Table of Content

Taxi Booking System

About Team

Introduction to Project

Nowadays everything is online. Every company has their online ticket purchase system. So I decided to build a small python project of a taxi booking system designed in tkinter(python) GUI. The main purpose of building this taxi booking system for a company or multiple company so that they can book their ticket faster.
If one user books multiple time tickets from the same company. Then the company can also find out the regular customer so that they can give some special kind of offers and discounts. This will improve company revenue.

Work to Do:

I have to create a GUI interface for a taxi booking system with 3 interfaces: admin, Customer, and Driver.

1. Admin
    a. Admin can register New Driver
    b. Admin can cancel the booking if he has no free driver.
    c. Admin can allocate drivers.
    d. View all booking
2. Customer
    a. Register.
    b. Booking taxi.
    c. View all old bookings.
3. Driver
    a. Check their upcoming trips.
    b. Confirm whether the trip is completed or not.

I have completed the above task, and I achieved all the conditions which I have mentioned above.

Task Description:

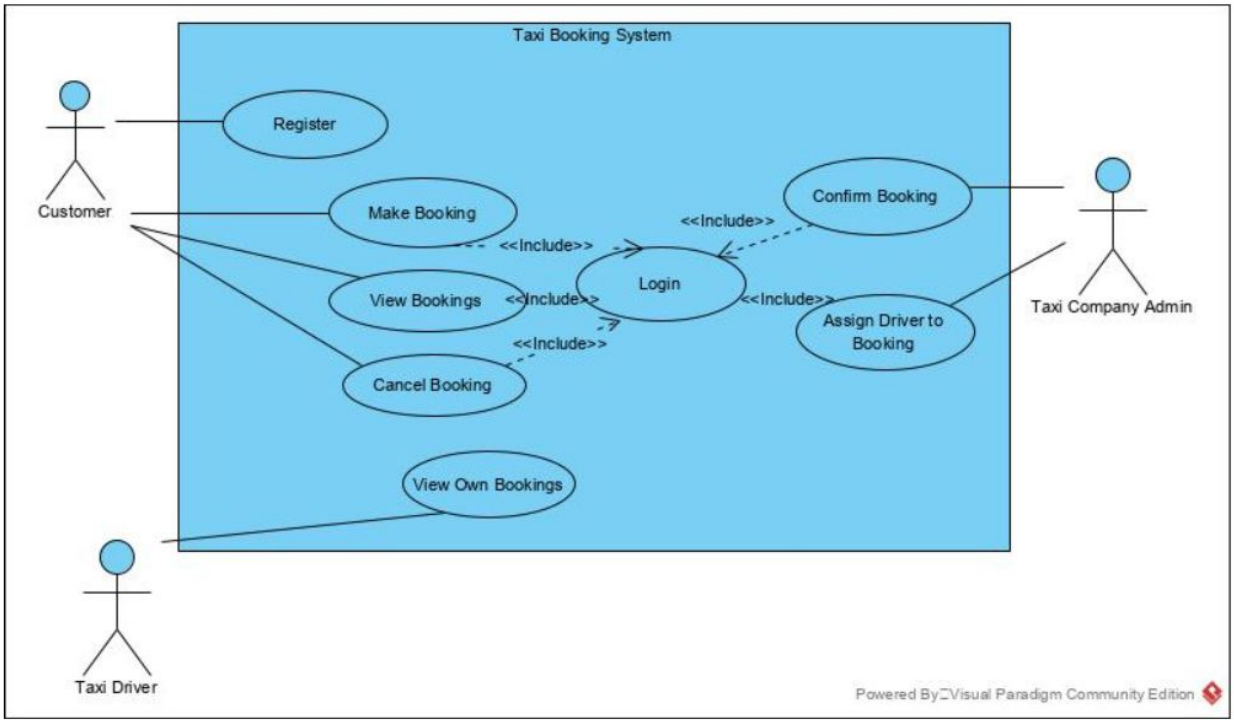
Given task is to create a GUI with following interfaces:

1. Admin
2. Customer
3. Driver

Customers can Register there themselves if he/she is not registered. They can also book their ticket and they can view their pending booking. We have to check if they submit blank fields then it will show an error message that 'all Fields are required'.

Drivers can check there today and upcoming booking through entering their licence number. If their trip is completed then he will update the status of the trip with 'done' that means the trip is finished and charges are taken from the customer.

Admin can view all bookings which have been done by the users and he can also cancel the booking if he has no driver available to allocate him in that case he can cancel the trip of the client. Admin can add a new Driver to the Driver list.



This is an UML diagram which we have to create the interface and the Design of the system.

In Tripinfo table there are 3 status column i have taken so that company can confirm there trips which are completed and which are pending. They are following

1. **Pending ->** Customer has booked now and the driver allocation is not done yet.

2. **Success ->** Admin has allocated drivers to the trip then the status becomes success.
3. **Done ->** when the trip is completed by the driver then the status of trip is finished so he/she will update the status of trip with done.

we have to store following data for **Customer** this are following:

1. Customer Name
2. Customer Address
3. Customer Email
4. Customer Telephone number
5. Customer password

We have to store following data for the **driver**:

1. Licence Number
2. Driver Name

We have to store following data for **admin login**:

1. Username
2. Password

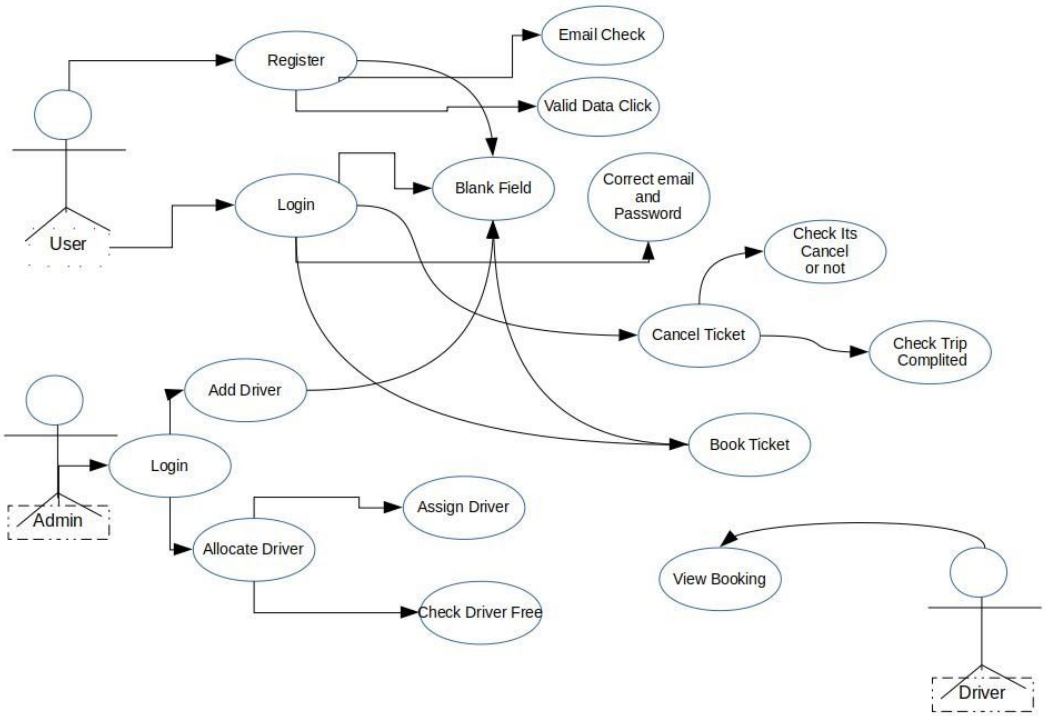
We have to store following data for **trip**:

1. pickUpAddress
2. pickUpDate
3. pickUpTime
4. dropAddress
5. dropDate
6. dropTime
7. Customer id who have booked this
8. Price/km
9. Trip status
10. Driver licence

Design

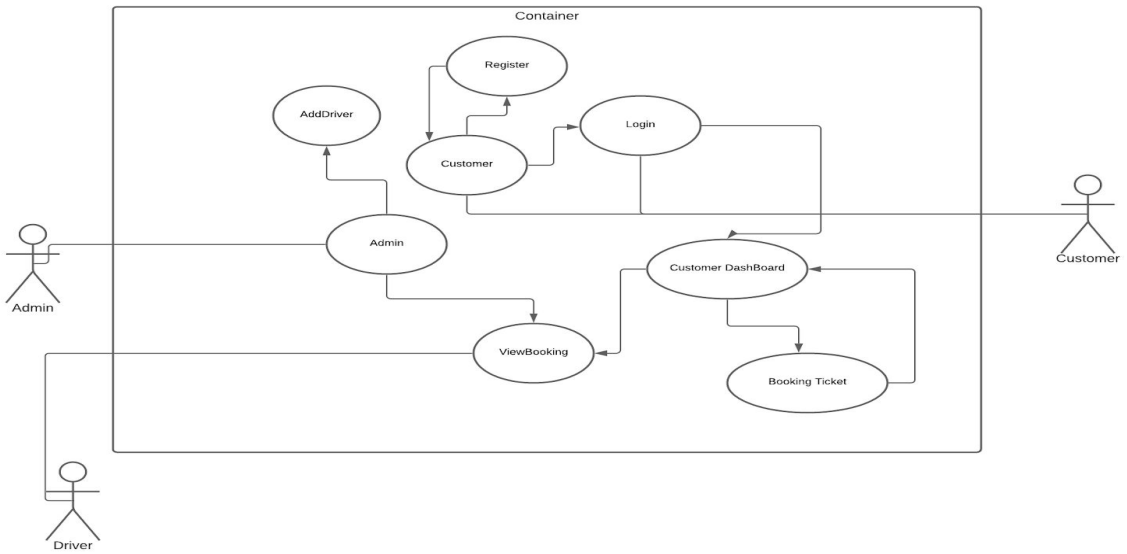
UML Diagram:

1. Use case diagram

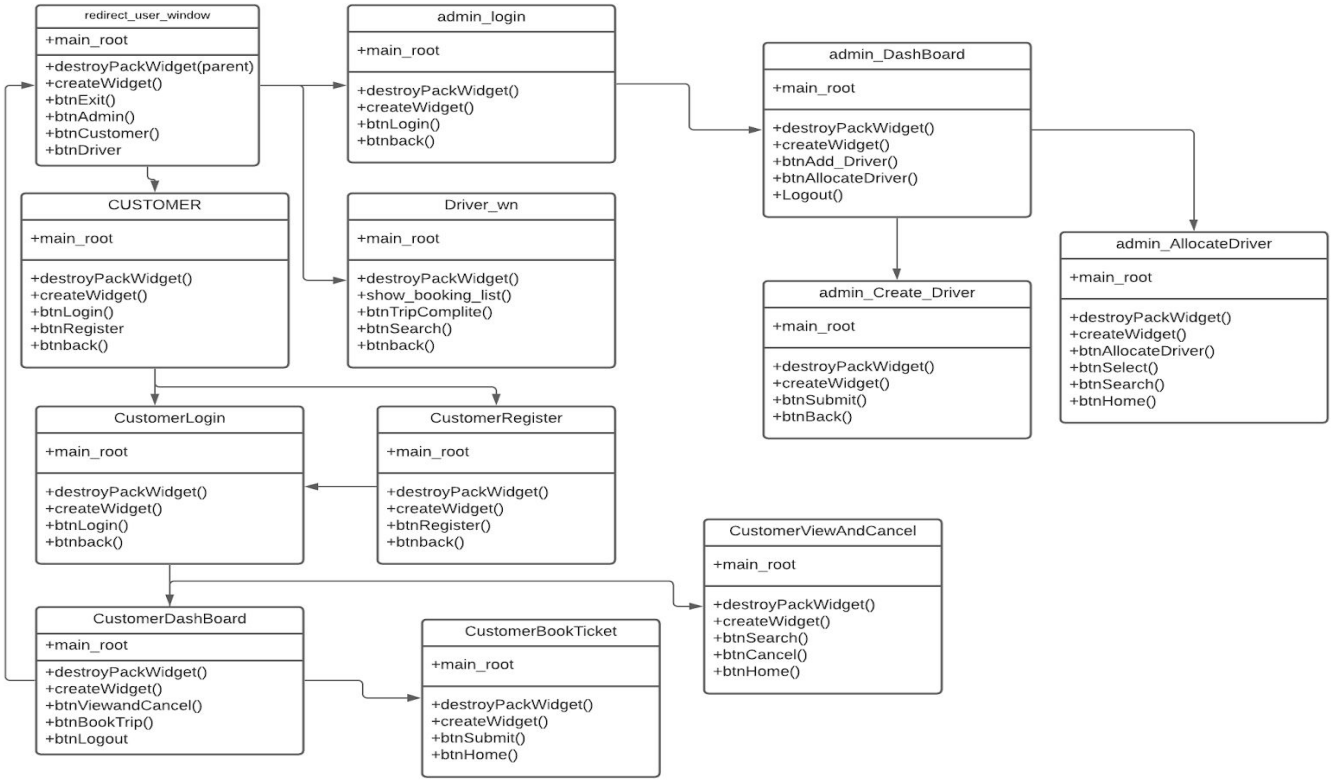


The above diagram is Use case diagram fish level which represent all the cases used in the taxi booking system software

Activity Diagrams

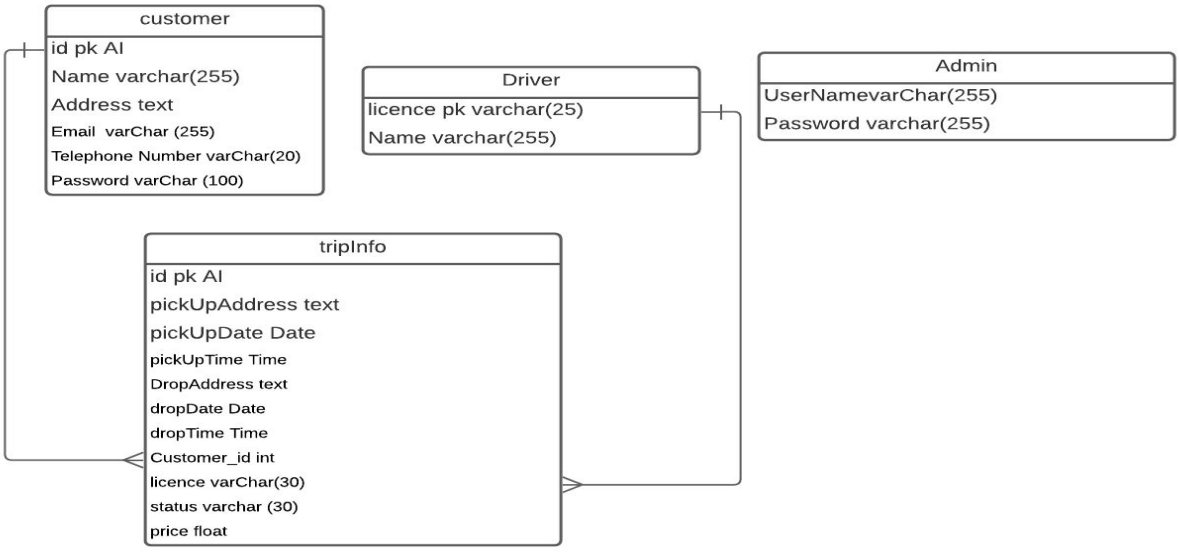


Class Diagram:

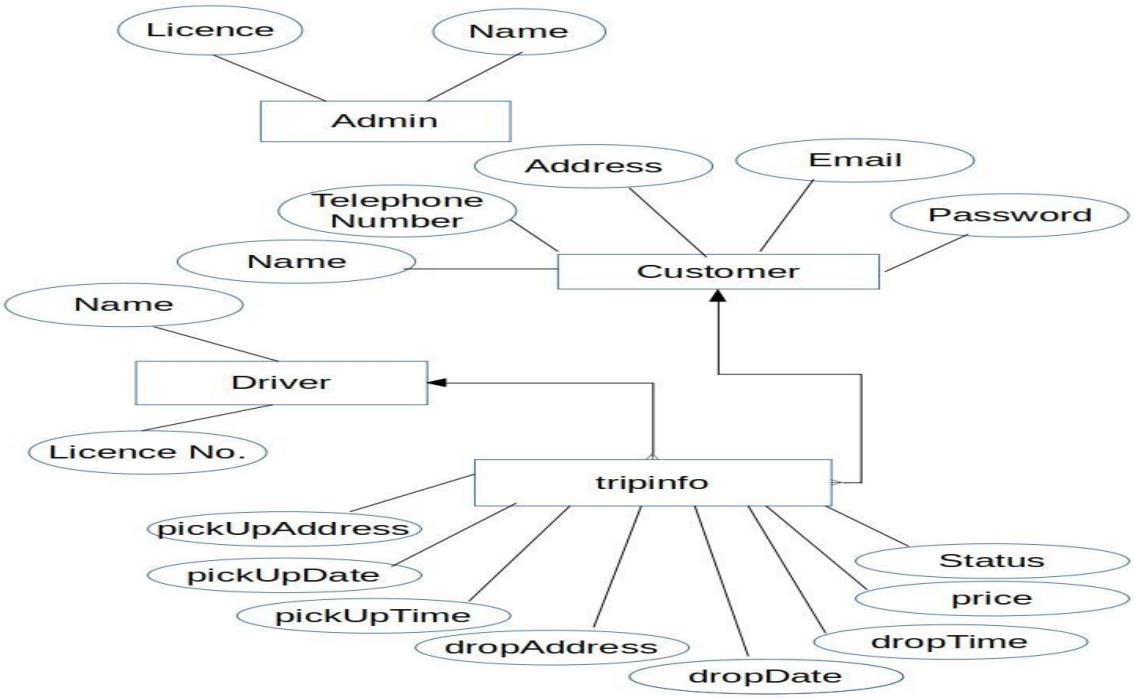


Database Design:

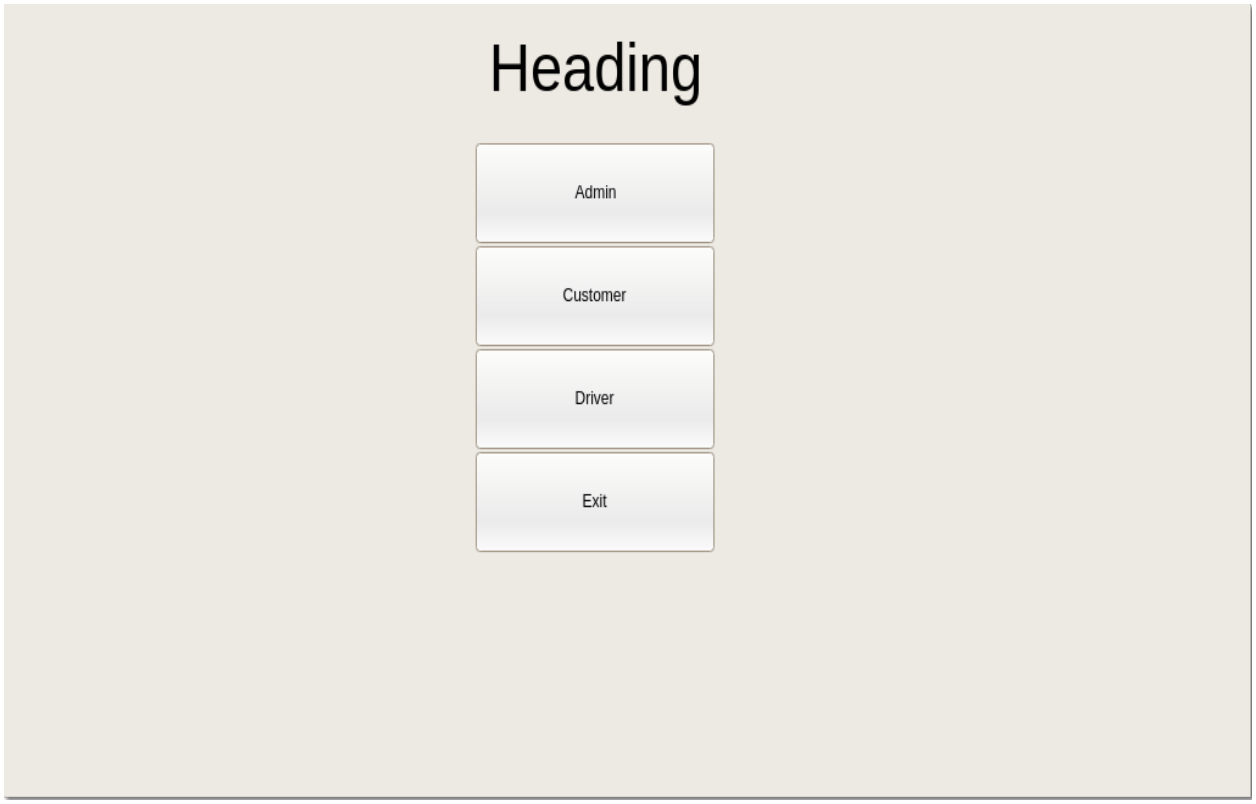
1. ERM Diagram

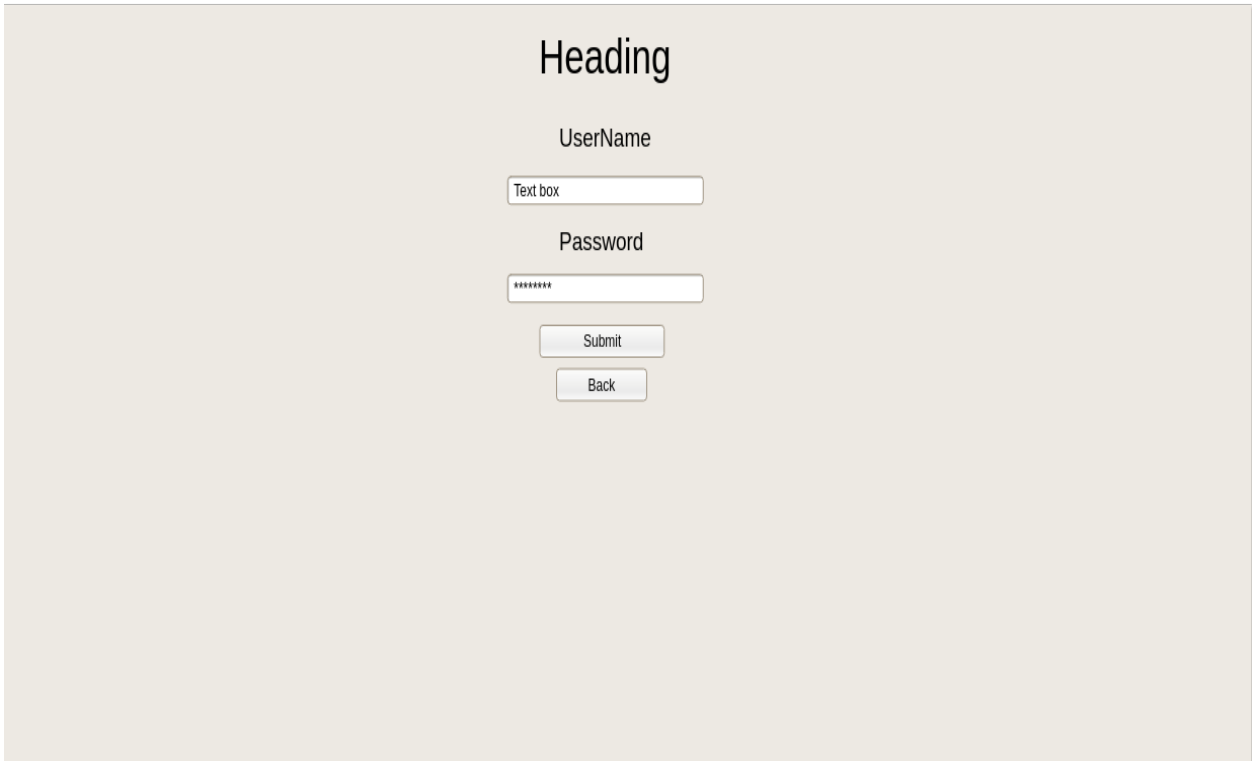


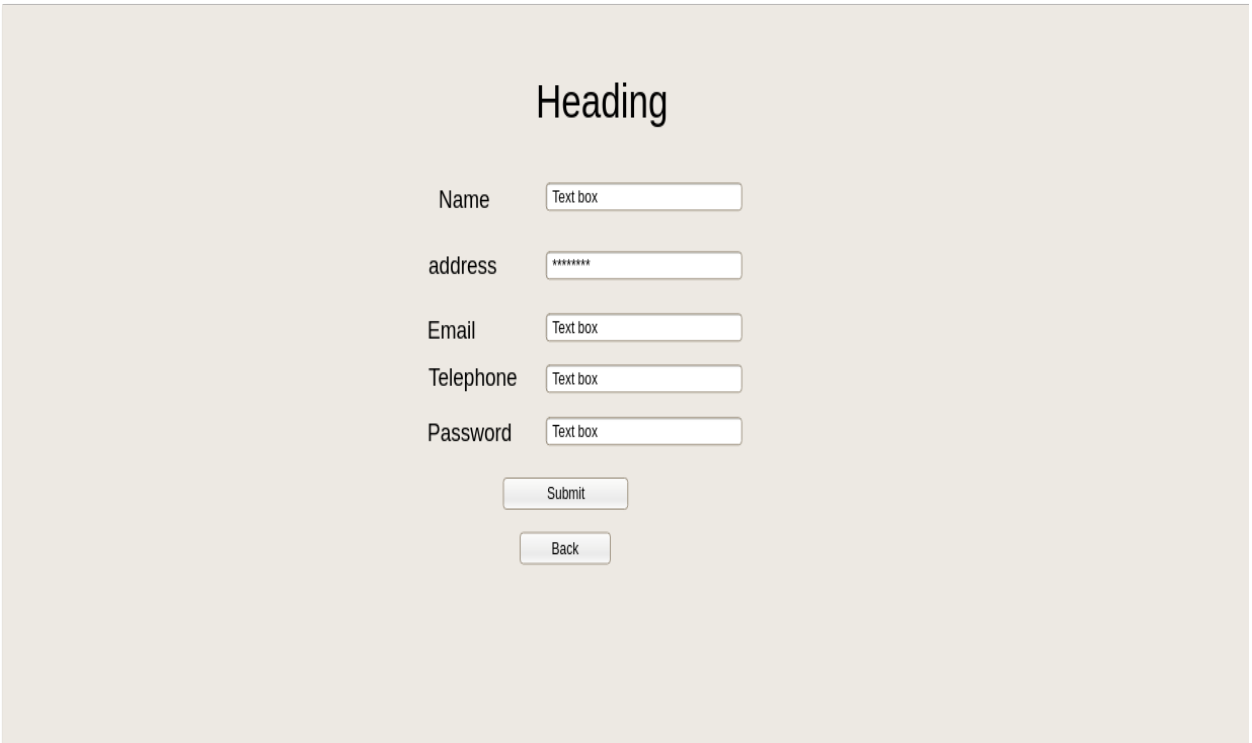
2. **Skeleton Table**

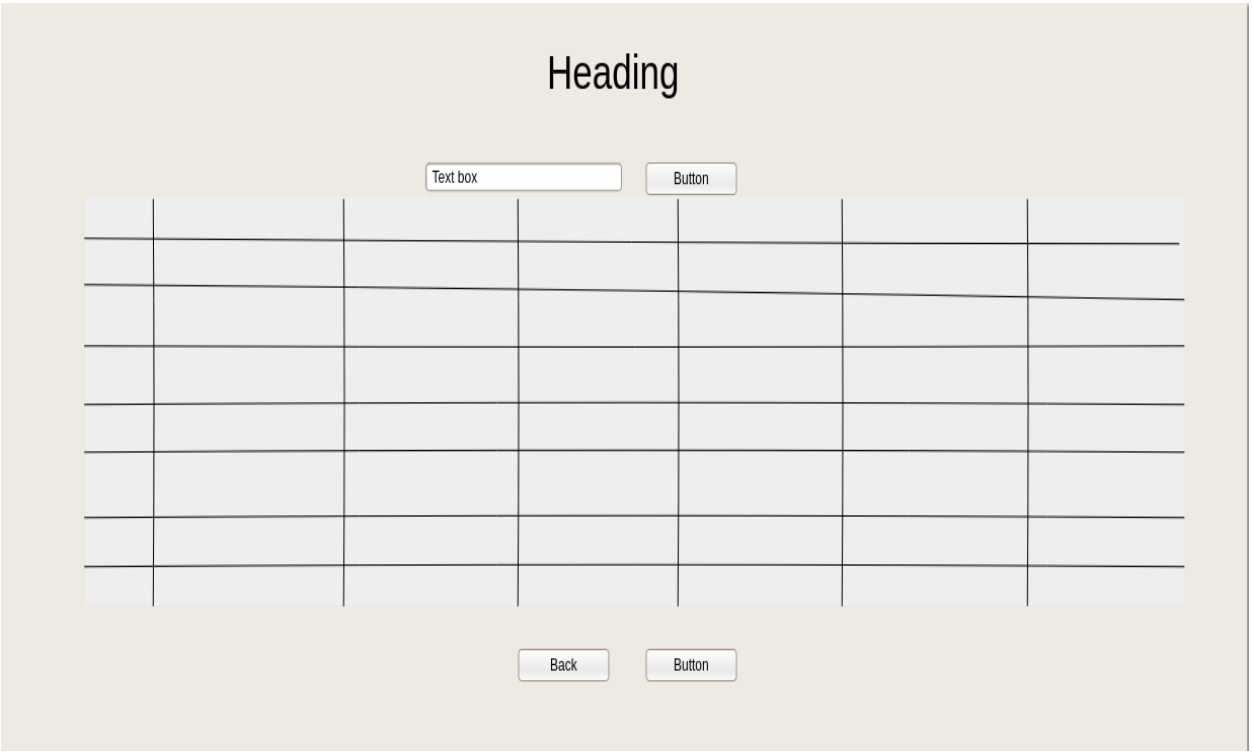


User InterFace Design:






Heading
Name
Text box
address
****
Email
Text box
Telephone
Text box
Password
Text box
Submit
Back

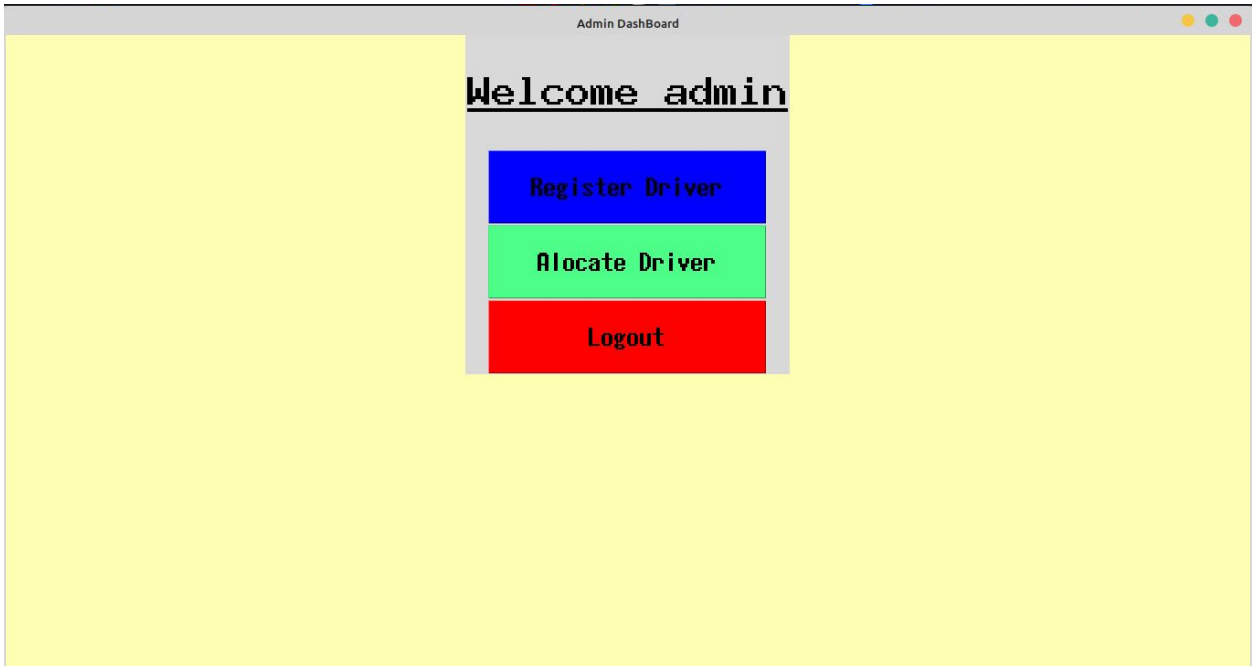

Heading
Text box
Button
Back
Button

Testing

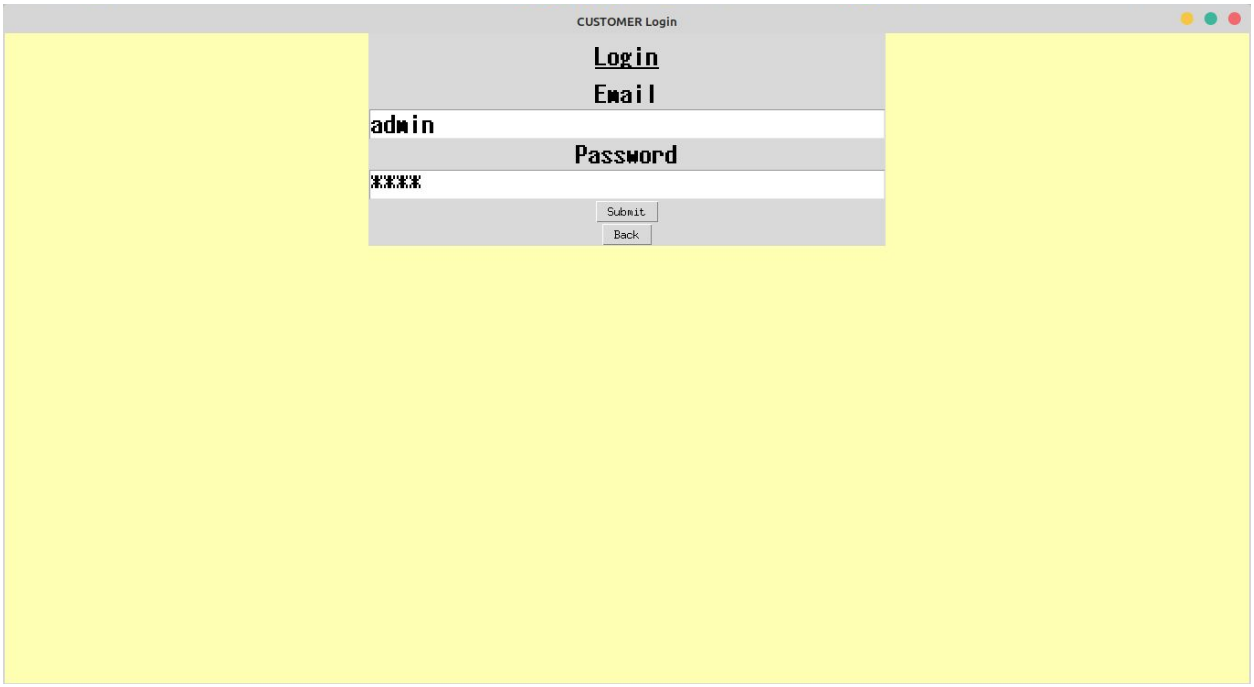
1. Admin login Test

Above given username and password is incorrect the output should be “username or password invalid”

2. When we give correct username and password output should be admin dashboard



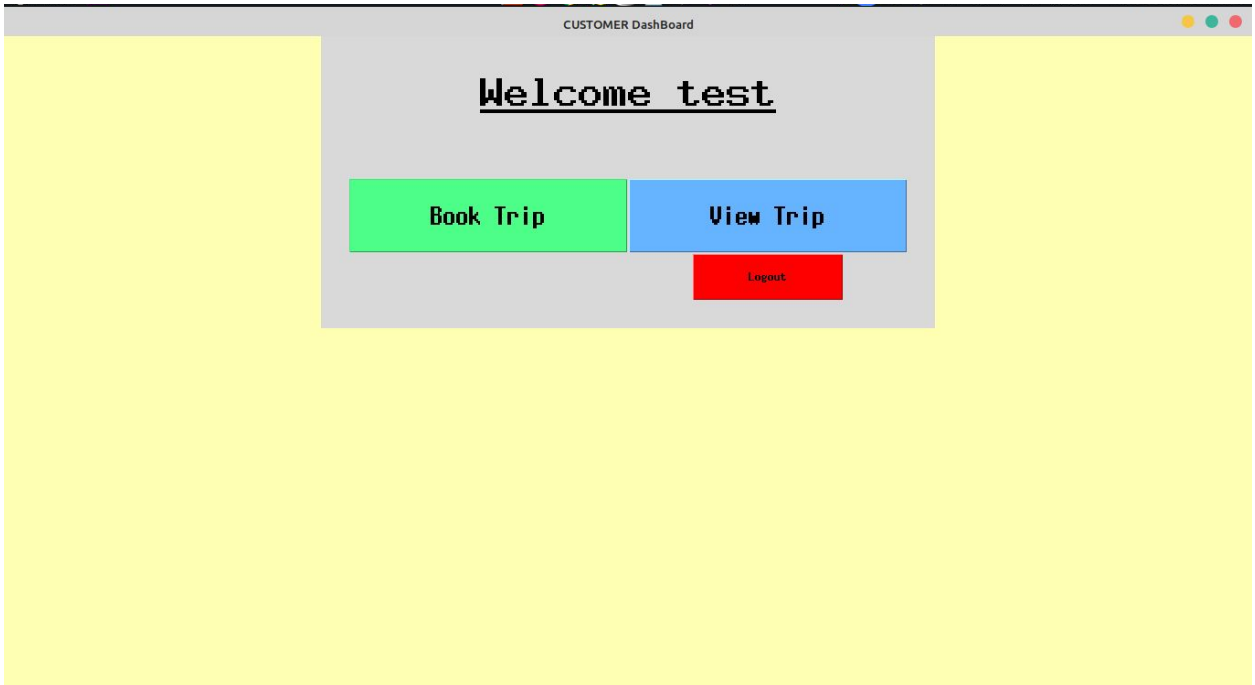
3. Customer login if he/she does not login with proper email id the output should be “invalid email id”



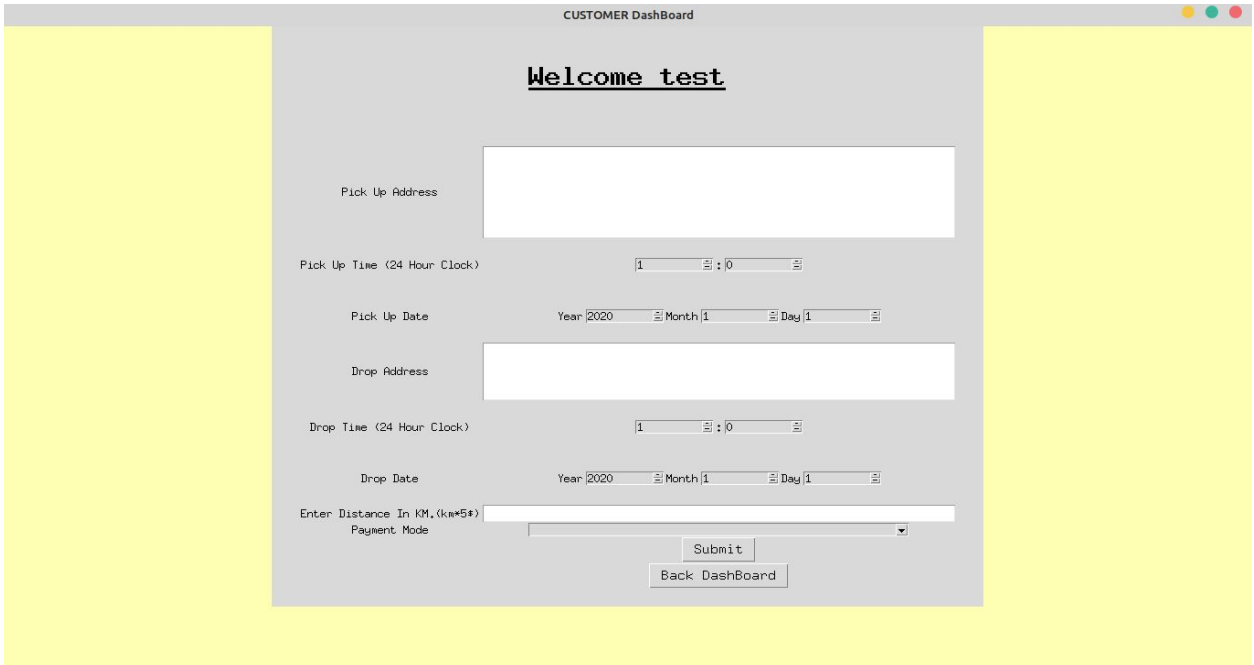
Output:

4. Correct email id and password entered

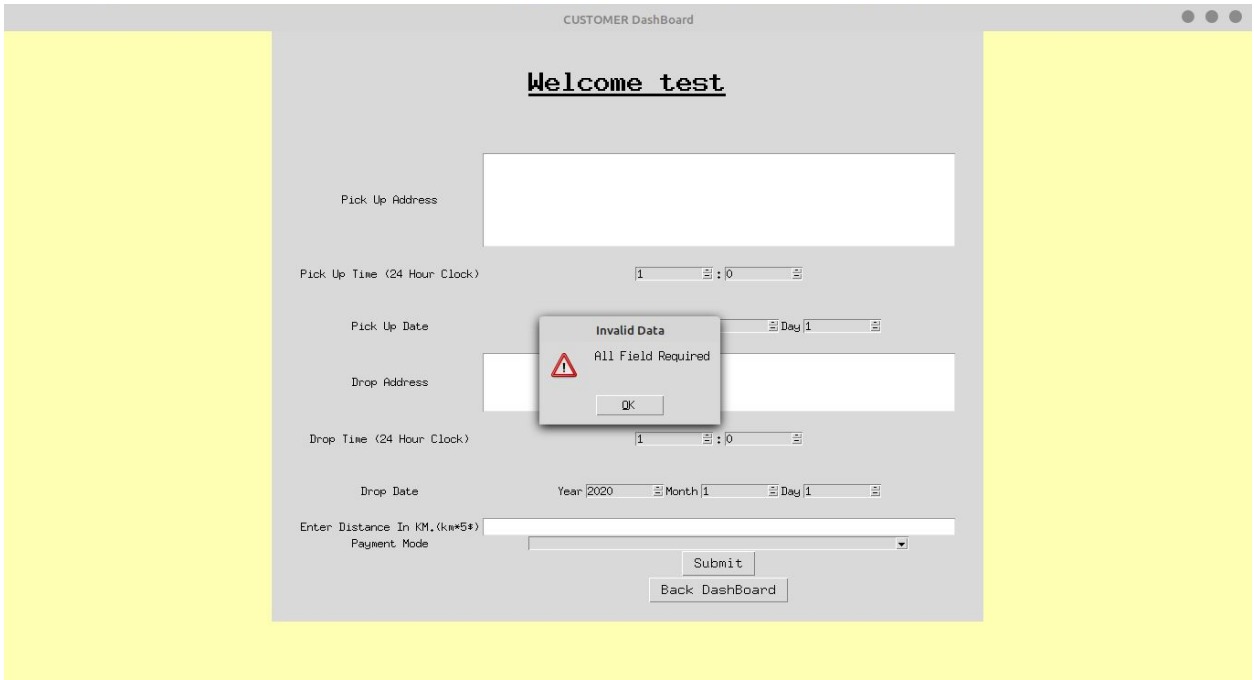
Output:



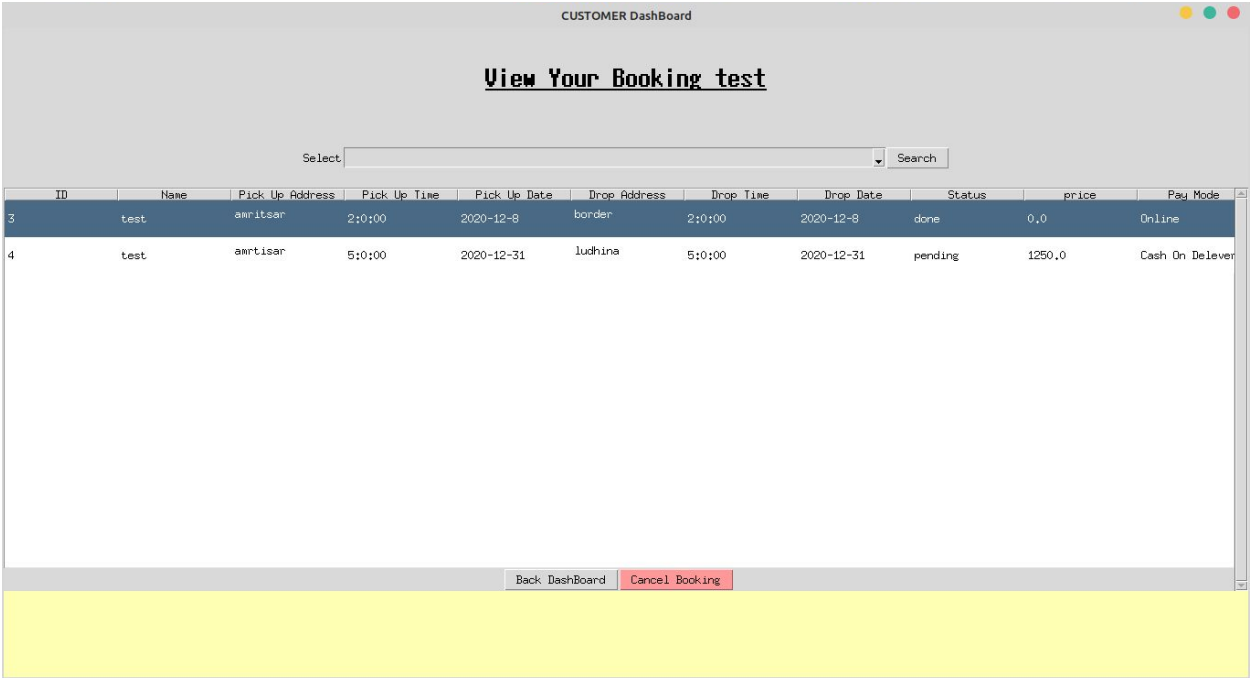
5. If we submit blank field then output should be “All FIeld Required”



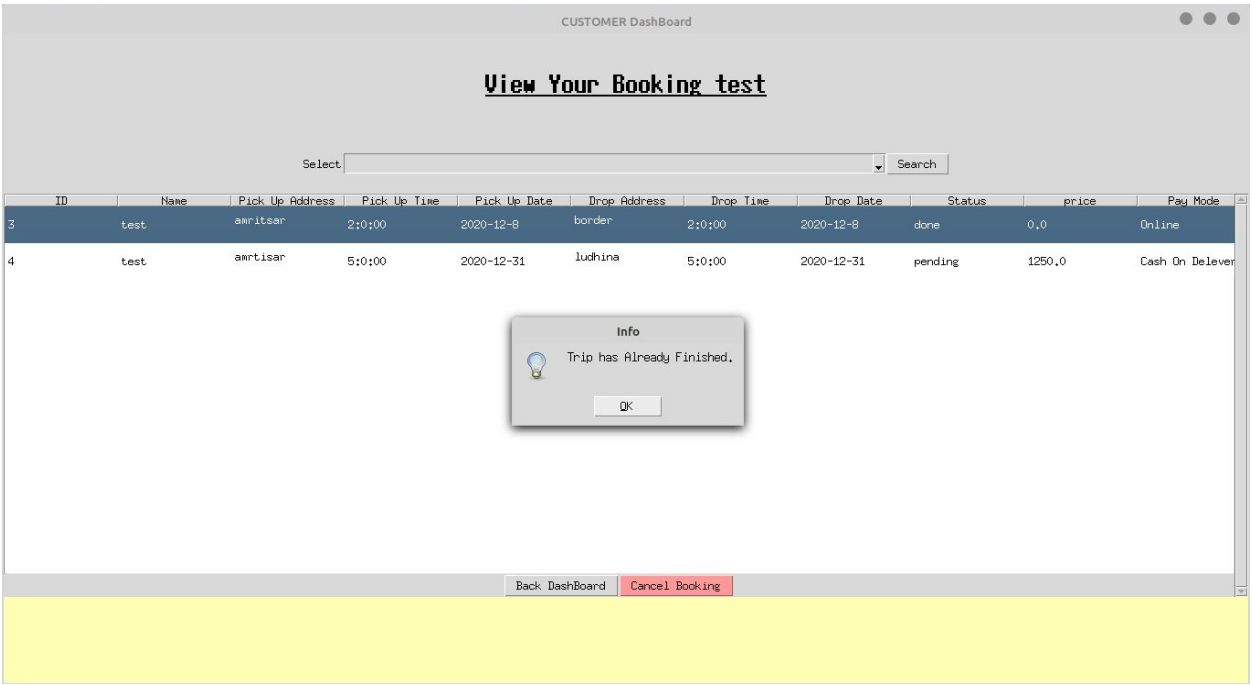
Output:



6. If trip status is “done” the user can not cancel the trip bcz trip is already finished or completed.


CUSTOMER DashBoard
View Your Booking test
Select
Search
ID Name Pick Up Address Pick Up Time Pick Up Date Drop Address Drop Time Drop Date Status price Pay Mode
3 test amritsar 2:0:00 2020-12-8 border 2:0:00 2020-12-8 done 0.0 Online
4 test amrtisar 5:0:00 2020-12-31 ludhina 5:0:00 2020-12-31 pending 1250.0 Cash On Delever
Back DashBoard
Cancel Booking

Output:


CUSTOMER DashBoard
View Your Booking test
Select
Search
ID Name Pick Up Address Pick Up Time Pick Up Date Drop Address Drop Time Drop Date Status price Pay Mode
3 test amritsar 2:0:00 2020-12-8 border 2:0:00 2020-12-8 done 0.0 Online
4 test amrtisar 5:0:00 2020-12-31 ludhina 5:0:00 2020-12-31 pending 1250.0 Cash On Delever
Info
Trip has Already Finished.
OK
Back DashBoard
Cancel Booking

Discussion, Critical Analysis and Reflection

The project came with a new idea and thus enhanced me to do something new.

After striving hard work, I was finally able to achieve and complete all the requirements of this project. Time management and dividing the project into smaller modules was a major task and I gave a lot of time initially thinking over the important aspects of the project.

Finally after a lot of hard work and time input by my side i was able to complete the project.
The project was interesting as per my expectations and I really enjoyed doing this project.

As I designed this project in python, One of the critical tasks that I encountered was setting up tk calendar and I handled it using the tkinter spinner of python programming language.
Thus, I learned a new aspect of doing things in a more enhanced way.

If I encounter something similar in the future, I will tackle it in a more comfortable way because I have now handled such a project and I know now how the time management is done in such a project.

For future enhancements of this project, I would recommend adding an analysis feature which analyzes and generates offers for regular customers. A slip generation system can also be added giving an additional feature.

Appendix

1. code

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import sqlite3

CUSTOMER_LOGIN = dict()
ADMIN_LOGIN = dict()



class redirect_user_window(Frame):
  main_Root = None


  def destroyPackWidget(self, parent):
      for e in parent.pack_slaves():
          e.destroy()

  def __init__(self, master=None):
      redirect_user_window.main_Root = master
      super().__init__(master=master)
      master.title("Login Screen")
      # master.config(background='white')
      master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
      self.createWidget()

  def createWidget(self):
      self.heading = Label(self, text='Texi Booking System',
                           font=("times new roman", 22, "bold", "underline"), bg='#ffffb3', pady=40)
      self.heading.pack()

      self.btnFrame = Frame(self, background='#ffffb3')
      self.btnFrame.pack()

      self.adminButton = Button(self.btnFrame, text='Admin Login', font=("times new roman", 20,
"bold",), width=20,
                                height=2, padx=10,
                                pady=10, bg='blue', command=self.btnAdmin)
```

```
        self.adminButton.pack()
        self.clientButton = Button(self.btnFrame, text='Customer Login', font=("times new roman",
20, "bold",),
                                   width=20,
                                   height=2, padx=10,
                                   pady=10, bg='orange', command=self.btnCustomer)
        self.clientButton.pack()
        self.driver = Button(self.btnFrame, text='Driver', font=("times new roman", 20, "bold",),
width=20,
                             height=2, padx=10,
                             pady=10, bg='#4dff88', command=self.btnDriver)
        self.driver.pack()
        self.distroy_win = Button(self.btnFrame, text="exit", font=("times new roman", 20, "bold",),
width=20,
                                  height=2, padx=10,
                                  pady=10, bg='red', command=self.btnexit)
        self.distroy_win.pack()

    def btnexit(self):
        c = messagebox.askokcancel("Exit?", "Do you Want to Exit?")
        if c:
            exit()

    def btnAdmin(self):
        self.destroyPackWidget(redirect_user_window.main_Root)
        frmadmin = admin_Login(redirect_user_window.main_Root)
        frmadmin.pack()

    def btnCustomer(self):
        self.destroyPackWidget(redirect_user_window.main_Root)
        frmCustomer = customer(redirect_user_window.main_Root)
        frmCustomer.pack()

    def btnDriver(self):
        self.destroyPackWidget(redirect_user_window.main_Root)
        frmDriver = driver_wn(redirect_user_window.main_Root)
        frmDriver.pack()


class customerBookTicket(Frame):
    main_root = None
    global CUSTOMER_LOGIN

    def destroyPackWidget(self, parent):
```

```
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        customerBookTicket.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER DashBoard")
        master.geometry("{0}x{0}+0+0".format(int(master.winfo_screenwidth()),
int(master.winfo_screenheight())))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Welcome {}'.format(CUSTOMER_LOGIN['name']),
                             font=("times new roman", 22, "bold", "underline"), pady=40)
        self.heading.pack()
        self.Frame_wn = Frame(self, padx=30, pady=20)
        self.Frame_wn.pack()

        self.pickupaddress = Label(self.Frame_wn, text="Pick Up Address")
        self.en_pickupaddress = Text(self.Frame_wn, pady=20, height=4, width=57, font=("times
new roman", 11))
        self.pickupaddress.grid(row=0, column=0)
        self.en_pickupaddress.grid(row=0, column=1)

        self.pickupTime = Label(self.Frame_wn, text="Pick Up Time (24 Hour Clock)")
        self.pickUpTimeFrame = Frame(self.Frame_wn, pady=20)
        self.pickupTime.grid(row=1, column=0)
        self.pickUpTimeFrame.grid(row=1, column=1)
        # self.picHourLabel = Label(self.pickUpTimeFrame, text="Hour")
        self.picMinLabel = Label(self.pickUpTimeFrame, text=":")
        self.picHour = Spinbox(self.pickUpTimeFrame, state="readonly", from_=1, to=24, width=10,
wrap=True)
        self.picMin = Spinbox(self.pickUpTimeFrame, state="readonly", from_=0, to=60, width=10,
wrap=True)
        # self.picHourLabel.grid(row=0, column=0)
        self.picHour.grid(row=0, column=1)
        self.picMinLabel.grid(row=0, column=2)
        self.picMin.grid(row=0, column=3)

        self.pickupDate = Label(self.Frame_wn, text="Pick Up Date")
        self.pickUpDateFrame = Frame(self.Frame_wn, pady=20)
        self.pickupDate.grid(row=2, column=0)
        self.pickUpDateFrame.grid(row=2, column=1)
        self.picYearLabel = Label(self.pickUpDateFrame, text="Year")
```

```
    self.picMonthLabel = Label(self.pickUpDateFrame, text="Month")
    self.picDayLabel = Label(self.pickUpDateFrame, text="Day")
    self.picYear = Spinbox(self.pickUpDateFrame, state="readonly", from_=2020, to=2040,
width=10, wrap=True)
    self.picMonth = Spinbox(self.pickUpDateFrame, state="readonly", from_=1, to=12,
width=10, wrap=True)
    self.picDay = Spinbox(self.pickUpDateFrame, state="readonly", from_=1, to=31, width=10,
wrap=True)
    self.picYearLabel.grid(row=0, column=1)
    self.picYear.grid(row=0, column=2)
    self.picMonthLabel.grid(row=0, column=3)
    self.picMonth.grid(row=0, column=4)
    self.picDayLabel.grid(row=0, column=5)
    self.picDay.grid(row=0, column=6)

    self.dropaddress = Label(self.Frame_wn, text="Drop Address")
    self.en_dropaddress = Text(self.Frame_wn, height=4, width=57, font=("times new roman",
11))
    self.dropaddress.grid(row=3, column=0)
    self.en_dropaddress.grid(row=3, column=1)

    self.dropTime = Label(self.Frame_wn, text="Drop Time (24 Hour Clock)")
    self.dropTimeFrame = Frame(self.Frame_wn, pady=20)
    self.dropTime.grid(row=4, column=0)
    self.dropTimeFrame.grid(row=4, column=1)
    # self.dropHourLabel = Label(self.dropTimeFrame, text="Hour")
    self.dropMinLabel = Label(self.dropTimeFrame, text=":")
    self.dropHour = Spinbox(self.dropTimeFrame, state="readonly", from_=1, to=24, width=10,
wrap=True)
    self.dropMin = Spinbox(self.dropTimeFrame, state="readonly", from_=0, to=60, width=10,
wrap=True)
    # self.dropHourLabel.grid(row=0, column=0)
    self.dropHour.grid(row=0, column=1)
    self.dropMinLabel.grid(row=0, column=2)
    self.dropMin.grid(row=0, column=3)

    self.dropDate = Label(self.Frame_wn, text="Drop Date")
    self.dropDateFrame = Frame(self.Frame_wn, pady=20)
    self.dropDate.grid(row=5, column=0)
    self.dropDateFrame.grid(row=5, column=1)
    self.dropYearLabel = Label(self.dropDateFrame, text="Year")
    self.dropMonthLabel = Label(self.dropDateFrame, text="Month")
    self.dropDayLabel = Label(self.dropDateFrame, text="Day")
```

```
        self.dropYear = Spinbox(self.dropDateFrame, state="readonly", from_=2020, to=2040,
width=10, wrap=True)
        self.dropMonth = Spinbox(self.dropDateFrame, state="readonly", from_=1, to=12, width=10,
wrap=True)
        self.dropDay = Spinbox(self.dropDateFrame, state="readonly", from_=1, to=31, width=10,
wrap=True)
        self.dropYearLabel.grid(row=0, column=1)
        self.dropYear.grid(row=0, column=2)
        self.dropMonthLabel.grid(row=0, column=3)
        self.dropMonth.grid(row=0, column=4)
        self.dropDayLabel.grid(row=0, column=5)
        self.dropDay.grid(row=0, column=6)

        self.km = Label(self.Frame_wn, text="Enter Distance In KM.(km*5$)")
        self.en_km = Entry(self.Frame_wn, width=57, font=("times new roman", 11))
        self.km.grid(row=6, column=0)
        self.en_km.grid(row=6, column=1)

        self.paymentMode = Label(self.Frame_wn, text='Payment Mode')
        self.en_paymentMode = ttk.Combobox(self.Frame_wn, values=('Online', 'Cash On
Delevery'), width=57, state="readonly")
        self.paymentMode.grid(row=7, column=0)
        self.en_paymentMode.grid(row=7, column=1)

        #
        # self.pickupdate = Label(self.Frame_wn, text="Pick Up Date")
        # self.en_pickupdate = Calendar(self.Frame_wn, pady=20, date_pattern='y-mm-dd')
        # self.pickupdate.grid(row=2, column=0)
        # self.en_pickupdate.grid(row=2, column=1)

        # self.dropdate = Label(self.Frame_wn, text="Drop Date")
        # self.en_dropdate = Calendar(self.Frame_wn, pady=20,date_pattern='y-mm-dd')
        # self.dropdate.grid(row=3, column=0)
        # self.en_dropdate.grid(row=3, column=1)

        self.submit = Button(self.Frame_wn, text="Submit", font=("times new roman", 11),
command=self.btnSubmit)
        self.submit.grid(row=8, column=1)

        self.Home = Button(self.Frame_wn, text="Back DashBoard", font=("times new roman", 11),
command=self.btnHome)
        self.Home.grid(row=9, column=1)

    def btnSubmit(self):
```

```
        pick_up_Address = self.en_pickupaddress.get('1.0', END)
        pic_up_time = f"{self.picHour.get()}:{self.picMin.get()}:00"
        pic_up_Day = f"{self.picYear.get()}-{self.picMonth.get()}-{self.picDay.get()}"
        drop_Address = self.en_dropaddress.get('1.0', END)
        drop_time = f"{self.dropHour.get()}:{self.dropMin.get()}:00"
        drop_Day = f"{self.dropYear.get()}-{self.dropMonth.get()}-{self.dropDay.get()}"
        # pickup_date = self.en_pickupdate.selection_get()
        # drop_date = self.en_dropdate.selection_get()
        km = self.en_km.get()
        payment = self.en_paymentMode.get()
        price = 0
        if km == "":
            messagebox.showwarning("Invalid Data", 'All Field Required')
        else:
            price = int(km) * 5
            if pick_up_Address == "" and drop_Address == "":
                messagebox.showwarning("Invalid Data", 'All Field Required')
            else:
                query = f"insert into tripinfo (pickUpAddress, pickupTime, pickUpDate, dropAddress,
dropTime, dropDate, customer_id, status,price, paymentMode) values
('{pick_up_Address}','{pic_up_time}','{pic_up_Day}','{drop_Address}','{pic_up_time}','{pic_up_Da
y}',{CUSTOMER_LOGIN['id']},'pending','{price}', '{payment}')"
                # print(query)
                conn = sqlite3.connect('taxiBooking')
                cr = conn.cursor()
                cr.execute(query)
                conn.commit()
                messagebox.showinfo("Booking", 'Request For Book taxt Done')
                self.destroyPackWidget(customerBookTicket.main_root)
                mainScreen = customerDashBoard(customerBookTicket.main_root)
                mainScreen.pack()

    def btnHome(self):
        self.destroyPackWidget(customerBookTicket.main_root)
        dashboard = customerDashBoard(customerBookTicket.main_root)
        dashboard.pack()


class customerViewAndCancel(Frame):
    main_root = None
    global CUSTOMER_LOGIN

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
```

```
        e.destroy()

    def __init__(self, master=None):
        customerViewAndCancel.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER DashBoard")
        master.geometry("{0}x{0}+0+0".format(int(master.winfo_screenwidth()),
int(master.winfo_screenheight())))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text="View Your Booking {}".format(CUSTOMER_LOGIN['name']),
                             font=("times new roman", 20, "bold", "underline"), pady=40)
        self.heading.pack()
        self.searchFrame_wn = Frame(self, padx=30, pady=20)
        self.searchFrame_wn.pack()

        self.searchLabel = Label(self.searchFrame_wn, text='Select')
        self.searchLabel.grid(row=0, column=0)

        self.chose = ttk.Combobox(self.searchFrame_wn, width=48, font=("times new roman", 14),
                                  values=('All', 'pending', 'success', 'Cancel', 'Done'), state='readonly')
        self.chose.grid(row=0, column=1)

        self.searchBtn = Button(self.searchFrame_wn, text="Search", command=self.btnSearch)
        self.searchBtn.grid(row=0, column=2)

        style = ttk.Style(self)
        style.configure('Treeview', rowheight=40)
        self.tree = ttk.Treeview(self, selectmode='browse',
                                 column=(
                                     "id", "name", "pickUpAddress", "pickUpTime", "pickUpDate",
"dropAddress",
                                     "dropTime",
                                     "dropDate", "status", 'price','payMode'))
        vsb = ttk.Scrollbar(self, orient="vertical", command=self.tree.yview)
        vsb.pack(side=RIGHT, fill=Y)

        self.tree.configure(yscrollcommand=vsb.set)

        self.tree.heading("id", text="ID")
        self.tree.heading("name", text="Name")
        self.tree.heading("pickUpAddress", text="Pick Up Address")
        self.tree.heading("pickUpTime", text="Pick Up Time")
```

```
        self.tree.heading("pickUpDate", text="Pick Up Date")
        self.tree.heading("dropAddress", text="Drop Address")
        self.tree.heading("dropTime", text="Drop Time")
        self.tree.heading("dropDate", text="Drop Date")
        self.tree.heading("status", text="Status")
        self.tree.heading("price", text="price")
        self.tree.heading("payMode", text="Pay Mode")
        self.tree.pack(side="top", fill="both", expand=1)
        self.tree.column('#0', stretch=False, minwidth=0, width=0)
        col_width = int(self.winfo_screenwidth() / 11)
        self.tree.column('id', stretch=False, minwidth=0, width=col_width)
        self.tree.column('name', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpAddress', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpTime', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpDate', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropAddress', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropTime', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropDate', stretch=False, minwidth=0, width=col_width)
        self.tree.column('status', stretch=False, minwidth=0, width=col_width)
        self.tree.column('price', stretch=False, minwidth=0, width=col_width)
        self.tree.column('payMode', stretch=False, minwidth=0, width=col_width)

        self.bottomFrame = Frame(self)
        self.bottomFrame.pack()

        self.cancel = Button(self.bottomFrame, text="Cancel Booking", bg='#ff9999',
command=self.btnCancel)
        self.cancel.grid(row=0, column=1)
        self.home = Button(self.bottomFrame, text="Back DashBoard", command=self.btnHome)
        self.home.grid(row=0, column=0)
        self.btnSearch()

    def btnHome(self):
        self.destroyPackWidget(customerViewAndCancel.main_root)
        dashboard = customerDashBoard(customerViewAndCancel.main_root)
        dashboard.pack()

    def btnSearch(self):
        option = self.chose.get()
        if option == '':
            query = "select * from tripinfo where customer_id ='{}'".format(CUSTOMER_LOGIN['id'])
        else:
            query = "select * from tripinfo where status='{}' and customer_id ='{}'".format(option,
                                                                                            CUSTOMER_LOGIN['id'])
```

```
        conn = sqlite3.connect('taxiBooking')
        cr = conn.cursor()
        cr.execute(query)
        p = cr.fetchall()
        all_data = []
        for item in p:
            print(item[10])
            # print(item[11])
            data = [item[0], CUSTOMER_LOGIN['name'], item[1], item[2], item[3], item[4], item[5],
item[6], item[9],
                    item[10],item[11]]
            all_data.append(data)
        for k in self.tree.get_children():
            self.tree.delete(k)
        for i in range(0, len(all_data)):
            self.tree.insert("", value=all_data[i], index=i)

    def btnCancel(self):
        curItem = self.tree.item(self.tree.focus())['values']
        if curItem == "":
            messagebox.showinfo("No select", ' Select One Item From Table.')
        else:
            que = messagebox.askokcancel('Question', 'Do You Want to Cancel This Booking?')
            if que:
                if 'Cancel' not in curItem:
                    if 'done' not in curItem:
                        query = "update tripinfo set status = 'Cancel' where id ='{}'".format(curItem[0])
                        conn = sqlite3.connect('taxiBooking')
                        cr = conn.cursor()
                        cr.execute(query)
                        conn.commit()
                        self.btnSearch()
                    else:
                        messagebox.showinfo("Info",
                                            "Trip has Already Finished.")
                else:
                    messagebox.showinfo("Info",
                                        "Driver Can Not Alocate To This Trip Because User has Already
Canceled this Trip.")


class customerDashBoard(Frame):
    main_root = None
```

```
    global CUSTOMER_LOGIN

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        customerDashBoard.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER DashBoard")
        master.geometry("{0}x{0}+0+0".format(int(master.winfo_screenwidth() / 2),
int(master.winfo_screenheight() / 2)))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Welcome {}'.format(CUSTOMER_LOGIN['name']),
                             font=("times new roman", 32, "bold", "underline"), pady=40)
        self.heading.pack()

        self.Frame_wn = Frame(self, padx=30, pady=30)
        self.Frame_wn.pack()

        self.add_trip = Button(self.Frame_wn, text='Book Trip', font=("times new roman", 20,
"bold",), width=20,
                               height=2, padx=10,
                               pady=10, bg='#4dff88', command=self.btnBookTrip)
        self.add_trip.grid(row=0, column=0)
        self.view_trip = Button(self.Frame_wn, text='View Trip', font=("times new roman", 20,
"bold",), width=20,
                                height=2, padx=10,
                                pady=10, bg='#66b3ff', command=self.btnViewAndCancel)
        self.view_trip.grid(row=0, column=1)
        self.view_trip = Button(self.Frame_wn, text='Logout', font=("times new roman", 10, "bold",),
width=20, height=2,
                                padx=10,
                                pady=10, bg='red', command=self.btnLogout)
        self.view_trip.grid(row=1, column=1)

    def btnViewAndCancel(self):
        self.destroyPackWidget(customerDashBoard.main_root)
        mainBook = customerViewAndCancel(customerDashBoard.main_root)
        mainBook.pack()

    def btnBookTrip(self):
```

```
        self.destroyPackWidget(customerDashBoard.main_root)
        mainBook = customerBookTicket(customerDashBoard.main_root)
        mainBook.pack()

    def btnLogout(self):
        self.destroyPackWidget(customerDashBoard.main_root)
        CUSTOMER_LOGIN.clear()
        mainScreen = redirect_user_window(customerDashBoard.main_root)
        mainScreen.pack()


class customerLogin(Frame):
    main_root = None
    global CUSTOMER_LOGIN

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        customerLogin.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER Login")
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Login', font=("times new roman", 20, "bold", "underline"),
pady=10)
        self.heading.pack()
        self.email = Label(self, text="Email", font=("times new roman", 20, "bold",))
        self.enEmail = Entry(self, font=("times new roman", 20, "bold",), width=40)
        self.password = Label(self, text="Password", font=("times new roman", 20, "bold",))
        self.enpassword = Entry(self, show='*',font=("times new roman", 20, "bold",), width=40)
        self.btn = Button(self, text='Submit', command=self.btnLogin)

        self.email.pack()
        self.enEmail.pack()
        self.password.pack()
        self.enpassword.pack()
        self.btn.pack()

        self.back = Button(self, text='Back', command=self.btnBack)
```

```
        self.back.pack(side=BOTTOM)

    def btnLogin(self):
        email = self.enEmail.get()
        password = self.enpassword.get()

        if email == "" and password == "":
            messagebox.showinfo("Invalid Data", "All Field Required")
        else:
            if '@' in email:
                conn = sqlite3.connect('taxiBooking')
                cr = conn.cursor()
                query = "select * from customer where email='{}' and password='{}'".format(email,
password)
                cr.execute(query)
                result = cr.fetchone()
                if result:
                    CUSTOMER_LOGIN['id'] = result[0]
                    CUSTOMER_LOGIN['name'] = result[1]
                    CUSTOMER_LOGIN['address'] = result[2]
                    CUSTOMER_LOGIN['email'] = result[3]
                    CUSTOMER_LOGIN['telephoneNumber'] = result[4]
                    self.destroyPackWidget(customerLogin.main_root)
                    frmCustomerDashBoard = customerDashBoard(customerLogin.main_root)
                    frmCustomerDashBoard.pack()
                else:
                    messagebox.showwarning('Invalid Data', 'Invalid Email or Password.')
            else:
                messagebox.showinfo("Invalid Data", "Invalid Email")

    def btnBack(self):
        self.destroyPackWidget(customerLogin.main_root)
        frmCustomer = customer(customerLogin.main_root)
        frmCustomer.pack()


class customerRegister(Frame):
    main_root = None

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
```

```
        customerRegister.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER REGISTER")
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Register', font=("times new roman", 20, "bold", "underline"),
pady=10)
        self.heading.pack()
        self.mainFrame = Frame(self)
        self.mainFrame.pack()

        # all widget grid into Main Freame start
        self.name = Label(self.mainFrame, text='Name', font=("times new roman", 20, "bold",))
        self.enName = Entry(self.mainFrame,font=("times new roman", 20, "bold",), width=40)
        self.name.grid(row=0, column=0)
        self.enName.grid(row=0, column=1)

        self.address = Label(self.mainFrame, text='Address', font=("times new roman", 20, "bold",))
        self.enAddress = Entry(self.mainFrame, font=("times new roman", 20, "bold",), width=40)
        self.address.grid(row=1, column=0)
        self.enAddress.grid(row=1, column=1)

        self.email = Label(self.mainFrame, text='Email', font=("times new roman", 20, "bold",))
        self.enEmail = Entry(self.mainFrame, font=("times new roman", 20, "bold",), width=40)
        self.email.grid(row=2, column=0)
        self.enEmail.grid(row=2, column=1)

        self.telephone = Label(self.mainFrame, text='Telephone', font=("times new roman", 20,
"bold",))
        self.enTelephone = Entry(self.mainFrame, font=("times new roman", 20, "bold",), width=40)
        self.telephone.grid(row=3, column=0)
        self.enTelephone.grid(row=3, column=1)

        self.password = Label(self.mainFrame, text='Password', font=("times new roman", 20,
"bold",))
        self.enPassword = Entry(self.mainFrame, show="*", font=("times new roman", 20, "bold",),
width=40)
        self.password.grid(row=4, column=0)
        self.enPassword.grid(row=4, column=1)

        self.btn = Button(self.mainFrame, text='Submit', command=self.save_data)
```

```
        self.btn.grid(row=5, column=1)

        self.back = Button(self, text='Back', command=self.btnBack)
        self.back.pack(side=BOTTOM)

    def btnBack(self):
        self.destroyPackWidget(customerRegister.main_root)
        frmCustomer = customer(customerRegister.main_root)
        frmCustomer.pack()

    def save_data(self):
        name = self.enName.get()
        address = self.enAddress.get()
        email = self.enEmail.get()
        telephone = self.enTelephone.get()
        password = self.enPassword.get()

        if name == "" and address == "" and email == "" and telephone == "" and password == "":
            messagebox.showinfo("Invalid Data", 'All Field Required')
        else:
            if '@' in email:
                if telephone.isnumeric():
                    if len(telephone) != 10:
                        messagebox.showinfo("Invalid Data", 'Invalid Phone Number.')
                    else:
                        conn = sqlite3.connect('taxiBooking')
                        cr = conn.cursor()
                        query = "insert into customer (name, address, email, telephoneNumber,
password) values('{}','{}','{}','{}','{}')".format(
                            name, address, email, telephone, password)
                        cr.execute(query)
                        conn.commit()
                        messagebox.showinfo("DataBase Work", 'Sucess Fully Added.')
                        self.destroyPackWidget(customerRegister.main_root)
                        loginScree = customerLogin(customerRegister.main_root)
                        loginScree.pack()
                else:
                    messagebox.showinfo("Invalid Data", 'Invalid Phone Number.')
            else:
                messagebox.showinfo("Invalid Data", 'Email must Contain @.')


class customer(Frame):
    main_root = None
```

```
    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        customer.main_root = master
        super().__init__(master=master)
        master.title("CUSTOMER")
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='CUSTOMER',
                             font=("times new roman", 22, "bold", "underline"), pady=40)
        self.heading.pack()
        self.Login = Button(self, bg='#99ccff', width='20', text="Login", command=self.btnLogin)
        self.register = Button(self, bg='#00cccc', width='20', text='Register',
command=self.btnRegister)
        self.Back = Button(self, bg='#ffad99', width='20', text='Back', command=self.btnBack)

        self.Login.pack()
        self.register.pack()
        self.Back.pack()

    def btnLogin(self):
        self.destroyPackWidget(customer.main_root)
        loginFrame = customerLogin(customer.main_root)
        loginFrame.pack()

    def btnRegister(self):
        self.destroyPackWidget(customer.main_root)
        regFrame = customerRegister(customer.main_root)
        regFrame.pack()

    def btnBack(self):
        self.destroyPackWidget(customer.main_root)
        mainScreen = redirect_user_window(customer.main_root)
        mainScreen.pack()


class admin_create_driver(Frame):
    main_root = None
```

```
    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        super().__init__(master=master)
        admin_create_driver.main_root = master
        master.title('Admin DashBoard')
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Add Driver',
                             font=("times new roman", 32, "bold", "underline"), pady=40)
        self.heading.pack()
        self.licence = Label(self, text='Licence Number')
        self.licence.pack()
        self.en_licence = Entry(self)
        self.en_licence.pack()
        self.name = Label(self, text='Name')
        self.name.pack()
        self.en_name = Entry(self)
        self.en_name.pack()
        self.submit = Button(self, text='Submit', bg='#66ff66', command=self.btnSubmit)
        self.submit.pack()
        self.back = Button(self, text='Back', bg='#ff9966', command=self.btnBack)
        self.back.pack()

    def btnBack(self):
        self.destroyPackWidget(admin_create_driver.main_root)
        admin_dash = admin_DashBoard(admin_create_driver.main_root)
        admin_dash.pack()

    def btnSubmit(self):
        licence = self.en_licence.get()
        name = self.en_name.get()
        if licence == "" and name == "":
            messagebox.showwarning("Invalid Data", 'All Field Are Required')
        else:
            query = "insert into driver(licence, name) values('{}','{}')".format(licence, name)
            conn = sqlite3.connect('taxiBooking')
            cr = conn.cursor()
```

```
        cr.execute(query)
        conn.commit()
        messagebox.showinfo("Success", 'Driver Added Success.')
        self.destroyPackWidget(admin_create_driver.main_root)
        admin_dash = admin_DashBoard(admin_create_driver.main_root)
        admin_dash.pack()


class admin_allocateDriver(Frame):
  main_root = None
  global ADMIN_LOGIN

  def destroyPackWidget(self, parent):
     for e in parent.pack_slaves():
        e.destroy()

  def __init__(self, master=None):
     super().__init__(master=master)
     admin_allocateDriver.main_root = master
     master.title('Allocate Driver')
     master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
     self.createWidget()

  def createWidget(self):
     self.heading = Label(self, text='View All Booking',
                          font=("times new roman", 32, "bold", "underline"), pady=40)
     self.heading.pack()
     self.searchFrame_wn = Frame(self, padx=30, pady=20)
     self.searchFrame_wn.pack()

     self.searchLabel = Label(self.searchFrame_wn, text='Select')
     self.searchLabel.grid(row=0, column=0)

     self.chose = ttk.Combobox(self.searchFrame_wn, width=48, font=("times new roman", 14),
                               values=('All', 'pending', 'success', 'Cancel', 'done'), state='readonly')
     self.chose.grid(row=0, column=1)

     self.searchBtn = Button(self.searchFrame_wn, text="Search", command=self.btnSearch)
     self.searchBtn.grid(row=0, column=2)

     style = ttk.Style(self)
     style.configure('Treeview', rowheight=40)
     self.tree = ttk.Treeview(self, selectmode='browse',
```

```
                        column=(
                            "id", "name", "pickUpAddress", "pickUpTime", "pickUpDate",
"dropAddress",
                            "dropTime",
                            "dropDate", 'licence', "status", 'price','payMode'))
    vsb = ttk.Scrollbar(self, orient="vertical", command=self.tree.yview)
    vsb.pack(side=RIGHT, fill=Y)

    self.tree.configure(yscrollcommand=vsb.set)

    self.tree.heading("id", text="ID")
    self.tree.heading("name", text="Name")
    self.tree.heading("pickUpAddress", text="Pick Up Address")
    self.tree.heading("pickUpTime", text="Pick Up Time")
    self.tree.heading("pickUpDate", text="Pick Up Date")
    self.tree.heading("dropAddress", text="Drop Address")
    self.tree.heading("dropTime", text="Drop Time")
    self.tree.heading("dropDate", text="Drop Date")
    self.tree.heading("licence", text="Licence")
    self.tree.heading("status", text="Status")
    self.tree.heading("price", text="Price")
    self.tree.heading("payMode", text="pay Mode")
    self.tree.pack(side="top", fill="both", expand=1)
    self.tree.column('#0', stretch=False, minwidth=0, width=0)
    col_width = int(self.winfo_screenwidth() / 12)
    self.tree.column('id', stretch=False, minwidth=0, width=col_width)
    self.tree.column('name', stretch=False, minwidth=0, width=col_width)
    self.tree.column('pickUpAddress', stretch=False, minwidth=0, width=col_width)
    self.tree.column('pickUpTime', stretch=False, minwidth=0, width=col_width)
    self.tree.column('pickUpDate', stretch=False, minwidth=0, width=col_width)
    self.tree.column('dropAddress', stretch=False, minwidth=0, width=col_width)
    self.tree.column('dropTime', stretch=False, minwidth=0, width=col_width)
    self.tree.column('dropDate', stretch=False, minwidth=0, width=col_width)
    self.tree.column('licence', stretch=False, minwidth=0, width=col_width)
    self.tree.column('status', stretch=False, minwidth=0, width=col_width)
    self.tree.column('price', stretch=False, minwidth=0, width=col_width)
    self.tree.column('payMode', stretch=False, minwidth=0, width=col_width)

    self.bottomFrame = Frame(self)
    self.bottomFrame.pack()

    self.cancel = Button(self.bottomFrame, text="Alocate Driver", bg='#ff9999',
command=self.btnAlocateDriver)
    self.cancel.grid(row=0, column=1)
```

```
        self.home = Button(self.bottomFrame, text="Back DashBoard", command=self.btnHome)
        self.home.grid(row=0, column=0)
        self.btnSearch()

    def btnHome(self):
        self.destroyPackWidget(admin_allocateDriver.main_root)
        dashboard = admin_DashBoard(admin_allocateDriver.main_root)
        dashboard.pack()

    def btnSearch(self):
        option = self.chose.get()
        if option == '' or option == 'All':
            query = "select tripinfo.id, customer.name ,tripinfo.pickUpAddress, tripinfo.pickUpDate,
tripinfo.pickupTime, tripinfo.dropAddress, tripinfo.dropDate, tripinfo.dropTime, tripinfo.licence,
tripinfo.status,tripinfo.price,tripinfo.paymentMode from tripinfo INNER JOIN customer on
tripinfo.customer_id=customer.id"
        else:
            query = "select tripinfo.id, customer.name ,tripinfo.pickUpAddress, tripinfo.pickUpDate,
tripinfo.pickupTime, tripinfo.dropAddress, tripinfo.dropDate, tripinfo.dropTime, tripinfo.licence,
tripinfo.status,tripinfo.price,tripinfo.paymentMode from tripinfo INNER JOIN customer on
tripinfo.customer_id=customer.id where tripinfo.status='{}'".format(
                option)
        conn = sqlite3.connect('taxiBooking')
        cr = conn.cursor()
        cr.execute(query)
        p = cr.fetchall()
        all_data = []
        for item in p:
            data = [item[0], item[1], item[2], item[3], item[4], item[5], item[6], item[7], item[8],
                    item[9], item[10],item[11]]
            all_data.append(data)
        for k in self.tree.get_children():
            self.tree.delete(k)
        for i in range(0, len(all_data)):
            self.tree.insert("", value=all_data[i], index=i)

    def btnCancel(self):
        curItem = self.tree.item(self.tree.focus())['values']
        if curItem == "":
            messagebox.showinfo("No select", ' Select One Item From Table.')
        else:
            que = messagebox.askokcancel('Question', 'Do You Want to Cancel This Booking?')
            if que:
                if 'Cancel' not in curItem:
```

```
            query = "update tripinfo set status = 'Cancel' where id ='{}'".format(curItem[0])
            conn = sqlite3.connect('taxiBooking')
            cr = conn.cursor()
            cr.execute(query)
            conn.commit()
            self.btnSearch()
        else:
            messagebox.showinfo("Info",
                                "Driver Can Not Alocate To This Trip Because User has Already
Canceled this Trip.")

  def btnAlocateDriver(self):
    curItem = self.tree.item(self.tree.focus())['values']
    if curItem == "":
        messagebox.showinfo("No select", ' Select One Item From Table.')
    else:
        que = messagebox.askokcancel('Question', 'Do You Want to Alocate Driver to This
Booking?')
        if que:
            if 'Cancel' not in curItem:
                if ('success' not in curItem):
                    if ('done' not in curItem):
                        admin_allocateDriver.main_root.withdraw()
                        self.wn = Toplevel(self)
                        self.wn.title('Allocate Driver')

                        query = f"select licence from driver where driver.licence not in (select
tripinfo.licence from tripinfo where tripinfo.pickupTime<='{curItem[4]}'  and
tripinfo.dropTime>='{curItem[7]}' and tripinfo.pickUpDate<='{curItem[3]}' and
tripinfo.dropDate>='{curItem[6]}' and tripinfo.status='Success')"
                        conn = sqlite3.connect('taxiBooking')
                        cr = conn.cursor()
                        cr.execute(query)
                        p = cr.fetchall()
                        all_licence = []
                        for i in p:
                            all_licence.append(i[0])
                        self.choseDriver = ttk.Combobox(self.wn, width=48, font=("times new roman",
14),
                                                        values=all_licence, state='readonly')
                        self.choseDriver.pack()
                        selected = Button(self.wn, text="Submit", command=lambda:
self.btnSelected(curItem[0]))
                        selected.pack()
```

```
                    self.wn.mainloop()
                else:
                    messagebox.showinfo("Info",
                                        "Driver Can Not Allocate To This Trip Because AllReady Allocated
or Trip is Completed.")
            else:
                messagebox.showinfo("Info",
                                    "Driver Can Not Allocate To This Trip Because AllReady Allocated or
Trip is Completed.")
        else:
            messagebox.showinfo("Info",
                                "Driver Can Not Allocate To This Trip Because User has Already
Canceled this Trip.")

    def btnSelected(self, id):
        licence = self.choseDriver.get()
        print(id)
        query = "Update tripinfo set licence='{}', status='success' where id='{}'".format(licence, id)
        conn = sqlite3.connect('taxiBooking')
        cr = conn.cursor()
        cr.execute(query)
        conn.commit()
        messagebox.showinfo("Admin", 'Driver Allocated')
        self.wn.destroy()
        admin_allocateDriver.main_root.deiconify()
        self.btnSearch()


class admin_DashBoard(Frame):
    main_root = None
    global ADMIN_LOGIN

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        super().__init__(master=master)
        admin_DashBoard.main_root = master
        master.title('Admin DashBoard')
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()
```

```
  def createWidget(self):
      self.heading = Label(self, text='Welcome {}'.format(ADMIN_LOGIN['name']),
                           font=("times new roman", 32, "bold", "underline"), pady=40)
      self.heading.pack()

      self.btnFrame = Frame(self)
      self.btnFrame.pack()
      self.add_Driver = Button(self.btnFrame, text='Register Driver', font=("times new roman", 20,
"bold",), width=20,
                               height=2, padx=10,
                               pady=10, bg='blue', command=self.btnAdd_Driver)
      self.add_Driver.pack()
      self.alocateDriver = Button(self.btnFrame, text='Allocate Driver', font=("times new roman",
20, "bold",),
                                  width=20,
                                  height=2, padx=10,
                                  pady=10, bg='#4dff88', command=self.btnAlocateDriver)
      self.alocateDriver.pack()
      self.Logout = Button(self.btnFrame, text='Logout', font=("times new roman", 20, "bold",),
width=20,
                           height=2, padx=10,
                           pady=10, bg='red', command=self.btnLogout)
      self.Logout.pack()

  def btnAdd_Driver(self):
      self.destroyPackWidget(admin_DashBoard.main_root)
      addDriver = admin_create_driver(admin_DashBoard.main_root)
      addDriver.pack()

  def btnAlocateDriver(self):
      self.destroyPackWidget(admin_DashBoard.main_root)
      addDriver = admin_allocateDriver(admin_DashBoard.main_root)
      addDriver.pack()

  def btnLogout(self):
      self.destroyPackWidget(admin_DashBoard.main_root)
      ADMIN_LOGIN.clear()
      home = redirect_user_window(admin_DashBoard.main_root)
      home.pack()


class admin_Login(Frame):
  main_root = None
  global ADMIN_LOGIN
```

```
    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        admin_Login.main_root = master
        super().__init__(master=master)
        master.title('Admin Login')

        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.createWidget()

    def createWidget(self):
        self.heading = Label(self, text='Login', font=("times new roman", 20, "bold", "underline"),
pady=10)
        self.heading.pack()
        self.username = Label(self, text="UserName", font=("times new roman", 20, "bold",))
        self.enusername = Entry(self, font=("times new roman", 20, "bold",), width=40)
        self.password = Label(self, text="Password", font=("times new roman", 20, "bold",))
        self.enpassword = Entry(self, show='*', font=("times new roman", 20, "bold",), width=40)
        self.btn = Button(self, text='Submit', command=self.btnLogin)

        self.username.pack()
        self.enusername.pack()
        self.password.pack()
        self.enpassword.pack()
        self.btn.pack()

        self.back = Button(self, text='Back', command=self.btnBack)
        self.back.pack(side=BOTTOM)

    def btnLogin(self):
        username = self.enusername.get()
        password = self.enpassword.get()

        if username == "" and password == "":
            messagebox.showinfo("Invalid Data", "All Field Required")
        else:
            conn = sqlite3.connect('taxiBooking')
            cr = conn.cursor()
            query = "select * from admin where username='{}' and password='{}'".format(username,
password)
```

```
        cr.execute(query)
        result = cr.fetchone()
        if result:
            ADMIN_LOGIN['name'] = result[0]
            self.destroyPackWidget(admin_Login.main_root)
            frmCustomerDashBoard = admin_DashBoard(admin_Login.main_root)
            frmCustomerDashBoard.pack()
        else:
            messagebox.showwarning('Invalid Data', 'Invalid Email or Password.')

    def btnBack(self):
        self.destroyPackWidget(admin_Login.main_root)
        frmMain = redirect_user_window(admin_Login.main_root)
        frmMain.pack()


class driver_wn(Frame):
    main_root = None

    licence = None

    def destroyPackWidget(self, parent):
        for e in parent.pack_slaves():
            e.destroy()

    def __init__(self, master=None):
        driver_wn.main_root = master
        super().__init__(master=master)
        master.title("Login Screen")
        master.config(background='white')
        master.geometry("{0}x{0}+0+0".format(master.winfo_screenwidth(),
master.winfo_screenheight()))
        self.show_booking_list()

    def show_booking_list(self):
        self.heading = Label(self, text='My Trips',
                             font=("times new roman", 32, "bold", "underline"), pady=40)
        self.heading.pack()

        headFrame = Frame(self)
        headFrame.pack()

        self.licence = Label(headFrame, text='Licence')
        self.grid(row=0, column=0)
```

```
        self.en_licence = Entry(headFrame)
        self.en_licence.grid(row=0, column=1)
        search = Button(headFrame, text='Search', command=self.btnSearch)
        search.grid(row=0, column=2)

        style = ttk.Style(self)
        style.configure('Treeview', rowheight=40)
        self.tree = ttk.Treeview(self, selectmode='browse',
                                 column=(
                                    "id", "name", "pickUpAddress", "pickUpTime", "pickUpDate",
"dropAddress",
                                    "dropTime",
                                    "dropDate", 'licence', "status"))
        vsb = ttk.Scrollbar(self, orient="vertical", command=self.tree.yview)
        vsb.pack(side=RIGHT, fill=Y)

        self.tree.configure(yscrollcommand=vsb.set)

        self.tree.heading("id", text="ID")
        self.tree.heading("name", text="Name")
        self.tree.heading("pickUpAddress", text="Pick Up Address")
        self.tree.heading("pickUpTime", text="Pick Up Time")
        self.tree.heading("pickUpDate", text="Pick Up Date")
        self.tree.heading("dropAddress", text="Drop Address")
        self.tree.heading("dropTime", text="Drop Time")
        self.tree.heading("dropDate", text="Drop Date")
        self.tree.heading("licence", text="Licence")
        self.tree.heading("status", text="Status")
        self.tree.pack(side="top", fill="both", expand=1)
        self.tree.column('#0', stretch=False, minwidth=0, width=0)
        col_width = int(self.winfo_screenwidth() / 10)
        self.tree.column('id', stretch=False, minwidth=0, width=col_width)
        self.tree.column('name', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpAddress', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpTime', stretch=False, minwidth=0, width=col_width)
        self.tree.column('pickUpDate', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropAddress', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropTime', stretch=False, minwidth=0, width=col_width)
        self.tree.column('dropDate', stretch=False, minwidth=0, width=col_width)
        self.tree.column('licence', stretch=False, minwidth=0, width=col_width)
        self.tree.column('status', stretch=False, minwidth=0, width=col_width)

        self.bottomFrame = Frame(self)
        self.bottomFrame.pack()
```

```
        self.btn = Button(self.bottomFrame, text='Back', command=self.btnBack)
        self.btn.grid(row=0, column=0)
        self.btn = Button(self.bottomFrame, text='Trip Complite', command=self.btnTripComplite)
        self.btn.grid(row=0, column=1)

    def btnTripComplite(self):
        curItem = self.tree.item(self.tree.focus())['values']
        if curItem == "":
            messagebox.showinfo("No select", ' Select One Item From Table.')
        else:
            que = messagebox.askokcancel('Question', 'Do You Want to Cancel This Booking?')
            if que:
                query = "Update tripinfo set status='done' where id='{}'".format(curItem[0])
                conn = sqlite3.connect('taxiBooking')
                cr = conn.cursor()
                cr.execute(query)
                conn.commit()
                messagebox.showinfo("Admin", 'Trip Finished')
                self.btnSearch()

    def btnSearch(self):
        option = self.en_licence.get()
        self.licence = option
        query = "select tripinfo.id, customer.name ,tripinfo.pickUpAddress, tripinfo.pickUpDate,
tripinfo.pickupTime, tripinfo.dropAddress, tripinfo.dropDate, tripinfo.dropTime, tripinfo.licence,
tripinfo.status from tripinfo INNER JOIN customer on tripinfo.customer_id=customer.id where
tripinfo.status='success' and tripinfo.licence='{}'".format(
            self.licence)
        conn = sqlite3.connect('taxiBooking')
        cr = conn.cursor()
        cr.execute(query)
        p = cr.fetchall()
        all_data = []
        for item in p:
            data = [item[0], item[1], item[2], item[3], item[4], item[5], item[6], item[7], item[8],
                    item[9]]
            all_data.append(data)
        if len(all_data) == 0:
            messagebox.showinfo("Data", 'Data Not Found.')
        for k in self.tree.get_children():
            self.tree.delete(k)
        for i in range(0, len(all_data)):
            self.tree.insert("", value=all_data[i], index=i)
```

```
    def btnBack(self):
        self.destroyPackWidget(driver_wn.main_root)
        frmLogin = redirect_user_window(driver_wn.main_root)
        frmLogin.pack()


if __name__ == '__main__':
    root = Tk()
    root.config(background='#ffffb3')
    frmLogin = redirect_user_window(root)
    frmLogin.pack()
    root.mainloop()
```

2. Files and class name

a. File name

i. Main.py
ii. taxiBooking

b. Class Names

i. redirect_user_window
ii. customerBookTicket
iii. customerViewAndCancel
iv. customerDashBoard
v. customerLogin
vi. customerRegister
vii. customer
viii. admin_create_driver
ix. admin_allocateDriver
x. admin_DashBoard
xi. admin_Login
xii. Driver_wn