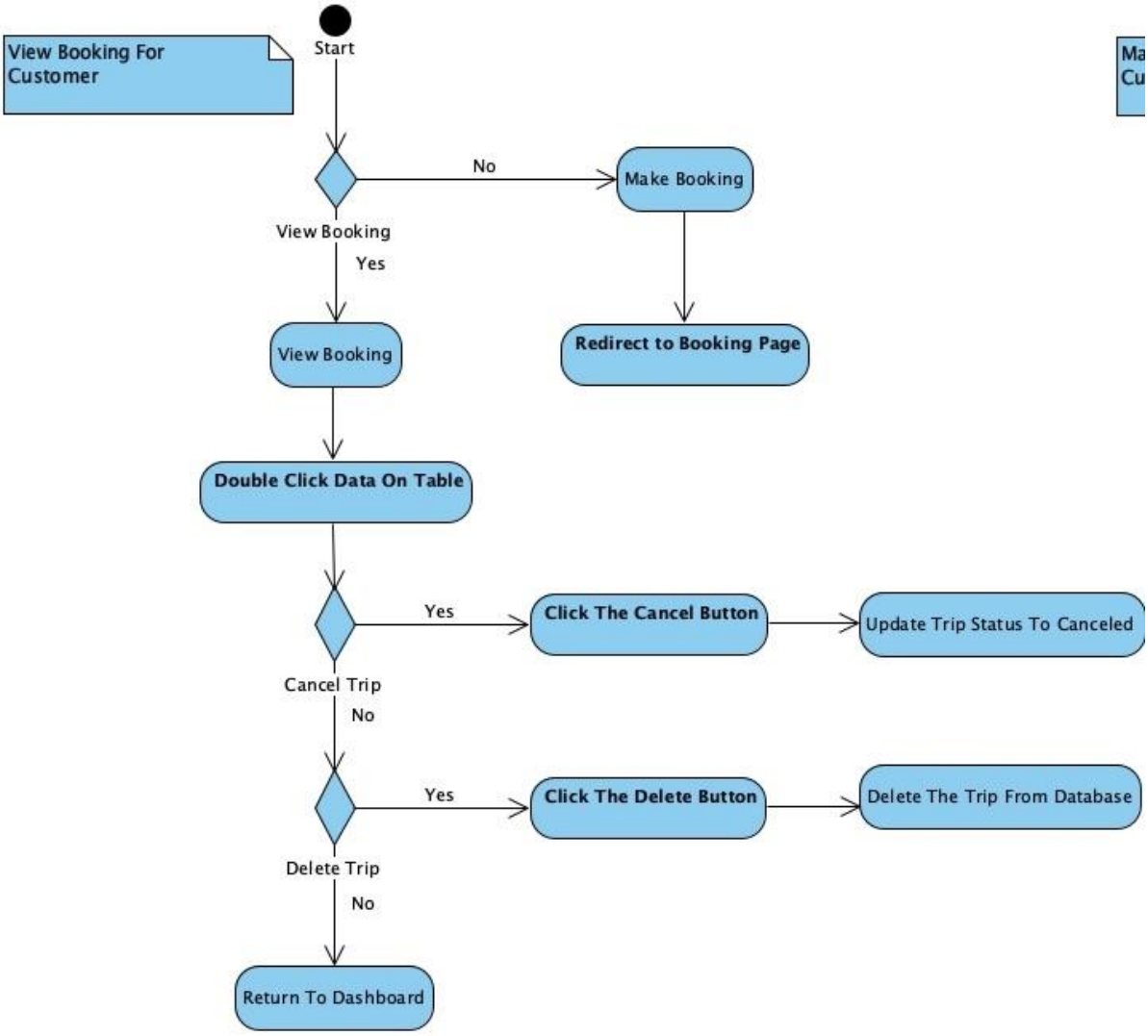


Fig.no.5. Activity Diagram To View Booked Trips

Users that are redirected to the dashboard can see that they are provided with two options: view the trip details and book a trip.

For the users trying to book a trip, they can press the book now button and they are redirected to the booking page.

For the users trying to see the trip details, they can choose one trip and double click on the trip. Then, they are provides two options: cancel the trip, delete the trip. The customer can simply cancel the trip by clicking the cancel button if the cancellation is an option. Similarly, customer can delete the trip by clicking the delete button if the deletion is allowed. Cancellation and deletion is allowed until trip is confirmed.

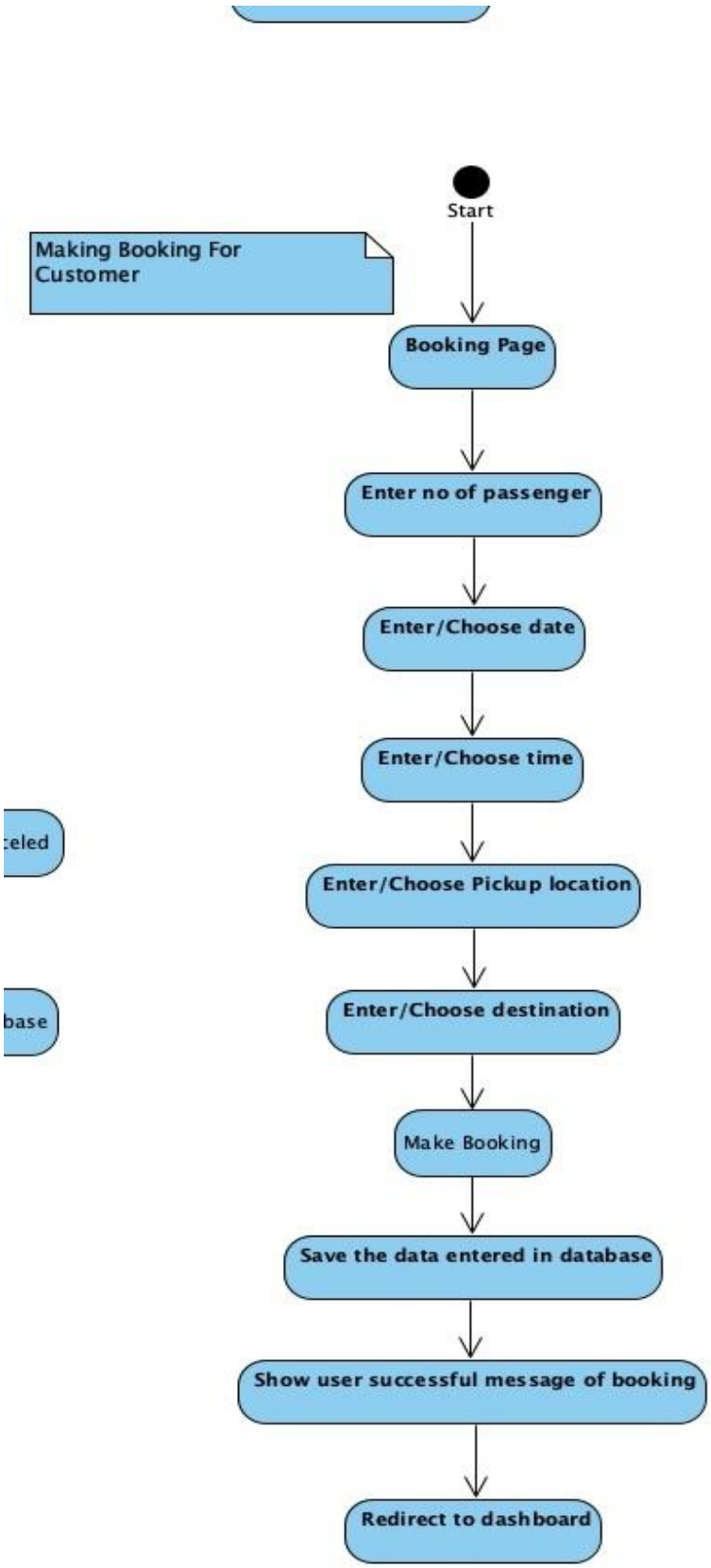


Fig.no.6. Activity Diagram For Booking A Trip

The users redirected to booking page can simply enter or choose the pickup location and destination. The can choose the number of passenger that are travelling. They can choose the pickup time. Having entered the required data, users can simply click the book button and a driver for the trip is requested.

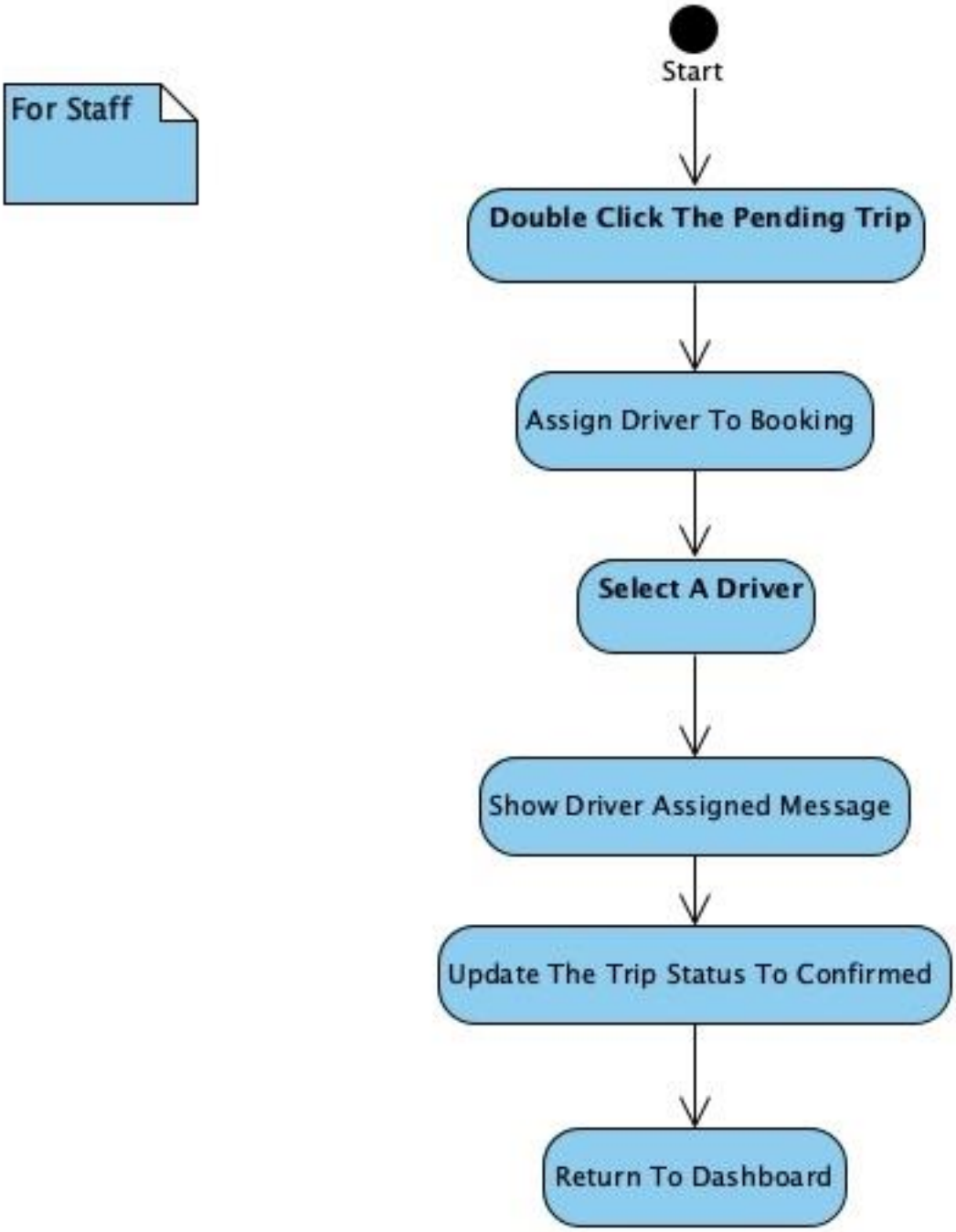


Fig.no.7. Activity Diagram For Staff

This flowchart shows the order of activity of the staff after logging. Staff double clicks the trip that is to be assigned a driver. They select one of the available diver and make the trip confirmed.

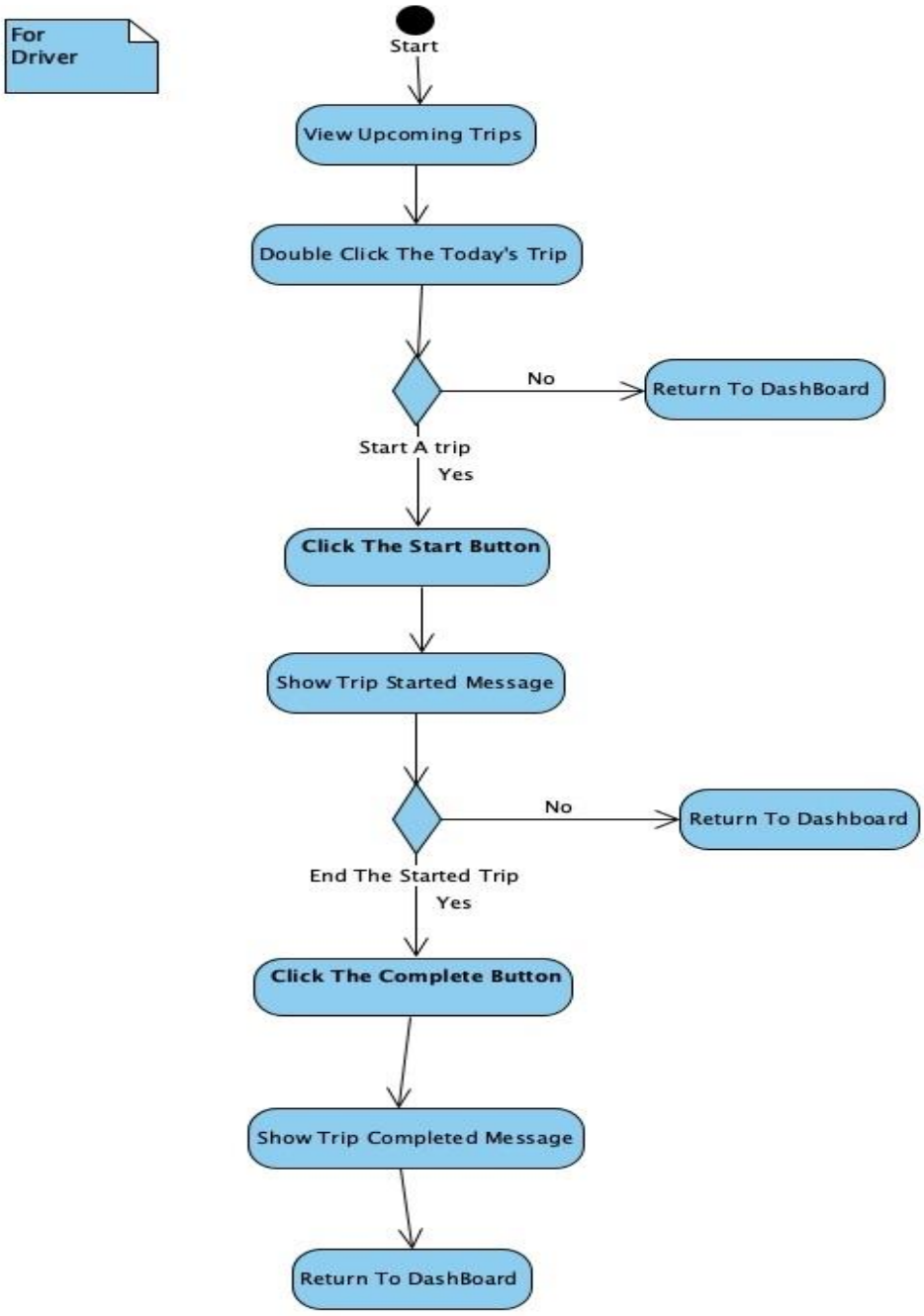


Fig.no.8. Activity Diagram Of Driver

Driver are redirected to driver dashboard after logging. The see their upcoming trips. To start and end the trip they must double click the trip to be started or ended.

Class Diagram

(Tutorialspoint, 2011) "The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them."

Fig.no.9. Class Diagram

This class diagram describes the system's classes and its attributes along with the relation with other classes. The class diagram represents almost all the aspect of the software.

Database Design

Logical Database Design

Entity Relationship Model(ERM)

ERM Diagram

(Visual Paradigm, 2022) "Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design."

In the ERM, the structure of the database is portrayed in the entity relationship. This helps to develop the conceptual design of the database.

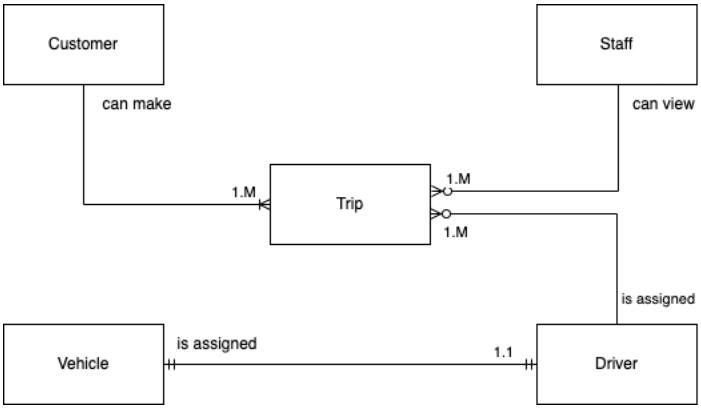


Fig.no.10. Logical Database

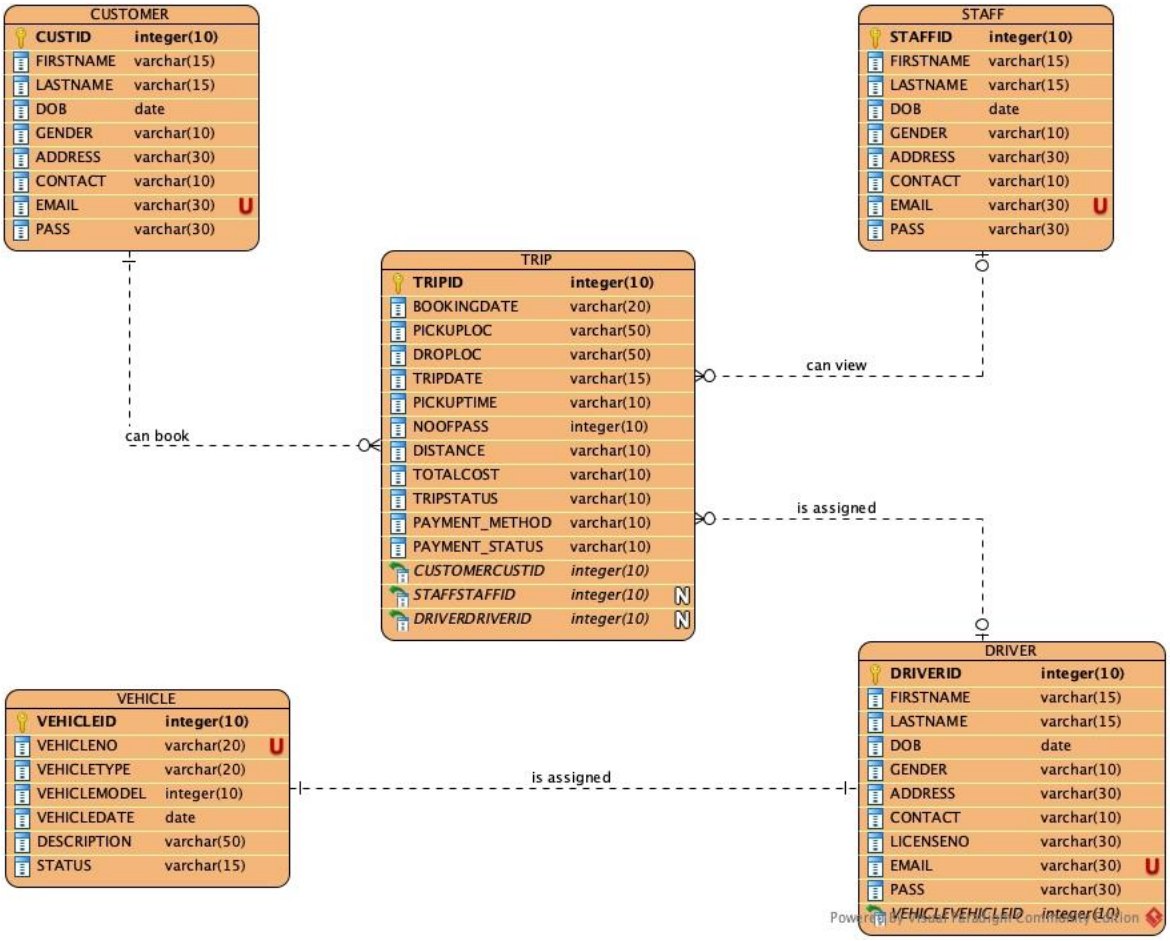


Fig.no.11. Entity Relationship Diagram

The above figure is the ERM diagram of the database creates to satisfy the need of this assignment. For this assignment five entities are identified by the normalization process. The entities are Customer, Staff, Trip, Driver and Vehicle. Each entities contains primary key to maintain data redundancy.

A customer can make many trips at a time.

A staff can view all the trips booked by the customer and assigns available drivers.

Driver can see all trips assigned to them.

A vehicle is assigned to a single driver only.

- List Of Entities

Customer (custid, firstname, lastname, gender, dob, address, contact, email, pass)

Vehicle(vehicleid, vehicleno, vehicletype, vehiclemodel, vehicledate, description, status)

Driver (driverid, firstname, lastname, gender, dob, address, contact, licenseno, vehicleis*, email, pass)

Staff(staffed, firstname, lastname, gender, dob, address, contact, email, pass)

Booking (tripid, bookingdate, pickuploc, droploc, tripdate, pickuptime, noofpass, distance, totalcost, tripstatus, payment_method, payment_status, custid*, driverid*, staffed*)

Physical Database Design

- ### Skeleton Table

Customer (custid, firstname, lastname, gender, dob, address, contact, email, pass)

Vehicle(vehicleid, vehicleno, vehicletype, vehiclemodel, vehicledate, description, status)

Driver (driverid, firstname, lastname, gender, dob, address, contact, licenseno, vehicleis*, email, pass)

Staff(staffed, firstname, lastname, gender, dob, address, contact, email, pass)

Booking (tripid, bookingdate, pickuploc, droploc, tripdate, pickuptime, noofpass, distance, totalcost, tripstatus, payment_method, payment_status, custid*, driverid*, staffed*)

- ### Data Dictionary

(Varma, 2022) “A data dictionary in Database Management System (DBMS) can be defined as a component that stores the collection of names, definitions, and attributes for data elements that are being used in a database.”

Customer							
Description: Customer details							
Field Name	**Datatype**	**Length**	**Index**	**Null**	**Default**	**Validation rule**	**Description**
CustID *(Primary)*	int (10) unsigned	10	PK	No			Autoincremented Uniquely identifies every customer
Firstname	varchar (15)	15		No		First letter must be capital, no space allowed	First name of customer
Lastname	varchar (15)	15		No		First letter must be capital, no space allowed	Last name of customer
Dob	date	10		No			Birthdate of customer
Contact	varchar (10)	10		No			Customer contact number
Address	varchar (30)	30		No			Address of customer
Gender	varchar (10)	10		No			Gender of customer
Email	varchar (30)	30		No		Must be email format containing an @ and a '.' Regex	Email of customer

				expression used	
Password	varchar (30)	30	No	Must contain 8 letters, a small, a capital letter, symbol and a number	TBS password

Indexes

Key name	Type	Unique	Column	Null
PRIMARY	BTREE	Yes	CustID	No

Staff

Description: Staff details

Field Name	Datatype	Length	Index	Null	Default	Validation rule	Description
StaffID *(Primary)*	int (10) unsigned	10	PK	No			Autoincremented Uniquely identifies every staff
Firstname	varchar (15)	15		No		First letter must be capital, no space allowed	First name of staff

Lastname	varchar (15)	15	No	First letter must be capital, no space allowed	Last name of staff
Dob	date	10	No		Birthdate of staff
Contact	varchar (10)	10	No		Staff contact number
Address	varchar (30)	30	No		Address of staff
Gender	varchar (10)	10	No		Gender of staff
Email	varchar (30)	30	No	Must be email format containing an @ and a '.' Regex expression used	Email of staff
Password	varchar (30)	30	No	Must contain 8 letters, a small, a capital letter, symbol and a number	TBS password

Indexes

Key name	Type	Unique	Column	Null
PRIMARY	BTREE	Yes	StaffID	No

Vehicle

Description: Vehicle details

Field Name	Datatype	Length	Index	Null	Default	Validation rule	Description
VehicleID ***(Primary)***	int (10) unsigned	10	PK	No			Autoincremented Uniquely identifies every vehicle
VehicleNo	varchar (15)	15		No		Must in format (Ba 23 Ja 1324)	Registered vehicle number
VehicleType	varchar (15)	15		No			Vehicle company name
VehicleModel	date	10		No			Vehicle model
VehicleDate	varchar (10)	10		No			Date vehicle registered to company
Description	varchar (30)	30		No			Description of the vehicle
Status	varchar (10)	10		No			Status of the vehicle

Indexes

Key name	Type	Unique	Column	Null
PRIMARY	BTREE	Yes	VehicleID	No

Driver							
Description: Driver details							
Field Name	**Datatype**	**Length**	**Index**	**Null**	**Defaul t**	**Validation rule**	**Description**
DriverId *(Primary)*	int (10) unsigned	10	PK	No			Autoincremented Uniquely identifies every driver
Firstname	varchar (15)	15		No		First letter must be capital, no space allowed	First name of driver
Lastname	varchar (15)	15		No		First letter must be capital, no space allowed	Last name of driver
Dob	date	10		No			Birthdate of driver
Contact	varchar (10)	10		No			Driver contact number
Address	varchar (30)	30		No			Address of driver
Gender	varchar (10)	10		No			Gender of driver
LicenseNo	varchar (30)	30		No			Registered license number of driver
Email	varchar	30		No		Must be	Email of driver

	(30)				email format containing an @ and a '.' Regex expression used	
Password	varchar (30)	30		No	Must contain 8 letters, a small, a capital letter, symbol and a number	TBS password
VehicleID	Int (10) unsigned	10	FK	No		Vehicle assigned to the driver

Indexes

Key name	**Type**	**Unique**	**Column**	**Null**
PRIMARY	BTREE	Yes	DriverID	No
FOREIGN	BTREE	Yes	VehicleID	No

BOOKING							
Description: Booking details							
Field Name	**Datatype**	**Length**	**Index**	**Null**	**Default**	**Validation rule**	**Description**
TripID	Int (10)	10	PK	No			Autoincremented Uniquely identifies

							every trip booked
BookingDate	varchar (20)	20		No			Date the trip is booked
Pickuploc	varchar (50)	50		No			Pickup address
Droploc	varchar (50)	50		No			Destination
TripDate	varchar (15)	15		No			The trip date
PickupTime	varchar (10)	10		No			Pickup time
NoofPass	int (10)	10		No		Cannot be more than 4	Number of passenger travelling
Distance	varchar (10)	10		No			Distance to cover for the trip
TotalCost	varchar (10)	10		No			Cost for the trip
TripStatus	varchar (10)	10		No			Status of the trip booked
Payment_Method	varchar (10)	10		No	Cash		Payment method for the trip
Payment_Status	varchar (10)	10		No			Payment status for the trip
CustID	int (10)	10	FK	No			Uniquely identifies every customers
StaffID	int (10)	10	FK	Yes			Uniquely identifies admin
DriverID	int (10)	10	FK	Yes			Uniquely identifies every driver

Indexes

Key name	Type	Unique	Column	Null
PRIMARY	BTREE	Yes	TripID	No
FOREIGN	BTREE	Yes	CustID	No
FOREIGN	BTREE	Yes	StaffID	Yes
FOREIGN	BTREE	Yes	DriverID	Yes

User Interface Design

(Tutorialspoint, 2022) "User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface."

(Interaction Design Foundation, 2022) "User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable."

For the fulfilment of the requirement a prototype was created using Balsamiq Wireframes. Designing a prototype was smooth using the Balsamiq Wireframes. The created prototype was converted to actual user interface, with the addition of some features. The screenshots of prototype created are given below.

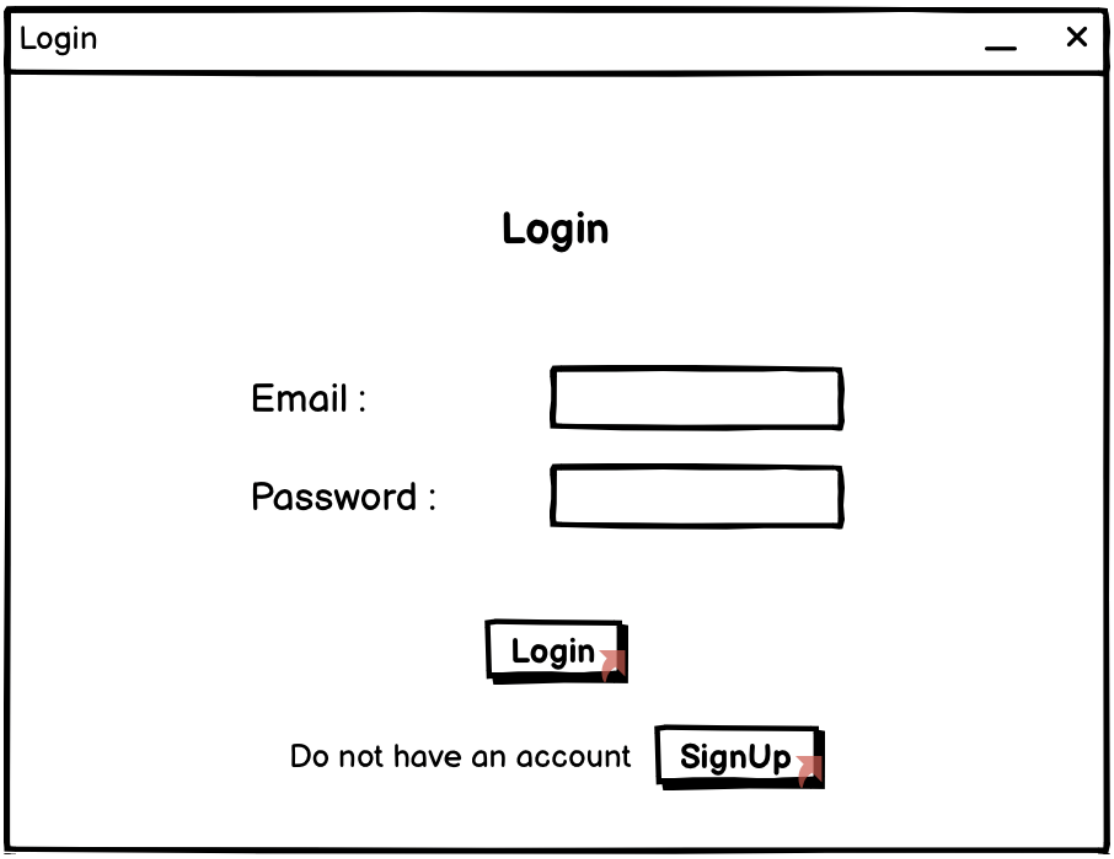


Fig.no.12 Login of Prototype

This is the login page created in the prototype phase.

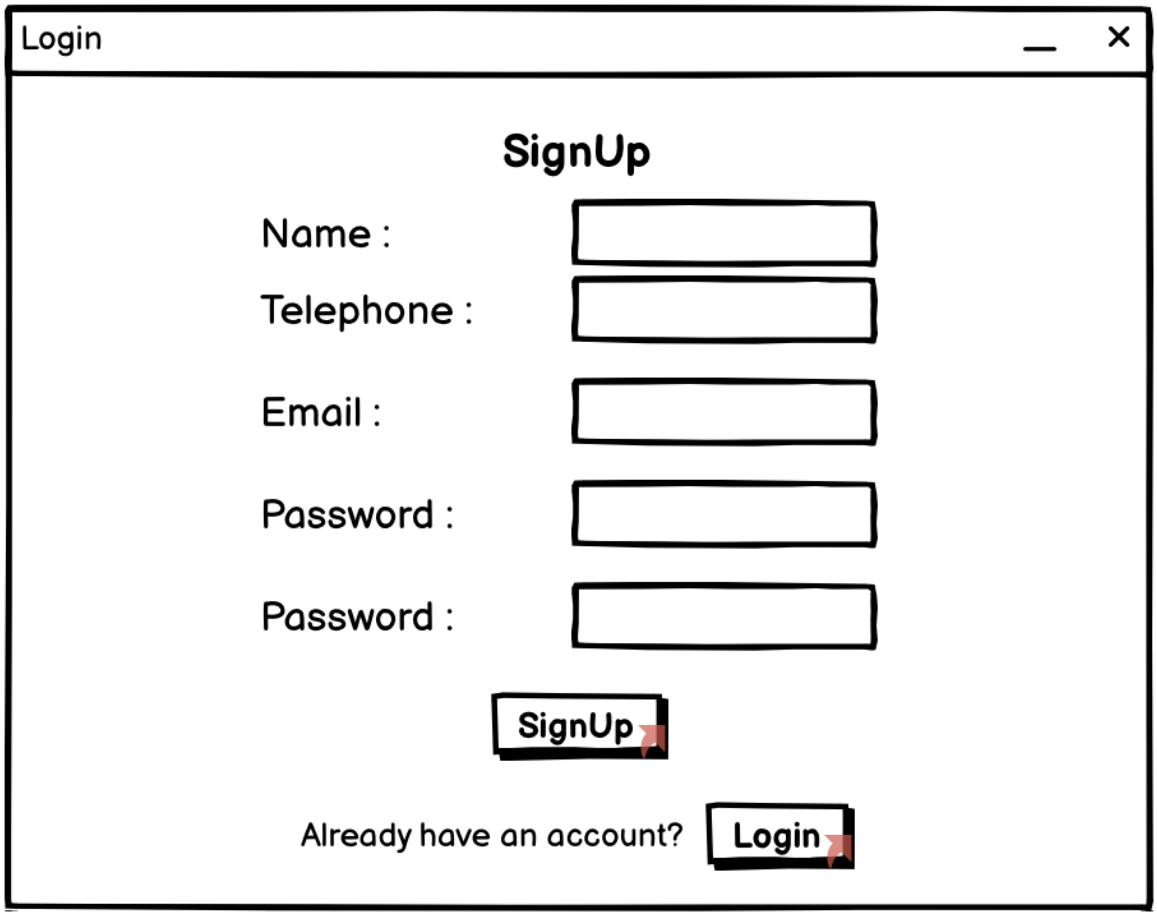


Fig.no.13 Registration of Prototype

This is the registration page where customers can register to login.

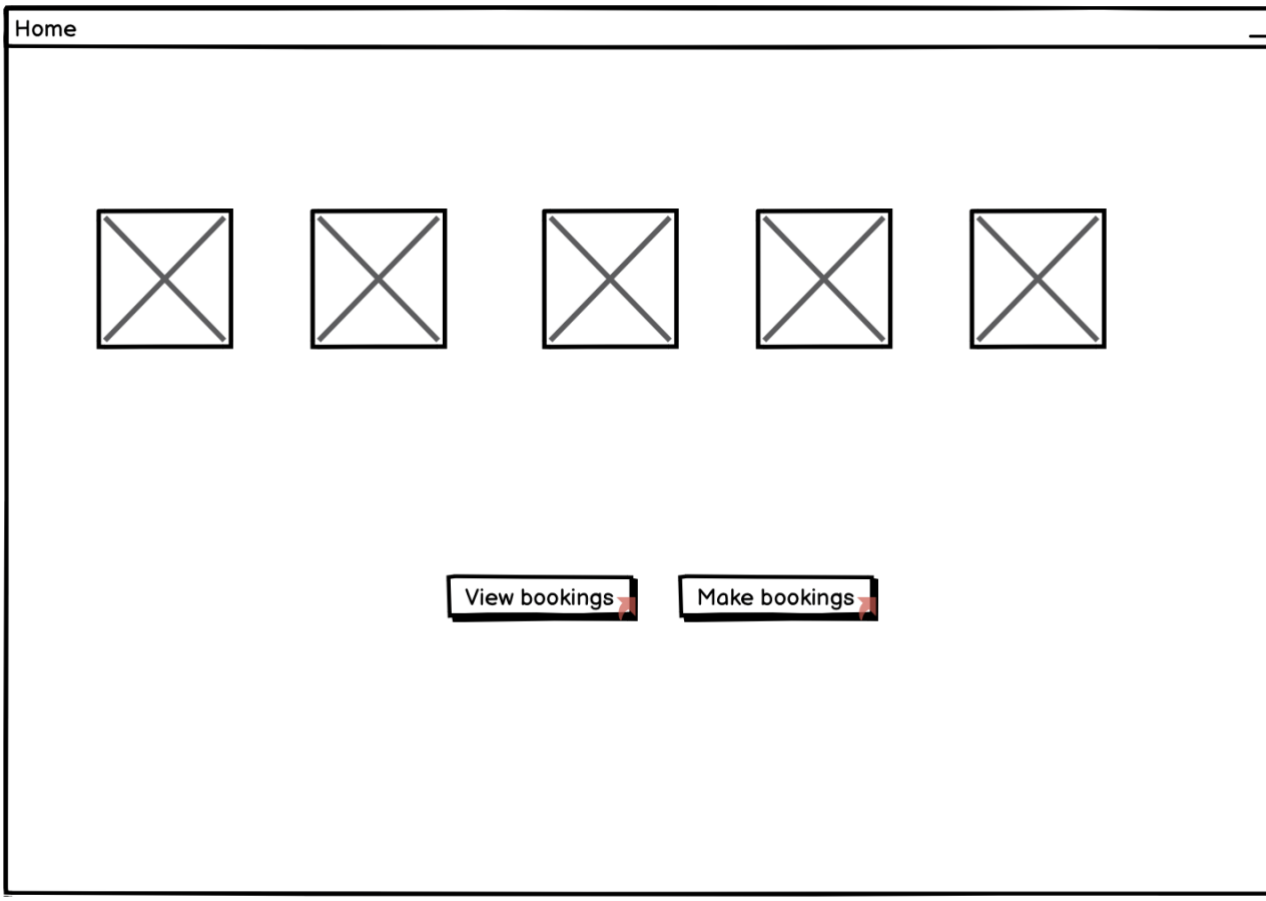


Fig.no.14 Customer Dashboard of Prototype

This is the home page of the customer after logging in.

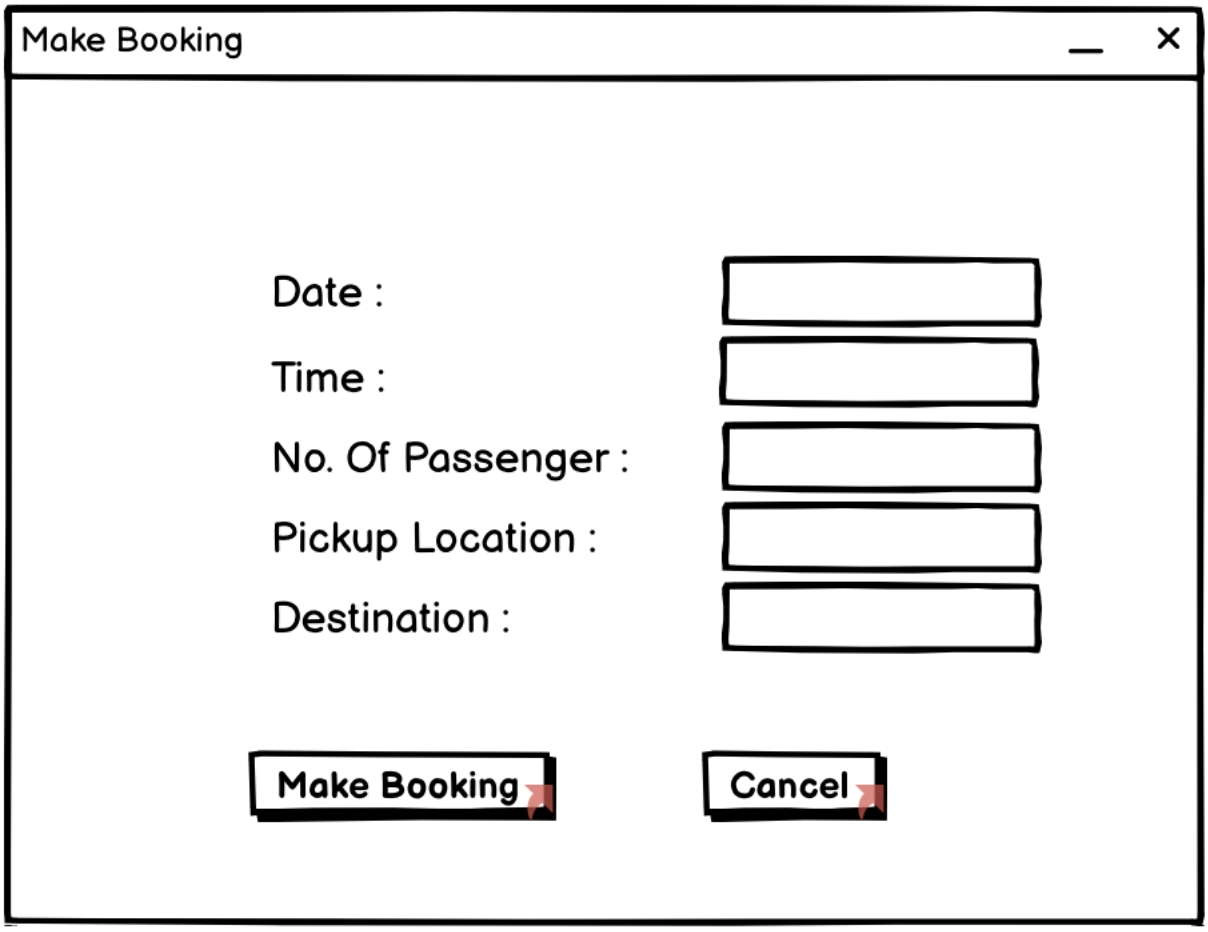


Fig.no.15 Booking Page

This is the booking page where customer can book trips.

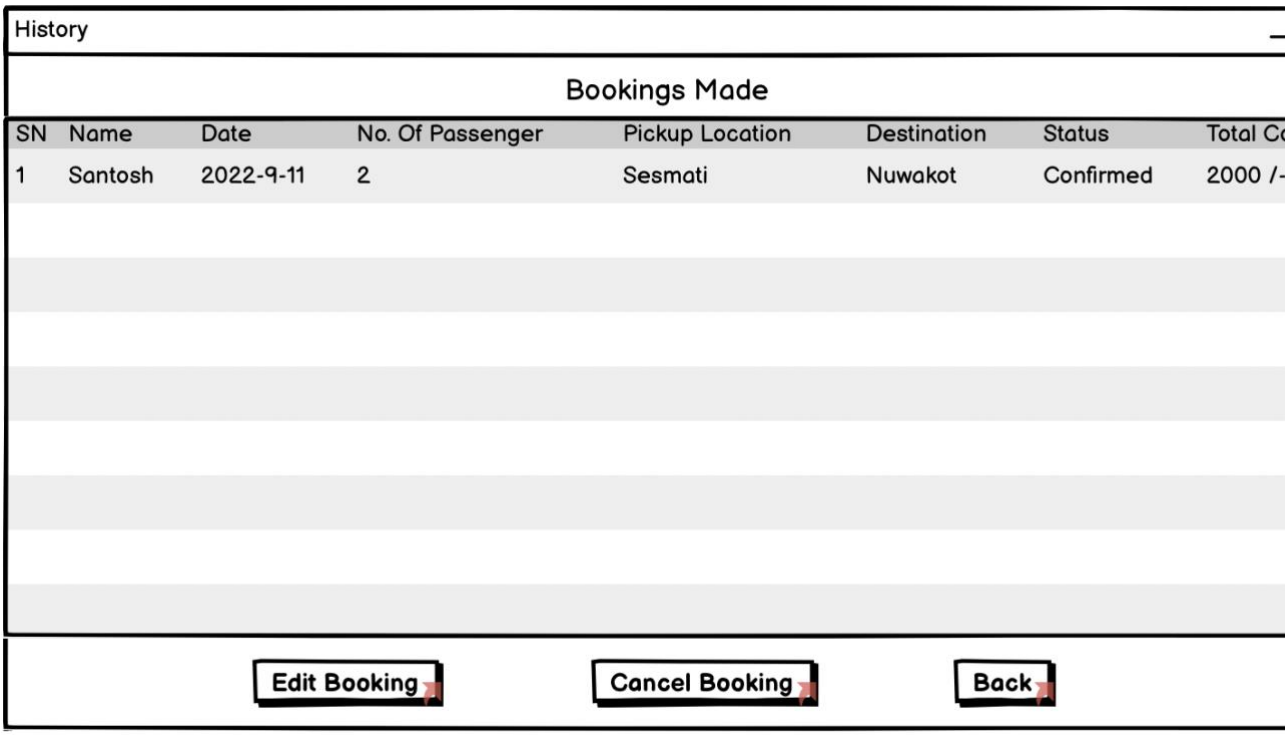


Fig.no.16. History Page of Prototype

This is the history page of customer where they can view all their trip. They can edit the trip if they wanted.

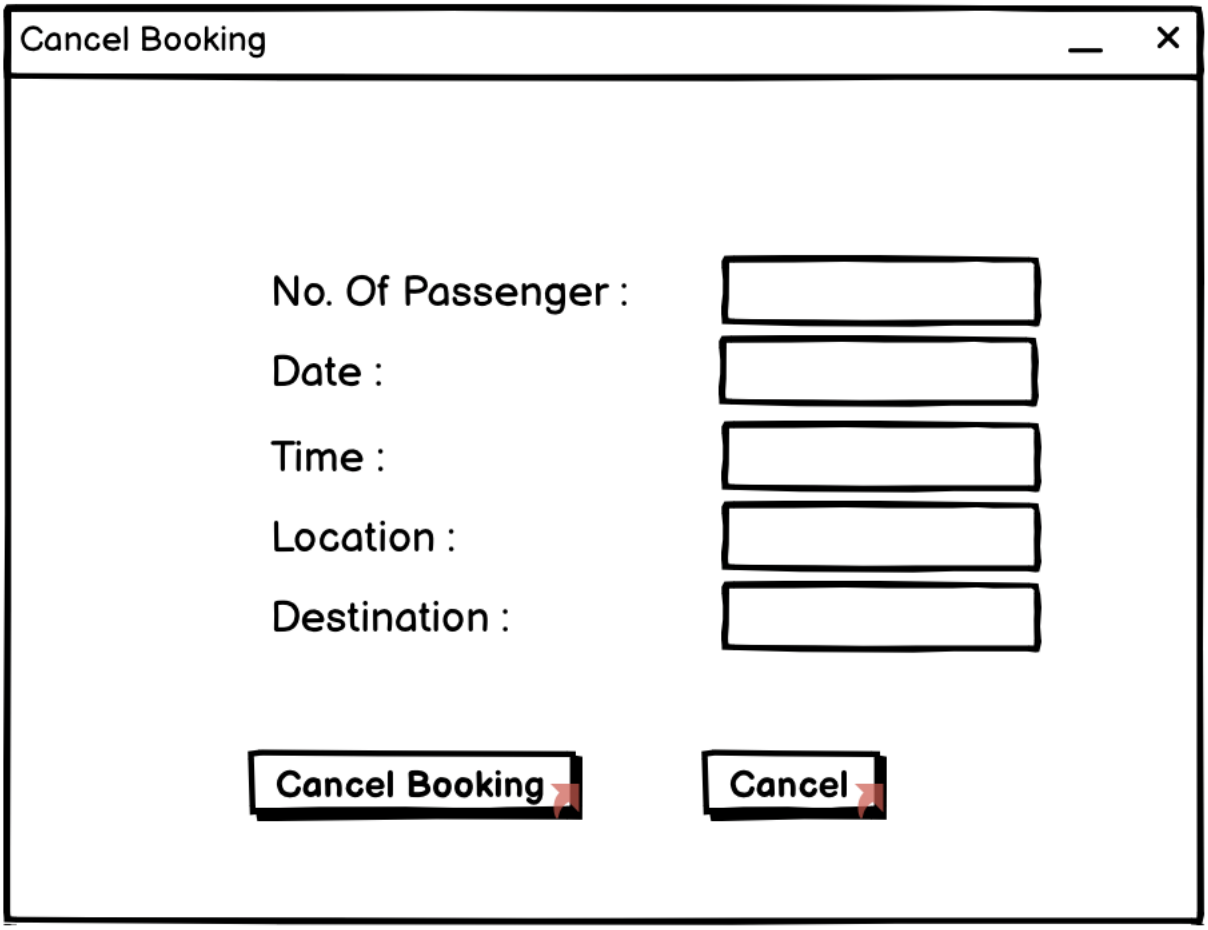


Fig.no.17 Cancel Trip of Prototype

This the trip cancellation page, where customer can cancel their trips.

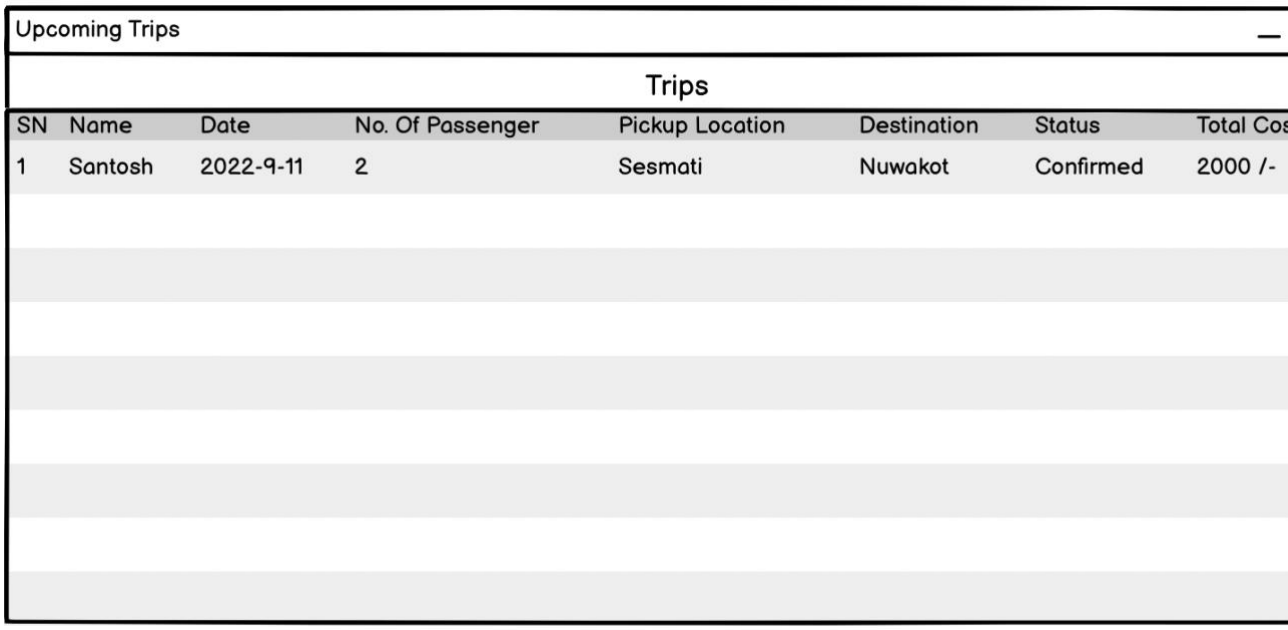


Fig.no.18 Driver Dashboard

This is the driver dashboard where they can see all their upcoming trips.

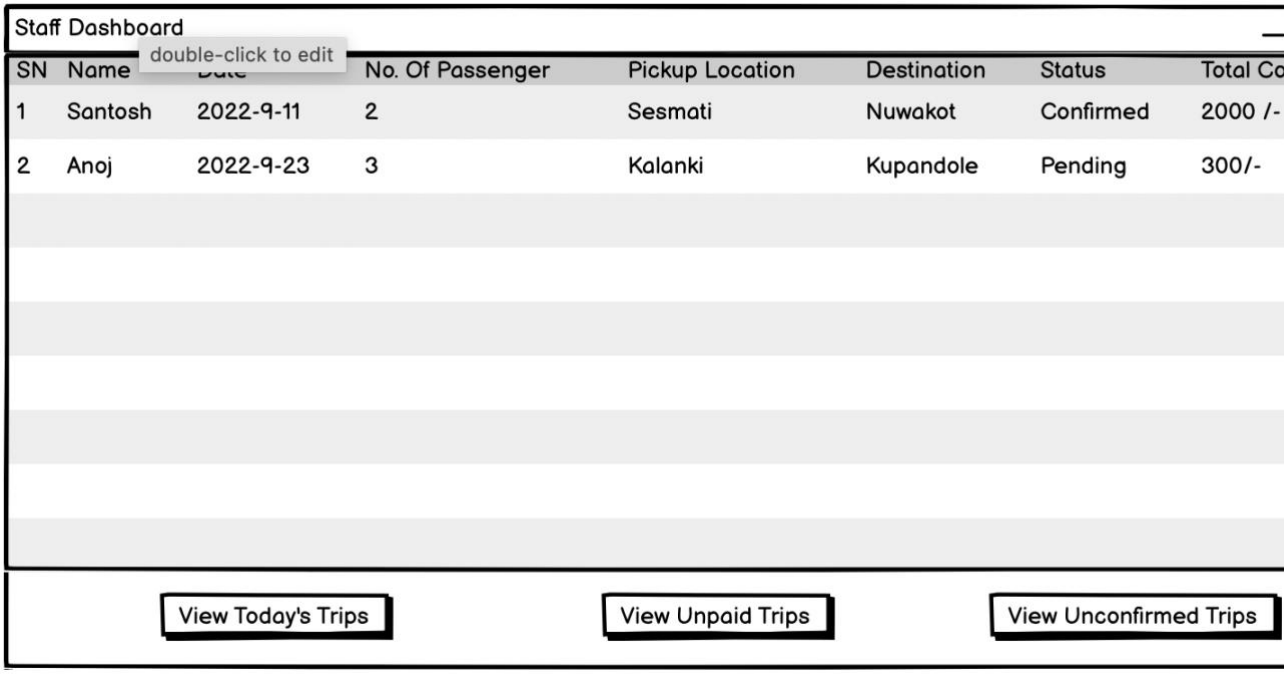


Fig.no.19. Staff Dashboard of Prototype

This is the staff dashboard where they can view all the trips and assign driver to the trips.

Implementation

Client-Server architecture was created for the development of the application. This architecture was considered the best option for the solution, as all users cannot access all the data stored in the database.

For the programming of the user interface and back-end part Python language was used. The language was implemented using PyCharm Integrated Development Environment (IDE). Development of the solution was simple due to its user-friendly interface and the range of its modules. Balsamiq was used to develop the prototype of the solution.

```python
# importing required modules
import psycopg2 as db
from psycopg2 import OperationalError


class DatabaseConnection:
    __conn = None
    __cur = None

    def __init__(self):

        # loading details of database
        self.host = 'localhost'
        self.db = 'xic'
        self.user = 'xic'
        self.port = 5432

        # connect to database
        self.__connect()
        self.__dbcur = DatabaseConnection.__cur
        self.__dbconn = DatabaseConnection.__conn

    def __connect(self):
        try:
            if DatabaseConnection.__conn is None:
                DatabaseConnection.__conn = db.connect(database=self.db, user=self.user,
                                                       host=self.host, port=self.port)
                DatabaseConnection.__conn.autocommit = True
                DatabaseConnection.__cur = DatabaseConnection.__conn.cursor()
        except OperationalError as e:
            raise e
```

Fig.no.20 Database Connector

The psycopg2 module was used to connect the application to the PostgreSQL database.

```python
def __init__(self):
    self.__custid = None
    self.__pickloc = None
    self.__droploc = None
    self.__tripdate = None
    self.__picktime = None
    self.__passno = None
    self.__cost = None
    self.__status = None
    self.__distance = None
    self.__cur = DatabaseConnection().cursor
```

Fig.no.21.1 Encapsulation1

```python
def getstatus(self):
    return self.__status

def getdistance(self):
    return self.__distance

# setter
def setcustid(self, custid):
    self.__custid = custid

def setpickloc(self, pickloc):
    self.__pickloc = pickloc
```

Fig.no.21.2 Encapulation2

The above two figures show the encapsulation of data. Similarly, encapsulation was used in various other places this is just an example.

At the start, Balsamiq Wireframe was used to design a desired prototype for the solution. Then the Tkinter module was used to create the desired GUI for the development of the desired solution for the assignment. For the final product various modules were used which made the GUI more attractive.

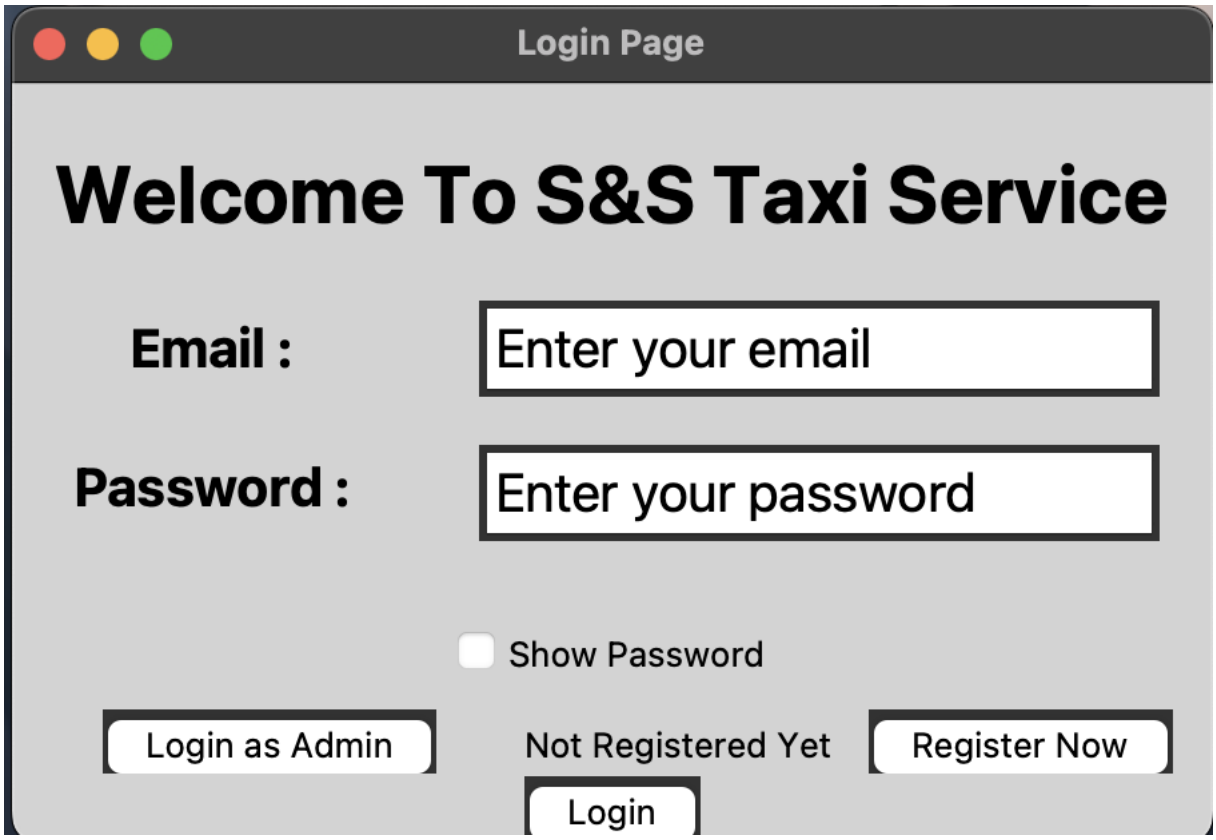


Fig.no.22 Login Page

The first page after opening the software.

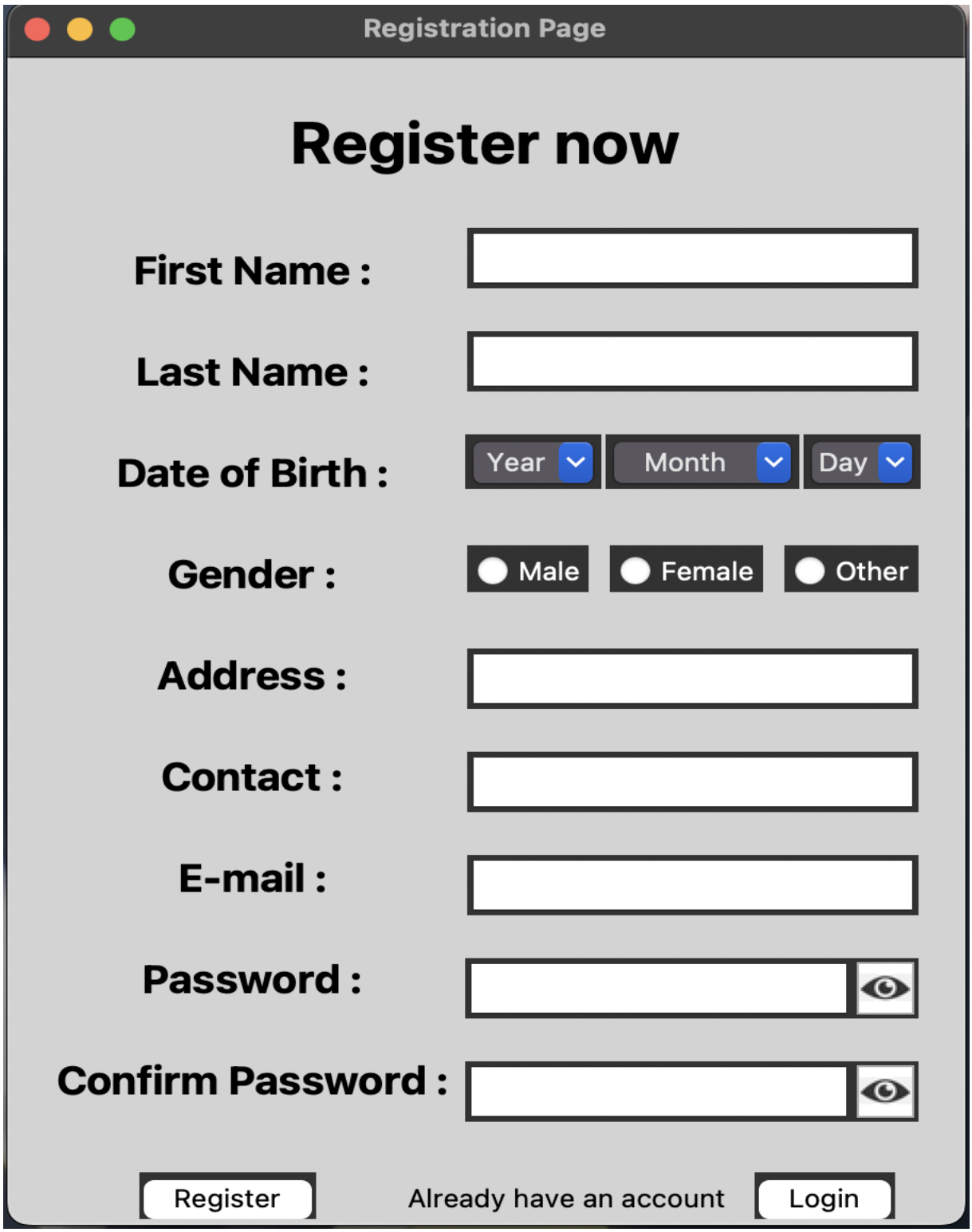


Fig.no.23 Registration Page for Customer

This is the registration page for the customer, as driver and staff are added by staff.

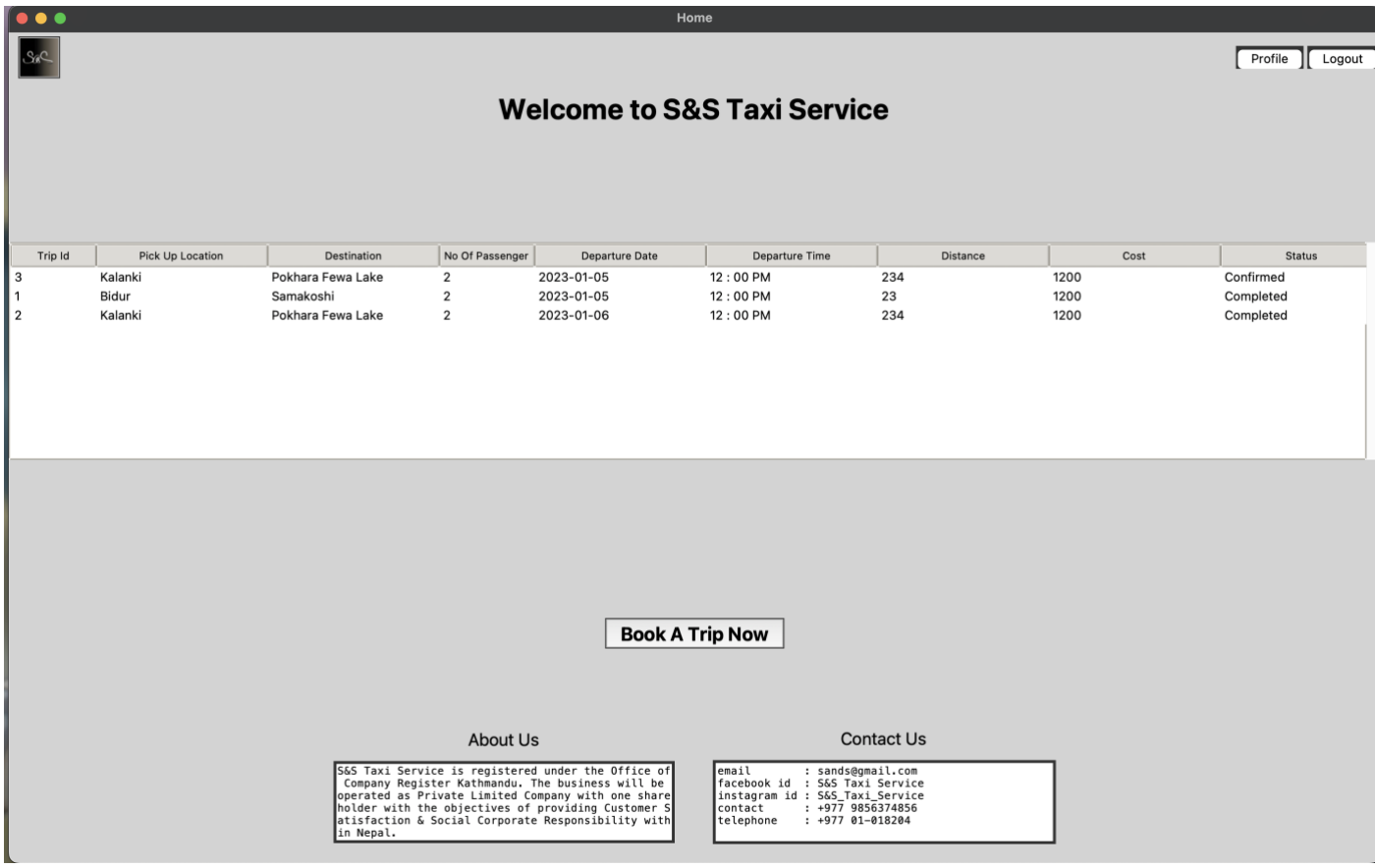


Fig.no.24 Dashboard of Customer

The page after logging in the application.

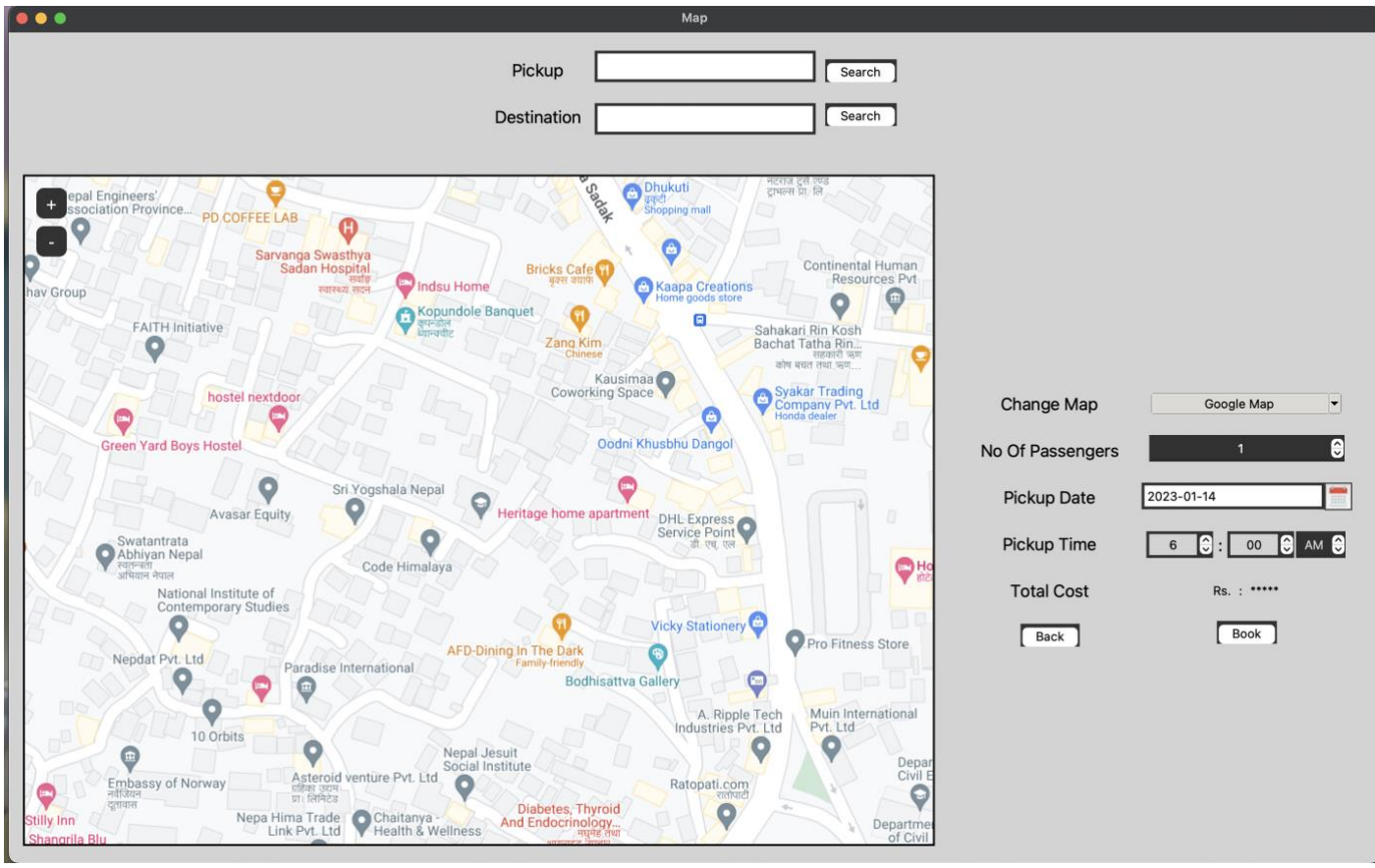


Fig.no.25 Booking Page

The page where customers can book as many trips as they like.

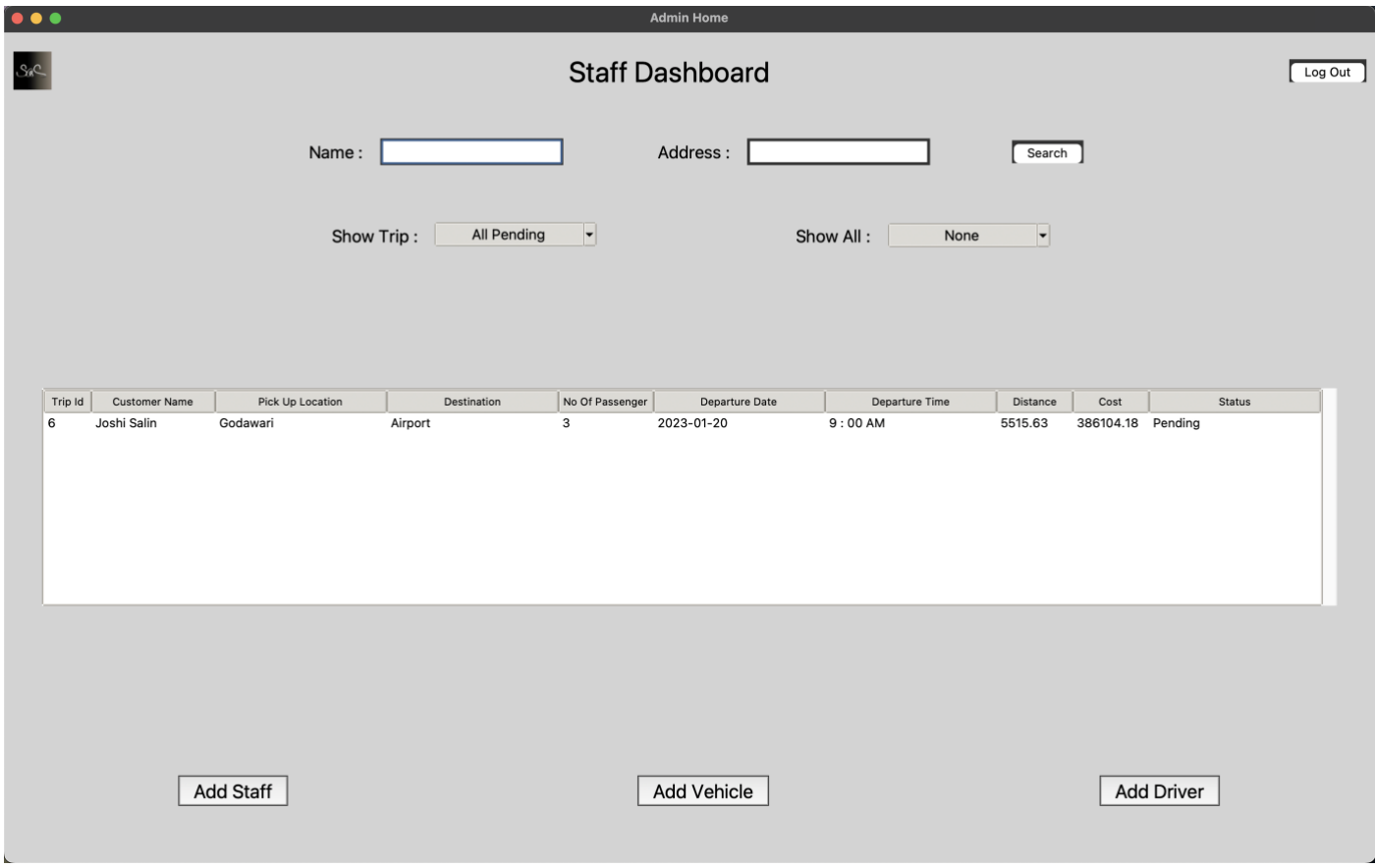
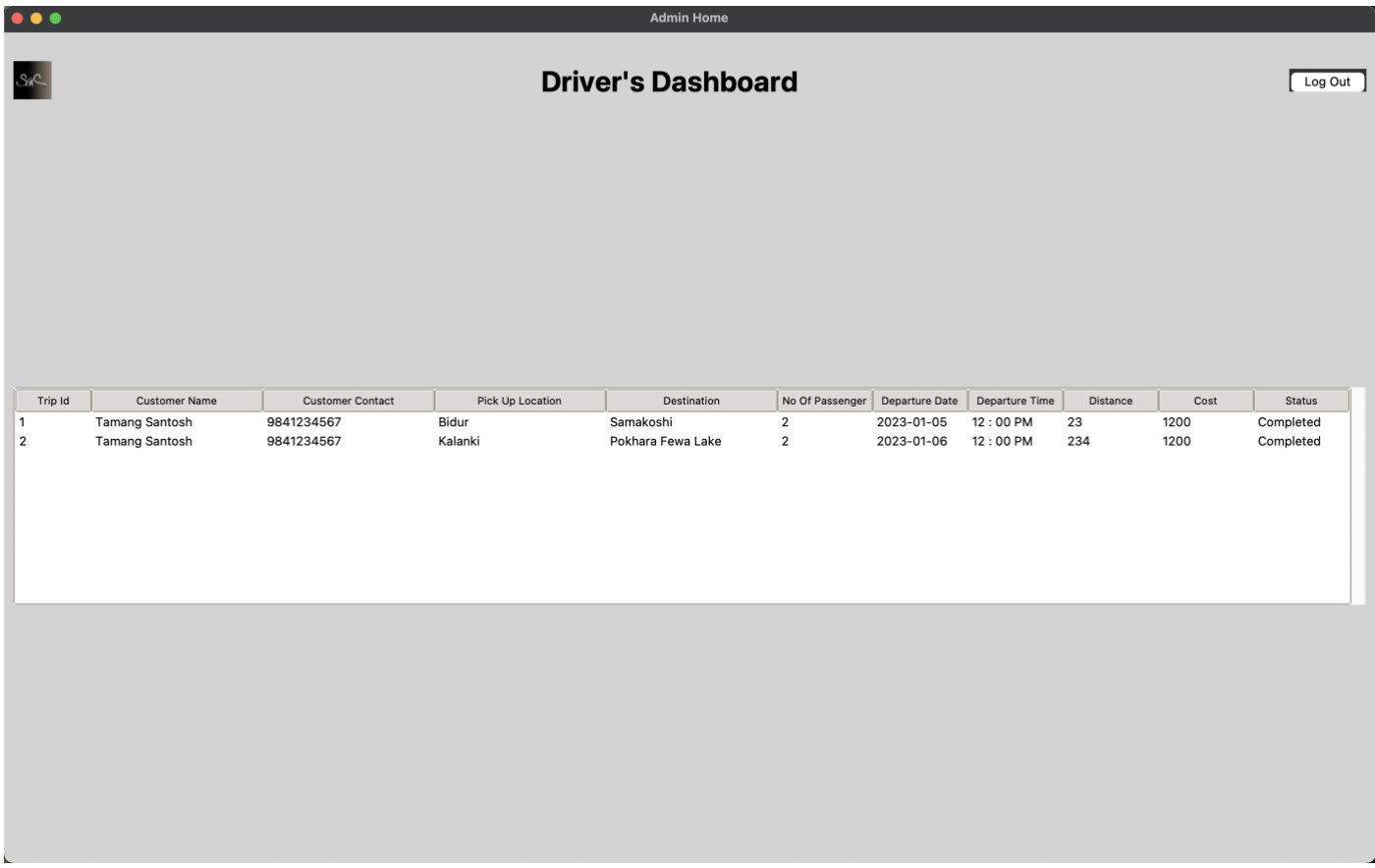


Fig.no.26 Staff Dashboard

This is the staff dashboard where staff adds driver, vehicle and staff and assigns driver to the booking.



Trip Id	Customer Name	Customer Contact	Pick Up Location	Destination	No Of Passenger	Departure Date	Departure Time	Distance	Cost	Status
1	Tamang Santosh	9841234567	Bidur	Samakoshi	2	2023-01-05	12 : 00 PM	23	1200	Completed
2	Tamang Santosh	9841234567	Kalanki	Pokhara Fewa Lake	2	2023-01-06	12 : 00 PM	234	1200	Completed

Fig.no.27 Driver Dashboard

This is page where the drivers can see their upcoming trips and start and end them.

Testing

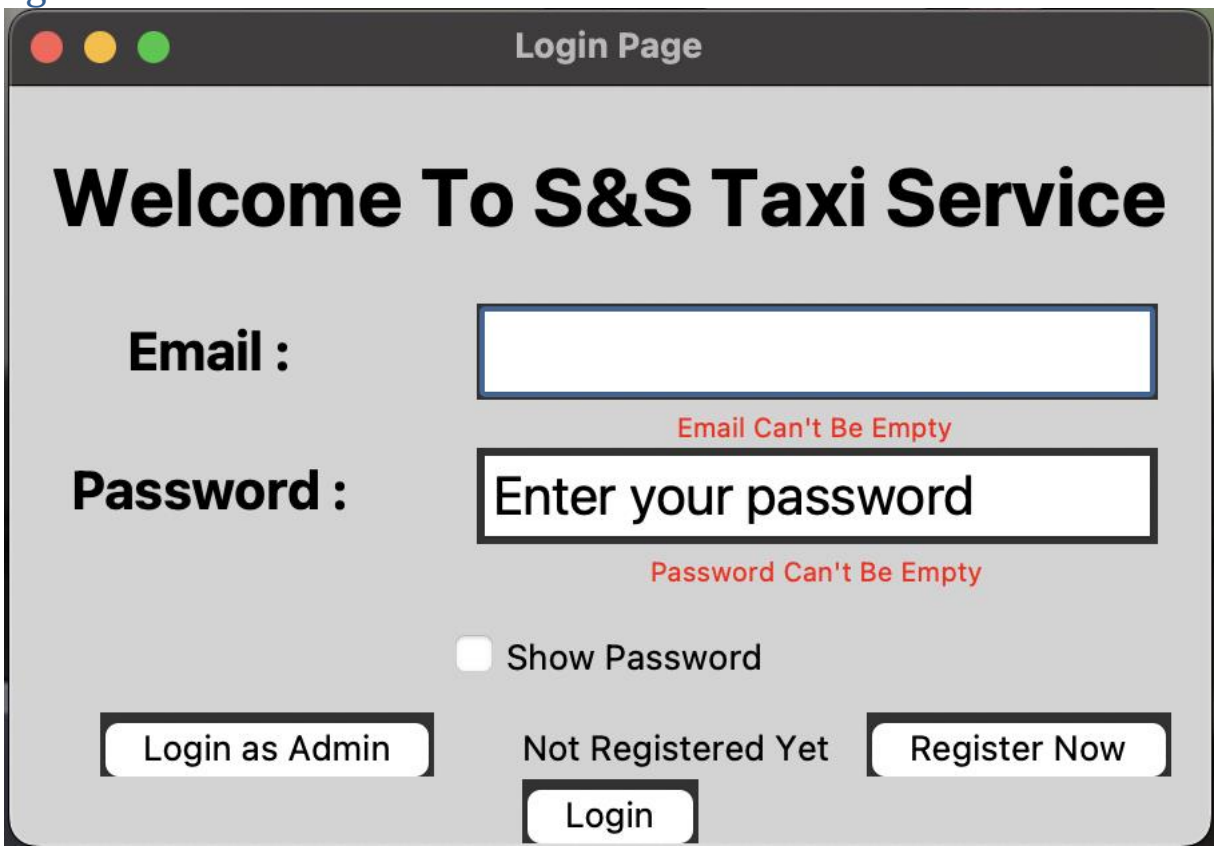


Fig.no.28 Logging with Empty Fields

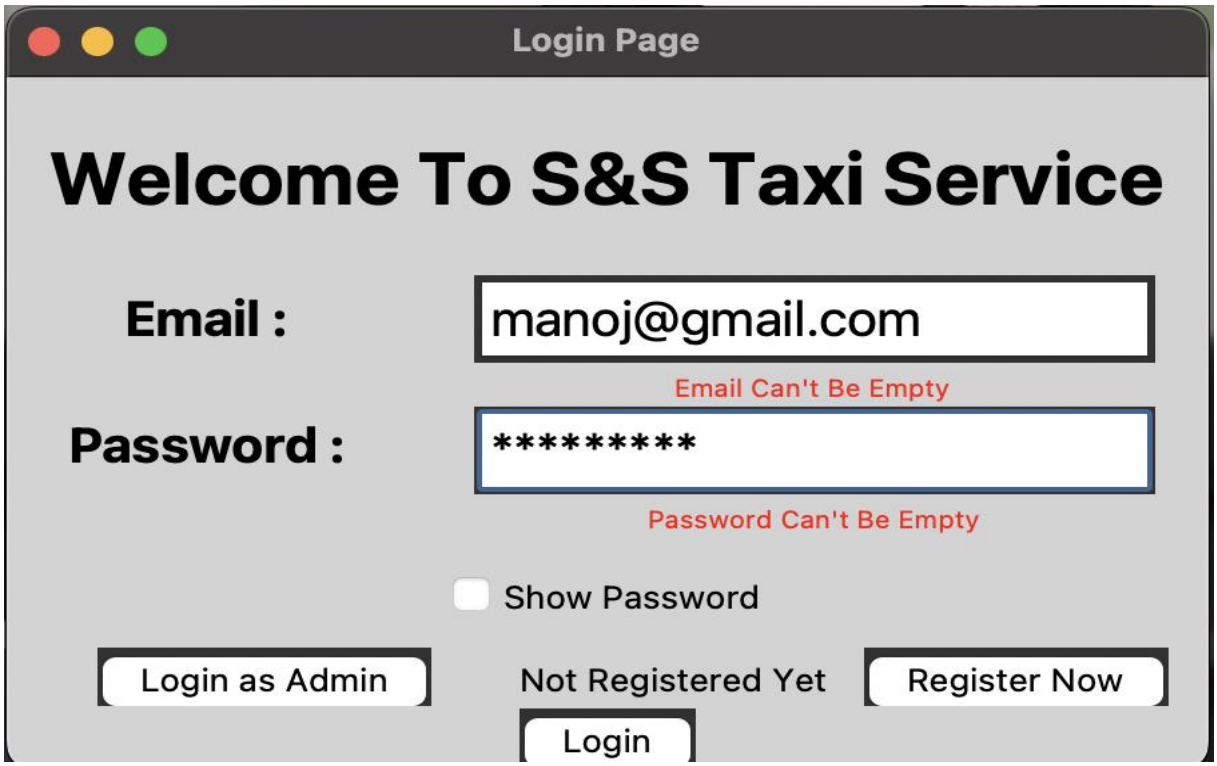


Fig.no.29 Logging with Incorrect Data

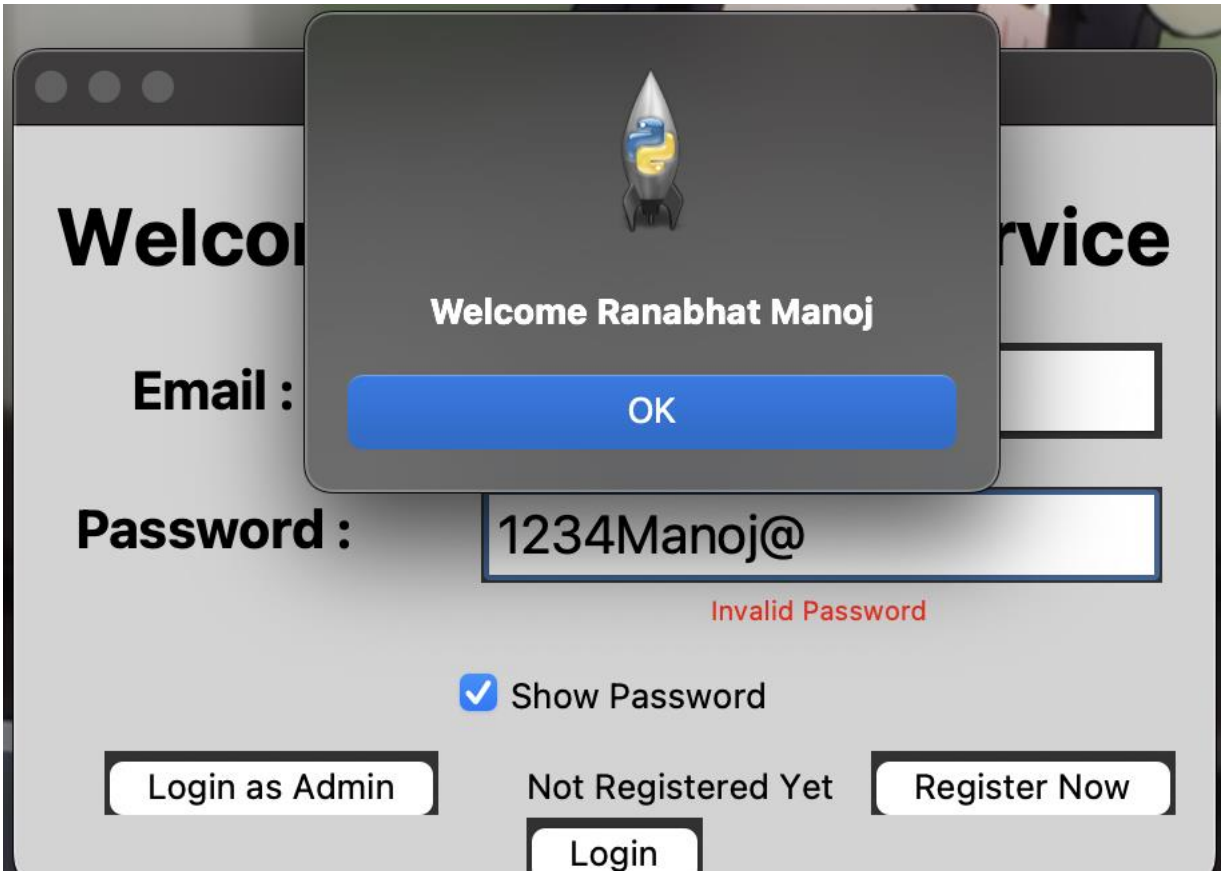


Fig.no.30 Logging with Correct Data

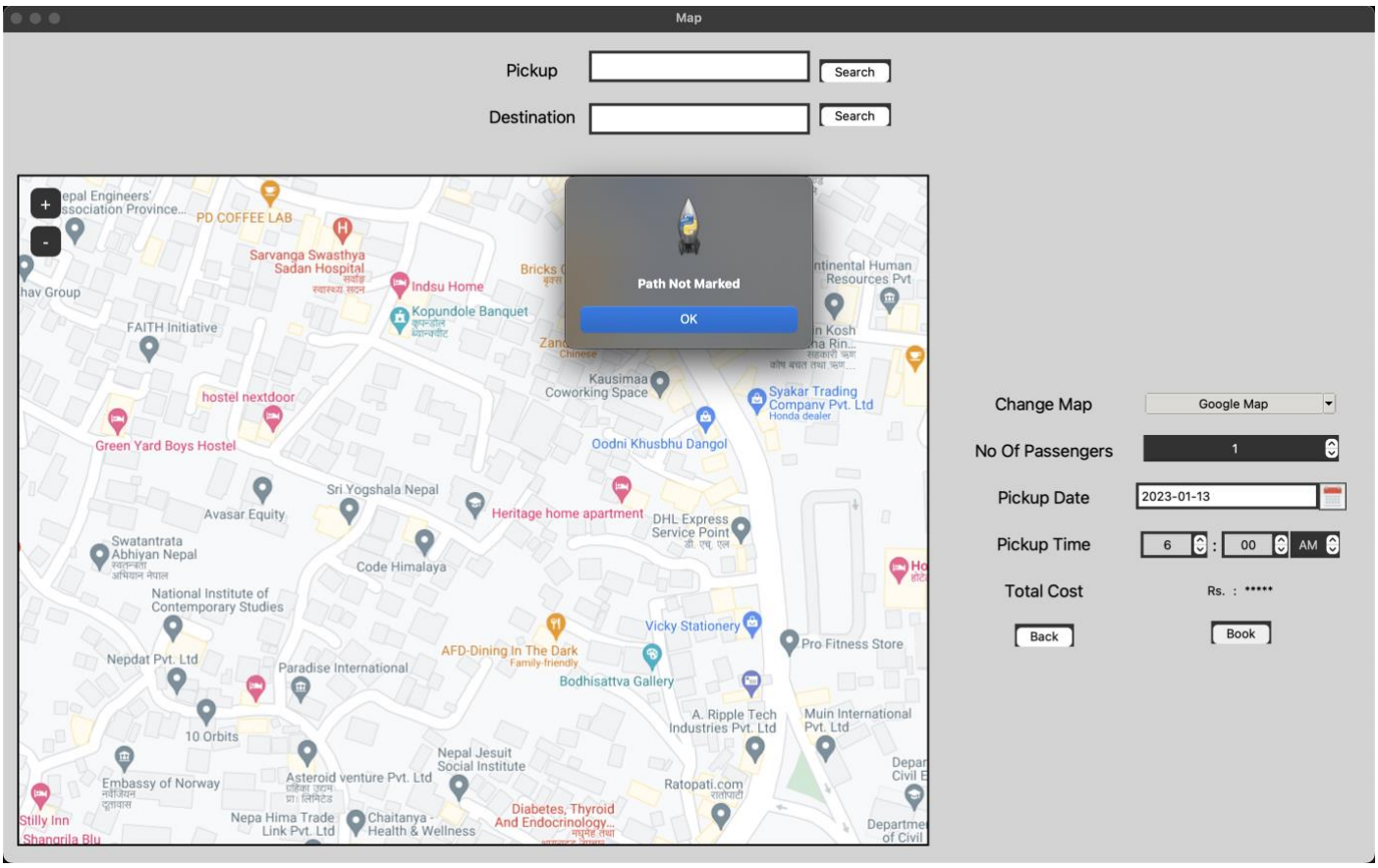


Fig.no.31 Booking Trip with Empty Field

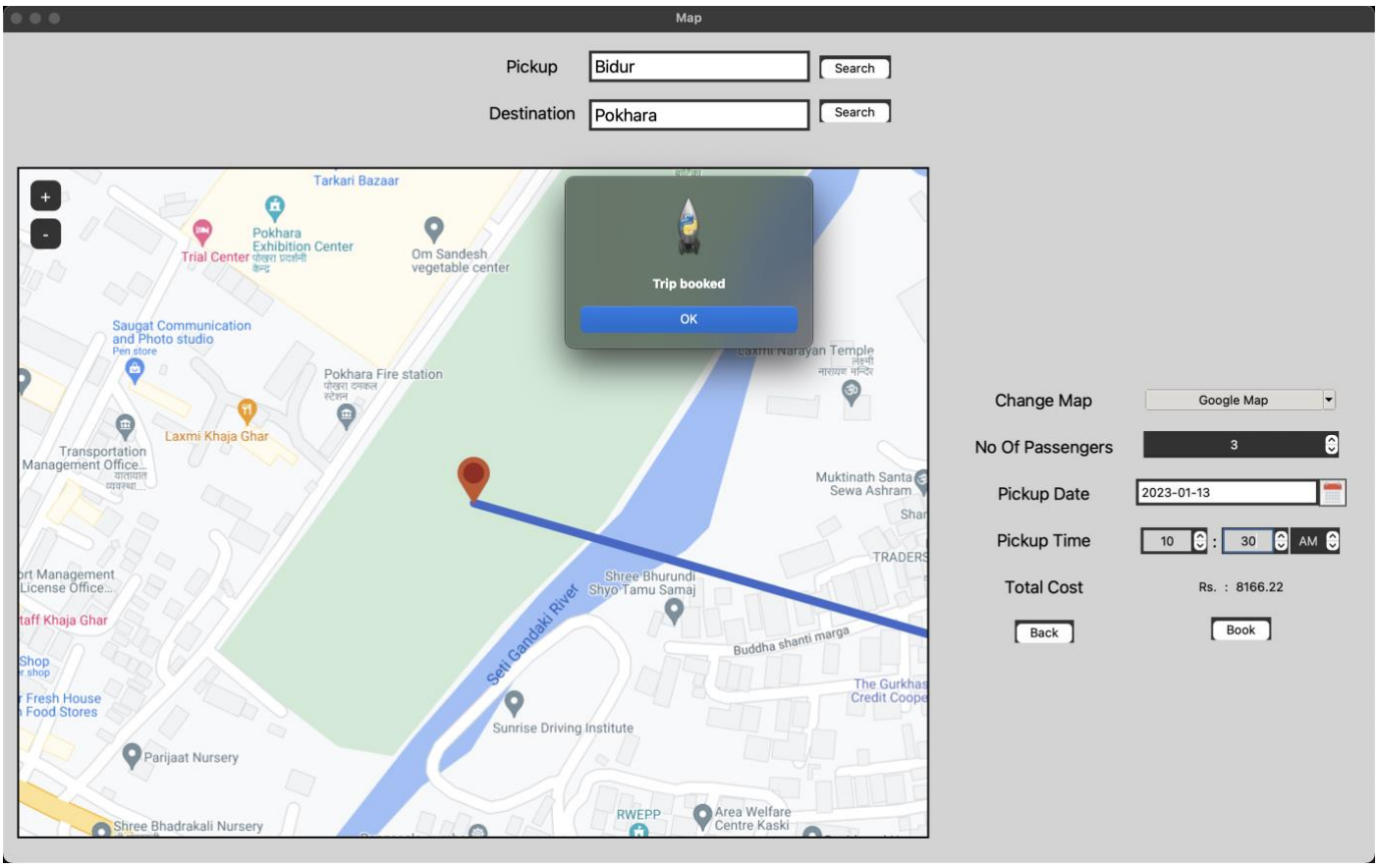


Fig.no.32 Booking with Correct Information

Fig.no.33 Registering with Empty Field

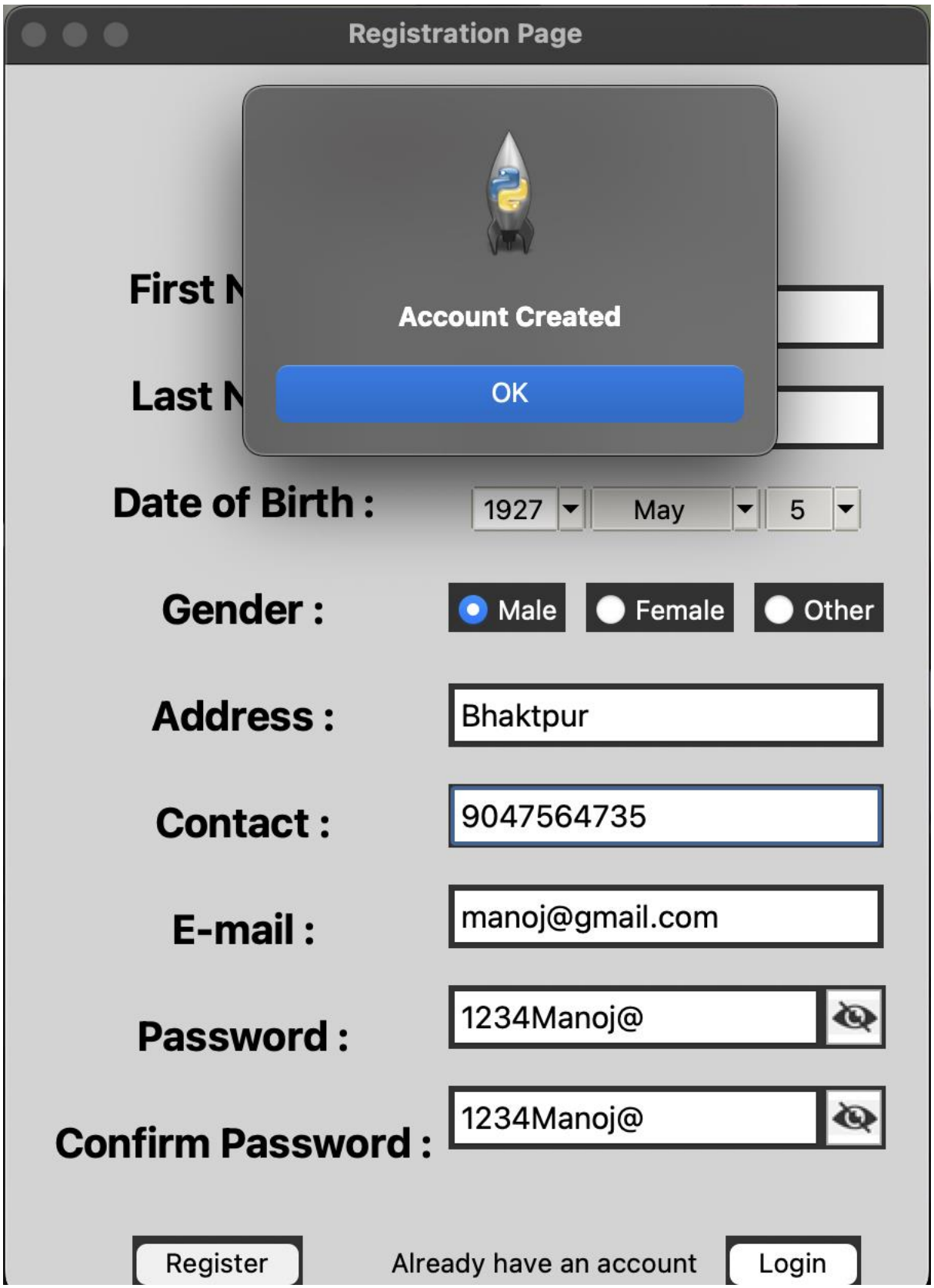


Fig.no.34 Registering with Correct Information

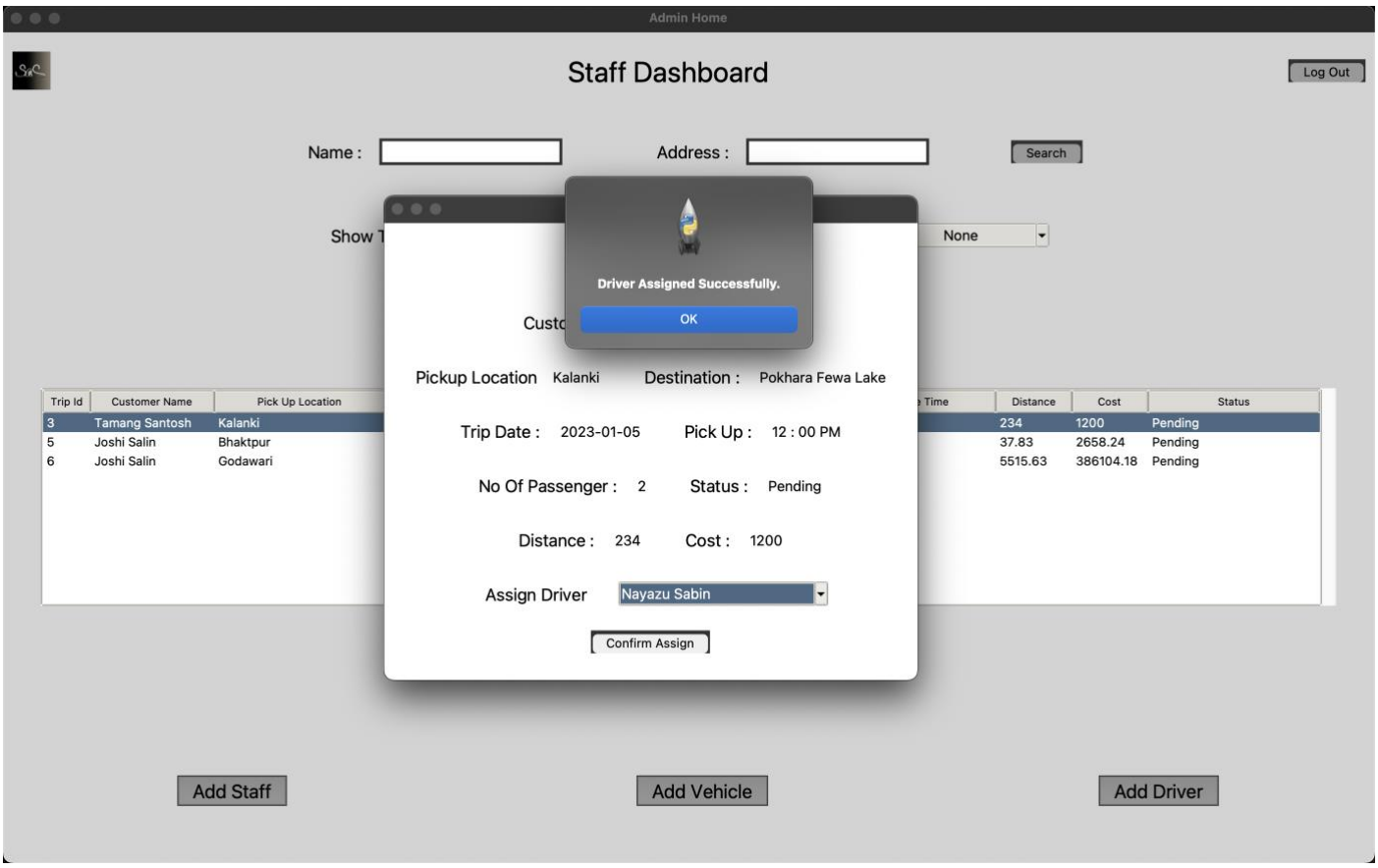


Fig.no.35 Assigning Driver

Test No.	Description	Expected Result	Obtained Result	Figure No.
1	Logging in with empty fields	Show error under entry field	Error shown under entry field	28
2	Logging in with incorrect data	Show error message	Error message shown	29
3	Logging in with correct data	Login successful	Login successful	30
4	Booking with empty location	Show error message	Error message shown	31
5	Booking with correct information	Booking successful	Booking successful	32
6	Registering with empty fields	Show error under entry field	Error shown under entry field	33
7	Registering with correct information	Register successful	Register successful	34
8	Assigning driver	Show message Driver assigned	Message shown driver assigned	35

Discussion / Reflection / Critical Analysis

For the assignment, the work was started at most 1 month prior to the submission date to provide more time till the due date to create a solution desired for the assignment. As the work started, I became confused about which architecture to follow. So, I consulted our lecturer, and he provided us with the demo model to follow and we were free to modify the model any way we want.

After getting the model, coding was not the hard part, debugging was. Importing various modules and using them to create a suitable GUI was easy, but many modules mean many debugging. To debug some code, I asked for help from friends, lecturers and even some websites.

Having completed the assignment on time feels so good. The assignment was completed prior to the due date, but the video was not recorded till the second last day.

Works Cited

TechTarget Contributor, 2023. *Tech Target.* [Online]
Available at: https://www.techtarget.com/whatis/definition/use-case-diagram
[Accessed 20 December 2022].
Tutorialspoint, 2022. *Tutorialspoint.* [Online]
Available at: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm
[Accessed 20 December 2022].
Tutorialspoint, 2011. *Tutorialspoint.* [Online]
Available at: https://www.javatpoint.com/uml-class-diagram
[Accessed 20 December 2022].
Visual Paradigm, 2022. *Visual Paradigm.* [Online]
Available at: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/
[Accessed 20 December 2022].
Varma, S., 2022. *Scaler Topics.* [Online]
Available at: https://www.scaler.com/topics/data-dictionary-in-dbms/
[Accessed 20 December 2022].
Tutorialspoint, 2022. *Tutorialspoint.* [Online]
Available at: https://www.tutorialspoint.com/software_engineering/software_user_interface_design.htm
[Accessed 20 December 2022].
Interaction Design Foundation, 2022. *Interaction Design Foundation.* [Online]
Available at: https://www.interaction-design.org/literature/topics/ui-design
[Accessed 20 December 2022].

Appendix

```
"""
This is the main file that starts the program.
Everything starts from here.
```

```
"""

# importing required modules
from View.Basewindow import Basewindow
from Controller.Controller import Controller


# running the code from here
# START POINT
if __name__ == '__main__':
    root = Basewindow()
    log = Controller(root)
    root.mainloop()
```

```
"""
    Module View
    This Is The Base Window Where All Frame Is Added
"""
# importing required modules
from tkinter import Tk


# creating basewindow class which creates a tk object
class Basewindow(Tk):

    # inheriting the tk and creating tk
    def __int__(self):
        super.__init__()
```

```
"""
    Module View
    This Is A Frontend Which User Sees And Input Data
"""

# importing required modules
import re
```

```
from tkinter import *
from tkinter import ttk
from tkinter.ttk import Combobox
from geopy.distance import geodesic
from datetime import date, timedelta
from Model.TripModel import TripModel
from tkcalendar.dateentry import Calendar
from Model.CustomerModel import CustomerModel
from tkinter.messagebox import askyesno, showinfo, showerror
from tkintermapview import TkinterMapView, convert_coordinates_to_address, convert_address_to_coordinates


# creating custdashboard
class CustDashboard(ttk.Frame):

    def __init__(self, window, controller, custid):

        self.__tripmod = TripModel()
        self.__custmod = CustomerModel()
        self.__custid = custid
        self.__window = window
        self.__window.deiconify()
        self.__controller = controller
        super().__init__(self.__window)
        self.__window.geometry('1920x1080')
        self.__window.title("Home")
        self.triptable = None
        self.__nmregex = ("[A-Z][a-z]{2,15}")  # firstname, lastname regex
        self.__conregx = ("[9]{1}[\d]{9}")  # contact regex

        mainf = Frame(self.__window, bg="Light Grey")
        mainf.pack(fill="both", expand=True)

        top = Frame(mainf, bg="Light Grey")
        top.pack(fill="x", pady=5)

        title = Frame(mainf, bg="Light Grey")
```

```
    title.pack(side="top", pady=5)

    titlelb = Label(title, text="Welcome to S&S Taxi Service", bg="Light Grey", fg="Black", font=("", 30, "bold"))
    titlelb.pack(anchor="center")

    self.logo = PhotoImage(file='Resource/logo.png')
    self.show = Button(top, image=self.logo, bd=0, command=self.call)
    self.show.pack(side='left', padx=10)

    logout = Button(top, text="Logout", bg="light Grey", fg="Black", bd=0, command=self.confirm)
    logout.pack(side="right", padx=2)

    profile = Button(top, text="Profile", bg="light Grey", fg="Black", bd=0, command=self.callprofile)
    profile.pack(side="right", padx=2)

    self.table = Frame(mainf, bg='Light Grey')
    self.table.pack(expand=True, pady=40)

    style = ttk.Style(self.table)
    # set ttk theme to "clam" which support the field background option
    style.theme_use("clam")
    style.configure("Treeview", background="White", foreground="Black")

    self.scrollbar = Scrollbar(self.table)
    self.scrollbar.pack(side='right', fill='y')

    self.owntriptable()

    main = Frame(mainf, bg="Light Grey")
    main.pack(pady=50)

    makeb = Frame(main, bg="Light Grey")
    makeb.pack(anchor='center')

    make = Button(makeb, text="Book A Trip Now", bg="light Grey", fg="Black", bd=0, font=("", 20, "bold"),
            command=self.call)
    make.pack()
```

```
    bottomf = Frame(mainf, bg='Light Grey')
    bottomf.pack(side='bottom', pady=20)

    aboutus = Frame(bottomf, bg='Light Grey')
    aboutus.pack(side='left', padx=20)

    aboutlb = Label(aboutus, text='About Us', bg='Light Grey', fg='Black', font=('', 18))
    aboutlb.pack(anchor='center', pady=10)

    aboucon = Text(aboutus, width=50, height=6, bg='White', fg='Black', bd=0)
    aboucon.pack(anchor='center')
    aboucon.insert(1.0, "S&S Taxi Service is registered under the Office of Company Register Kathmandu. The "
                   "business will be operated as Private Limited Company with one shareholder with the "
                   "objectives of providing Customer Satisfaction & Social Corporate Responsibility "
                   "within Nepal.")
    aboucon.config(state='disabled')

    contactus = Frame(bottomf, bg='Light Grey')
    contactus.pack(side='right', padx=20)

    contactlb = Label(contactus, text='Contact Us', bg='Light Grey', fg='Black', font=('', 18))
    contactlb.pack(anchor='center', pady=10)

    concon = Text(contactus, width=50, height=6, bg='White', fg='Black', bd=1)
    concon.pack(anchor='center')
    concon.insert(1.0, "email        : sands@gmail.com                    "
                  "facebook id  : S&S Taxi Service                  "
                  "instagram id : S&S_Taxi_Service                  "
                  "contact      : +977 9856374856                   "
                  "telephone    : +977 01-018204")
    concon.config(state='disabled')

# calling profile top level
def callprofile(self):
    top = Toplevel()
    top.geometry('550x500+450+250')
```

```
        top.title("Profile")

        # always top
        top.transient(self.__window)
        top.grab_set()
        self.profile(top)

    # calling booking page
    def call(self):
        self.__controller.book(self.__custid)

    # creating table and adding
    def owntriptable(self):

        self.triptable = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.triptable.pack()

        self.scrollbar.config(command=self.triptable.yview)

        # defining columns
        self.triptable["columns"] = ("Trip ID", "Pick Up Location", "Destination", "No Of Passenger", "Departure Date",
                                     "Departure Time", "Distance", "Cost", "Status")

        # formatting columns
        self.triptable.column("#0", width=0, stretch=NO)
        self.triptable.column("Trip ID", width=80, minwidth=90, anchor=W)
        self.triptable.column("Pick Up Location", width=180, minwidth=90, anchor=W)
        self.triptable.column("Destination", width=180, minwidth=90, anchor=W)
        self.triptable.column("No Of Passenger", width=100, minwidth=90, anchor=W)
        self.triptable.column("Departure Date", width=180, minwidth=90, anchor=W)
        self.triptable.column("Departure Time", width=180, minwidth=90, anchor=W)
        self.triptable.column("Distance", width=180, minwidth=90, anchor=W)
        self.triptable.column("Cost", width=180, minwidth=90, anchor=W)
        self.triptable.column("Status", width=180, minwidth=90, anchor=W)

        # creating heading
        self.triptable.heading("#0", text='Label')
```

```
        self.triptable.heading("Trip ID", text="Trip Id")
        self.triptable.heading("Pick Up Location", text="Pick Up Location")
        self.triptable.heading("Destination", text="Destination")
        self.triptable.heading("No Of Passenger", text="No Of Passenger")
        self.triptable.heading("Departure Date", text="Departure Date")
        self.triptable.heading("Departure Time", text="Departure Time")
        self.triptable.heading("Distance", text="Distance")
        self.triptable.heading("Cost", text="Cost")
        self.triptable.heading("Status", text="Status")

        # reading data from table
        record = self.__tripmod.custrip(self.__custid)

        if record:

            # inserting data into table
            for data in record:
                self.triptable.insert("", index="end",
                                      values=(data[0], data[2], data[3], data[6], data[4], data[5], data[7], data[8]
                                              , data[9]))

            self.triptable.bind('<Double-1>', self.data)

    # confirm logging out
    def confirm(self):
        ans = askyesno("Conformation", "Are You Sure You Want To Log Out?")
        if ans:
            self.__controller.log()

    # creating top level for specific trip
    def data(self, event):
        tripd = None
        try:
            record = None
            if self.triptable.selection()[0]:
                for data in self.triptable.selection():
                    value = self.triptable.item(data)
```

```
            tripd = value['values']
        tp = Toplevel()

        # always top
        tp.transient(self.__window)
        tp.grab_set()
        if tripd[8] == 'Pending' or tripd[8] == 'Cancelled':
            record = self.__tripmod.tripdetail(tripd[0])
        else:
            record = self.__tripmod.tripdetails(tripd[0])
        self.details(tp, record)
    except Exception as e:
        showinfo("Message", "Select A Data")
        print(e)

def details(self, top, record):
    for data in record:

        top = top
        top.geometry('500x380+450+235')
        top.title('Trip Details')

        mainf = Frame(top, bg='White')
        mainf.pack(fill='both', expand=True)

        titlef = Frame(mainf, bg='White')
        titlef.pack(pady=10)

        title = Label(titlef, text='Trip Detail', bg='White', fg='Black', font=('EB Garamond', 25, 'bold'))
        title.pack(anchor='center')

        locationf = Frame(mainf, bg='White')
        locationf.pack(pady=10)

        picklocation = Frame(locationf, bg='White')
        picklocation.pack(side='left', padx=10)
```

```
pickloc = Label(picklocation, text='Pickup : ', bg='White', fg='Black', font=('Crimson Text', 18))
pickloc.pack(side='left', padx=10)

picklocdata = Label(picklocation, text=data[2], bg='White', fg='Black', font=('Crimson Text', 16))
picklocdata.pack(side='left')

droplocation = Frame(locationf, bg='White')
droplocation.pack(side='right', padx=10)

droploc = Label(droplocation, text='Destination : ', bg='White', fg='Black', font=('Crimson Text', 18))
droploc.pack(side='left')

droplocdata = Label(droplocation, text=data[3], bg='White', fg='Black', font=('Crimson Text', 16))
droplocdata.pack(side='left', padx=10)

datime = Frame(mainf, bg='White')
datime.pack(pady=10)

datef = Frame(datime, bg='White')
datef.pack(side='left', padx=10)

datelb = Label(datef, text='Date : ', bg='White', fg='Black', font=('Crimson Text', 18))
datelb.pack(side='left', padx=10)

datedata = Label(datef, text=data[4], bg='White', fg='Black', font=('Crimson Text', 16))
datedata.pack(side='left')

timef = Frame(datime, bg='White', padx=10)
timef.pack(side='right')

timelb = Label(timef, text='Time : ', bg='White', fg='Black', font=('Crimson Text', 18))
timelb.pack(side='left')

timedata = Label(timef, text=data[5], bg='White', fg='Black', font=('Crimson Text', 16))
timedata.pack(side='left', padx=10)

statupass = Frame(mainf, bg='White')
```

```
statupass.pack(pady=10)

passengerf = Frame(statupass, bg='White')
passengerf.pack(side='left', padx=10)

passengerlb = Label(passengerf, text='Passenger : ', bg='White', fg='Black', font=('Crimson Text', 18))
passengerlb.pack(side='left', padx=10)

passegerdata = Label(passengerf, text=data[6], bg='White', fg='Black', font=('Crimson Text', 16))
passegerdata.pack(side='left')

statusf = Frame(statupass, bg='White')
statusf.pack(side='right', padx=10)

statuslb = Label(statusf, text='Status : ', bg='White', fg='Black', font=('Crimson Text', 18))
statuslb.pack(side='left')

statusdata = Label(statusf, text=data[9], bg='White', fg='Black', font=('Crimson Text', 16))
statusdata.pack(side='left', padx=10)

distcost = Frame(mainf, bg='White')
distcost.pack(pady=10)

distf = Frame(distcost, bg='White')
distf.pack(side='left', padx=10)

distlb = Label(distf, text="Distance : ", bg='White', fg='Black', font=('Crimson Text', 18))
distlb.pack(side='left', padx=10)

distdata = Label(distf, text=data[7], bg='White', fg='Black', font=('Crimson Text', 16))
distdata.pack(side='left')

costf = Frame(distcost, bg='White')
costf.pack(side='right', padx=10)

costlb = Label(costf, text="Cost : ", bg='White', fg='Black', font=('Crimson Text', 18))
costlb.pack(side='left')
```

```
costdata = Label(costf, text=data[8], bg='White', fg='Black', font=('Crimson Text', 16))
costdata.pack(side='left', padx=10)

driverf = Frame(mainf, bg='White')
driverf.pack(pady=10)

drivernamef = Frame(driverf, bg='White')
drivernamef.pack(side='left', padx=10)

drivernamelb = Label(drivernamef, text='Driver Name : ', bg='White', fg='Black', font=('Crimson Text', 18))
drivernamelb.pack(side='left', padx=10)

drivernamedata = Label(drivernamef, text='Pending', bg='White', fg='Black', font=('Crimson Text', 16))
drivernamedata.pack(side='left')

driverconf = Frame(driverf, bg='White')
driverconf.pack(side='right', padx=10)

driverconlb = Label(driverconf, text="Driver Contact : ", bg='White', fg='Black', font=('Crimson Text', 18))
driverconlb.pack(side='left')

drivercondata = Label(driverconf, text='Pending', bg='White', fg='Black', font=('Crimson Text', 16))
drivercondata.pack(side='left', padx=10)

vehiclef = Frame(mainf, bg='White')
vehiclef.pack(pady=10)

vehiclenamef = Frame(vehiclef, bg='White')
vehiclenamef.pack(side='left', padx=10)

vehiclenamelb = Label(vehiclenamef, text='Vehicle Type : ', bg='White', fg='Black', font=('Crimson Text', 18))
vehiclenamelb.pack(side='left', padx=20)

vehiclenamedata = Label(vehiclenamef, text='Pending', bg='White', fg='Black', font=('Crimson Text', 16))
vehiclenamedata.pack(side='left')
```

```
    vehiclenof = Frame(vehiclef, bg='White')
    vehiclenof.pack(side='right')

    vehiclenolb = Label(vehiclenof, text='Vehicle No : ', bg='White', fg='Black', font=('Crimson Text', 18))
    vehiclenolb.pack(side='left')

    vehiclenodata = Label(vehiclenof, text='Pending', bg='White', fg='Black', font=('Crimson Text', 16))
    vehiclenodata.pack(side='left', padx=10)

    if not data[9] == 'Pending' and not data[9] == 'Cancelled':
        dname = data[17] + " " + data[16]
        vname = data[28] + " " + data[29]
        drivernamedata.config(text=dname)
        drivercondata.config(text=data[21])
        vehiclenamedata.config(text=vname)
        vehiclenodata.config(text=data[27])
    else:
        drivernamedata.config(text='Pending')
        drivercondata.config(text='Pending')
        vehiclenamedata.config(text='Pending')
        vehiclenodata.config(text='Pending')

    btn = Frame(mainf, bg='White')
    btn.pack(expand=True, pady=10)

    cancelf = Frame(btn, bg='White')
    cancelf.pack(side='left', padx=10)

    cancel = Button(cancelf, text='Cancel Trip', bg='Light Grey', fg='Black', bd=0
                    , command=lambda: canceltrip(top))
    cancel.pack(anchor='center')

    deletef = Frame(btn, bg='White')
    deletef.pack(side='right', padx=10)

    delete = Button(deletef, text='Delete Trip', bg='Light Grey', fg='Black', bd=0
                    , command=lambda: deletetrip(top))
```

```
        delete.pack(anchor='center')

    # cancelling trip
    def canceltrip(top):
        if data[9] == "Cancelled":
            showinfo("Message", 'Trip Already Cancelled')
        elif data[9] == "Pending":
            if self.__tripmod.canceltrip(data[0]):
                ans = askyesno("Conformation", "Are You Sure You Want To Cancel The Trip?")
                if ans:
                    showinfo("Messsage", "Trip Cancelled Succefully")
                    top.destroy()
                    self.triptable.destroy()
                    self.owntriptable()
        else:
            showinfo("Message", "Trip Can Not Be Cancelled Now")
            top.destroy()

    def deletetrip(top):
        if data[9] == "Cancelled" or data[9] == "Pending":
            if self.__tripmod.deletetrip(data[0]):
                ans = askyesno("Conformation", "Are You Sure You Want To Delete The Trip?")
                if ans:
                    showinfo("Message", "Trip Deleted Successfully")
                    top.destroy()
                    self.triptable.destroy()
                    self.owntriptable()
        else:
            showinfo("Message", "Trip Can Not Be Deleted Now")
            top.destroy()

# creating profile top level
def profile(self, topf):

    top = topf
    mainf = Frame(top, bg="Light Grey")
    mainf.pack(fill="both", expand=True)
```

```
title = Frame(mainf, bg="Light Grey")
title.pack(side="top", pady=5)

titlelb = Label(title, text="Profile", bg="Light Grey", fg="Black", font=("", 30, "bold"))
titlelb.pack(anchor="center")

lbdata = Frame(mainf, bg='Light Grey')
lbdata.pack(pady=10)

lb = Frame(lbdata, bg='Light Grey')
lb.pack(side='left', padx=20, pady=10)

fnamelb = Label(lb, text='First Name : ', bg='Light Grey', fg='Black', font=('', 18))
fnamelb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()

lnamelb = Label(lb, text='Last Name : ', bg='Light Grey', fg='Black', font=('', 18))
lnamelb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()

doblb = Label(lb, text='Date Of Birth : ', bg='Light Grey', fg='Black', font=('', 18))
doblb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()

genderlb = Label(lb, text='Gender : ', bg='Light Grey', fg='Black', font=('', 18))
genderlb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()

addresslb = Label(lb, text='Address : ', bg='Light Grey', fg='Black', font=('', 18))
addresslb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()
```

```
contactlb = Label(lb, text='Contact : ', bg='Light Grey', fg='Black', font=('', 18))
contactlb.pack(anchor='center')

Label(lb, bg='Light Grey').pack()

emailb = Label(lb, text='Email : ', bg='Light Grey', fg='Black', font=('', 18))
emailb.pack(anchor='center')

data = Frame(lbdata, bg='Light Grey')
data.pack(side='right', padx=20, pady=10)

fn = StringVar()
fname = Entry(data, state='readonly', fg='Light Grey', readonlybackground='Black', textvariable=fn)
fname.pack(anchor='center')

Label(data, bg='Light Grey').pack()

ln = StringVar()
lname = Entry(data, state='readonly', fg='Light Grey', readonlybackground='Black', textvariable=ln)
lname.pack(anchor='center')

Label(data, bg='Light Grey').pack()

db = StringVar()
dob = Entry(data, state='readonly', fg='White', readonlybackground='Black', textvariable=db)
dob.pack(anchor='center')

Label(data, bg='Light Grey').pack()

gen = StringVar()
gender = Entry(data, state='readonly', fg='Light Grey', readonlybackground='Black', textvariable=gen)
gender.pack(anchor='center')

Label(data, bg='Light Grey').pack()

add = StringVar()
```

```
address = Entry(data, state='readonly', fg='Light Grey', readonlybackground='Black', textvariable=add)
address.pack(anchor='center')

Label(data, bg='Light Grey').pack()

con = StringVar()
contact = Entry(data, state='readonly', fg='Light Grey', readonlybackground='Black', textvariable=con)
contact.pack(anchor='center')

Label(data, bg='Light Grey').pack()

em = StringVar()
email = Entry(data, state='readonly', fg='White', readonlybackground='Black', textvariable=em)
email.pack(anchor='center')

record = self.__custmod.details(self.__custid)
for data in record:
    fn.set(data[1])
    ln.set(data[2])
    db.set(data[3])
    gen.set(data[4])
    add.set(data[5])
    con.set(data[6])
    em.set(data[7])

btn = Frame(mainf, bg='Light Grey')
btn.pack(pady=10)

editbtn = Frame(btn, bg='Light Grey')
editbtn.pack(side='left', padx=10, fill='both')

edit = Button(editbtn, text='Edit', bd=0, bg='#F2F3F5', fg='Black', command=lambda: editdetail(top))
edit.pack(anchor='center')

deletebtn = Frame(btn, bg='Light Grey')
deletebtn.pack(side='right', padx=10, fill='both')
```

```
    delete = Button(deletebtn, text='Delete', bd=0, bg='#F2F3F5', fg='Black', command=lambda: deleteacc())
    delete.pack(anchor='center')

    def editdetail(top):
        edit.destroy()
        delete.destroy()
        fname.config(state='normal')
        fname.config(bg='White', fg='Black')
        lname.config(state='normal')
        lname.config(bg='White', fg='Black')
        address.config(state='normal')
        address.config(bg='White', fg='Black')
        contact.config(state='normal')
        contact.config(bg='White', fg='Black')
        confirmedit = Button(editbtn, text='Confirm Edit', bd=0, bg='#F2F3F5', fg='Black',
                             command=lambda: insertdetail(top))
        confirmedit.pack(anchor='center')
        canceledit = Button(deletebtn, text='Cancel Edit', bd=0, bg='#F2F3F5', fg='Black',
                            command=lambda: cancel(top))
        canceledit.pack(anchor='center')

    def insertdetail(top):
        a = fnval()
        b = lnval()
        c = addval()
        d = conval()
        if a and b and c and d:
            if self.__custmod.updatedetatil(self.__custid):
                ans = askyesno("Conformation", "Are You Sure You Want To Change Te Detail?")
                if ans:
                    showinfo("Messasge", "Profile Succefully Updated")
                    top.destroy()

    def fnval():
        if not fn.get():
            showinfo("Message", "First Name Can Not Be Empty.")
            return False
```

```
        elif not re.match(self.__nmregex, fn.get()):
            showinfo("Message", 'Invalid First Name')
            return False
        else:
            self.__custmod.setfn(fn.get())
            return True

    def lnval():
        if not ln.get():
            showinfo("Message", "Last Name Can Not Be Empty.")
            return False
        elif not re.match(self.__nmregex, ln.get()):
            showinfo("Message", 'Invalid Last Name')
            return False
        else:
            self.__custmod.setln(ln.get())
            return True

    def addval():
        if not add.get():
            showinfo("Message", 'Address Can Not Be Empty.')
            return False
        else:
            self.__custmod.setadd(add.get())
            return True

    def conval():
        if not con.get():
            showinfo("Message", 'Contact Can Not Be Empty.')
            return False
        elif not re.match(self.__conregx, con.get()):
            showinfo("Message", 'Invalid Contact')
            return False
        else:
            self.__custmod.setcon(con.get())
            return True
```

```
    def cancel(top):
        top.destroy()

    def deleteacc():
        if self.__custmod.delete(self.__custid):
            ans = askyesno("Conformation", "Are You Sure You Want To Delete The Account?")
            if ans:
                showinfo("Message", 'Account Deleted Successfully')
                self.__controller.log()


# creating page with map
# creating booking page class
class BookingPage(ttk.Frame):

    # managing the top level
    def __init__(self, window, controller, custid):

        self.__custid = custid
        self.cord1 = None
        self.cord2 = None
        self.mark1 = None
        self.mark2 = None
        self.path1 = None
        self.distance = None
        self.cal = None
        self.top = None
        self.__ndate = None
        self.__mod = TripModel()
        self.__window = window
        self.__controller = controller
        super().__init__(self.__window)
        self.__window.geometry('1920x1080')
        self.__window.title("Map")
        self.__numregex = "[0-9]"  # checking for number only

        mainf = Frame(window, bg='Light Grey')
```

```
mainf.pack(expand=True, fill='both')

lbentry = Frame(mainf, bg='Light Grey')
lbentry.pack(pady=10)

lb = Frame(lbentry, bg="Light Grey")
lb.pack(side='left', pady=10, padx=10)

pickuplb = Label(lb, text='Pickup', bg='Light Grey', fg='Black', font=("", 18))
pickuplb.pack()

Label(lb, bg='Light Grey').pack()

droplb = Label(lb, text='Destination', bg='Light Grey', fg='Black', font=("", 18))
droplb.pack()

Label(lb, bg='Light Grey').pack()

btn = Frame(lbentry, bg='Light Grey')
btn.pack(side='right', padx=10)

pickbtn = Button(btn, text='Search', bd=0, command=self.pick_up)
pickbtn.pack()

Label(btn, bg='Light Grey').pack()

dropbtn = Button(btn, text='Search', bd=0, command=self.drop_loc)
dropbtn.pack()

Label(btn, bg='Light Grey').pack()

entry = Frame(lbentry, bg='Light Grey')
entry.pack(side='right', pady=10)

self.pickup = Entry(entry, bg='White', fg='Black', font=("", 18))
self.pickup.pack()
```

```
    self.pickupemt = Label(entry, bg='Light Grey')
    self.pickupemt.pack()

    self.drop = Entry(entry, bg='White', fg='Black', font=("", 18))
    self.drop.pack()

    self.dropemt = Label(entry, bg='Light Grey')
    self.dropemt.pack()

    details = Frame(mainf, bg='Light Grey')
    details.pack(side='right', padx=20, pady=20)

    label_frame = Frame(details, bg='Light Grey')
    label_frame.pack(side='left', padx=10)

    tile = Label(label_frame, text='Change Map', bg='Light Grey', fg='Black', font=("", 18))
    tile.pack()

    Label(label_frame, bg='Light Grey').pack()

    passlb = Label(label_frame, text='No Of Passengers', bg='Light Grey', fg='Black', font=("", 18))
    passlb.pack()

    Label(label_frame, bg='Light Grey').pack()

    pickdate = Label(label_frame, text='Pickup Date', bg='Light Grey', fg='Black', font=("", 18))
    pickdate.pack()

    Label(label_frame, bg='Light Grey').pack()

    picktime = Label(label_frame, text='Pickup Time', bg='Light Grey', fg='Black', font=("", 18))
    picktime.pack()

    Label(label_frame, bg='Light Grey').pack()

    total_cost = Label(label_frame, text='Total Cost', bg='Light Grey', fg='Black', font=("", 18))
    total_cost.pack()
```

```
Label(label_frame, bg='Light Grey').pack()

entry_frame = Frame(details, bg='Light Grey')
entry_frame.pack(side='right', padx=10)

self.map_type = StringVar()
map_types = ['Google Map', 'Google satellite', 'Open Street Map']
self.__choice = Combobox(entry_frame, textvariable=self.map_type, values=map_types, state='readonly',
                         justify='center')
self.__choice.set('Google Map')
self.__choice.pack()
self.__choice.bind("<<ComboboxSelected>>", self.map_change)

Label(entry_frame, bg='Light Grey').pack()

self.__no = list((range(1, 5)))
self.__noofpass = IntVar()
self.__passanger = Spinbox(entry_frame, values=self.__no, state='readonly', justify='center', wrap=True,
                           textvariable=self.__noofpass)
self.__passanger.pack()

Label(entry_frame, bg='Light Grey').pack()

datef = Frame(entry_frame, bg='Light Grey')
datef.pack()

now = date.today() + timedelta(days=1)
self.pd = StringVar()
self.pickdate = Entry(datef, state='readonly', readonlybackground='White', fg='Black', textvariable=self.pd)
self.pd.set(str(now))
self.pickdate.pack(side='left')

self.showcale = PhotoImage(file='Resource/calendar.png')
self.__calendar = Button(datef, image=self.showcale, command=self.showcalendar, bd=0)
self.__calendar.pack()
```

```
    self.datemt = Label(entry_frame, bg='Light Grey')
    self.datemt.pack()

    timef = Frame(entry_frame, bg='Light Grey')
    timef.pack()

    hrs = list(range(1, 13))
    self.__hour = StringVar()
    self.hour = Spinbox(timef, values=hrs, textvariable=self.__hour, justify='center', width=5, wrap=True,
                        bg='Light Grey', fg='Black')
    self.__hour.set("6")
    self.hour.pack(side='left')

    Label(timef, text=':', bg='Light Grey', fg='Black', font=('', 18)).pack(side='left', padx=2)

    mins = list(range(1, 60))
    mins.insert(0, '00')
    self.__minute = StringVar()
    self.minute = Spinbox(timef, values=mins, textvariable=self.__minute, justify='center', width=5, wrap=True,
                          bg='Light Grey', fg='Black')
    self.minute.pack(side='left')

    apm = list(("AM", "PM"))
    self.__apm = StringVar()
    self.apm = Spinbox(timef, values=apm, textvariable=self.__apm, state='readonly', justify='center', width=3,
                       wrap=True, bg='Black', fg='Light Grey')
    self.apm.pack(side='left', padx=2)

    self.timemt = Label(entry_frame, bg='Light Grey')
    self.timemt.pack()

    costf = Frame(entry_frame, bg="Light Grey")
    costf.pack()

    Label(costf, text="Rs.", bg='Light Grey', fg='Black').pack(side='left')

    Label(costf, text=":", bg='Light Grey', fg='Black').pack(side='left', padx=2)
```

```
    self.__cost = Label(costf, text='*****', bg='Light Grey', fg='Black')
    self.__cost.pack(side='left')

    Label(entry_frame, bg='Light Grey').pack()

    back = Button(label_frame, text='Back', command=self.call, bg='Light Grey',
                  fg='Black', bd=0)
    back.pack()

    bookbtn = Button(entry_frame, text='Book', bg="Light Grey", fg="BLack", bd=0, command=self.book)
    bookbtn.pack()

    frame = Frame(mainf, bg="Light Grey", highlightbackground="BLack", highlightthickness=2)
    frame.pack(padx=15)

    mapf = Frame(frame, bg='Light Grey')
    mapf.pack(fill='both', anchor='center')

    # adding map
    self.map = TkinterMapView(mapf, width=1000, height=700, corner_radius=0)

    # setting tile to google map
    self.map.set_tile_server("https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",
                             max_zoom=22)  # Google Normal Map

    # setting opening address
    self.map.set_position(27.6845, 85.3170)  # PCPS, Kupandole
    self.map.set_zoom(18)  # setting zoom level to 18
    self.map.pack()
    self.map.add_left_click_map_command(self.address)  # binding event

# changing map tile
def map_change(self, _):
    if self.map_type.get() == 'Open Street Map':
        self.map.set_tile_server("https://a.tile.openstreetmap.org/{z}/{x}/{y}.png")  # OpenStreetMap (default)
    elif self.map_type.get() == 'Google Map':
```

```
        self.map.set_tile_server("https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",
                                 max_zoom=22)  # Google Normal Map
    else:
        self.map.set_tile_server("https://mt0.google.com/vt/lyrs=s&hl=en&x={x}&y={y}&z={z}&s=Ga",
                                 max_zoom=22)  # google satellite

# converting coordinates to address
def address(self, cords):
    adr = convert_coordinates_to_address(cords[0], cords[1])
    if not self.pickup.get():
        self.cord1 = cords
        if self.mark1:
            self.mark1.delete()
            self.path1.delete()
        if adr.street:
            ls = adr.street + ',' + adr.city
            self.pickup.insert(0, ls)
        else:
            ls = adr.latlng
            self.pickup.insert(0, ls)
        self.marker()
    elif not self.drop.get():
        self.cord2 = cords
        if self.mark2:
            self.mark2.delete()
            self.path1.delete()
        if adr.street:
            ls = adr.street + ',' + adr.city
            self.drop.insert(0, ls)
        else:
            ls = adr.latlng
            self.drop.insert(0, ls)
        self.marker()

# marking the location entered
def pick_up(self):
    if not self.pickup.get():
```

```
        self.pickupemt.config(text='Enter A Location', font=('', 10), fg='Red')
    else:
        self.pickupemt.config(text='')
        self.cord1 = convert_address_to_coordinates(self.pickup.get())
        if self.cord1:
            cords = self.cord1
            self.map.set_position(cords[0], cords[1])
            self.map.set_zoom(18)
            if self.mark1:
                self.mark1.delete()
                if self.path1:
                    self.path1.delete()
            self.marker()
        else:
            self.pickupemt.config(text='Location Can Not Be Found.', font=('', 10), fg='Red')

# marking the location entered
def drop_loc(self):
    if not self.drop.get():
        self.dropemt.config(text='Enter A Location', font=('', 10), fg='Red')
    else:
        self.dropemt.config(text='')
        self.cord2 = convert_address_to_coordinates(self.drop.get())
        if self.cord2:
            cords = self.cord2
            self.map.set_position(cords[0], cords[1])
            self.map.set_zoom(18)
            if self.mark2:
                self.mark2.delete()
                if self.path1:
                    self.path1.delete()
            self.marker()
        else:
            self.dropemt.config(text='Location Can Not Be Found.', font=('', 10), fg='Red')

# making marker
def marker(self):
```

```
        if self.pickup.get():
            cords = self.cord1
            self.mark1 = self.map.set_marker(cords[0], cords[1])
        if self.drop.get():
            cords = self.cord2
            self.mark2 = self.map.set_marker(cords[0], cords[1])
        if self.pickup.get() and self.drop.get():
            self.path()

    # creating path when there is two markers
    def path(self):
        if self.cord1 and self.cord2:
            self.path1 = self.map.set_path([self.cord1, self.cord2])
            self.distance = geodesic(self.cord1, self.cord2).km
            self.cost()

    # calculating the total cost
    def cost(self):
        mindistance = 2
        mincost = 150
        price = 0
        if self.distance == mindistance:
            price = mincost
        elif mindistance < self.distance <= 10:
            price = mincost + (self.distance - mindistance) * 30
        elif self.distance <= 20:
            price = mincost + (self.distance - mindistance) * 50
        elif self.distance > 20:
            price = mincost + (self.distance - mindistance) * 70
        format_float = "{:.2f}".format(price)
        self.__cost.config(text=format_float)

    # showing calendar and filling the entry with date
    def showcalendar(self):
        tp = Toplevel()

        # always top
```

```
        tp.transient(self.__window)
        tp.grab_set()
        self.create(tp)

    # creating calendar
    def create(self, top):
        self.top = top
        self.top.overrideredirect(1)  # Remove border
        self.top.geometry('210x135+1195+532')

        mainf = Frame(self.top, bg='Light Grey')
        mainf.pack(fill='both', expand=True)

        min = date.today() + timedelta(days=1)
        max = date.today() + timedelta(days=15)
        self.cal = Calendar(mainf, selectmode='day', date_pattern='yyy-mm-dd', mindate=min, maxdate=max,
                            showweeknumbers=False, weekendforeground='Red', normalforeground='Black')
        self.cal.bind('<<CalendarSelected>>', self.asd)
        self.cal.pack(fill='both')

    # selecting day and destroying the toplevel
    def asd(self, event):
        self.__ndate = self.cal.get_date()
        self.pd.set(str(self.__ndate))
        self.top.destroy()

    # calling customer dashboard
    def call(self):
        self.__controller.custlog(self.__custid)

    # validating and booking a trip
    def book(self):
        a = self.pickuplocation()
        b = self.droplocation()
        c = self.picktime()
        d = self.totalcost()
        if a and b and c and d:
```

```
        self.__mod.setcustid(self.__custid)
        self.__mod.setpassno(self.__passanger.get())
        self.__mod.settripdate(self.pickdate.get())
        if self.__mod.booktrip():
            showinfo('message', "Trip booked")
            self.call()

    def pickuplocation(self):
        if not self.pickup.get():
            self.pickupemt.config(text='Location Can Not Be Empty.', font=('', 10), fg='Red')
            return False
        else:
            self.pickupemt.config(text='')
            self.__mod.setpickloc(self.pickup.get())
            return True

    def droplocation(self):
        if not self.drop.get():
            self.dropemt.config(text='Location Can Not Be Empty.', font=('', 10), fg='Red')
            return False
        else:
            self.dropemt.config(text='')
            self.__mod.setdroploc(self.drop.get())
            return True

    def picktime(self):
        if not re.match(self.__numregex, self.__hour.get()):
            self.timemt.config(text='Invalid Input', font=('', 10), fg='Red')
            return False
        elif int(self.__hour.get()) > 12 or int(self.__hour.get()) < 1:
            self.timemt.config(text='Invalid Input', font=('', 10), fg='Red')
            return False
        elif not re.match(self.__numregex, self.__minute.get()):
            self.timemt.config(text='Invalid Input', font=('', 10), fg='Red')
            return False
        elif int(self.__minute.get()) >= 60 or int(self.__minute.get()) < 0:
            self.timemt.config(text='Invalid Input', font=('', 10), fg='Red')
```

```
            return False
        else:
            self.timemt.config(text='')
            time = self.__hour.get() + " : " + self.__minute.get() + " " + self.__apm.get()
            self.__mod.setpicktime(time)
            return True

    def totalcost(self):
        cost = self.__cost.cget("text")
        if cost == '*****':
            showerror('Invalid', "Path Not Marked")
            return False
        else:
            format_float = "{:.2f}".format(self.distance)
            self.__mod.setdistance(format_float)
            self.__mod.setcost(cost)
            return True
```

```
"""
    Module View
    This Is A Frontend Which User Sees And Input Data
"""
from datetime import date
# importing required modules
from tkinter import *
from tkinter import ttk
from tkinter.messagebox import askyesno, showinfo, showerror
from Model.TripModel import TripModel


# creating class Driver Home
# this is driver home
class DriverHome(ttk.Frame):

    # creating frame for admin home
    def __init__(self, window, controller, did):
```

```
        self.__did = did
        self.tripdata = None
        self.__window = window
        self.__window.deiconify()
        self.__tripmod = TripModel()
        self.__controller = controller
        super().__init__(self.__window)
        self.__window.geometry("1920x1080")
        self.__window.title("Admin Home")

        mainf = Frame(self.__window, bg="Light Grey")
        mainf.pack(expand=True, fill='both')

        topf = Frame(mainf, bg='Light Grey')
        topf.pack(side='top', fill='both', pady=20)

        self.logo = PhotoImage(file='Resource/logo.png')
        self.show = Label(topf, image=self.logo, bd=0)
        self.show.pack(side='left', padx=10)

        logout = Button(topf, text='Log Out', bg='Light Grey', fg='Black', bd=0, command=self.confirm)
        logout.pack(side='right', padx=10, pady=10)

        titlelb = Label(topf, text="Driver's Dashboard", bg='Light Grey', fg='Black', font=('', 30, "bold"))
        titlelb.pack(side='bottom', pady=10)

        self.table = Frame(mainf, bg='Light Grey')
        self.table.pack(expand=True, pady=40)

        style = ttk.Style(self.table)
        # set ttk theme to "clam" which support the field background option
        style.theme_use("clam")
        style.configure("Treeview", background="White", foreground="Black")

        self.scrollbar = Scrollbar(self.table)
        self.scrollbar.pack(side='right', fill='y')
```

```
    self.triptable()

# creating table and adding
def triptable(self):

    self.tripdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
    self.tripdata.pack()

    self.scrollbar.config(command=self.tripdata.yview)

    # defining columns
    self.tripdata["columns"] = ("Trip ID", "Customer Name", "Customer Contact", "Pick Up Location", "Destination",
                                "No Of Passenger", "Departure Date", "Departure Time", "Distance", "Cost"
                                , "Status")

    # formatting columns
    self.tripdata.column("#0", width=0, stretch=NO)
    self.tripdata.column("Trip ID", width=80, minwidth=50, anchor=W)
    self.tripdata.column("Customer Name", width=180, minwidth=90, anchor=W)
    self.tripdata.column("Customer Contact", width=180, minwidth=90, anchor=W)
    self.tripdata.column("Pick Up Location", width=180, minwidth=90, anchor=W)
    self.tripdata.column("Destination", width=180, minwidth=90, anchor=W)
    self.tripdata.column("No Of Passenger", width=100, minwidth=90, anchor=W)
    self.tripdata.column("Departure Date", width=100, minwidth=90, anchor=W)
    self.tripdata.column("Departure Time", width=100, minwidth=90, anchor=W)
    self.tripdata.column("Distance", width=100, minwidth=90, anchor=W)
    self.tripdata.column("Cost", width=100, minwidth=90, anchor=W)
    self.tripdata.column("Status", width=100, minwidth=90, anchor=W)

    # creating heading
    self.tripdata.heading("#0", text='Label')
    self.tripdata.heading("Trip ID", text="Trip Id")
    self.tripdata.heading("Customer Name", text="Customer Name")
    self.tripdata.heading("Customer Contact", text="Customer Contact")
    self.tripdata.heading("Pick Up Location", text="Pick Up Location")
    self.tripdata.heading("Destination", text="Destination")
    self.tripdata.heading("No Of Passenger", text="No Of Passenger")
```

```
        self.tripdata.heading("Departure Date", text="Departure Date")
        self.tripdata.heading("Departure Time", text="Departure Time")
        self.tripdata.heading("Distance", text="Distance")
        self.tripdata.heading("Cost", text="Cost")
        self.tripdata.heading("Status", text="Status")

        # reading data from table
        record = self.__tripmod.driverti(self.__did)
        if record:

            # inserting data into table
            for data in record:
                name = data[2] + " " + data[1]
                self.tripdata.insert("", index="end", values=(data[9], name, data[6], data[11], data[12], data[15],
                                                              data[13], data[14], data[16], data[17], data[18]))

            self.tripdata.bind('<Double-1>', self.data)

    # creating top level for specific trip
    def data(self, event):
        tripd = None
        try:
            if self.tripdata.selection()[0]:
                for data in self.tripdata.selection():
                    value = self.tripdata.item(data)
                    tripd = value['values']
                tp = Toplevel()

                # always top
                tp.transient(self.__window)
                tp.grab_set()

                self.details(tp, tripd)
        except Exception as e:
            showerror("Invalid", "Select A Data")
            print(e)
```

```
def details(self, tp, tripd):
    top = tp
    top.geometry('670x380+350+235')
    top.title('Trip Details')

    mainf = Frame(top, bg='White')
    mainf.pack(fill='both', expand=True)

    titlef = Frame(mainf, bg='White')
    titlef.pack(pady=10)

    title = Label(titlef, text='Trip Detail', bg='White', fg='Black', font=('EB Garamond', 25, 'bold'))
    title.pack(anchor='center')

    lbdataf = Frame(mainf, bg='White')
    lbdataf.pack(pady=10)

    leftf = Frame(lbdataf, bg='White')
    leftf.pack(side='left', padx=10, pady=10)

    leftlbf = Frame(leftf, bg='White')
    leftlbf.pack(side='left', padx=10)

    namelb = Label(leftlbf, text='Customer Name :', bg='White', fg='Black', font=('Crimson Text', 18))
    namelb.pack(pady=10)

    pickloc = Label(leftlbf, text='Pickup        : ', bg='White', fg='Black', font=('Crimson Text', 18))
    pickloc.pack(pady=10)

    datelb = Label(leftlbf, text='Date          : ', bg='White', fg='Black', font=('Crimson Text', 18))
    datelb.pack(pady=10)

    passengerlb = Label(leftlbf, text='Passenger     : ', bg='White', fg='Black', font=('Crimson Text', 18))
    passengerlb.pack(pady=10)

    costlb = Label(leftlbf, text="Cost          : ", bg='White', fg='Black', font=('Crimson Text', 18))
    costlb.pack(pady=10)
```

```
leftdataf = Frame(leftf, bg='White')
leftdataf.pack(side='right', padx=10)

namedata = Label(leftdataf, text=tripd[1], bg='White', fg='Black', font=('Crimson Text', 16))
namedata.pack(pady=10)

picklocdata = Label(leftdataf, text=tripd[3], bg='White', fg='Black', font=('Crimson Text', 16))
picklocdata.pack(pady=10)

datedata = Label(leftdataf, text=tripd[6], bg='White', fg='Black', font=('Crimson Text', 16))
datedata.pack(pady=10)

passegerdata = Label(leftdataf, text=tripd[5], bg='White', fg='Black', font=('Crimson Text', 16))
passegerdata.pack(pady=10)

costdata = Label(leftdataf, text=tripd[9], bg='White', fg='Black', font=('Crimson Text', 16))
costdata.pack(pady=10)

rightf = Frame(lbdataf, bg='White')
rightf.pack(side='right', padx=10, pady=10)

rightlbf = Frame(rightf, bg='White')
rightlbf.pack(side='left', padx=10)

conlb = Label(rightlbf, text="Customer Number : ", bg='White', fg='Black', font=('Crimson Text', 18))
conlb.pack(pady=10)

droploc = Label(rightlbf, text='Destination     : ', bg='White', fg='Black', font=('Crimson Text', 18))
droploc.pack(pady=10)

timelb = Label(rightlbf, text='Time           : ', bg='White', fg='Black', font=('Crimson Text', 18))
timelb.pack(pady=10)

distlb = Label(rightlbf, text="Distance       : ", bg='White', fg='Black', font=('Crimson Text', 18))
distlb.pack(pady=10)
```

```
    statuslb = Label(rightlbf, text='Status        : ', bg='White', fg='Black', font=('Crimson Text', 18))
    statuslb.pack(pady=10)

    rightdataf = Frame(rightf, bg='White')
    rightdataf.pack(side='right', padx=10)

    condata = Label(rightdataf, text=tripd[2], bg='White', fg='Black', font=('Crimson Text', 16))
    condata.pack(pady=10)

    droplocdata = Label(rightdataf, text=tripd[4], bg='White', fg='Black', font=('Crimson Text', 16))
    droplocdata.pack(pady=10)

    timedata = Label(rightdataf, text=tripd[7], bg='White', fg='Black', font=('Crimson Text', 16))
    timedata.pack(pady=10)

    distdata = Label(rightdataf, text=tripd[8], bg='White', fg='Black', font=('Crimson Text', 16))
    distdata.pack(pady=10)

    statusdata = Label(rightdataf, text=tripd[10], bg='White', fg='Black', font=('Crimson Text', 16))
    statusdata.pack(pady=10)

    btnf = Frame(mainf, bg='White')
    btnf.pack(pady=10)

    start = None
    complete = None
    now = date.today()

    if tripd[10] == "Confirmed":
        if complete:
            complete.destroy()
        start = Button(btnf, text='Start The Trip', bg='Light Grey', fg="Black", bd=0, command=lambda: startrip())
        start.pack()
    elif tripd[10] == "Started":
        if start:
            start.destroy()
        complete = Button(btnf, text='Complete', bg='Light Grey', fg='Black', bd=0, command=lambda: completetrip())
```

```
        complete.pack()
    else:
        if complete:
            complete.destroy()
        if start:
            start.destroy()

    # starting trip
    def startrip():
        if tripd[6] == now:
            if self.__tripmod.startrip(tripd[0]):
                showinfo("Message", "Trip Started")
                top.destroy()
                self.tripdata.destroy()
                self.triptable()
        else:
            showerror("Invalid", "Trip Can Not Be Started Now.")

    def completetrip():
        if self.__tripmod.completetrip(tripd[0]):
            showinfo("Message", "Trip Completed")
            top.destroy()
            self.tripdata.destroy()
            self.triptable()

# confirming log out
def confirm(self):
    ans = askyesno("Conformation", "Are You Sure You Want To Log Out?")
    if ans:
        self.__controller.log()
```

```
"""
    Module View
    This Is A Frontend Which User Sees And Input Data
"""
```

```
# importing required modules
import re
from tkinter import *
from Model.LoginModel import LoginModel
from tkinter.messagebox import showinfo, showerror


# creating login page class
class LoginPage:

    # creating frames and placing in the tk
    def __init__(self, window, controller):

        self.__id = None
        self.__mod = LoginModel()
        self.__window = window
        self.__controller = controller
        self.__window.title("Login Page")
        self.__window.geometry('450x285+450+200')
        self.__emregex = ('^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$')  # regex for email only
        self.__passregex = ("^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$")  # regex for password only

        mainf = Frame(self.__window, bg="Light Grey")
        mainf.pack(fill="both", expand=True)

        title = Frame(mainf, bg="Light Grey")
        title.pack(expand=True)

        titlelb = Label(title, text="Welcome To S&S Taxi Service", font=("", 30, "bold"), bg="Light Grey", fg="Black")
        titlelb.pack(pady=20)

        lbtxt = Frame(mainf, bg="Light Grey")
        lbtxt.pack(expand=True, anchor="center")

        empw = Frame(lbtxt, bg="Light Grey")
        empw.pack(padx=20, side="left")
```

```
em = Label(empw, text="Email : ", font=("", 20, "bold"), bg="Light Grey", fg="Black")
em.pack()
Label(empw, bg="light Grey").pack()
pw = Label(empw, text="Password : ", font=("", 20, "bold"), bg="Light Grey", fg="Black")
pw.pack()
Label(empw, bg="light Grey").pack()

txtfield = Frame(lbtxt, bg="Light Grey")
txtfield.pack(padx=20, side="right")

self.emtext = Entry(txtfield, bg="White", fg="Black", font=("", 20))
self.emtext.insert(0, "Enter your email")
self.emtext.pack()
self.emtext.bind('<FocusIn>', self.clear_text)

self.emt = Label(txtfield, font=("", 10), bg="Light Grey", fg="Black")
self.emt.pack()

self.__pwtext = Entry(txtfield, bg="White", fg="Black", font=("", 20))
self.__pwtext.insert(0, "Enter your password")
self.__pwtext.pack()
self.__pwtext.bind('<FocusIn>', self.clear_text1)

self.emt1 = Label(txtfield, font=("", 10), bg="Light Grey", fg="Black")
self.emt1.pack()

log = Frame(mainf, bg="Light Grey")
log.pack(expand=True)

self.checkvar = IntVar()
checkbox = Checkbutton(log, text="Show Password", variable=self.checkvar, onvalue=1, offvalue=0,
                       bg="Light Grey", fg="Black")
checkbox.pack(pady=10)
checkbox.bind('<Button-1>', self.showpass)

btn = Frame(log, bg="Light Grey")
btn.pack()
```

```
        admin = Button(btn, text="Login as Admin", bg="Light Grey", command=self.__controller.admin, bd=0)
        admin.pack(padx=30, side="left")

        signup = Button(btn, text="Register Now", bg="Light Grey", command=controller.reg, bd=0)
        signup.pack(padx=10, side="right")

        nry = Label(btn, text="Not Registered Yet", bg="Light Grey", fg="Black")
        nry.pack(side="right")

        ad = Frame(mainf, bg="Light Grey")
        ad.pack(expand=True)

        logb = Button(ad, text='Login', bg="Light Grey", command=self.verify, bd=0)
        logb.pack()

        self.__window.bind('<Return>', self.callverify)

    # calling verify
    def callverify(self, _):
        self.verify()

    # verifying the data entered
    def verify(self):
        a = self.em()
        b = self.pas()
        if a and b:
            custid = self.__mod.cust()
            did = self.__mod.driver()
            if custid:
                for data in custid:
                    self.__id = data[0]
                    fname = data[1]
                    lname = data[2]
                    message = "Welcome " + lname + " " + fname
                    showinfo("Message", message)
                self.__controller.custlog(self.__id)
```

```
        elif did:
            for data in did:
                self.__id = data[0]
                fname = data[1]
                lname = data[2]
                message = "Welcome " + lname + " " + fname
                showinfo("Message", message)
            self.__controller.driverdash(self.__id)
        else:
            showerror('Invalid', "Invalid Email or Password")

    # verifying the email
    def em(self):
        email = self.emtext.get().lower()
        if not self.emtext.get() or self.emtext.get() == "Enter your email":
            self.emt.config(text="Email Can't Be Empty", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__emregex, email):
            self.emt.config(text="Invalid Email", font=("", 10), fg="Red")
            return False
        else:
            self.emt.config(text="")
            self.__mod.setem(email)
            return True

    # verifying the password
    def pas(self):
        if not self.__pwtext.get() or self.__pwtext.get() == "Enter your password":
            self.emt1.config(text="Password Can't Be Empty", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__passregex, self.__pwtext.get()):
            self.emt1.config(text="Invalid Password", font=("", 10), fg="Red")
            return False
        else:
            self.emt1.config(text="")
            self.__mod.setpas(self.__pwtext.get())
            return True
```

```
    # clearing email textfield
    def clear_text(self, event):
        if self.emtext.get() == "Enter your email":
            self.emtext.delete(0, END)
        if not self.__pwtext.get():
            self.__pwtext.insert(0, "Enter your password")
            self.__pwtext.config(show="")

    # clearing password textfield
    def clear_text1(self, event):
        if self.__pwtext.get() == "Enter your password":
            self.__pwtext.delete(0, END)
            if not self.checkvar.get():
                self.__pwtext.config(show="*")
        if not self.emtext.get():
            self.emtext.insert(0, "Enter your email")

    # showing and hiding password
    def showpass(self, event):
        if self.__pwtext.get() != "Enter your password":
            if self.checkvar.get():
                self.__pwtext.config(show="*")
            else:
                self.__pwtext.config(show="")
        else:
            showinfo("Message", "Password not entered")
            if self.checkvar.get():
                self.checkvar.set(1)
            else:
                self.checkvar.set(0)
```

"""
Module View
This Is A Frontend Which User Sees And Input Data
"""

```
# importing required modules
import re
from tkinter import *
from tkinter import ttk
from datetime import datetime
from tkinter.ttk import Combobox
from Model.TripModel import TripModel
from Model.StaffModel import StaffModel
from Model.DriverModel import DriverModel
from Model.VehicleModel import VehicleModel
from Model.CustomerModel import CustomerModel
from Model.RegistrationModel import RegistrationModel
from tkinter.messagebox import askyesno, showinfo, showerror


# creating class Admin Home
# this is admin home
class StaffHome(ttk.Frame):

    # creating frame for staff home
    def __init__(self, window, controller, stid):

        self.__stid = stid
        self.__tripmod = TripModel()
        self.__custmod = CustomerModel()
        self.__stmod = StaffModel()
        self.__dmod = DriverModel()
        self.__vmod = VehicleModel()
        self.__window = window
        self.__window.deiconify()
        self.__controller = controller
        super().__init__(self.__window)
        self.__window.geometry("1920x1080")
        self.__window.title("Admin Home")
        self.pendingdata = None
        self.alltripdata = None
        self.allcustdata = None
        self.allstaffdata = None
        self.allvehicledata = None
        self.alldriverdata = None
        self.todaytripdata = None
        self.searchcusr = None

        mainf = Frame(self.__window, bg="Light Grey")
        mainf.pack(expand=True, fill='both')

        topf = Frame(mainf, bg='Light Grey')
```

```
topf.pack(side='top', pady=20, fill='both')

logout = Button(topf, text='Log Out', bg='Light Grey', fg='Black', bd=0, command=self.confirm)
logout.pack(side='right', padx=10)

self.logo = PhotoImage(file='Resource/logo.png')
self.show = Label(topf, image=self.logo, bd=0)
self.show.pack(side='left', padx=10)

title = Label(topf, text='Staff Dashboard', bg='Light Grey', fg='Black', font=("", 30))
title.pack()

searchcf = Frame(mainf, bg='Light Grey')
searchcf.pack(pady=30)

fnf = Frame(searchcf, bg='Light Grey')
fnf.pack(side='left')

fnlb = Label(fnf, text='Name : ', bg='Light Grey', fg='Black', font=("", 18))
fnlb.pack(side='left', padx=10)

self.nameentry = Entry(fnf, bg='White', fg='Black')
self.nameentry.pack(side='right')

emptf1 = Frame(searchcf, bg='Light Grey')
emptf1.pack(side='left')

Label(emptf1, bg='Light Grey').pack(padx=40)

lnf = Frame(searchcf, bg='Light Grey')
lnf.pack(side='left')

lnlb = Label(lnf, text='Address : ', bg='Light Grey', fg='Black', font=("", 18))
lnlb.pack(side='left', padx=10)

self.addentry = Entry(lnf, bg='White', fg='Black')
self.addentry.pack(side='right')

emptf2 = Frame(searchcf, bg='Light Grey')
emptf2.pack(side='left')

Label(emptf2, bg='Light Grey').pack(padx=40)

searchbtn = Button(searchcf, text="Search", bg='Light Grey', fg='Black', bd=0, command=self.searchc)
searchbtn.pack(side='left')

choosing = Frame(mainf, bg='Light Grey')
```

```
        choosing.pack(pady=30)

        tripf = Frame(choosing, bg='Light Grey')
        tripf.pack(side='left', padx=20)

        triplb = Label(tripf, text="Show Trip : ", bg='Light Grey', fg='Black', font=("", 18))
        triplb.pack(side='left', padx=10)

        tripvalue = ["All Pending", "Today's", "All"]
        self.selectedtrip = StringVar()
        choosetrip = Combobox(tripf, values=tripvalue, textvariable=self.selectedtrip, width=15,
justify='center',
                        state='readonly', font=("", 15))
        choosetrip.pack()
        self.selectedtrip.set("All Pending")
        choosetrip.bind("<<ComboboxSelected>>", self.showaboutrip)

        emptf2 = Frame(choosing, bg='Light Grey')
        emptf2.pack(side='left')

        Label(emptf2, bg='Light Grey').pack(padx=80)

        allf = Frame(choosing, bg='Light Grey')
        allf.pack(side='right', padx=20)

        alllb = Label(allf, text="Show All : ", bg='Light Grey', fg='Black', font=("", 18))
        alllb.pack(side='left')

        value = ["Vehicle", "Staff", "Driver", "Customer"]
        self.selectedall = StringVar()
        all = Combobox(allf, values=value, textvariable=self.selectedall, width=15, justify='center',
                state='readonly', font=(", 15))
        self.selectedall.set("None")
        all.pack(side='left', padx=10)
        all.bind("<<ComboboxSelected>>", self.showall)

        self.table = Frame(mainf, bg='Light Grey')
        self.table.pack(expand=True, pady=40)

        style = ttk.Style(self.table)
        # set ttk theme to "clam" which support the field background option
        style.theme_use("clam")
        style.configure("Treeview", background="White", foreground="Black")

        self.scrollbar = Scrollbar(self.table)
        self.scrollbar.pack(side='right', fill='y')
```

```
        self.pendingtable()

        adds = Frame(mainf, bg="Light Grey")
        adds.pack(side='left', expand=True, padx=20, pady=60)

        addst = Frame(adds, bg="Light Grey")
        addst.pack(anchor='center')

        addstaff = Button(addst, text="Add Staff", bg="light Grey", fg="Black", bd=0, font=("", 20),
                          command=self.addstaff)
        addstaff.pack()

        addV = Frame(mainf, bg="light Grey")
        addV.pack(side='left', padx=20, expand=True, pady=60)

        addv = Frame(addV, bg='LighT Grey')
        addv.pack(anchor='center')

        addvehicle = Button(addv, text="Add Vehicle", bg="light Grey", fg="Black", bd=0, font=("", 20),
                            command=self.addvehicle)
        addvehicle.pack()

        addD = Frame(mainf, bg="Light Grey")
        addD.pack(side='right', expand=True, pady=60)

        add = Frame(addD, bg="Light Grey")
        add.pack(anchor='center')

        addriver = Button(add, text="Add Driver", bg="light Grey", fg="Black", bd=0, font=("", 20),
                          command=self.addriver)
        addriver.pack()

    # pending trip table
    def pendingtable(self):

        self.pendingdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.pendingdata.pack()

        self.scrollbar.config(command=self.pendingdata.yview)

        # defining columns
        self.pendingdata["columns"] = ("Trip ID", "Customer Name", "Pick Up Location", "Destination", "No
Of Passenger",
                                       "Departure Date", "Departure Time", "Distance", "Cost", "Status")

        # formatting columns
        self.pendingdata.column("#0", width=0, stretch=NO)
```

```
        self.pendingdata.column("Trip ID", width=50, minwidth=40, anchor=W)
        self.pendingdata.column("Customer Name", width=130, minwidth=90, anchor=W)
        self.pendingdata.column("Pick Up Location", width=180, minwidth=90, anchor=W)
        self.pendingdata.column("Destination", width=180, minwidth=90, anchor=W)
        self.pendingdata.column("No Of Passenger", width=100, minwidth=90, anchor=W)
        self.pendingdata.column("Departure Date", width=180, minwidth=90, anchor=W)
        self.pendingdata.column("Departure Time", width=180, minwidth=90, anchor=W)
        self.pendingdata.column("Distance", width=80, minwidth=50, anchor=W)
        self.pendingdata.column("Cost", width=80, minwidth=50, anchor=W)
        self.pendingdata.column("Status", width=180, minwidth=90, anchor=W)

        # creating heading
        self.pendingdata.heading("#0", text='Label')
        self.pendingdata.heading("Trip ID", text="Trip Id")
        self.pendingdata.heading("Customer Name", text="Customer Name")
        self.pendingdata.heading("Pick Up Location", text="Pick Up Location")
        self.pendingdata.heading("Destination", text="Destination")
        self.pendingdata.heading("No Of Passenger", text="No Of Passenger")
        self.pendingdata.heading("Departure Date", text="Departure Date")
        self.pendingdata.heading("Departure Time", text="Departure Time")
        self.pendingdata.heading("Distance", text="Distance")
        self.pendingdata.heading("Cost", text="Cost")
        self.pendingdata.heading("Status", text="Status")

        # reading data from table
        record = self.__tripmod.pendingtrip()
        if record:

            # inserting data into table
            for data in record:
                name = data[17] + " " + data[16]
                self.pendingdata.insert("", index="end",
                                        values=(data[0], name, data[2], data[3], data[6], data[4], data[5], data[7],
                                                data[8], data[9]))

            self.pendingdata.bind('<Double-1>', self.minipendingtable)

    # creating toplevel
    def minipendingtable(self, event):
        tripd = None
        try:
            if self.pendingdata.selection()[0]:
                for data in self.pendingdata.selection():
                    value = self.pendingdata.item(data)
                    tripd = value['values']
                tp = Toplevel()
```

```
            # always top
            tp.transient(self.__window)
            tp.grab_set()
            self.pendingdetails(tp, tripd)
        except Exception as e:
            showinfo("Message", "Select A Data")
            print(e)

    # full detailed
    def pendingdetails(self, tp, tripd):
        top = tp
        top.geometry("560x480+400+200")
        top.title("Trip Details")

        mainf = Frame(top, bg='White')
        mainf.pack(fill='both', expand=True)

        titlef = Frame(mainf, bg="White")
        titlef.pack(pady=20)

        titlelb = Label(titlef, text='Trip Details', bg='White', fg='Black', font=("", 25, 'bold'))
        titlelb.pack()

        custnamef = Frame(mainf, bg='White')
        custnamef.pack(pady=15)

        custnamelb = Label(custnamef, text='Customer Name : ', bg='White', fg='Black', font=("", 18))
        custnamelb.pack(side='left', padx=10)

        custnamedata = Label(custnamef, text=tripd[1], bg='White', fg='Black', font=("", 15))
        custnamedata.pack(side='left')

        locationf = Frame(mainf, bg='White')
        locationf.pack(pady=15)

        pickf = Frame(locationf, bg='White')
        pickf.pack(side='left', padx=20)

        picklb = Label(pickf, text='Pickup Location', bg='White', fg='Black', font=("", 18))
        picklb.pack(side='left', padx=10)

        pickdata = Label(pickf, text=tripd[2], bg='White', fg='Black', font=("", 15))
        pickdata.pack(side='left')

        dropf = Frame(locationf, bg='White')
        dropf.pack(side='right', padx=20)
```

```
droplb = Label(dropf, text='Destination : ', bg='White', fg='Black', font=("", 18))
droplb.pack(side='left')

dropdata = Label(dropf, text=tripd[3], bg='White', fg='Black', font=("", 15))
dropdata.pack(side='left', padx=10)

datime = Frame(mainf, bg='White')
datime.pack(pady=15)

datef = Frame(datime, bg='White')
datef.pack(side='left', padx=20)

datelb = Label(datef, text='Trip Date : ', bg='White', fg='Black', font=("", 18))
datelb.pack(side='left', padx=10)

datedata = Label(datef, text=tripd[5], bg='White', fg='Black', font=("", 15))
datedata.pack(side='left')

timef = Frame(datime, bg='White')
timef.pack(side='right', padx=20)

timelb = Label(timef, text='Pick Up : ', bg='White', fg='Black', font=("", 18))
timelb.pack(side='left')

timedata = Label(timef, text=tripd[6], bg='White', fg='Black', font=("", 15))
timedata.pack(side='left', padx=10)

passtatus = Frame(mainf, bg='White')
passtatus.pack(pady=15)

passf = Frame(passtatus, bg='White')
passf.pack(side='left', padx=20)

passlb = Label(passf, text='No Of Passenger : ', bg='White', fg='Black', font=("", 18))
passlb.pack(side='left', padx=10)

passdata = Label(passf, text=tripd[4], fg='Black', bg='White', font=("", 15))
passdata.pack(side='left')

statusf = Frame(passtatus, bg='White')
statusf.pack(side='right', padx=20)

statuslb = Label(statusf, text='Status : ', bg='White', fg='Black', font=("", 18))
statuslb.pack(side='left')

statusdata = Label(statusf, text=tripd[9], bg='White', fg='Black', font=("", 15))
statusdata.pack(side='left', padx=10)
```

```
distcost = Frame(mainf, bg='White')
distcost.pack(pady=15)

distf = Frame(distcost, bg='White')
distf.pack(side='left', padx=20)

distlb = Label(distf, text="Distance : ", bg='White', fg='Black', font=("", 18))
distlb.pack(side='left', padx=10)

distdata = Label(distf, text=tripd[7], bg='White', fg='Black', font=("", 15))
distdata.pack(side='left')

costf = Frame(distcost, bg='White')
costf.pack(side='left', padx=20)

costlb = Label(costf, text='Cost : ', bg='White', fg='Black', font=("", 18))
costlb.pack(side='left')

costdata = Label(costf, text=tripd[8], bg='White', fg='Black', font=("", 15))
costdata.pack(side='left', padx=10)

assignf = Frame(mainf, bg='White')
assignf.pack(pady=15)

assignlb = Label(assignf, text='Assign Driver', bg='White', fg='Black', font=("", 18))
assignlb.pack(side='left', padx=20)

named = []
record = self.__dmod.emptydriver(tripd[5])
if record:
    for data in record:
        name = data[2] + " " + data[1]
        named.insert(0, name)
if not named:
    named.insert(0, "No Driver Available")
driver = StringVar()
driver.set("Driver's Name")
assign = ttk.Combobox(assignf, values=named, textvariable=driver, font=("", 15), state='readonly')
assign.pack(side='left', padx=10)

btn = Frame(mainf, bg='White')
btn.pack(pady=10)

assignbtn = Button(btn, text='Confirm Assign', bd=0, command=lambda: assignd())
assignbtn.pack()
```

```
        def assignd():
            did = None
            if not driver.get() or driver.get() == "Driver's Name" or driver.get() == "No Driver Available":
                showerror('Invalid', "Select A Driver")
            else:
                driverids = self.__dmod.getid(driver.get())
                for data in driverids:
                    did = data[0]
                if self.__tripmod.assigndr(self.__stid, did, tripd[0]):
                    showinfo("Message", "Driver Assigned Successfully.")
                    top.destroy()
                    self.pendingdata.destroy()
                    self.pendingtable()

    # all trip table
    def alltriptable(self):

        self.alltripdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.alltripdata.pack()

        self.scrollbar.config(command=self.alltripdata.yview)

        # defining columns
        self.alltripdata["columns"] = ("Trip ID", "Customer Name", "Pick Up Location", "Destination", "No Of
Passenger",
                                       "Departure Date", "Departure Time", "Distance", "Cost", "Status", "Driver Id")

        # formatting columns
        self.alltripdata.column("#0", width=0, stretch=NO)
        self.alltripdata.column("Trip ID", width=50, minwidth=40, anchor=W)
        self.alltripdata.column("Customer Name", width=130, minwidth=90, anchor=W)
        self.alltripdata.column("Pick Up Location", width=160, minwidth=90, anchor=W)
        self.alltripdata.column("Destination", width=160, minwidth=90, anchor=W)
        self.alltripdata.column("No Of Passenger", width=100, minwidth=90, anchor=W)
        self.alltripdata.column("Departure Date", width=160, minwidth=90, anchor=W)
        self.alltripdata.column("Departure Time", width=160, minwidth=90, anchor=W)
        self.alltripdata.column("Distance", width=80, minwidth=50, anchor=W)
        self.alltripdata.column("Cost", width=80, minwidth=50, anchor=W)
        self.alltripdata.column("Status", width=180, minwidth=90, anchor=W)
        self.alltripdata.column("Driver Id", width=80, minwidth=50, anchor=W)

        # creating heading
        self.alltripdata.heading("#0", text='Label')
        self.alltripdata.heading("Trip ID", text="Trip Id")
        self.alltripdata.heading("Customer Name", text="Customer Name")
        self.alltripdata.heading("Pick Up Location", text="Pick Up Location")
        self.alltripdata.heading("Destination", text="Destination")
```

```
        self.alltripdata.heading("No Of Passenger", text="No Of Passenger")
        self.alltripdata.heading("Departure Date", text="Departure Date")
        self.alltripdata.heading("Departure Time", text="Departure Time")
        self.alltripdata.heading("Distance", text="Distance")
        self.alltripdata.heading("Cost", text="Cost")
        self.alltripdata.heading("Status", text="Status")
        self.alltripdata.heading("Driver Id", text="Driver Id")

        # reading data from table
        record = self.__tripmod.alltrip()
        if record:

            # inserting data into table
            for data in record:
                name = data[17] + " " + data[16]
                self.alltripdata.insert("", index="end",
                                        values=(data[0], name, data[2], data[3], data[6], data[4], data[5], data[7],
                                                data[8], data[9], data[14]))

        self.alltripdata.bind('<Double-1>', self.minialltrip)

    # creating toplevel
    def minialltrip(self, event):
        tripd = None
        try:
            if self.alltripdata.selection()[0]:
                for data in self.alltripdata.selection():
                    value = self.alltripdata.item(data)
                    tripd = value['values']
                tp = Toplevel()

                # always top
                tp.transient(self.__window)
                tp.grab_set()
                self.alltripdetails(tp, tripd)
        except Exception as e:
            showinfo("Message", "Select A Data")
            print(e)

    # full detailed
    def alltripdetails(self, tp, tripd):
        top = tp
        top.title("Trip Details")
        top.geometry("560x480+400+200")

        mainf = Frame(top, bg='White')
        mainf.pack(fill='both', expand=True)
```

```
titlef = Frame(mainf, bg="White")
titlef.pack(pady=20)

titlelb = Label(titlef, text='Trip Details', bg='White', fg='Black', font=("", 25, 'bold'))
titlelb.pack()

custnamef = Frame(mainf, bg='White')
custnamef.pack(pady=15)

custnamelb = Label(custnamef, text='Customer Name : ', bg='White', fg='Black', font=("", 18))
custnamelb.pack(side='left', padx=10)

custnamedata = Label(custnamef, text=tripd[1], bg='White', fg='Black', font=("", 15))
custnamedata.pack(side='left')

locationf = Frame(mainf, bg='White')
locationf.pack(pady=15)

pickf = Frame(locationf, bg='White')
pickf.pack(side='left', padx=20)

picklb = Label(pickf, text='Pickup Location', bg='White', fg='Black', font=("", 18))
picklb.pack(side='left', padx=10)

pickdata = Label(pickf, text=tripd[2], bg='White', fg='Black', font=("", 15))
pickdata.pack(side='left')

dropf = Frame(locationf, bg='White')
dropf.pack(side='right', padx=20)

droplb = Label(dropf, text='Destination : ', bg='White', fg='Black', font=("", 18))
droplb.pack(side='left')

dropdata = Label(dropf, text=tripd[3], bg='White', fg='Black', font=("", 15))
dropdata.pack(side='left', padx=10)

datime = Frame(mainf, bg='White')
datime.pack(pady=15)

datef = Frame(datime, bg='White')
datef.pack(side='left', padx=20)

datelb = Label(datef, text='Trip Date : ', bg='White', fg='Black', font=("", 18))
datelb.pack(side='left', padx=10)

datedata = Label(datef, text=tripd[5], bg='White', fg='Black', font=("", 15))
```

```
datedata.pack(side='left')

timef = Frame(datime, bg='White')
timef.pack(side='right', padx=20)

timelb = Label(timef, text='Pick Up : ', bg='White', fg='Black', font=("", 18))
timelb.pack(side='left')

timedata = Label(timef, text=tripd[6], bg='White', fg='Black', font=("", 15))
timedata.pack(side='left', padx=10)

passtatus = Frame(mainf, bg='White')
passtatus.pack(pady=15)

passf = Frame(passtatus, bg='White')
passf.pack(side='left', padx=20)

passlb = Label(passf, text='No Of Passenger : ', bg='White', fg='Black', font=("", 18))
passlb.pack(side='left', padx=10)

passdata = Label(passf, text=tripd[4], fg='Black', bg='White', font=("", 15))
passdata.pack(side='left')

statusf = Frame(passtatus, bg='White')
statusf.pack(side='right', padx=20)

statuslb = Label(statusf, text='Status : ', bg='White', fg='Black', font=("", 18))
statuslb.pack(side='left')

statusdata = Label(statusf, text=tripd[9], bg='White', fg='Black', font=("", 15))
statusdata.pack(side='left', padx=10)

distcost = Frame(mainf, bg='White')
distcost.pack(pady=15)

distf = Frame(distcost, bg='White')
distf.pack(side='left', padx=20)

distlb = Label(distf, text="Distance : ", bg='White', fg='Black', font=("", 18))
distlb.pack(side='left', padx=10)

distdata = Label(distf, text=tripd[7], bg='White', fg='Black', font=("", 15))
distdata.pack(side='left')

costf = Frame(distcost, bg='White')
costf.pack(side='left', padx=20)
```

```
    costlb = Label(costf, text='Cost : ', bg='White', fg='Black', font=("", 18))
    costlb.pack(side='left')

    costdata = Label(costf, text=tripd[8], bg='White', fg='Black', font=("", 15))
    costdata.pack(side='left', padx=10)
    assignf = None
    if tripd[9] == "Pending":
        assignf = Frame(mainf, bg='White')
        assignf.pack(pady=15)

        assignlb = Label(assignf, text='Assign Driver', bg='White', fg='Black', font=("", 18))
        assignlb.pack(side='left', padx=20)

        named = []
        record = self.__dmod.emptydriver(tripd[5])
        if record:
            for data in record:
                name = data[2] + " " + data[1]
                named.insert(0, name)
        driver = StringVar()
        driver.set("Driver's Name")
        if not named:
            named.insert(0, "No Driver Available")
        assign = ttk.Combobox(assignf, values=named, textvariable=driver, font=("", 15), state='readonly')
        assign.pack(side='left', padx=10)

        btn = Frame(mainf, bg='White')
        btn.pack(pady=10)

        assignbtn = Button(btn, text='Confirm Assign', bd=0, command=lambda: assignd())
        assignbtn.pack()
    else:
        if assignf:
            assignf.destroy()

    def assignd():
        did = None
        if not driver.get() or driver.get() == "Driver's Name" or driver.get() == "No Driver Available":
            showerror('Invalid', "Select A Driver")
        else:
            driverids = self.__dmod.getid(driver.get())
            for data in driverids:
                did = data[0]
            if self.__tripmod.assigndr(self.__stid, did, tripd[0]):
                showinfo("Message", "Driver Assigned Successfully.")
                top.destroy()
                self.alltripdata.destroy()
```

```
            self.alltriptable()

    # today trip table
    def todaytriptable(self):

        self.todaytripdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set,
selectmode="extended")
        self.todaytripdata.pack()

        self.scrollbar.config(command=self.todaytripdata.yview)

        # defining columns
        self.todaytripdata["columns"] = (
            "Trip ID", "Customer Name", "Pick Up Location", "Destination", "No Of Passenger",
            "Departure Date", "Departure Time", "Status", "Cost", "Driver Id")

        # formatting columns
        self.todaytripdata.column("#0", width=0, stretch=NO)
        self.todaytripdata.column("Trip ID", width=80, minwidth=90, anchor=W)
        self.todaytripdata.column("Customer Name", width=140, minwidth=90, anchor=W)
        self.todaytripdata.column("Pick Up Location", width=180, minwidth=90, anchor=W)
        self.todaytripdata.column("Destination", width=180, minwidth=90, anchor=W)
        self.todaytripdata.column("No Of Passenger", width=100, minwidth=90, anchor=W)
        self.todaytripdata.column("Departure Date", width=180, minwidth=90, anchor=W)
        self.todaytripdata.column("Departure Time", width=180, minwidth=90, anchor=W)
        self.todaytripdata.column("Status", width=180, minwidth=90, anchor=W)
        self.todaytripdata.column("Cost", width=80, minwidth=90, anchor=W)
        self.todaytripdata.column("Driver Id", width=80, minwidth=90, anchor=W)

        # creating heading
        self.todaytripdata.heading("#0", text='Label')
        self.todaytripdata.heading("Trip ID", text="Trip Id")
        self.todaytripdata.heading("Customer Name", text="Customer Name")
        self.todaytripdata.heading("Pick Up Location", text="Pick Up Location")
        self.todaytripdata.heading("Destination", text="Destination")
        self.todaytripdata.heading("No Of Passenger", text="No Of Passenger")
        self.todaytripdata.heading("Departure Date", text="Departure Date")
        self.todaytripdata.heading("Departure Time", text="Departure Time")
        self.todaytripdata.heading("Status", text="Status")
        self.todaytripdata.heading("Cost", text="Cost")
        self.todaytripdata.heading("Driver Id", text="Driver Id")

        # reading data from table
        record = self.__tripmod.todaytrip()
        if record:

            # inserting data into table
```

```
            for data in record:
                name = data[17] + " " + data[16]
                self.todaytripdata.insert("", index="end",
                                          values=(data[0], name, data[2], data[3], data[6], data[4], data[5], data[7],
                                                  data[8], data[9], data[14]))

        self.todaytripdata.bind('<Double-1>', self.minitodaytrip)

    def minitodaytrip(self, event):
        tripd = None
        try:
            if self.todaytripdata.selection()[0]:
                for data in self.todaytripdata.selection():
                    value = self.todaytripdata.item(data)
                    tripd = value['values']
                tp = Toplevel()

                # always top
                tp.transient(self.__window)
                tp.grab_set()
                self.todaytripdetails(tp, tripd)
        except Exception as e:
            showinfo("Message", "Select A Data")
            print(e)

    def todaytripdetails(self, tp, tripd):
        top = tp
        top.title("Trip Details")
        top.geometry("560x480+400+200")

        mainf = Frame(top, bg='White')
        mainf.pack(fill='both', expand=True)

        titlef = Frame(mainf, bg="White")
        titlef.pack(pady=20)

        titlelb = Label(titlef, text='Trip Details', bg='White', fg='Black', font=("", 25, 'bold'))
        titlelb.pack()

        custnamef = Frame(mainf, bg='White')
        custnamef.pack(pady=15)

        custnamelb = Label(custnamef, text='Customer Name : ', bg='White', fg='Black', font=("", 18))
        custnamelb.pack(side='left', padx=10)

        custnamedata = Label(custnamef, text=tripd[1], bg='White', fg='Black', font=("", 15))
        custnamedata.pack(side='left')
```

```
locationf = Frame(mainf, bg='White')
locationf.pack(pady=15)

pickf = Frame(locationf, bg='White')
pickf.pack(side='left', padx=20)

picklb = Label(pickf, text='Pickup Location', bg='White', fg='Black', font=("", 18))
picklb.pack(side='left', padx=10)

pickdata = Label(pickf, text=tripd[2], bg='White', fg='Black', font=("", 15))
pickdata.pack(side='left')

dropf = Frame(locationf, bg='White')
dropf.pack(side='right', padx=20)

droplb = Label(dropf, text='Destination : ', bg='White', fg='Black', font=("", 18))
droplb.pack(side='left')

dropdata = Label(dropf, text=tripd[3], bg='White', fg='Black', font=("", 15))
dropdata.pack(side='left', padx=10)

datime = Frame(mainf, bg='White')
datime.pack(pady=15)

datef = Frame(datime, bg='White')
datef.pack(side='left', padx=20)

datelb = Label(datef, text='Trip Date : ', bg='White', fg='Black', font=("", 18))
datelb.pack(side='left', padx=10)

datedata = Label(datef, text=tripd[5], bg='White', fg='Black', font=("", 15))
datedata.pack(side='left')

timef = Frame(datime, bg='White')
timef.pack(side='right', padx=20)

timelb = Label(timef, text='Pick Up : ', bg='White', fg='Black', font=("", 18))
timelb.pack(side='left')

timedata = Label(timef, text=tripd[6], bg='White', fg='Black', font=("", 15))
timedata.pack(side='left', padx=10)

passtatus = Frame(mainf, bg='White')
passtatus.pack(pady=15)

passf = Frame(passtatus, bg='White')
```

```
    passf.pack(side='left', padx=20)

    passlb = Label(passf, text='No Of Passenger : ', bg='White', fg='Black', font=("", 18))
    passlb.pack(side='left', padx=10)

    passdata = Label(passf, text=tripd[4], fg='Black', bg='White', font=("", 15))
    passdata.pack(side='left')

    statusf = Frame(passtatus, bg='White')
    statusf.pack(side='right', padx=20)

    statuslb = Label(statusf, text='Status : ', bg='White', fg='Black', font=("", 18))
    statuslb.pack(side='left')

    statusdata = Label(statusf, text=tripd[9], bg='White', fg='Black', font=("", 15))
    statusdata.pack(side='left', padx=10)

    distcost = Frame(mainf, bg='White')
    distcost.pack(pady=15)

    distf = Frame(distcost, bg='White')
    distf.pack(side='left', padx=20)

    distlb = Label(distf, text="Distance : ", bg='White', fg='Black', font=("", 18))
    distlb.pack(side='left', padx=10)

    distdata = Label(distf, text=tripd[7], bg='White', fg='Black', font=("", 15))
    distdata.pack(side='left')

    costf = Frame(distcost, bg='White')
    costf.pack(side='left', padx=20)

    costlb = Label(costf, text='Cost : ', bg='White', fg='Black', font=("", 18))
    costlb.pack(side='left')

    costdata = Label(costf, text=tripd[8], bg='White', fg='Black', font=("", 15))
    costdata.pack(side='left', padx=10)
    assignf = None
    if tripd[9] == "Pending":
        assignf = Frame(mainf, bg='White')
        assignf.pack(pady=15)

        assignlb = Label(assignf, text='Assign Driver', bg='White', fg='Black', font=("", 18))
        assignlb.pack(side='left', padx=20)

        named = []
        record = self.__dmod.emptydriver(tripd[5])
```

```
            if record:
                for data in record:
                    name = data[2] + " " + data[1]
                    named.insert(0, name)
            if not named:
                named.insert(0, "No Driver Available")
            driver = StringVar()
            driver.set("Driver's Name")
            assign = ttk.Combobox(assignf, values=named, textvariable=driver, font=("", 15), state='readonly')
            assign.pack(side='left', padx=10)

            btn = Frame(mainf, bg='White')
            btn.pack(pady=10)

            assignbtn = Button(btn, text='Confirm Assign', bd=0, command=lambda: assignd())
            assignbtn.pack()
        else:
            if assignf:
                assignf.destroy()

        def assignd():
            did = None
            if not driver.get() or driver.get() == "Driver's Name" or driver.get() == "No Driver Available":
                showerror('Invalid', "Select A Driver")
            else:
                driverids = self.__dmod.getid(driver.get())
                for data in driverids:
                    did = data[0]
                if self.__tripmod.assigndr(self.__stid, did, tripd[0]):
                    showinfo("Message", "Driver Assigned Successfully.")
                    top.destroy()
                    self.todaytripdata.destroy()
                    self.todaytriptable()

    # showing all customer details
    def allcustable(self):
        self.allcustdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.allcustdata.pack()

        self.scrollbar.config(command=self.allcustdata.yview)

        # defining columns
        self.allcustdata["columns"] = ("Cust ID", "Customer Name", "DOB", "Gender", "Address", "Contact",
"Email")

        # formatting columns
        self.allcustdata.column("#0", width=0, stretch=NO)
```

```
        self.allcustdata.column("Cust ID", width=80, minwidth=90, anchor=W)
        self.allcustdata.column("Customer Name", width=140, minwidth=90, anchor=W)
        self.allcustdata.column("DOB", width=180, minwidth=90, anchor=W)
        self.allcustdata.column("Gender", width=180, minwidth=90, anchor=W)
        self.allcustdata.column("Address", width=100, minwidth=90, anchor=W)
        self.allcustdata.column("Contact", width=100, minwidth=90, anchor=W)
        self.allcustdata.column("Email", width=180, minwidth=90, anchor=W)

        # creating heading
        self.allcustdata.heading("#0", text='Label')
        self.allcustdata.heading("Cust ID", text="Customer Id")
        self.allcustdata.heading("Customer Name", text="Customer Name")
        self.allcustdata.heading("DOB", text="DOB")
        self.allcustdata.heading("Gender", text="Gender")
        self.allcustdata.heading("Address", text="Address")
        self.allcustdata.heading("Contact", text="Contact")
        self.allcustdata.heading("Email", text="Email")

        # reading data from table
        record = self.__custmod.allcust()
        if record:

            # inserting data into table
            for data in record:
                name = data[2] + " " + data[1]
                self.allcustdata.insert("", index="end",
                                        values=(data[0], name, data[3], data[4], data[5], data[6], data[7]))

    # showing all staff details
    def allstafftable(self):
        self.allstaffdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.allstaffdata.pack()

        self.scrollbar.config(command=self.allstaffdata.yview)

        # defining columns
        self.allstaffdata["columns"] = ("Staff ID", "Staff Name", "DOB", "Gender", "Address", "Contact",
"Email")

        # formatting columns
        self.allstaffdata.column("#0", width=0, stretch=NO)
        self.allstaffdata.column("Staff ID", width=80, minwidth=90, anchor=W)
        self.allstaffdata.column("Staff Name", width=140, minwidth=90, anchor=W)
        self.allstaffdata.column("DOB", width=180, minwidth=90, anchor=W)
        self.allstaffdata.column("Gender", width=180, minwidth=90, anchor=W)
        self.allstaffdata.column("Address", width=100, minwidth=90, anchor=W)
        self.allstaffdata.column("Contact", width=100, minwidth=90, anchor=W)
```

```
        self.allstaffdata.column("Email", width=180, minwidth=90, anchor=W)

        # creating heading
        self.allstaffdata.heading("#0", text='Label')
        self.allstaffdata.heading("Staff ID", text="Staff Id")
        self.allstaffdata.heading("Staff Name", text="Staff Name")
        self.allstaffdata.heading("DOB", text="DOB")
        self.allstaffdata.heading("Gender", text="Gender")
        self.allstaffdata.heading("Address", text="Address")
        self.allstaffdata.heading("Contact", text="Contact")
        self.allstaffdata.heading("Email", text="Email")

        # reading data from table
        record = self.__stmod.allstaff()
        if record:

            # inserting data into table
            for data in record:
                name = data[2] + " " + data[1]
                self.allstaffdata.insert("", index="end",
                                         values=(data[0], name, data[3], data[4], data[5], data[6], data[7]))

    # showing all driver details
    def alldrivertable(self):
        self.alldriverdata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.alldriverdata.pack()

        self.scrollbar.config(command=self.alldriverdata.yview)

        # defining columns
        self.alldriverdata["columns"] = ("Driver ID", "Driver Name", "DOB", "Gender", "Address", "Contact",
"License No"
                                         , "Email")

        # formatting columns
        self.alldriverdata.column("#0", width=0, stretch=NO)
        self.alldriverdata.column("Driver ID", width=80, minwidth=90, anchor=W)
        self.alldriverdata.column("Driver Name", width=140, minwidth=90, anchor=W)
        self.alldriverdata.column("DOB", width=180, minwidth=90, anchor=W)
        self.alldriverdata.column("Gender", width=180, minwidth=90, anchor=W)
        self.alldriverdata.column("Address", width=100, minwidth=90, anchor=W)
        self.alldriverdata.column("Contact", width=100, minwidth=90, anchor=W)
        self.alldriverdata.column("License No", width=100, minwidth=90, anchor=W)
        self.alldriverdata.column("Email", width=180, minwidth=90, anchor=W)

        # creating heading
        self.alldriverdata.heading("#0", text='Label')
```

```
        self.alldriverdata.heading("Driver ID", text="Driver Id")
        self.alldriverdata.heading("Driver Name", text="Driver Name")
        self.alldriverdata.heading("DOB", text="DOB")
        self.alldriverdata.heading("Gender", text="Gender")
        self.alldriverdata.heading("Address", text="Address")
        self.alldriverdata.heading("Contact", text="Contact")
        self.alldriverdata.heading("License No", text="License No")
        self.alldriverdata.heading("Email", text="Email")

        # reading data from table
        record = self.__dmod.alldriver()
        if record:

            # inserting data into table
            for data in record:
                name = data[2] + " " + data[1]
                self.alldriverdata.insert("", index="end",
                                          values=(data[0], name, data[3], data[4], data[5], data[6], data[7], data[8]))

            # self.pendingdata.bind('<Double-1>', self.data)

    # showing all vehicle details
    def allvehicletable(self):
        self.allvehicledata = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set,
selectmode="extended")
        self.allvehicledata.pack()

        self.scrollbar.config(command=self.allvehicledata.yview)

        # defining columns
        self.allvehicledata["columns"] = ("Vehicle ID", "Vehicle No", "Vehicle Type", "Vehicle Model", "Date
Registered"
                                          , "Status")

        # formatting columns
        self.allvehicledata.column("#0", width=0, stretch=NO)
        self.allvehicledata.column("Vehicle ID", width=100, minwidth=50, anchor=W)
        self.allvehicledata.column("Vehicle No", width=180, minwidth=90, anchor=W)
        self.allvehicledata.column("Vehicle Type", width=180, minwidth=90, anchor=W)
        self.allvehicledata.column("Vehicle Model", width=180, minwidth=90, anchor=W)
        self.allvehicledata.column("Date Registered", width=180, minwidth=90, anchor=W)
        self.allvehicledata.column("Status", width=180, minwidth=90, anchor=W)

        # creating heading
        self.allvehicledata.heading("#0", text='Label')
        self.allvehicledata.heading("Vehicle ID", text="Vehicle ID")
        self.allvehicledata.heading("Vehicle No", text="Vehicle No")
```

```
        self.allvehicledata.heading("Vehicle Type", text="Vehicle Type")
        self.allvehicledata.heading("Vehicle Model", text="Vehicle Model")
        self.allvehicledata.heading("Date Registered", text="Date Registered")
        self.allvehicledata.heading("Status", text="Status")

        # reading data from table
        record = self.__vmod.allvehicle()
        if record:

            # inserting data into table
            for data in record:
                self.allvehicledata.insert("", index="end",
                                           values=(data[0], data[1], data[2], data[3], data[4], data[6]))

    def showaboutrip(self, _):
        self.nameentry.delete(0, 'end')
        self.addentry.delete(0, 'end')
        self.selectedall.set("None")
        if self.pendingdata:
            self.pendingdata.destroy()
        if self.alltripdata:
            self.alltripdata.destroy()
        if self.todaytripdata:
            self.todaytripdata.destroy()
        if self.allcustdata:
            self.allcustdata.destroy()
        if self.allstaffdata:
            self.allstaffdata.destroy()
        if self.alldriverdata:
            self.alldriverdata.destroy()
        if self.allvehicledata:
            self.allvehicledata.destroy()
        if self.searchcusr:
            self.searchcusr.destroy()
        if self.selectedtrip.get() == "All Pending":
            self.pendingtable()
        if self.selectedtrip.get() == "Today's":
            self.todaytriptable()
        if self.selectedtrip.get() == "All":
            self.alltriptable()

    # showing table according to selected
    def showall(self, _):
        self.nameentry.delete(0, 'end')
        self.addentry.delete(0, 'end')
        self.selectedtrip.set("None")
        if self.pendingdata:
```

```
            self.pendingdata.destroy()
        if self.alltripdata:
            self.alltripdata.destroy()
        if self.todaytripdata:
            self.todaytripdata.destroy()
        if self.allcustdata:
            self.allcustdata.destroy()
        if self.allstaffdata:
            self.allstaffdata.destroy()
        if self.alldriverdata:
            self.alldriverdata.destroy()
        if self.allvehicledata:
            self.allvehicledata.destroy()
        if self.searchcusr:
            self.searchcusr.destroy()
        if self.selectedall.get() == "Customer":
            self.allcustable()
        elif self.selectedall.get() == "Staff":
            self.allstafftable()
        elif self.selectedall.get() == "Driver":
            self.alldrivertable()
        elif self.selectedall.get() == "Vehicle":
            self.allvehicletable()

    # searching customer from name
    def searchc(self):
        self.selectedall.set("None")
        self.selectedtrip.set("None")
        if self.pendingdata:
            self.pendingdata.destroy()
        if self.alltripdata:
            self.alltripdata.destroy()
        if self.todaytripdata:
            self.todaytripdata.destroy()
        if self.allcustdata:
            self.allcustdata.destroy()
        if self.allstaffdata:
            self.allstaffdata.destroy()
        if self.alldriverdata:
            self.alldriverdata.destroy()
        if self.allvehicledata:
            self.allvehicledata.destroy()
        if self.searchcusr:
            self.searchcusr.destroy()
        if not self.nameentry.get():
            name = None
        else:
```

```
            name = self.nameentry.get()
        if not self.addentry.get():
            add = None
        else:
            add = self.addentry.get()
        self.nameentry.delete(0, 'end')
        self.addentry.delete(0, 'end')
        record = self.__custmod.search(name, add)
        self.searchcust(record)

    def searchcust(self, rcrd):
        self.searchcusr = ttk.Treeview(self.table, yscrollcommand=self.scrollbar.set, selectmode="extended")
        self.searchcusr.pack()

        self.scrollbar.config(command=self.searchcusr.yview)

        # defining columns
        self.searchcusr["columns"] = ("Cust ID", "Customer Name", "DOB", "Gender", "Address", "Contact",
"Email")

        # formatting columns
        self.searchcusr.column("#0", width=0, stretch=NO)
        self.searchcusr.column("Cust ID", width=80, minwidth=90, anchor=W)
        self.searchcusr.column("Customer Name", width=140, minwidth=90, anchor=W)
        self.searchcusr.column("DOB", width=180, minwidth=90, anchor=W)
        self.searchcusr.column("Gender", width=180, minwidth=90, anchor=W)
        self.searchcusr.column("Address", width=100, minwidth=90, anchor=W)
        self.searchcusr.column("Contact", width=100, minwidth=90, anchor=W)
        self.searchcusr.column("Email", width=180, minwidth=90, anchor=W)

        # creating heading
        self.searchcusr.heading("#0", text='Label')
        self.searchcusr.heading("Cust ID", text="Customer Id")
        self.searchcusr.heading("Customer Name", text="Customer Name")
        self.searchcusr.heading("DOB", text="DOB")
        self.searchcusr.heading("Gender", text="Gender")
        self.searchcusr.heading("Address", text="Address")
        self.searchcusr.heading("Contact", text="Contact")
        self.searchcusr.heading("Email", text="Email")

        record = rcrd
        if record:

            # inserting data into table
            for data in record:
                name = data[2] + " " + data[1]
                self.searchcusr.insert("", index="end",
```

```
                    values=(data[0], name, data[3], data[4], data[5], data[6], data[7]))

    # top level to add staff
    def addstaff(self):
        add = Toplevel()

        # always top
        add.transient(self.__window)
        add.grab_set()
        AddStaff(add)

    # top level to add vehicle
    def addvehicle(self):
        add = Toplevel()

        # always top
        add.transient(self.__window)
        add.grab_set()
        AddVehicle(add)

    # top level to add driver
    def addriver(self):
        add = Toplevel()

        # always top
        add.transient(self.__window)
        add.grab_set()
        AddDriver(add)

    # confirming log out
    def confirm(self):
        ans = askyesno("Conformation", "Are You Sure You Want To Log Out?")
        if ans:
            self.__controller.log()


# declaring class add staff
class AddStaff:

    # creating a top level to be displayed over root window
    def __init__(self, window):

        self.__stmod = StaffModel()
        self.__window = window
        self.__window.title("Add Staff")
        self.__window.geometry('440x525+500+200')
        self.__nmregex = ("[A-Z][a-z]{2,10}")  # firstname, lastname regex
```

```
    self.__conregx = ("[9]{1}[\d]{9}")  # contact regex
    self.__emregex = ('^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$')  # regex for email only
    self.__passregex = ("^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$")  # regex for
password only

    mainf = Frame(self.__window, bg='Light Grey')
    mainf.pack(expand=True, fill='both')

    title = Frame(mainf, bg='Light Grey')
    title.pack(expand=True)

    titlelb = Label(title, text='Add Staff', bg='Light Grey', fg='Black', font=("", 20, 'bold'))
    titlelb.pack(pady=20)

    lbtx = Frame(mainf, bg='Light Grey')
    lbtx.pack(expand=True)

    lb = Frame(lbtx, bg='Light Grey')
    lb.pack(side='left', padx=30)

    stname = Label(lb, text='First Name', bg='Light Grey', fg='Black', font=("", 18))
    stname.pack()

    Label(lb, bg='Light Grey').pack()

    stlname = Label(lb, text='Last Name', bg='Light Grey', fg='Black', font=("", 18))
    stlname.pack()

    Label(lb, bg='Light Grey').pack()

    stdob = Label(lb, text='Date of Birth', bg='Light Grey', fg='Black', font=("", 18))
    stdob.pack()

    Label(lb, bg='Light Grey').pack()

    stgender = Label(lb, text='Gender', bg='Light Grey', fg='Black', font=("", 18))
    stgender.pack()

    Label(lb, bg='Light Grey').pack()

    stadd = Label(lb, text='Address', bg='Light Grey', fg='Black', font=("", 18))
    stadd.pack()

    Label(lb, bg='Light Grey').pack()

    stcon = Label(lb, text='Contact', bg='Light Grey', fg='Black', font=("", 18))
    stcon.pack()
```

```
    Label(lb, bg='Light Grey').pack()

    stemail = Label(lb, text='Email', bg='Light Grey', fg='Black', font=("", 18))
    stemail.pack()

    Label(lb, bg='Light Grey').pack()

    stpass = Label(lb, text='Password', bg='Light Grey', fg='Black', font=("", 18))
    stpass.pack()

    Label(lb, bg='Light Grey').pack()

    txt = Frame(lbtx, bg='Light Grey')
    txt.pack(side='right', padx=30)

    self.__stfname = Entry(txt, bg='White', fg='Black')
    self.__stfname.pack()

    self.__stfnlb = Label(txt, bg='Light Grey')
    self.__stfnlb.pack()

    self.__stlname = Entry(txt, bg='White', fg='Black')
    self.__stlname.pack()

    self.__stlnlb = Label(txt, bg='Light Grey')
    self.__stlnlb.pack()

    date = Frame(txt, bg="Light Grey")
    date.pack()

    # using datetime module to find the current year
    x = datetime.now()
    y = x.year - 18
    z = x.year - 100
    yea = list(range(z, y))
    self.__yearvar = StringVar()
    year = Combobox(date, values=yea, textvariable=self.__yearvar, width=4, justify="left",
state='readonly')
    year.set("Year")
    year.pack(side="left")

    mon = ["January", "February", "March", "April", "May", "June", "July", "August", "September",
"October",
        "November", "December"]
    self.__monthvar = StringVar()
```

```
        self.month = Combobox(date, values=mon, textvariable=self.__monthvar, width=7, justify="left",
state='readonly')
        self.month.set("Month")
        self.month.pack(side="left", padx=2)
        self.month.bind("<<ComboboxSelected>>", self.determine)

        self.__dayvar = StringVar()
        self.day = Combobox(date, values=[], textvariable=self.__dayvar, width=3, justify="left",
state='readonly')
        self.day.set("Day")
        self.day.pack(side="left")

        self.__datelb = Label(txt, bg="Light Grey")
        self.__datelb.pack()

        self.__emt = Label(txt, bg='Light Grey', font=('', 1))
        self.__emt.pack()

        radiio = Frame(txt, bg="Light Grey")
        radiio.pack()

        self.__genval = StringVar()
        self.__genval.set('1')
        self.__male = Radiobutton(radiio, text="Male", variable=self.__genval, value="Male")
        self.__male.pack(side='left')
        self.__female = Radiobutton(radiio, text="Female", variable=self.__genval, value="Female")
        self.__female.pack(side='left', padx=10)
        self.__other = Radiobutton(radiio, text="Other", variable=self.__genval, value="Others")
        self.__other.pack(side='left')

        self.__gen = Label(txt, bg="Light Grey")
        self.__gen.pack()

        self.__emt2 = Label(txt, bg='Light Grey', font=('', 1))
        self.__emt2.pack()

        self.__stadd = Entry(txt, bg='White', fg='Black')
        self.__stadd.pack()

        self.__add = Label(txt, bg='Light Grey')
        self.__add.pack()

        self.__stcon = Entry(txt, bg='White', fg='Black')
        self.__stcon.pack()

        self.__con = Label(txt, bg='Light Grey')
        self.__con.pack()
```

```
        self.__stemail = Entry(txt, bg='White', fg='Black')
        self.__stemail.pack()

        self.__stemlb = Label(txt, bg='Light Grey')
        self.__stemlb.pack()

        self.__stpass = Entry(txt, show='*', bg='White', fg='Black')
        self.__stpass.pack()

        self.__stpasslb = Label(txt, bg='Light Grey')
        self.__stpasslb.pack()

        btn = Frame(mainf, bg='Light Grey')
        btn.pack(expand=True)

        add = Button(btn, text='Add Staff', bg='Light Grey', fg='Black', bd=0, command=self.create)
        add.pack(side='left', padx=30, pady=10)

        cancel = Button(btn, text="Cancel", bg='Light Grey', fg="Black", bd=0, command=self.close)
        cancel.pack(side='right', padx=30, pady=10)

        self.__window.bind('<Return>', self.callcreate)

    # calling creating
    def callcreate(self, _):
        self.create()

    # determining the day to show
    def determine(self, _):
        if self.__yearvar.get() == 'Year':
            showinfo("Message", "Select a Year First")
            self.month.set("Month")
        match self.__monthvar.get():

            case "January":
                self.day.config(values=list(range(1, 32)))

            case "February":
                if int(self.__yearvar.get()) % 400 == 0:
                    self.day.config(values=list(range(1, 30)))

                else:
                    self.day.config(values=list(range(1, 29)))

            case "March":
                self.day.config(values=list(range(1, 32)))
```

```
        case "April":
            self.day.config(values=list(range(1, 31)))

        case "May":
            self.day.config(values=list(range(1, 32)))

        case "June":
            self.day.config(values=list(range(1, 31)))

        case "July":
            self.day.config(values=list(range(1, 32)))

        case "August":
            self.day.config(values=list(range(1, 32)))

        case "September":
            self.day.config(values=list(range(1, 31)))

        case "October":
            self.day.config(values=list(range(1, 32)))

        case "November":
            self.day.config(values=list(range(1, 31)))

        case "December":
            self.day.config(values=list(range(1, 32)))

    # validating and creating account
    def create(self):
        if self.validate():
            mod = RegistrationModel()
            if mod.double(self.__stemail.get()):
                showinfo("Message", "Email Already Used")
            else:
                if self.__stmod.create():
                    showinfo("Message", "Account Created")
                    self.__window.destroy()

    # validating the data entered
    def validate(self):
        a = self.fn()
        b = self.ln()
        c = self.dob()
        d = self.gen()
        e = self.con()
        f = self.add()
```

```
        g = self.em()
        h = self.pas()
        if a and b and c and d and e and f and g and h:
            return True

    def fn(self):
        if not self.__stfname.get():
            self.__stfnlb.config(text="First Name Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__nmregex, self.__stfname.get()):
            self.__stfnlb.config(text="Invalid First Name", font=(", 10), fg='Red')
            return False
        else:
            self.__stfnlb.config(text="")
            self.__stmod.setfn(self.__stfname.get())
            return True

    def ln(self):
        if not self.__stlname.get():
            self.__stlnlb.config(text="Last Name Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__nmregex, self.__stlname.get()):
            self.__stlnlb.config(text="Invalid Last Name", font=(", 10), fg='Red')
            return False
        else:
            self.__stlnlb.config(text="")
            self.__stmod.setln(self.__stlname.get())
            return True

    def dob(self):
        if self.__dayvar.get() == 'Day':
            self.__datelb.config(text="Select Date Of Birth.", font=("", 10), fg="Red")
            self.__emt.config(font=', 5')
            return False
        else:
            self.__datelb.config(text=")
            self.__emt.config(font=', 1')
            date = self.__yearvar.get() + "-" + self.__monthvar.get() + "-" + self.__dayvar.get()
            self.__stmod.setdate(date)
            return True

    def gen(self):
        if self.__genval.get() == '1':
            self.__gen.config(text="Select A Gender.", font=("", 10), fg="Red")
            self.__emt2.config(font=', 5')
            return False
        else:
```

```
        self.__gen.config(text='')
        self.__emt2.config(font=', 1')
        self.__stmod.setgen(self.__genval.get())
        return True

    def con(self):
        if not self.__stcon.get():
            self.__con.config(text="Contact Can Not Be Empty.", font=('', 10), fg='Red')
            return False
        elif not re.match(self.__conregx, self.__stcon.get()):
            self.__con.config(text="Invalid Contact", font=('', 10), fg='Red')
            return False
        else:
            self.__con.config(text="")
            self.__stmod.setcon(self.__stcon.get())
            return True

    def add(self):
        if not self.__stadd.get():
            self.__add.config(text='Address Can Not Be Empty.', font=('', 10), fg='Red')
            return False
        else:
            self.__add.config(text='')
            self.__stmod.setadd(self.__stadd.get())
            return True

    def em(self):
        email = self.__stemail.get().lower()
        if not email:
            self.__stemlb.config(text="Email Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__emregex, email):
            self.__stemlb.config(text="Enter Correct Email", font=("", 10), fg="Red")
            return False
        else:
            self.__stemlb.config(text="")
            self.__stmod.setem(email)
            return True

    def pas(self):
        if not self.__stpass.get():
            self.__stpasslb.config(text="Password Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__passregex, self.__stpass.get()):
            showinfo('Message', "Password must contain 8 letters, a capital letter, a small letter, a number and a
"
                     "symbol")
```

```
            self.__stpasslb.config(text="Invalid Password", font=("", 10), fg="Red")
            return False
        else:
            self.__stpasslb.config(text="")
            self.__stmod.setpas(self.__stpass.get())
            return True

    # closing the top level if wrong click
    def close(self):
        self.__window.destroy()


# this is to add vehicle
class AddVehicle:

    # creating a top level to be displayed over root window
    def __init__(self, window):

        self.__vmod = VehicleModel()
        self.__window = window
        self.__window.title("Add Vehicle")
        self.__window.geometry('400x350+500+230')
        self.__numrex = ("[A-Z]{1}[a-z]{1}\s[0-9]{1,3}\s[A-Za-z]{1,3}\s[0-9]{1,5}")  # regex for number
only

        mainf = Frame(self.__window, bg='Light Grey')
        mainf.pack(expand=True, fill='both')

        title = Frame(mainf, bg='Light Grey')
        title.pack(expand=True)

        titlelb = Label(title, text='Add Vehicle', bg='Light Grey', fg='Black', font=("", 20, 'bold'))
        titlelb.pack(pady=20)

        lbtx = Frame(mainf, bg='Light Grey')
        lbtx.pack(expand=True)

        lb = Frame(lbtx, bg='Light Grey')
        lb.pack(side='left', padx=20)

        self.__vehiclenolb = Label(lb, text='Vehicle No', bg='Light Grey', fg='Black', font=("", 18))
        self.__vehiclenolb.pack()

        Label(lb, bg='Light Grey').pack()

        self.__vehicletypelb = Label(lb, text='Vehicle Type', bg='Light Grey', fg='Black', font=("", 18))
        self.__vehicletypelb.pack()
```

```
Label(lb, bg='Light Grey').pack()

self.__vehiclemodellb = Label(lb, text='Vehicle Model', bg='Light Grey', fg='Black', font=("", 18))
self.__vehiclemodellb.pack()

Label(lb, bg='Light Grey').pack()
Label(lb, bg='Light Grey').pack()

self.__vehicledeslb = Label(lb, text='Description', bg='Light Grey', fg='Black', font=("", 18))
self.__vehicledeslb.pack()

Label(lb, bg='Light Grey').pack()

txt = Frame(lbtx, bg='Light Grey')
txt.pack(side='right', padx=20)

self.__vehicleno = Entry(txt, bg='Light Grey', fg='Black')
self.__vehicleno.pack()

self.vehicleno = Label(txt, bg='Light Grey')
self.vehicleno.pack()

self.__vehiclet = StringVar()
type = list(("Hundai", "Maruti", "Honda"))
self.__vehicletype = Combobox(txt, values=type, textvariable=self.__vehiclet, state='readonly')
self.__vehicletype.set("Select Vehicle")
self.__vehicletype.pack()
self.__vehicletype.bind("<<ComboboxSelected>>", self.vehicleM)
self.vehicletype = Label(txt, bg='Light Grey')
self.vehicletype.pack()

self.__vehiclemod = StringVar()
self.__vehiclemodel = Combobox(txt, values=[], textvariable=self.__vehiclemod, state='readonly')
self.__vehiclemodel.set("Select Model")
self.__vehiclemodel.pack()

self.vehiclemodel = Label(txt, bg='Light Grey')
self.vehiclemodel.pack()

self.__vehicledes = Text(txt, width=30, height=4, bg='Light Grey', fg='Black')
self.__vehicledes.pack()

btn = Frame(mainf, bg='Light Grey')
btn.pack(expand=True)

add = Button(btn, text='Add Vehicle', bg='Light Grey', fg='Black', bd=0, command=self.addv)
```

```
        add.pack(side='left', padx=30, pady=10)

        cancel = Button(btn, text="Cancel", bg='Light Grey', fg="Black", bd=0, command=self.close)
        cancel.pack(side='right', padx=30, pady=10)

        self.__window.bind("<Return>", self.calladdv)

    # calling addv method
    def calladdv(self, _):
        self.addv()

    # setting model according to type
    def vehicleM(self, _):
        match self.__vehiclet.get():

            case "Hundai":
                self.__vehiclemodel.config(values=["Grand 110 NIOS", "Aura", "Venue"])

            case "Maruti":
                self.__vehiclemodel.config(values=["Swift dzire", "Eeco", "Ritz"])

            case "Honda":
                self.__vehiclemodel.config(values=["Amaze V", "city E", "mobilio S"])

    # add vehicle details
    def addv(self):
        if self.validate():
            if self.__vmod.double(self.__vehicleno.get()):
                showinfo('Message', "Vehicle Already Added")
            else:
                if self.__vehicledes.get("1.0", END):
                    self.__vmod.setdescription(self.__vehicledes.get("1.0", END))
                self.__vmod.addV()
                showinfo("Message", "Vehicle Added")
                self.__window.destroy()

    # validating the entered data
    def validate(self):
        a = self.vno()
        b = self.vtype()
        c = self.vmodel()
        if a and b and c:
            return True

    def vno(self):
        if not self.__vehicleno.get():
            self.vehicleno.config(text='Vehicle No Can Not Be Empty.', font=("", 10), fg="Red")
```

```
            return False
        elif not re.match(self.__numrex, self.__vehicleno.get()):
            self.vehicleno.config(text='Invalid Vehicle Number.', font=("", 10), fg="Red")
            return False
        else:
            self.vehicleno.config(text='')
            self.__vmod.setvehicleno(self.__vehicleno.get())
            return True

    def vtype(self):
        if self.__vehiclet.get() == "Select Vehicle":
            self.vehicletype.config(text='Vehicle Type Can Not Be Empty.', font=("", 10), fg="Red")
            return False
        else:
            self.vehicletype.config(text='')
            self.__vmod.setvehicletype(self.__vehiclet.get())
            return True

    def vmodel(self):
        if self.__vehiclemod.get() == "Select Model":
            self.vehiclemodel.config(text='Vehicle Model Can Not Be Empty.', font=("", 10), fg="Red")
            return False
        else:
            self.vehiclemodel.config(text='')
            self.__vmod.setvehiclemodel(self.__vehiclemod.get())
            return True

    # closing the top level if wrong click
    def close(self):
        self.__window.destroy()


# declaring class add staff
class AddDriver:

    # creating a top level to be displayed over root window
    def __init__(self, window):

        self.__dmod = DriverModel()
        self.__vmod = VehicleModel()
        self.__window = window
        self.__window.title("Add Driver")
        self.__window.geometry('480x660+500+150')
        self.__nmregex = ("[A-Z][a-z]{2,10}")  # firstname, lastname regex
        self.__conregx = ("[9]{1}[\d]{9}")  # contact regex
        self.__emregex = ("^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$")  # regex for email only
```

```
self.__passregex = ("^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$")  # regex for
password only

mainf = Frame(self.__window, bg='Light Grey')
mainf.pack(expand=True, fill='both')

title = Frame(mainf, bg='Light Grey')
title.pack(expand=True)

titlelb = Label(title, text='Add Driver', bg='Light Grey', fg='Black', font=("", 20, 'bold'))
titlelb.pack(pady=20)

lbtx = Frame(mainf, bg='Light Grey')
lbtx.pack(expand=True)

lb = Frame(lbtx, bg='Light Grey')
lb.pack(side='left', padx=40)

dfname = Label(lb, text='First Name', bg='Light Grey', fg='Black', font=("", 18))
dfname.pack()

Label(lb, bg='Light Grey').pack()

dlname = Label(lb, text='Last Name', bg='Light Grey', fg='Black', font=("", 18))
dlname.pack()

Label(lb, bg='Light Grey').pack()

ddob = Label(lb, text='Date of Birth', bg='Light Grey', fg='Black', font=("", 18))
ddob.pack()

Label(lb, bg='Light Grey').pack()

dgender = Label(lb, text='Gender', bg='Light Grey', fg='Black', font=("", 18))
dgender.pack()

Label(lb, bg='Light Grey').pack()

dadd = Label(lb, text='Address', bg='Light Grey', fg='Black', font=("", 18))
dadd.pack()

Label(lb, bg='Light Grey').pack()

dcon = Label(lb, text='Contact', bg='Light Grey', fg='Black', font=("", 18))
dcon.pack()

Label(lb, bg='Light Grey').pack()
```

```
dlicense = Label(lb, text='License No', bg='Light Grey', fg='Black', font=("", 18))
dlicense.pack()

Label(lb, bg='Light Grey').pack()

dvehicle = Label(lb, text='Vehicle No', bg='Light Grey', fg='Black', font=("", 18))
dvehicle.pack()

Label(lb, bg='Light Grey').pack()

demail = Label(lb, text='Email', bg='Light Grey', fg='Black', font=("", 18))
demail.pack()

Label(lb, bg='Light Grey').pack()

dpass = Label(lb, text='Password', bg='Light Grey', fg='Black', font=("", 18))
dpass.pack()

Label(lb, bg='Light Grey').pack()

txt = Frame(lbtx, bg='Light Grey')
txt.pack(side='right', padx=40)

self.__dfname = Entry(txt, bg='White', fg='Black')
self.__dfname.pack()

self.__dfnlb = Label(txt, bg='Light Grey')
self.__dfnlb.pack()

self.__dlname = Entry(txt, bg='White', fg='Black')
self.__dlname.pack()

self.__dlnlb = Label(txt, bg='Light Grey')
self.__dlnlb.pack()

date = Frame(txt, bg="Light Grey")
date.pack()

# using datetime module to find the current year
x = datetime.now()
y = x.year - 18
z = x.year - 100
yea = list(range(z, y))
self.__yearvar = StringVar()
year = Combobox(date, values=yea, textvariable=self.__yearvar, width=4, justify="left",
state='readonly')
```

```
        year.set("Year")
        year.pack(side="left")

        mon = ["January", "February", "March", "April", "May", "June", "July", "August", "September",
"October",
              "November", "December"]
        self.__monthvar = StringVar()
        self.month = Combobox(date, values=mon, textvariable=self.__monthvar, width=7, justify="left",
state='readonly')
        self.month.set("Month")
        self.month.pack(side="left", padx=2)
        self.month.bind("<<ComboboxSelected>>", self.determine)

        self.__dayvar = StringVar()
        self.day = Combobox(date, values=[], textvariable=self.__dayvar, width=3, justify="left",
state='readonly')
        self.day.set("Day")
        self.day.pack(side="left")

        self.__datelb = Label(txt, bg="Light Grey")
        self.__datelb.pack()

        self.__emt = Label(txt, bg='Light Grey', font=('', 1))
        self.__emt.pack()

        radiio = Frame(txt, bg="Light Grey")
        radiio.pack()

        self.__genval = StringVar()
        self.__genval.set('1')
        self.__male = Radiobutton(radiio, text="Male", variable=self.__genval, value="Male")
        self.__male.pack(side='left')
        self.__female = Radiobutton(radiio, text="Female", variable=self.__genval, value="Female")
        self.__female.pack(side='left', padx=10)
        self.__other = Radiobutton(radiio, text="Other", variable=self.__genval, value="Others")
        self.__other.pack(side='left')

        self.__genderlb = Label(txt, bg="Light Grey")
        self.__genderlb.pack()

        self.__emt2 = Label(txt, bg='Light Grey', font=('', 1))
        self.__emt2.pack()

        self.__dadd = Entry(txt, bg='White', fg='Black')
        self.__dadd.pack()

        self.__add = Label(txt, bg='Light Grey')
```

```
        self.__add.pack()

        self.__dcon = Entry(txt, bg='White', fg='Black')
        self.__dcon.pack()

        self.__con = Label(txt, bg='Light Grey')
        self.__con.pack()

        self.__dlicense = Entry(txt, bg='White', fg='Black')
        self.__dlicense.pack()

        self.__dlicenselb = Label(txt, bg='Light Grey')
        self.__dlicenselb.pack()

        self.__dvehiclevar = StringVar()
        self.__vehicleno = []
        record = self.__vmod.assigning()
        for data in record:
            vehivleno = data[1]
            self.__vehicleno.insert(0, vehivleno)
        self.__dvehicle = Combobox(txt, values=self.__vehicleno, textvariable=self.__dvehiclevar, width=18,
                                   justify='center', state='readonly')
        self.__dvehicle.set("Select Vehicle")
        self.__dvehicle.pack()

        self.__vehicle = Label(txt, bg='Light Grey')
        self.__vehicle.pack()

        self.__demail = Entry(txt, bg='White', fg='Black')
        self.__demail.pack()

        self.__demlb = Label(txt, bg='Light Grey')
        self.__demlb.pack()

        self.__dpass = Entry(txt, bg='White', fg='Black', show='*')
        self.__dpass.pack()

        self.__dpasslb = Label(txt, bg='Light Grey')
        self.__dpasslb.pack()

        btn = Frame(mainf, bg='Light Grey')
        btn.pack(expand=True)

        add = Button(btn, text='Add Driver', bg='Light Grey', fg='Black', bd=0, command=self.create)
        add.pack(side='left', padx=30, pady=10)

        cancel = Button(btn, text="Cancel", bg='Light Grey', fg="Black", bd=0, command=self.close)
```

```
        cancel.pack(side='right', padx=30, pady=10)

        self.__window.bind('<Return>', self.callcreate)

    # calling creating
    def callcreate(self, _):
        self.create()

    # determining the day to show
    def determine(self, _):
        if self.__yearvar.get() == 'Year':
            showinfo("Message", "Select a Year First")
            self.month.set("Month")
        match self.__monthvar.get():

            case "January":
                self.day.config(values=list(range(1, 32)))

            case "February":
                if int(self.__yearvar.get()) % 400 == 0:
                    self.day.config(values=list(range(1, 30)))

                else:
                    self.day.config(values=list(range(1, 29)))

            case "March":
                self.day.config(values=list(range(1, 32)))

            case "April":
                self.day.config(values=list(range(1, 31)))

            case "May":
                self.day.config(values=list(range(1, 32)))

            case "June":
                self.day.config(values=list(range(1, 31)))

            case "July":
                self.day.config(values=list(range(1, 32)))

            case "August":
                self.day.config(values=list(range(1, 32)))

            case "September":
                self.day.config(values=list(range(1, 31)))

            case "October":
```

```
            self.day.config(values=list(range(1, 32)))

        case "November":
            self.day.config(values=list(range(1, 31)))

        case "December":
            self.day.config(values=list(range(1, 32)))

    # validating and creating account
    def create(self):
        if self.validate():
            mod = RegistrationModel()
            if mod.double(self.__demail.get()):
                showinfo("Message", "Email Already Used")
            else:
                if self.__dmod.create():
                    if self.__vmod.assign(self.__dvehiclevar.get()):
                        showinfo("Message", "Account Created")
                        self.__window.destroy()

    # validating the data entered
    def validate(self):
        a = self.fn()
        b = self.ln()
        c = self.dob()
        d = self.gen()
        e = self.con()
        f = self.add()
        g = self.license()
        h = self.vehicle()
        i = self.em()
        j = self.pas()
        if a and b and c and d and e and f and g and h and i and j:
            return True

    def fn(self):
        if not self.__dfname.get():
            self.__dfnlb.config(text="First Name Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__nmregex, self.__dfname.get()):
            self.__dfnlb.config(text="Invalid First Name", font=(", 10), fg='Red')
            return False
        else:
            self.__dfnlb.config(text="")
            self.__dmod.setfn(self.__dfname.get())
            return True
```

```
    def ln(self):
        if not self.__dlname.get():
            self.__dlnlb.config(text="Last Name Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__nmregex, self.__dlname.get()):
            self.__dlnlb.config(text="Invalid Last Name", font=(", 10), fg='Red')
            return False
        else:
            self.__dlnlb.config(text="")
            self.__dmod.setln(self.__dlname.get())
            return True

    def dob(self):
        if self.__dayvar.get() == 'Day':
            self.__datelb.config(text="Select Date Of Birth.", font=("", 10), fg="Red")
            self.__emt.config(font=', 5')
            return False
        else:
            self.__datelb.config(text=")
            self.__emt.config(font=', 1')
            date = self.__yearvar.get() + "-" + self.__monthvar.get() + "-" + self.__dayvar.get()
            self.__dmod.setdate(date)
            return True

    def gen(self):
        if self.__genval.get() == '1':
            self.__genderlb.config(text="Select A Gender.", font=("", 10), fg="Red")
            self.__emt2.config(font=', 5')
            return False
        else:
            self.__genderlb.config(text=")
            self.__emt2.config(font=', 1')
            self.__dmod.setgen(self.__genval.get())
            return True

    def con(self):
        if not self.__dcon.get():
            self.__con.config(text="Contact Can Not Be Empty.", font=(", 10), fg='Red')
            return False
        elif not re.match(self.__conregx, self.__dcon.get()):
            self.__con.config(text="Invalid Contact", font=(", 10), fg='Red')
            return False
        else:
            self.__con.config(text="")
            self.__dmod.setcon(self.__dcon.get())
            return True
```

```
    def add(self):
        if not self.__dadd.get():
            self.__add.config(text='Address Can Not Be Empty.', font=('', 10), fg='Red')
            return False
        else:
            self.__add.config(text='')
            self.__dmod.setadd(self.__dadd.get())
            return True

    def license(self):
        if not self.__dlicense.get():
            self.__dlicenselb.config(text='License Can Not Be Empty.', font=("", 10), fg="Red")
            return False
        else:
            self.__dlicenselb.config(text='')
            self.__dmod.setlicense(self.__dlicense.get())
            return True

    def vehicle(self):
        if not self.__dvehiclevar.get():
            self.__vehicle.config(text='Select A Vehicle', font=("", 10), fg="Red")
            return False
        else:
            self.__vehicle.config(text='')
            vehicleid = self.__vmod.getid(self.__dvehiclevar.get())
            for data in vehicleid:
                vid = data[0]
                self.__dmod.setvehicle(vid)
                return True

    def em(self):
        email = self.__demail.get().lower()
        if not email:
            self.__demlb.config(text="Email Can Not Be Empty.", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__emregex, email):
            self.__demlb.config(text="Enter Correct Email", font=("", 10), fg="Red")
            return False
        else:
            self.__demlb.config(text="")
            self.__dmod.setem(email)
            return True

    def pas(self):
        if not self.__dpass.get():
            self.__dpasslb.config(text="Password Can Not Be Empty.", font=("", 10), fg="Red")
            return False
```

```
        elif not re.match(self.__passregex, self.__dpass.get()):
            showinfo('Message', "Password must contain 8 letters, a capital letter, a small letter, a number and a "
                                "symbol")
            self.__dpasslb.config(text="Invalid Password", font=("", 10), fg="Red")
            return False
        else:
            self.__dpasslb.config(text="")
            self.__dmod.setpas(self.__dpass.get())
            return True

    # closing the top level if wrong click
    def close(self):
        self.__window.destroy()
```

```
"""
    Module View
    This Is A Frontend Which User Sees And Input Data
"""

# importing required modules
import re
from tkinter import *
from Model.StaffModel import StaffModel
from tkinter.messagebox import showinfo, showerror


# creating class admin login
class StaffLogin:

    # creating frames and placing in the tk
    def __init__(self, window, controller):

        self.__id = None
        self.__mod = StaffModel()
        self.__window = window
        self.__controller = controller
        self.__window.geometry('450x300+450+200')
        self.__window.title("Admin Login")
        self.__emregex = ('^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$')  # regex for email only
```

```
self.__passregex = ("^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$")  # regex for password only

mainf = Frame(self.__window, bg="Light Grey")
mainf.pack(fill="both", expand=True)

title = Frame(mainf, bg="Light Grey")
title.pack(expand=True)

titlelb = Label(title, text="Login To S&S Taxi Service", font=("", 30, "bold"), bg="Light Grey", fg="Black")
titlelb.pack(pady=20)

lbtxt = Frame(mainf, bg="Light Grey")
lbtxt.pack(expand=True, anchor="center")

empw = Frame(lbtxt, bg="Light Grey")
empw.pack(padx=20, side="left")

em = Label(empw, text="Email : ", font=("", 20, "bold"), bg="Light Grey", fg="Black")
em.pack()
Label(empw, bg="light Grey").pack()
pw = Label(empw, text="Password : ", font=("", 20, "bold"), bg="Light Grey", fg="Black")
pw.pack()
Label(empw, bg="light Grey").pack()

txtfield = Frame(lbtxt, bg="Light Grey")
txtfield.pack(padx=20, side="right")

self.emtext = Entry(txtfield, bg="White", fg="Black", font=("", 20))
self.emtext.insert(0, "Enter your email")
self.emtext.pack()
self.emtext.bind('<FocusIn>', self.clear_text)

self.emt = Label(txtfield, font=("", 10), bg="Light Grey", fg="Black")
self.emt.pack()

self.__pwtext = Entry(txtfield, bg="White", fg="Black", font=("", 20))
self.__pwtext.insert(0, "Enter your password")
```

```
        self.__pwtext.pack()
        self.__pwtext.bind('<FocusIn>', self.clear_text1)

        self.emt1 = Label(txtfield, font=("", 10), bg="Light Grey", fg="Black")
        self.emt1.pack()

        log = Frame(mainf, bg="Light Grey")
        log.pack()

        self.checkvar = IntVar()
        checkbox = Checkbutton(log, text="Show Password", variable=self.checkvar, onvalue=1, offvalue=0,
                               bg="Light Grey", fg="Black")
        checkbox.pack(pady=10)
        checkbox.bind('<Button-1>', self.showpass)

        button = Frame(mainf, bg='Light Grey')
        button.pack(expand=True)

        btn = Frame(button, bg='Light Grey')
        btn.pack(side='left', padx=30)

        log = Button(btn, text='Login', bg='Light Grey', fg="Black", bd=0, command=self.verify)
        log.pack(anchor='center')

        back = Frame(button, bg='Light Grey')
        back.pack(side='right', padx=40)

        wrc = Label(back, text='Wrong Click', bg='Light Grey', fg='Black')
        wrc.pack(side='left', padx=10)

        back_btn = Button(back, text='Back', bg='Light Grey', fg='Black', bd=0, command=self.__controller.log)
        back_btn.pack(side='left')

        self.__window.bind('<Return>', self.callverify)

    # calling verify
    def callverify(self, _):
```

```
        self.verify()

    # verifying the data entered
    def verify(self):
        a = self.em()
        b = self.pas()
        if a and b:
            record = self.__mod.staff()
            if record:
                for data in record:
                    self.__id = data[0]
                    fname = data[1]
                    lname = data[2]
                    message = "Welcome " + lname + " " + fname
                    showinfo("Message", message)
                self.__controller.admindash(self.__id)
            else:
                showerror('Invalid', "Invalid Email or Password")

    # verifying the email
    def em(self):
        email = self.emtext.get().lower()
        if not self.emtext.get() or self.emtext.get() == "Enter your email":
            self.emt.config(text="Email Can't Be Empty", font=("", 10), fg="Red")
            return False
        elif not re.match(self.__emregex, email):
            self.emt.config(text="Invalid Email", font=("", 10), fg="Red")
            return False
        else:
            self.emt.config(text="")
            self.__mod.setem(email)
            return True

    # verifying the password
    def pas(self):
        if not self.__pwtext.get() or self.__pwtext.get() == "Enter your password":
            self.emt1.config(text="Password Can't Be Empty", font=("", 10), fg="Red")
```

```
        return False
    elif not re.match(self.__passregex, self.__pwtext.get()):
        self.emt1.config(text="Invalid Password", font=("", 10), fg="Red")
        return False
    else:
        self.emt1.config(text="")
        self.__mod.setpas(self.__pwtext.get())
        return True

# clearing email textfield
def clear_text(self, event):
    if self.emtext.get() == "Enter your email":
        self.emtext.delete(0, END)
    if not self.__pwtext.get():
        self.__pwtext.insert(0, "Enter your password")
        self.__pwtext.config(show="")

# clearing password textfield
def clear_text1(self, event):
    if self.__pwtext.get() == "Enter your password":
        self.__pwtext.delete(0, END)
        if not self.checkvar.get():
            self.__pwtext.config(show="*")
    if not self.emtext.get():
        self.emtext.insert(0, "Enter your email")

# showing and hiding password
def showpass(self, event):
    if self.__pwtext.get() != "Enter your password":
        if self.checkvar.get():
            self.__pwtext.config(show="*")
        else:
            self.__pwtext.config(show="")
    else:
        showinfo("Message", "Password not entered")
        if self.checkvar.get():
            self.checkvar.set(1)
```

```
        else:
            self.checkvar.set(0)
```

```
"""
    Module Utilis
    This Connects With The Database
"""

# importing required modules
import psycopg2 as db
from psycopg2 import OperationalError


class DatabaseConnection:
    __conn = None
    __cur = None

    def __init__(self):

        # loading details of database
        self.host = 'localhost'
        self.db = 'xic'
        self.user = 'xic'
        self.port = 5432

        # connect to database
        self.__connect()
        self.__dbcur = DatabaseConnection.__cur
        self.__dbconn = DatabaseConnection.__conn

    def __connect(self):
        try:
            if DatabaseConnection.__conn is None:
                DatabaseConnection.__conn = db.connect(database=self.db, user=self.user,
                                                       host=self.host, port=self.port)
                DatabaseConnection.__conn.autocommit = True
                DatabaseConnection.__cur = DatabaseConnection.__conn.cursor()
```

```
        except OperationalError as e:
            raise e

    @property
    def cursor(self):
        return self.__dbcur

    def close(self):
        self.__dbconn.close()
```

```
"""
    Module Controller
    This Controls The Flow Of The Windows
"""
# importing required modules
from tkinter import Toplevel
from View.LoginPage import LoginPage
from View.StaffLogin import StaffLogin
from View.StaffDashboard import StaffHome
from View.DriverDashboard import DriverHome
from View.RegistrationPage import RegistrationPage
from View.CustDashboard import CustDashboard, BookingPage


# creating class controller
class Controller:

    # creating login page
    def __init__(self, root):
        self.__root = root
        self.__root.withdraw()
        self.__clear_root()
        lg = Toplevel()
        log = LoginPage(lg, self)

    # creating registration page
    def reg(self):
```

```
        self.__clear_root()
        rg = Toplevel()
        reg = RegistrationPage(rg, self)

    # creating login page
    def log(self):
        lg = Controller(self.__root)

    # creating customer home page
    def custlog(self, custid):
        self.__clear_root()
        dash = CustDashboard(self.__root, self, custid)

    # creating profile page
    def book(self, custid):
        self.__clear_root()
        bk = BookingPage(self.__root, self, custid)

    # creating admin login page
    def admin(self):
        self.__clear_root()
        add = Toplevel()
        ad = StaffLogin(add, self)

    # creating admin home page
    def admindash(self, stid):
        self.__clear_root()
        addh = StaffHome(self.__root, self, stid)

    # creating admin home page
    def driverdash(self, did):
        self.__clear_root()
        ddh = DriverHome(self.__root, self, did)

    # clearing the tk window
    def __clear_root(self):
        for child in self.__root.winfo_children():
```

```
        child.destroy()
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection


# declaring model class
class CustomerModel:

    def __init__(self):

        # declaring variables to be used
        self.__custid = None
        self.__fn = None
        self.__ln = None
        self.__date = None
        self.__gen = None
        self.__add = None
        self.__con = None
        self.__em = None
        self.__pas = None
        self.__cur = DatabaseConnection().cursor

    # getters for customer
    def getcustid(self):
        return self.__custid

    def getfn(self):
        return self.__fn

    def getln(self):
        return self.__ln
```

```
    def getdate(self):
        return self.__date

    def getgen(self):
        return self.__gen

    def getadd(self):
        return self.__add

    def getcon(self):
        return self.__con

    def getem(self):
        return self.__em

    def getpas(self):
        return self.__pas

    # setter for customer
    def setcustid(self, id):
        self.__custid = id

    def setfn(self, fn):
        self.__fn = fn

    def setln(self, ln):
        self.__ln = ln

    def setdate(self, date):
        self.__date = date

    def setgen(self, gen):
        self.__gen = gen

    def setadd(self, add):
        self.__add = add
```

```
    def setcon(self, con):
        self.__con = con

    def setem(self, em):
        self.__em = em

    def setpas(self, pas):
        self.__pas = pas

    # select all customers
    def allcust(self):
        query = """SELECT * FROM CUSTOMER"""
        self.__cur.execute(query)
        record = self.__cur.fetchall()
        if record:
            return record

    # getting details
    def details(self, custid):
        query = """SELECT * FROM CUSTOMER WHERE CUSTID=%s ORDER BY CUSTID"""
        self.__cur.execute(query, [custid])
        record = self.__cur.fetchall()
        if record:
            return record

    # updating details
    def updatedetatil(self, custid):
        query = """UPDATE CUSTOMER SET FIRSTNAME=%s, LASTNAME=%s, ADDRESS=%s, CONTACT=%s WHERE
CUSTID=%s"""
        values = (self.getfn(), self.getln(), self.getadd(), self.getcon(), custid)
        self.__cur.execute(query, values)
        return True

    # searching for customer
    def search(self, name, add):
```

```
        query = """SELECT * FROM CUSTOMER WHERE CONCAT(FIRSTNAME,' ', LASTNAME) ILIKE CONCAT('%%',%s,'%%')
AND
        ADDRESS ILIKE CONCAT('%%',%s,'%%')"""
        value = (name, add)
        self.__cur.execute(query, value)
        record = self.__cur.fetchall()
        if record:
            return record

    # deleting one customer
    def delete(self, custid):
        query = """DELETE FROM CUSTOMER WHERE CUSTID=%s"""
        self.__cur.execute(query, [custid])
        return True
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection


# declaring model class
class DriverModel:

    def __init__(self):
        self.__drid = None
        self.__fn = None
        self.__ln = None
        self.__date = None
        self.__gen = None
        self.__add = None
        self.__con = None
        self.__license = None
        self.__vehicle = None
```

```
        self.__em = None
        self.__pas = None
        self.__cur = DatabaseConnection().cursor

    # getters for customer
    def getdrid(self):
        return self.__drid

    def getfn(self):
        return self.__fn

    def getln(self):
        return self.__ln

    def getdate(self):
        return self.__date

    def getgen(self):
        return self.__gen

    def getadd(self):
        return self.__add

    def getcon(self):
        return self.__con

    def getlicense(self):
        return self.__license

    def getvehicle(self):
        return self.__vehicle

    def getem(self):
        return self.__em

    def getpas(self):
        return self.__pas
```

```
# setter for customer
def setdrid(self, did):
    self.__drid = did

def setfn(self, fn):
    self.__fn = fn

def setln(self, ln):
    self.__ln = ln

def setdate(self, date):
    self.__date = date

def setgen(self, gen):
    self.__gen = gen

def setadd(self, add):
    self.__add = add

def setcon(self, con):
    self.__con = con

def setlicense(self, no):
    self.__license = no

def setvehicle(self, id):
    self.__vehicle = id

def setem(self, em):
    self.__em = em

def setpas(self, pas):
    self.__pas = pas

# creating account for Driver
def create(self):
```

```
        query = """INSERT INTO DRIVER(FIRSTNAME, LASTNAME, DOB, GENDER, ADDRESS, CONTACT, LICENSENO, EMAIL,
PASS,
        VEHICLEID)
        VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
        values = (self.getfn(), self.getln(), self.getdate(), self.getgen(), self.getadd(), self.getcon(),
                  self.getlicense(), self.getem(), self.getpas(), self.getvehicle())
        self.__cur.execute(query, values)
        return True

    # selecting all driver
    def alldriver(self):
        query = """SELECT * FROM DRIVER ORDER BY DRIVERID"""
        self.__cur.execute(query)
        record = self.__cur.fetchall()
        if record:
            return record

    # selecting all driver who are empty on particular day
    def emptydriver(self, when):
        query = """SELECT * FROM DRIVER WHERE DRIVERID NOT IN (SELECT DRIVERID FROM TRIP WHERE TRIPDATE=%s
AND
        DRIVERID IS NOT NULL ORDER BY DRIVERID)"""
        self.__cur.execute(query, [when])
        record = self.__cur.fetchall()
        if record:
            return record

    # returning id from name
    def getid(self, name):
        query = """SELECT DRIVERID FROM DRIVER WHERE CONCAT(LASTNAME,' ', FIRSTNAME) ILIKE
CONCAT('%%',%s,'%%')"""
        self.__cur.execute(query, [name])
        record = self.__cur.fetchall()
        if record:
            return record

    # deleting the driver details
```

```
    def delete(self):
        query = """DELETE FROM DRIVER WHERE DRIVERID=%s"""
        self.__cur.execute(query, [self.getdrid()])
        return True
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection


# declaring model class
class LoginModel:

    def __init__(self):
        self.__em = None
        self.__pas = None
        self.__id = None
        self.__cur = DatabaseConnection().cursor

    # gettter
    def getem(self):
        return self.__em

    def getpas(self):
        return self.__pas

    # setter
    def setem(self, em):
        self.__em = em

    def setpas(self, pas):
        self.__pas = pas
```

```
    # login as customer
    def cust(self):
        query = """SELECT * FROM CUSTOMER WHERE EMAIL=%s AND PASS=%s"""
        value = (self.getem(), self.getpas())
        self.__cur.execute(query, value)
        record = self.__cur.fetchall()
        if record:
            return record

    # login as driver
    def driver(self):
        query = """SELECT * FROM DRIVER WHERE EMAIL=%s AND PASS=%s"""
        value = (self.getem(), self.getpas())
        self.__cur.execute(query, value)
        record = self.__cur.fetchall()
        if record:
            return record
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection


# declaring model class
class RegistrationModel:

    def __init__(self):

        # declaring variables tobe used
        self.__fn = None
        self.__ln = None
        self.__date = None
        self.__gen = None
```

```
        self.__add = None
        self.__con = None
        self.__em = None
        self.__pas = None
        self.__cur = DatabaseConnection().cursor

    # getters for customer
    def getfn(self):
        return self.__fn

    def getln(self):
        return self.__ln

    def getdate(self):
        return self.__date

    def getgen(self):
        return self.__gen

    def getadd(self):
        return self.__add

    def getcon(self):
        return self.__con

    def getem(self):
        return self.__em

    def getpas(self):
        return self.__pas

    # setter for customer
    def setfn(self, fn):
        self.__fn = fn

    def setln(self, ln):
        self.__ln = ln
```

```
    def setdate(self, date):
        self.__date = date

    def setgen(self, gen):
        self.__gen = gen

    def setadd(self, add):
        self.__add = add

    def setcon(self, con):
        self.__con = con

    def setem(self, em):
        self.__em = em

    def setpas(self, pas):
        self.__pas = pas

    # creating account for customer
    def create(self):
        query = """INSERT INTO CUSTOMER(FIRSTNAME, LASTNAME, DOB, GENDER, ADDRESS, CONTACT, EMAIL, PASS)
VALUES(
        %s, %s, %s, %s, %s, %s, %s, %s)"""
        values = (self.getfn(), self.getln(), self.getdate(), self.getgen(), self.getadd(), self.getcon(), self.getem(),
                  self.getpas())
        self.__cur.execute(query, values)
        return True

    # checking for same email
    def double(self, email):
        query = """SELECT * FROM CUSTOMER WHERE EMAIL=%s"""
        self.__cur.execute(query, [email])
        record = self.__cur.fetchall()
        if record:
            return True
```

```
        query = """SELECT * FROM STAFF WHERE EMAIL=%s"""
        self.__cur.execute(query, [email])
        record = self.__cur.fetchall()
        if record:
            return True

        query = """SELECT * FROM DRIVER WHERE EMAIL=%s"""
        self.__cur.execute(query, [email])
        record = self.__cur.fetchall()
        if record:
            return True
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection


# declaring model class
class StaffModel:
    def __init__(self):
        self.__stid = None
        self.__fn = None
        self.__ln = None
        self.__date = None
        self.__gen = None
        self.__add = None
        self.__con = None
        self.__em = None
        self.__pas = None
        self.__cur = DatabaseConnection().cursor

    # getters for customer
    def getstid(self):
```

```
        return self.__stid

    def getfn(self):
        return self.__fn

    def getln(self):
        return self.__ln

    def getdate(self):
        return self.__date

    def getgen(self):
        return self.__gen

    def getadd(self):
        return self.__add

    def getcon(self):
        return self.__con

    def getem(self):
        return self.__em

    def getpas(self):
        return self.__pas

    # setter for customer
    def setstid(self, stid):
        self.__stid = stid

    def setfn(self, fn):
        self.__fn = fn

    def setln(self, ln):
        self.__ln = ln

    def setdate(self, date):
```

```
        self.__date = date

    def setgen(self, gen):
        self.__gen = gen

    def setadd(self, add):
        self.__add = add

    def setcon(self, con):
        self.__con = con

    def setem(self, em):
        self.__em = em

    def setpas(self, pas):
        self.__pas = pas

    # login as staff
    def staff(self):
        query = """SELECT * FROM STAFF WHERE EMAIL=%s AND PASS=%s"""
        value = (self.getem(), self.getpas())
        self.__cur.execute(query, value)
        record = self.__cur.fetchall()
        if record:
            return record

    # creating account for staff
    def create(self):
        query = """INSERT INTO STAFF(FIRSTNAME, LASTNAME, DOB, GENDER, ADDRESS, CONTACT, EMAIL, PASS)
        VALUES(%s, %s, %s, %s, %s, %s, %s, %s)"""
        values = (self.getfn(), self.getln(), self.getdate(), self.getgen(), self.getadd(), self.getcon(),
                  self.getem(), self.getpas())
        self.__cur.execute(query, values)
        return True

    # showing all staff detail
    def allstaff(self):
```

```
        query = """SELECT * FROM STAFF ORDER BY STAFFID"""
        self.__cur.execute(query)
        record = self.__cur.fetchall()
        if record:
            return record

    # deleting staff details
    def delete(self, stid):
        query = """DELETE FROM STAFF WHERE STAFFID=%s"""
        self.__cur.execute(query, [stid])
        return True
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from Utilis.DatabaseConnector import DatabaseConnection
from datetime import date


# declaring class trip model
class TripModel:

    def __init__(self):
        self.__custid = None
        self.__pickloc = None
        self.__droploc = None
        self.__tripdate = None
        self.__picktime = None
        self.__passno = None
        self.__cost = None
        self.__status = None
        self.__distance = None
        self.__cur = DatabaseConnection().cursor
```

```
    # getters
    def getcustid(self):
        return self.__custid

    def getpickloc(self):
        return self.__pickloc

    def getdroploc(self):
        return self.__droploc

    def gettripdate(self):
        return self.__tripdate

    def getpicktime(self):
        return self.__picktime

    def getpassno(self):
        return self.__passno

    def getcost(self):
        return self.__cost

    def getstatus(self):
        return self.__status

    def getdistance(self):
        return self.__distance

    # setter
    def setcustid(self, custid):
        self.__custid = custid

    def setpickloc(self, pickloc):
        self.__pickloc = pickloc

    def setdroploc(self, droploc):
        self.__droploc = droploc
```

```
    def settripdate(self, tripdate):
        self.__tripdate = tripdate

    def setpicktime(self, picktime):
        self.__picktime = picktime

    def setpassno(self, passno):
        self.__passno = passno

    def setcost(self, cost):
        self.__cost = cost

    def setdistance(self, dis):
        self.__distance = dis

    # book a trip
    def booktrip(self):
        now = str(date.today())
        query = """INSERT INTO TRIP(CUSTID, BOOKINGDATE, PICKUPLOC, DROPLOC, TRIPDATE, PICKUPTIME,
NOOFPASS, TOTALCOST,
        TRIPSTATUS, PAYMENT_METHOD, PAYMENT_STATUS, DISTANCE) VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s)"""
        values = (self.getcustid(), now, self.getpickloc(), self.getdroploc(), self.gettripdate(), self.getpicktime(),
                  self.getpassno(), self.getcost(), "Pending", "Cash", "Pending", self.getdistance())
        self.__cur.execute(query, values)
        return True

    # showing trip of a customer
    def custrip(self, custid):
        query = """SELECT * FROM TRIP WHERE CUSTID=%s ORDER BY TRIPDATE"""
        self.__cur.execute(query, [custid])
        record = self.__cur.fetchall()
        if record:
            return record

    # showing all trip assigned to a driver with customer details
```

```
    def driverti(self, did):
        query = """SELECT * FROM CUSTOMER C JOIN TRIP T ON T.CUSTID=C.CUSTID WHERE T.DRIVERID=%s ORDER BY
T.TRIPDATE"""
        self.__cur.execute(query, [did])
        record = self.__cur.fetchall()
        if record:
            return record

    # starting the trip
    def startrip(self, tripid):
        query = """UPDATE TRIP SET TRIPSTATUS=%s WHERE TRIPID=%s"""
        values = ("Started", tripid)
        self.__cur.execute(query, values)
        return True

    # marking trip completed
    def completetrip(self, tripid):
        query = """UPDATE TRIP SET TRIPSTATUS=%s, PAYMENT_STATUS=%s WHERE TRIPID=%s"""
        values = ("Completed", "Paid", tripid)
        self.__cur.execute(query, values)
        return True

    # showing all trips
    def alltrip(self):
        query = """SELECT * FROM TRIP T JOIN CUSTOMER C ON T.CUSTID=C.CUSTID ORDER BY T.TRIPDATE"""
        self.__cur.execute(query)
        record = self.__cur.fetchall()
        if record:
            return record

    # showing trip detail where trip status is cancelled or pending
    def tripdetail(self, tripid):
        query = """SELECT * FROM TRIP WHERE TRIPID=%s ORDER BY TRIPDATE"""
        self.__cur.execute(query, [tripid])
        record = self.__cur.fetchall()
        if record:
            return record
```

```
    # showing all details related to trip where status is not cancelled or pending
    def tripdetails(self, tripid):
        query = """SELECT * FROM TRIP T JOIN DRIVER D ON D.DRIVERID=T.DRIVERID JOIN VEHICLE V ON
        D.VEHICLEID=V.VEHICLEID WHERE T.TRIPID=%s ORDER BY T.TRIPDATE"""
        self.__cur.execute(query, [tripid])
        record = self.__cur.fetchall()
        if record:
            return record

    # showing all pending trips
    def pendingtrip(self):
        query = """SELECT * FROM TRIP T JOIN CUSTOMER C ON T.CUSTID=C.CUSTID WHERE TRIPSTATUS=%s ORDER BY
T.TRIPDATE"""
        self.__cur.execute(query, ['Pending'])
        record = self.__cur.fetchall()
        if record:
            return record

    # assign driver and making trip status confirmed
    def assigndr(self, stid, did, tpid):
        query = """UPDATE TRIP SET STAFFID=%s, DRIVERID=%s, TRIPSTATUS=%s WHERE TRIPID=%s"""
        values = (stid, did, "Confirmed", tpid)
        self.__cur.execute(query, values)
        return True

    # showing all today's trip
    def todaytrip(self):
        now = date.today()
        query = """SELECT * FROM TRIP T JOIN CUSTOMER C ON T.CUSTID=C.CUSTID WHERE TRIPDATE=%s ORDER BY
T.TRIPDATE"""
        self.__cur.execute(query, [str(now)])
        record = self.__cur.fetchall()
        if record:
            return record

    # cancelling selected trip
```

```
    def canceltrip(self, tripid):
        query = """UPDATE TRIP SET TRIPSTATUS=%s WHERE TRIPID=%s"""
        values = ("Cancelled", tripid)
        self.__cur.execute(query, values)
        return True

    # deleting selected trip
    def deletetrip(self, tripid):
        query = """DELETE FROM TRIP WHERE TRIPID=%s"""
        self.__cur.execute(query, [tripid])
        return True
```

```
"""
    Module Model
    This Connects With The Database Connector
"""

# importing required modules
from datetime import date
from Utilis.DatabaseConnector import DatabaseConnection


# declaring vehicle model class
class VehicleModel:

    def __init__(self):
        self.__vehicleid = None
        self.__vehicleno = None
        self.__vehicletype = None
        self.__vehiclemodel = None
        self.__vehicledate = None
        self.__description = None
        self.__vehiclestatus = None
        self.__cur = DatabaseConnection().cursor

    # getters
    def getvehicleid(self):
```

```
        return self.__vehicleid

    def getvehicleno(self):
        return self.__vehicleno

    def getvehicletype(self):
        return self.__vehicletype

    def getvehiclemodel(self):
        return self.__vehiclemodel

    def getvehicledate(self):
        return self.__vehicledate

    def getdescription(self):
        return self.__description

    def getstatus(self):
        return self.__vehiclestatus

    # setter
    def setvehicleid(self, vid):
        self.__vehicleid = vid

    def setvehicleno(self, vno):
        self.__vehicleno = vno

    def setvehicletype(self, vtype):
        self.__vehicletype = vtype

    def setvehiclemodel(self, mod):
        self.__vehiclemodel = mod

    def setvehicledate(self, date):
        self.__vehicledate = date

    def setdescription(self, des):
```

```
        self.__description = des

    def setstatus(self, stat):
        self.__vehiclestatus = stat

    # inserting data into vehicle
    def addV(self):
        query = """INSERT INTO VEHICLE(VEHICLENO, VEHICLETYPE, VEHICLEMODEL, VEHICLEDATE, DESCRIPTION,
STATUS)
        VALUES(%s, %s, %s, %s, %s, %s)"""
        now = date.today()
        values = (self.getvehicleno(), self.getvehicletype(), self.getvehiclemodel(), now, self.getdescription(),
                  "Not Assigned")
        self.__cur.execute(query, values)
        return True

    def allvehicle(self):
        query = """SELECT * FROM VEHICLE"""
        self.__cur.execute(query)
        record = self.__cur.fetchall()
        if record:
            return record

    # checking for double
    def double(self, vecno):
        query = """SELECT * FROM VEHICLE WHERE VEHICLENO=%s ORDER BY VEHICLEID"""
        self.__cur.execute(query, [vecno])
        record = self.__cur.fetchall()
        if record:
            return True

    # for assigning
    def assigning(self):
        query = """SELECT * FROM VEHICLE WHERE STATUS=%s"""
        self.__cur.execute(query, ["Not Assigned"])
        record = self.__cur.fetchall()
        return record
```

```
# assigning
def assign(self, vehicleno):
    query = """UPDATE VEHICLE SET STATUS=%s WHERE VEHICLENO=%s"""
    value = ("Assigned", vehicleno)
    self.__cur.execute(query, value)
    return True

# returning a vehicleid
def getid(self, vehicleno):
    query="""SELECT VEHICLEID FROM VEHICLE WHERE VEHICLENO=%s"""
    self.__cur.execute(query, [vehicleno])
    record = self.__cur.fetchall()
    if record:
        return record

# deleting vehicle details
def delete(self):
    query = """DELETE FROM VEHICLE WHERE VEHICLEID=%s"""
    self.__cur.execute(query, [self.getvehicleid()])
    return True
```