

CIS020-1 – Introduction to Software Development

Taxi Booking System – Case Study

Date- Jan 13, 2023

Name - Yaman Mahrajan

University Code - 2212387

# Table of Contents

Introduction/Overview…………………………………………………..3

TaskDescription………………………………………………………...3

Project Plan / Schedule……………………………………………….3

Tasks:

Requirements Analysis……………………………………………..4-6

Overview of Functional, Technical (Non-Functional) and Usability Requirements Design:……………………………………………….7-32

* UML Diagrams ………………7

o Use Case Diagram…………..7

* Activity Diagrams ………….14-18
* o Class Diagram………….19-21
* Database Design …………………22-26
* o Data Dictionary…….……-26
* User Interface Design………27

Implementation…………………………………………………………..31-35

Testing-------------------------------------------------------------------------------35

Discussion / Reflection / Critical Analysis-------------------------------------36

Conclusion---------------------------------------------------------------------------37

Appendix ……………………………………38-100

# Introduction/Overview

This report is all about the assignment work that I have done as a final assessment of my first semester. The task was, to develop a Taxi Booking System for a Taxi company using python programming language to provide online taxi booking service. According to the task, I have completed all the requirement and develop a user-friendly software. After meeting every criterion, I have successfully developed the software where users, driver and admin can perform their respective task.

# Task Description

A taxi firm would like to offer an online service that allows clients to book trips and taxi drivers to check their future bookings. Data must be kept in an external file, such as a database or text file. at least the following information would be required. Customer information (name, address, email, phone number, and payment method) and trip details (pickup and drop-off addresses, as well as pickup time and date) must be stored and easily accessible. Additionally, you will need to keep driver data (name and license plate). There must be exactly one driver assigned to each trip, and the trips assigned to a driver cannot overlap. Customers who haven't signed up yet must do so by entering their name, address, phone number, email address, and preferred payment method. A registered user should be able to schedule a trip, i.e., specify the pick-up address, the time of the pick-up, and the drop-off address, preferably through a graphical user interface. Additionally, customers ought to be able to see their excursions and cancel them if necessary.

A consumer can only do any of these functions after signing in, which necessitates registration. The administrator of the taxi company must approve each new trip booking and assign a driver for it. Each confirmed booking must have a single driver assigned to it. Taxi drivers must be able to see a list of their upcoming rides after logging in.

# Project Plan/Schedule

|  |  |  |
| --- | --- | --- |
| Week No. | Tasks | Priority |
| 1 | Design UML Diagrams | SHOULD |
| 2 | Class Diagram, Activity Diagram | MUST |
| 3 | Database Design | MUST |
| 4 | UI Design | SHOULD |
| 5 | Implementation | MUST |
| 6 | Testing | MUST |
|  | Submit Group Report, Project Code and Video Recording (if required) |  |
|  | Project Presentation (if required) |  |

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

Functional Requirements TBS = Taxi Booking System

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | A customer must be able to register on the TBS | MUST |
| 2 | A customer must be able to log in to the TBS | MUST |
| 3 | A customer must be able to log out of the TBS | COULD |
| 4 | An administrator must be able log in to the TBS | SHOULD |
| 5 | An administrator must be able to log out of the  TBS | COULD |
| 6 | An administrator must be able to confirm a booking | SHOULD |
| 7 | An administrator must be able to allocate a driver to a booking | SHOULD |
| 8 | An administrator must be able to view all bookings | SHOULD |
| 9 | An administrator must be able to cancel a booking | SHOULD |
| 10 | A customer must be able to view his/her bookings | MUST |
| 11 | A customer must be able to make a booking | MUST |
| 12 | A customer must be able to cancel a booking | MUST |
| 13 | A taxi driver must be able to log in to the TBS | MUST |
| 14 | An administrator must be able to log out of the | SHOULD |
|  | TBS |  |
| 15 | A taxi driver must be able to view his bookings | MUST |

# Non-functional Requirements

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | The TBS should process input and return results within 10 seconds |  |
| 2 | The TBS should run on a range of devices from PCs to mobile phones |  |
| 3 | The TBS design should be sufficiently scalable and flexible to allow for further future  enhancements |  |
| 4 | The TBS users should not experience critical system failures. 99.99% ‘uptime’ should be achieved. |  |

# Usability Requirements

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | The TBS should incorporate a user-centric design |  |
| 2 | The design should demonstrate evidence of a good understanding of interface design issues – for example, a consistent design for each form, layout of content, use of colour schemes and images, navigational methods, usability when viewed at various screen resolutions and various monitor dimensions. |  |
| 3 | All data entry forms should be short and easy to complete and there should be entry validation. |  |
| 4 | The TBS should have clear and intuitive navigation |  |
| 5 | The TBS should comply with WW3 Web  Accessibility Standards (WCAG)  Text easy to read and language and language style should be appropriate with absence of grammar / spelling errors |  |
|  | There should be a clear layout which remains |  |
|  | consistent throughout the application. Style, layout and content should be appropriate for the purpose of the application. |  |

\*MOSCOW Notation:

M = MUST

S = SHOULD

C =COULD W =WON’T

# Design o UML Diagrams

## o Uses Case Diagrams(s)

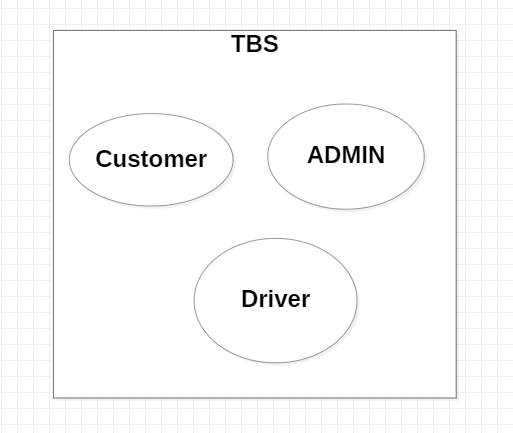


Fig 1: Taxi Booking System - Use Case Diagram (Sea Level)

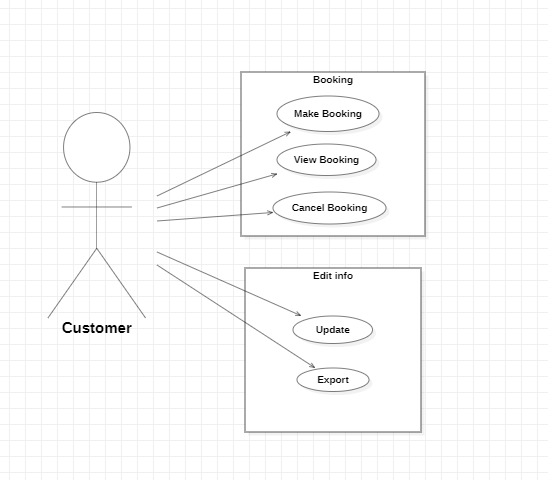


Fig 2: Taxi Booking System - Use Case Diagram (Fish Level Customer)

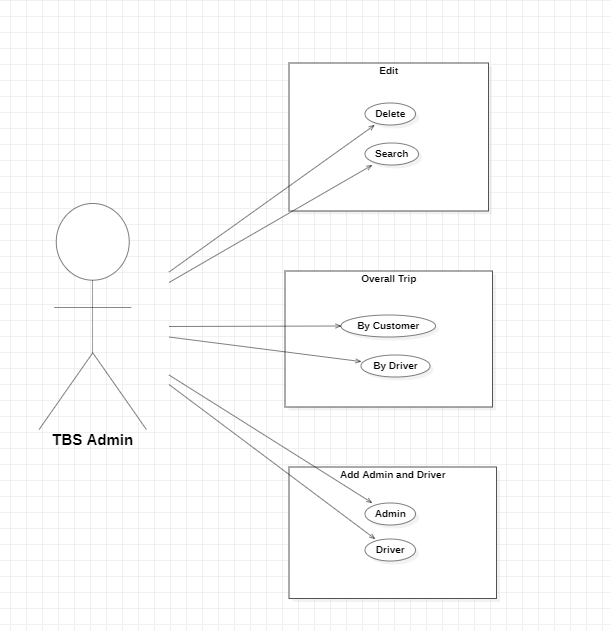


Fig 3: Taxi Booking System - Use Case Diagram (Fish Level Admin)

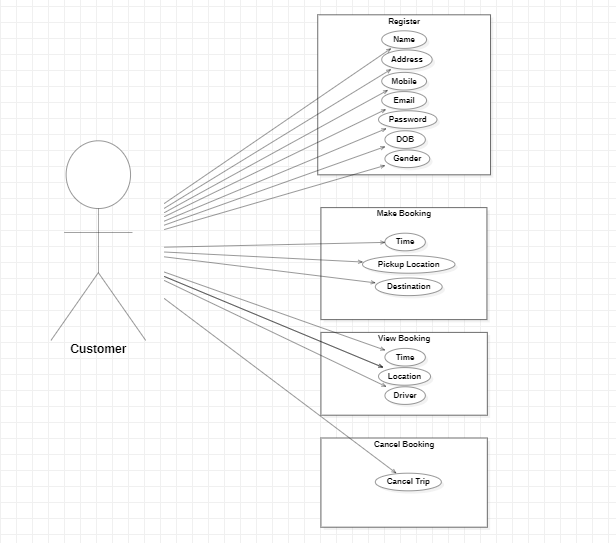


Fig 4.1: Taxi Booking System - Use Case Diagram (Clam Level Customer)

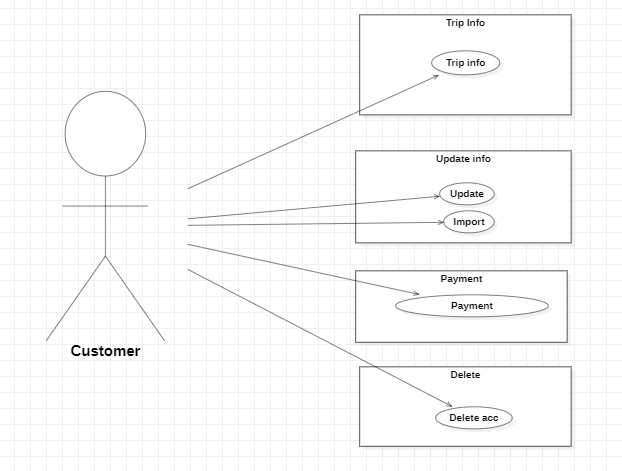


Fig 4.2: Taxi Booking System - Use Case Diagram (Clam Level Customer)

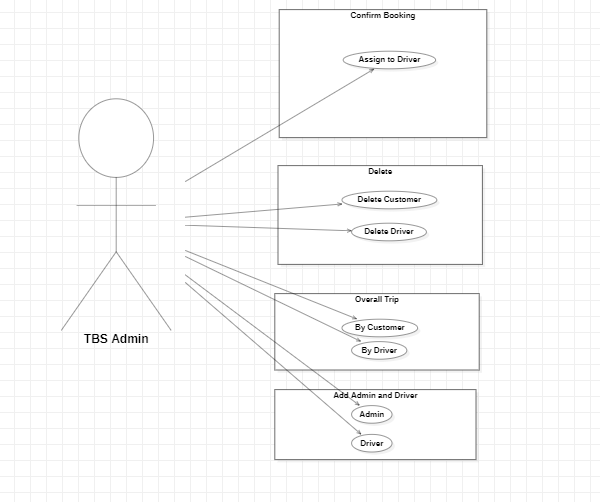


Fig 5: Taxi Booking System - Use Case Diagram (Clam Level Admin)

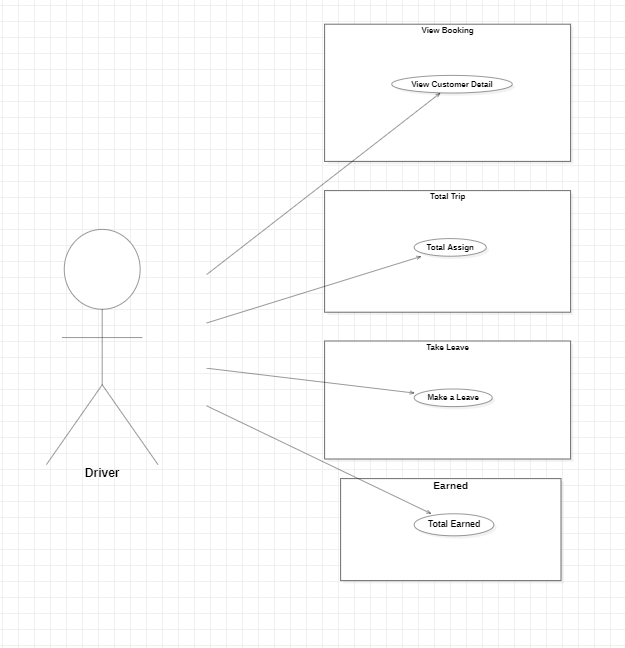


Fig 5: Taxi Booking System - Use Case Diagram (Clam Level Driver)

o Activity Diagram(s)

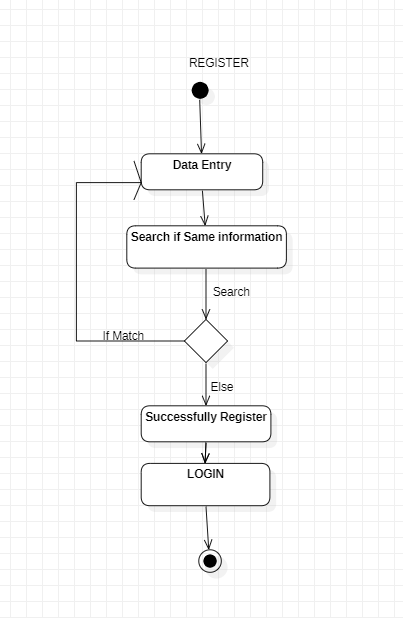


Fig 6: Taxi Booking System - Activity Diagram (Register)

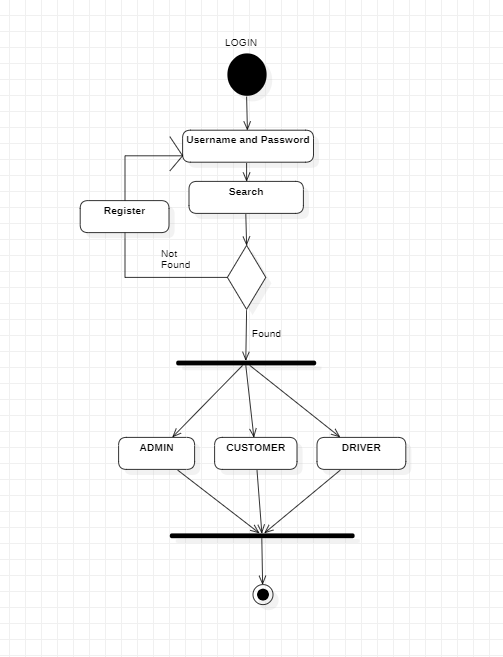


Fig 7: Taxi Booking System - Activity Diagram (Login)

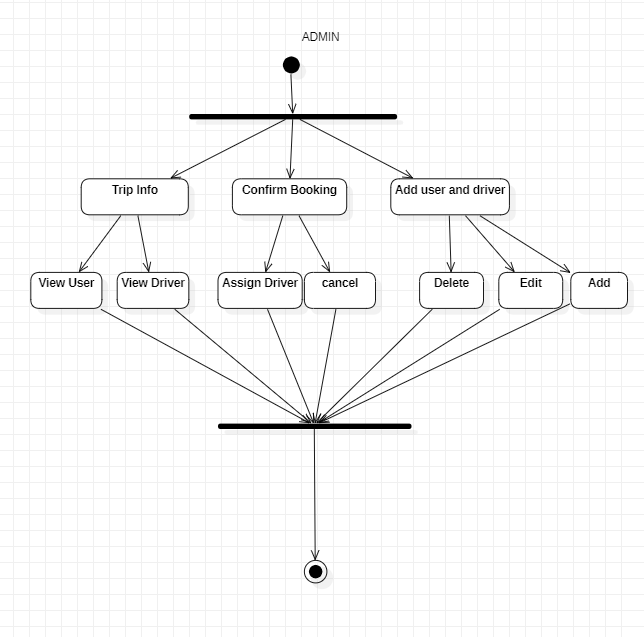


Fig 8: Taxi Booking System - Activity Diagram (Admin)

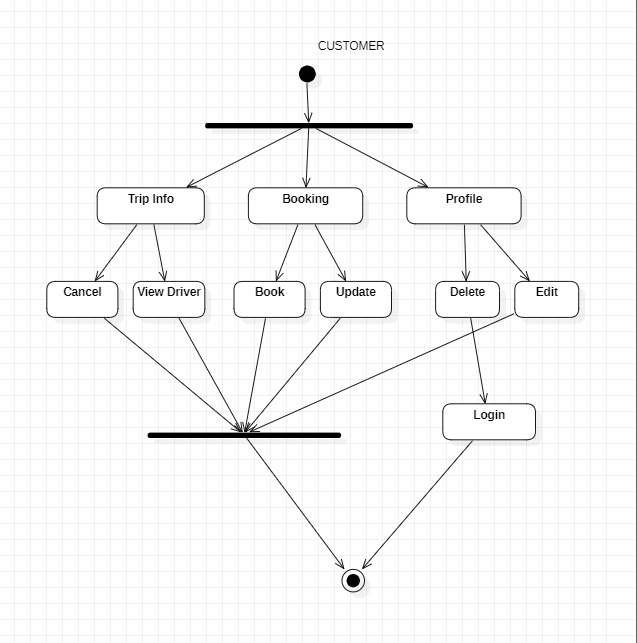


Fig 9: Taxi Booking System - Activity Diagram (Customer)

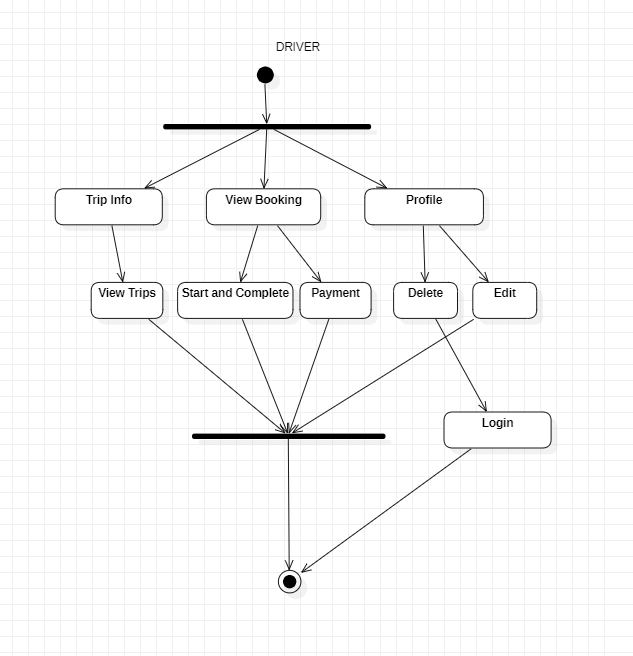


Fig 10: Taxi Booking System - Activity Diagram (Driver)

## o Class Diagram(s)



Fig 11: Taxi Booking System – Class Diagram (Customer)

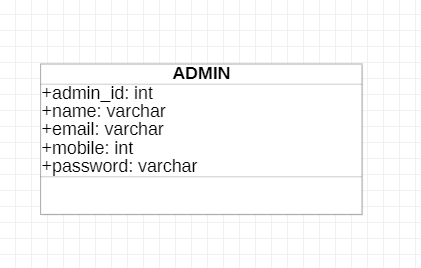


Fig 12: Taxi Booking System - Class Diagram (Admin)

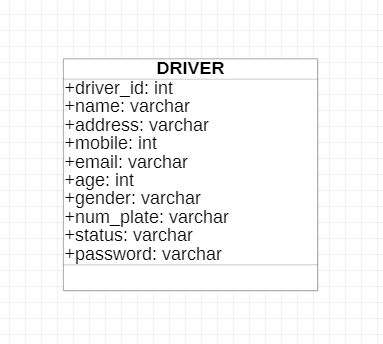


Fig 13: Taxi Booking System - Class Diagram (Driver)

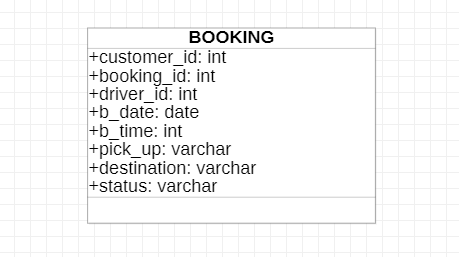


Fig 14: Taxi Booking System - Class Diagram (Booking)

# o Classes (omitting methods / operations)

* Customer (customertid, name, address, email, mobile, dob, gender, password, paymentmethod)

o Administrator (adminid, name, email, mobile, password)

* Booking (customer\_id, booking\_id, driver\_id, date, time, pick\_up, destination, status, paid)
* Taxidriver (driverid, name, address, mobile, email, age, gender, num\_plate, status, password)

o Database Design

## o Data Dictionary Design

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Customer | |  |  |  |  |  |  |
| **Description: Customer details** | |  |  |  |  |  |  |
| **Field Name** | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| customerid (Primary) | int (10)  unsigned | 10 | PK | No |  |  | Autoincremented  Uniquely identifies every customer |
| name | varchar | 255 |  | No |  |  | Name of customer |
| address | varchar | 255 |  | No |  |  | Address of customer |
| mobile | varchar | 255 |  | No |  |  | Mobile of customer |
| email | varchar | 50 |  | No |  | Must be email format containing an @ and a ‘.’ Regex expression used | Email of customer |
| dob | DATE | 20 |  | No |  |  | Date of birth of customer |
| gender | varchar | 255 |  | No |  |  | Gender of the customer |
| password | varchar | 255 |  | No |  |  | TBS password |
| payment | varchar | 255 |  | No |  |  | Payment method |

# Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | customerid | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Booking | |  |  |  |  |  |  |
| **Description: Booking**  **details** | |  |  |  |  |  |  |
| **Field Name** | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| Booking\_id (Primary) | int (10)  unsigned | 10 | PK | No |  |  | Autoincremented  Uniquely identifies every booking |
| customer\_id | int | 10 |  | No |  |  | Customer id for the booking |
| driver\_id | int | 10 |  | No |  |  | Driver id to assign |
| date | date |  |  | No |  |  | Pick\_up date |
| time | time |  |  | No |  |  | Pick\_up time |
| pick\_up | varchar | 255 |  | No |  |  | Pick\_up address |
| destination | varchar | 255 |  | No |  |  | Destination address |
| status | varchar | 255 |  | No |  |  | Status of booking |

# Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | Booking\_id | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Driver | |  |  |  |  |  |  |
| **Description: Driver**  **details** | |  |  |  |  |  |  |
| **Field Name** | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| Driver\_id (Primary) | int (10)  unsigned | 10 | PK | No |  |  | Autoincremented  Uniquely identifies every driver |
| name | varchar | 255 |  | No |  |  | Name of customer |
| address | varchar | 255 |  | No |  |  | Address of customer |
| mobile | Int | 11 |  | No |  |  | Mobile of customer |
| email | varchar | 50 |  | No |  | Must be email format containing an @ and a ‘.’ Regex expression used | Email of customer |
| age | int | 2 |  | No |  |  | Date of birth of customer |
| Number\_plate | varchar | 255 |  | No |  |  | Number plate of driver |
| gender | varchar | 255 |  | No |  |  | Gender of the customer |
| password | varchar | 255 |  | No |  |  | TBS password |
| Payment receive | varchar | 255 |  | No |  |  | Payment receive |

# Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | Driver\_id | No |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Admin | |  |  |  |  |  |  |
| **Description: Admin**  **details** | |  |  |  |  |  |  |
| **Field Name** | **Datatype** | **Length** | **Index** | **Null** | **Default** | **Validation rule** | **Description** |
| Admin\_id (Primary) | int (10)  unsigned | 10 | PK | No |  |  | Autoincremented  Uniquely identifies |
| name | varchar | 255 |  | No |  |  | Name of customer |
| mobile | Int | 11 |  | No |  |  | Mobile of customer |
| email | varchar | 50 |  | No |  | Must be email format containing an @ and a ‘.’ Regex expression used | Email of customer |
| password | varchar | 255 |  | No |  |  | TBS password |

# Indexes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Keyname** | **Type** | **Unique** | **Column** | **Null** |
| PRIMARY | BTREE | Yes | Admin\_id | No |

## o User Interface Design

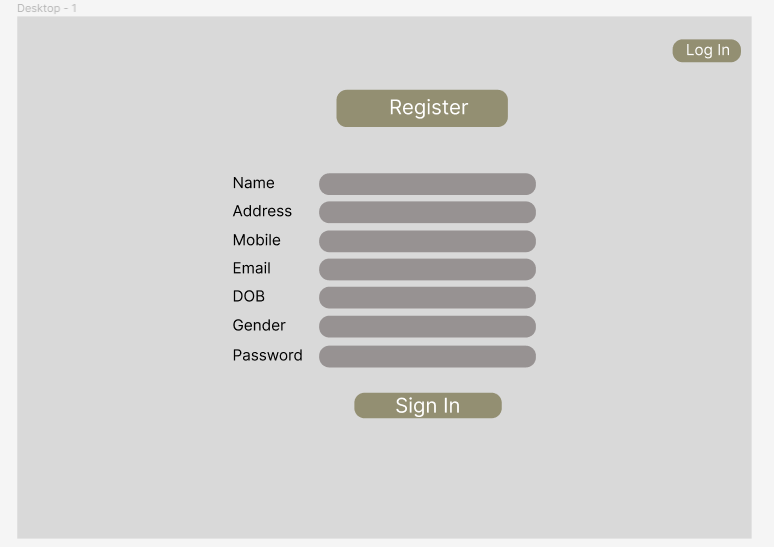


Fig 15: Taxi Booking System – UI (Register)

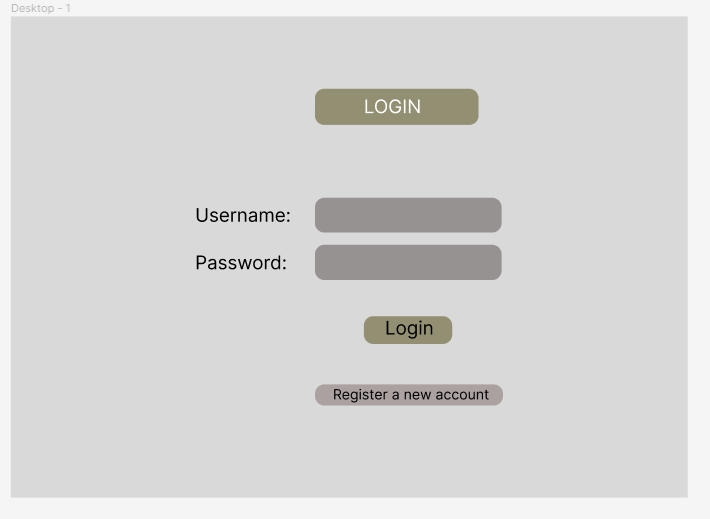


Fig 16: Taxi Booking System – UI (Login)

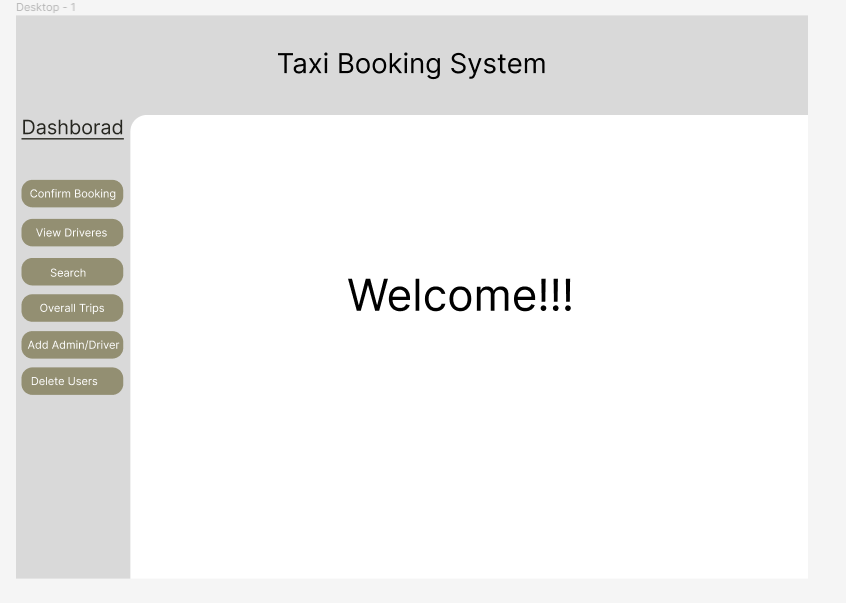


Fig 17: Taxi Booking System – UI (Admin Dashboard)

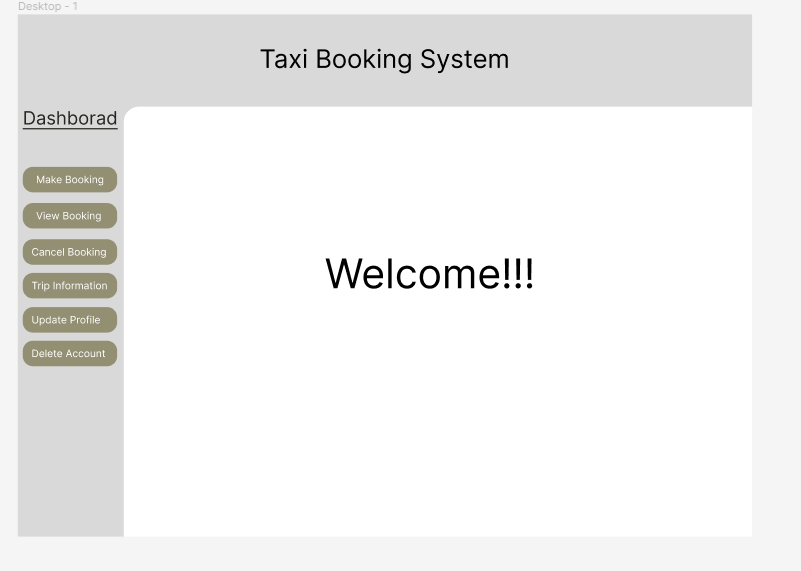


Fig 18: Taxi Booking System – UI (Customer Dashboard)

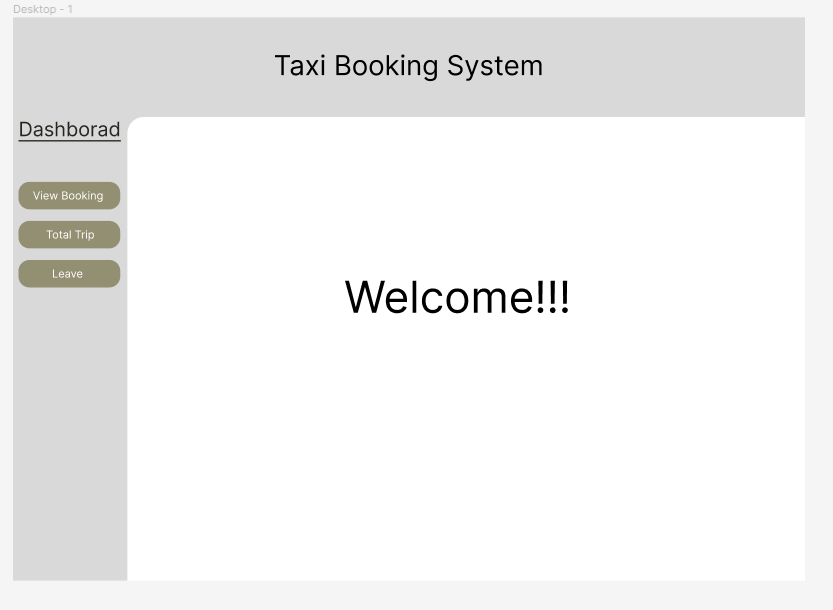


Fig 19: Taxi Booking System – UI (Driver Dashboard)

## Implementation

The application is user-friendly and easy to understand the working mechanism of the application. The application helps to book trips and assign driver as per requested. The program was created using the Software Development Life Cycle (SDLC) method and using python programming language. The development stages of the program are: Requirements & Analysis, Project Planning, Design, Coding & Implementation, Testing.

I have used pycharm as an Integrated Development Environment (IDE). Though it was specially made for python language, it has many various features which has helped me during the development of this application. For the GUI I have used tkinter as a framework and different libraries and package which is available, they are:

* tkinter
* tkcalender
* messagebox
* re
* dateentry
* ttk

During the coding, I faced different kinds of problem while sending and receiving data from database, like while displaying specific category data from certain table and showing in treeview format. As well as while updating. For this problem I searched form many solution by going through youtube, stackoverflow, online tutorial, and by asking seniors and teacher. This all have helped me to learn new technic for the solution and helped in overall software development.



Fig 20: Taxi Booking System – treeview table data show

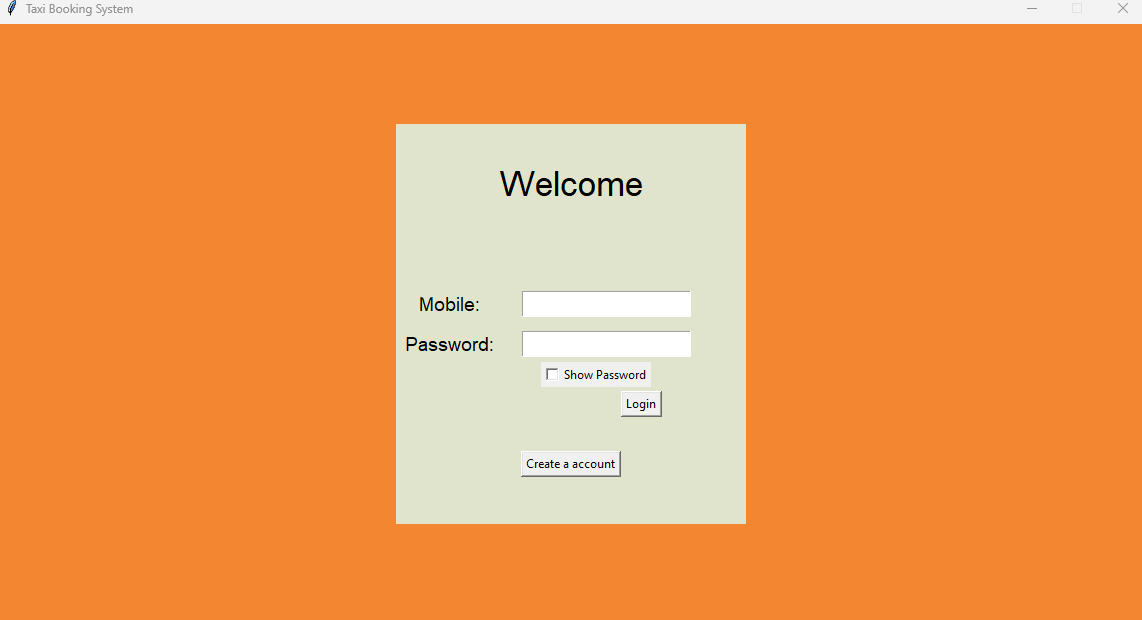


Fig 21: Taxi Booking System – UI(LOGIN)

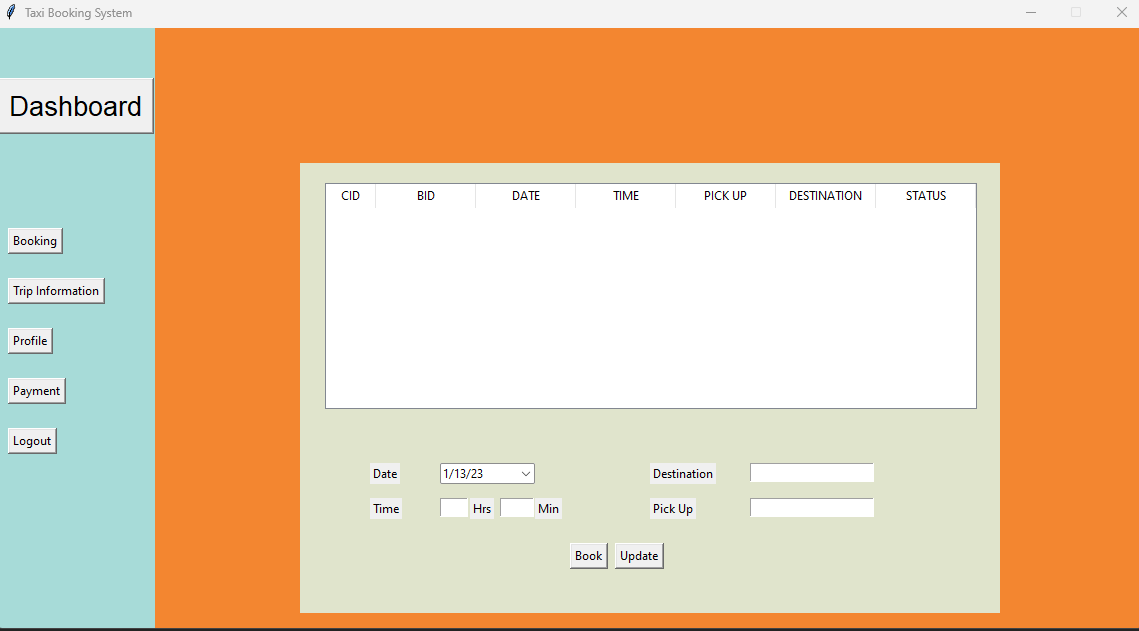


Fig 22: Taxi Booking System – UI(CUSTOMER DASHBOARD)



Fig 23: Taxi Booking System – UI(ADMIN)

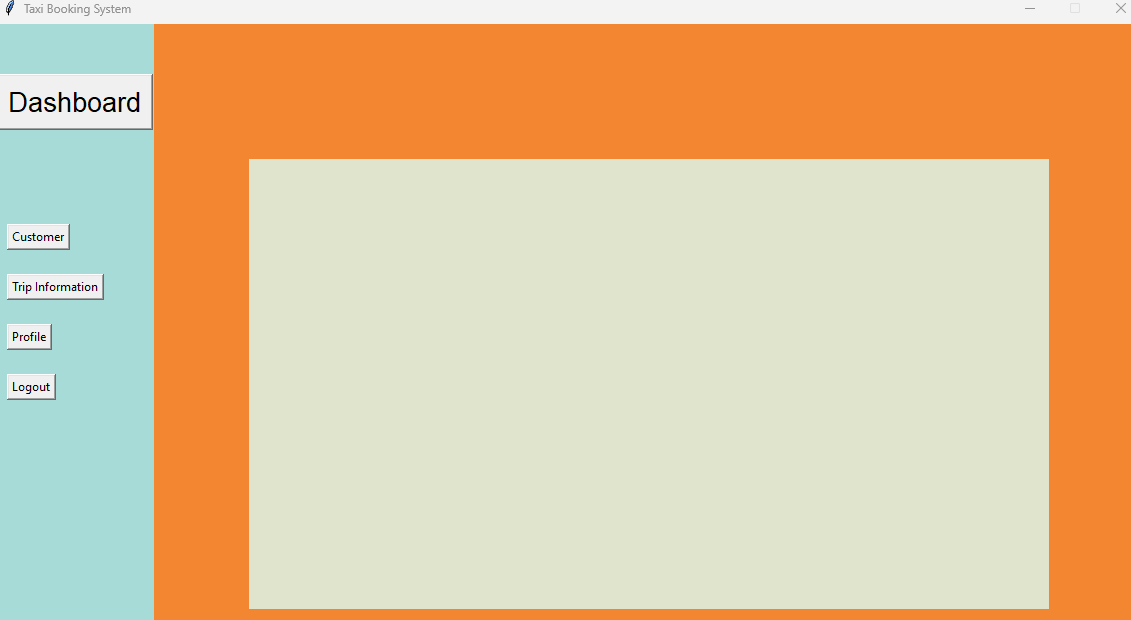


Fig 24: Taxi Booking System – UI(Driver)

As a database system I used MySQL to store the date. To operate MySQL I used XAMPP which I was quite familiar.

During this project I have experience many new things which have help me to be more confident in my analyzing and thinking skill. Moreover, that it has help me to grow my problem-solving skill. While coding I faced various problems which was quite hard for me to understand in the beginning. But later on, after doing research, watched video, read article and self-analyzing now I am sure that I can face the problem with ease. The experience while coding was fun while developing and testing the program, it taught me how to handle as well as manage file properly.

## Testing

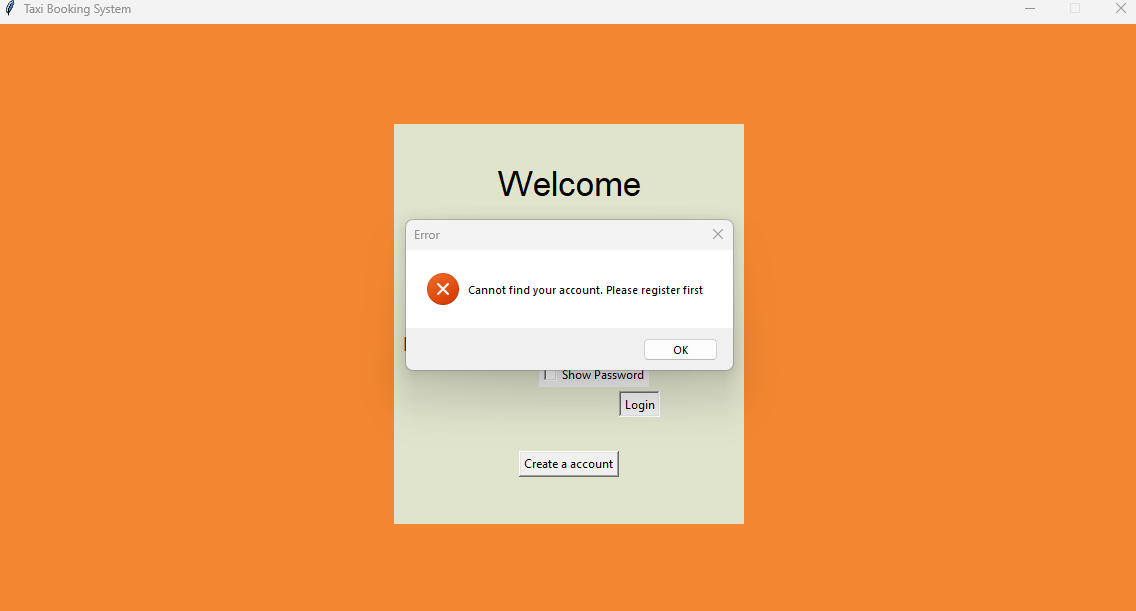


Fig 25: Taxi Booking System –Test during Login

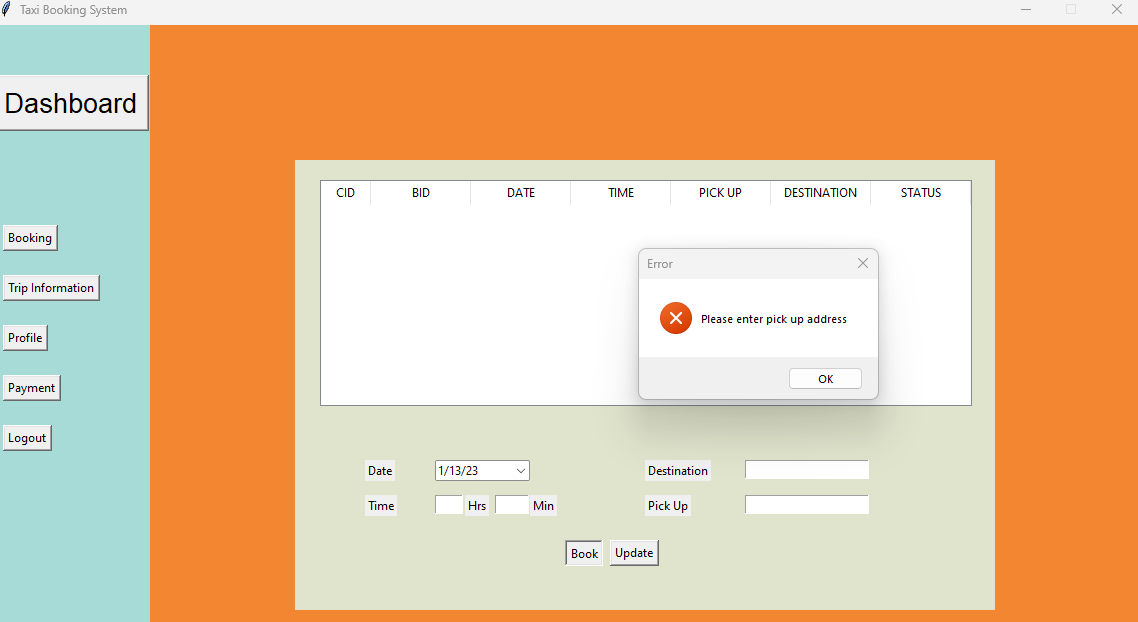


Fig 26: Taxi Booking System – Test during booking

## Discussion / Reflection / Critical Analysis

During the project, almost everything went well. Most of the part for the project was per-schedule and worked in SDLC method. First of all, the designation part went absolutely great. It helps me in the further development of the program. The problems start to occur during coding and implementation. Most of the database parts used to make things complicated while transferring and receiving the data but, after regular interaction I get the hang of the overall process that I was working with.

Basically, I have created the online booking system using python programming language. The application's interface is clear and its operation is simple to comprehend. The tool will enable trip planning and driver assignment. As per all the requirement I have included every possible requirement for the best performance of the application.

I encountered a slew of problems while sending and receiving data from databases, such as when displaying a certain category of data from a particular table in a treeview format. Additionally, while updating. I looked for solutions to this issue on YouTube, StackOverflow, in online tutorials, and by consulting teachers and elders. All of this has aided in my learning of new techniques for the problem and in the development of software as a whole.

The part that stops me to achieve what I hoped to achieve was the smooth switch between frames and while calling modules which was not much important for a while but I think it happened because of lack of good experience and proper training. I will continuously learn over this topic for overall understand in this topic.

For me, I think there was sufficient time to accomplish this project. It was a little bit hard then what I imagine but it was quite fun to learn all those terms and technique to solve this assignment. I have improved in my critical thinking and problem-solving skill during this project period.

This work has enhanced my understanding of python programming language in absolutely great way and encourage me to learn even more deeply in this programming language. The concept from python have also motivate me to learn other programming language like: Java, C++ etc.

If again I have to undertake in similar again, I will definitely approach in new method to tackle the problem. For the future improvement or enhancement I want to recommend to provide overall style and concept as well as the necessary or possible way for the project while considering various possible result.

Conclusion

The project was to develop a software for taxi booking company to run their online booking service. I have completed the software after meeting all the required information given.

# Appendix

UI.py

from tkinter import \*  
import tkinter as tk  
import re  
import messagebox  
from tkinter import ttk  
from tkcalendar import DateEntry  
from Assignment.Manager.admin.admin\_database import adminsearch  
from Assignment.UI.Customer\_Dashboard import CUSTOMER  
from Assignment.UI.Admin\_Dashboard import ADMIN  
from Assignment.Classes.Driver.driver\_class import driverregister  
from Assignment.UI.DRIVER\_Dashboard import DRIVER\_DASHBOARD  
from Assignment.Manager.customer.database import \*  
from Assignment.Manager.driver.driver\_database import \*  
  
  
  
  
  
# create a class for login  
# every frame will work inside this class where tkinter is pass while calling the class LOGIN  
class LOGIN():  
  
 def \_\_init\_\_(self, master):  
 super().\_\_init\_\_()  
 # passing the master to create a framework  
 self.Window\_main = master  
  
 #create frame  
 self.frame = Frame(self.Window\_main, bg="#E0E4CC", height=400, width=350)  
 self.frame.place(relx=0.5, rely=0.5, anchor=CENTER)  
  
 #create label and text box  
 self.headlbl = Label(self.frame, text="Welcome", font=("San Francisco", 26), bg="#E0E4CC")  
 self.usernamelbl = Label(self.frame, text="Mobile:", font=("San Francisco", 14), bg="#E0E4CC")  
 self.passwordlbl = Label(self.frame, text="Password:", font=("San Francisco", 14), bg="#E0E4CC")  
 self.lblMessage = Label(self.frame, text='', bg="#E0E4CC")  
  
 # Create a variable to store the password visibility  
 self.show\_password\_var = tk.IntVar()  
  
 self.usernametxt = Entry(self.frame, width=15, font=("San Francisco", 14), )  
 self.passwordtxt = Entry(self.frame, width=15, show="\*",font=("San Francisco", 14))  
  
 self.loginbtn = Button(self.frame, text="Login", command=self.searchUser)  
 self.registerbtn = Button(self.frame, text="Create a account", command=self.register\_btn)  
  
 self.headlbl.place(relx=0.5, rely=0.15, anchor=CENTER)  
 self.usernamelbl.place(relx=0.15, rely=0.45, anchor=CENTER)  
 self.passwordlbl.place(relx=0.15, rely=0.55, anchor=CENTER)  
 self.usernametxt.place(relx=0.6, rely=0.45, anchor=CENTER)  
 self.passwordtxt.place(relx=0.6, rely=0.55, anchor=CENTER)  
 self.loginbtn.place(relx=0.7, rely=0.7, anchor=CENTER)  
  
 self.registerbtn.place(relx=0.5, rely=0.85, anchor=CENTER)  
  
 # Create the "show password" checkbutton  
 self.show\_password\_checkbutton = tk.Checkbutton(self.frame, text="Show Password",  
 variable=self.show\_password\_var,  
 command=self.show\_password)  
 self.show\_password\_checkbutton.place(x=200, y=250, anchor=CENTER)  
  
 #function to show password  
 def show\_password(self):  
 if self.show\_password\_var.get() == 1:  
 self.passwordtxt.config(show="")  
 else:  
 self.passwordtxt.config(show="\*")  
  
  
  
 def searchUser(self):  
 # Read values from Window  
 # Read value from TextBox  
 mobile = (self.usernametxt.get())  
 password = (self.passwordtxt.get())  
  
 # Send values to search (Middleware)  
 result = customersearch(mobile, password)  
 admin = adminsearch(mobile, password)  
 record = driversearch(mobile, password)  
 print(result)  
  
 def validate\_mobile(mobile):  
 if len(mobile) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid name")  
 return False  
 return True  
  
 def validate\_password(password):  
 if len(password) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid address")  
 return False  
 return True  
  
 MOBILE = validate\_mobile(mobile)  
 PASSWORD = validate\_password(password)  
  
 if MOBILE==True and PASSWORD==True:  
  
 # Display Message  
 #check users from mobile and password  
 if mobile == "admin" and password == "admin":  
 #passing root in admin dashboard after declaring neccessary window frame  
 Window\_main\_admin = self.Window\_main  
 Window\_main\_admin.title("Taxi Booking System")  
 mywidth = 1150  
 myheight = 600  
 screen\_width = Window\_main\_admin.winfo\_screenwidth()  
 screen\_height = Window\_main\_admin.winfo\_screenheight()  
  
 xCordinate = int((screen\_width / 2) - (mywidth / 2))  
 yCordinate = int((screen\_height / 2) - (myheight / 2))  
 Window\_main\_admin.geometry('{}x{}+{}+{}'.format(mywidth, myheight, xCordinate, yCordinate))  
 Window\_main\_admin.resizable(False, False)  
 window = ADMIN(Window\_main\_admin)  
 Window\_main\_admin.mainloop()  
  
 return window  
  
  
 #check for customer users  
 if result == None:  
 self.lblMessage['text'] = "Record not found"  
 else:  
 id = []  
 global cid  
 #getting customer id for specific customer information while login  
 for i in result:  
 j = str(i)  
 id.append(j[0])  
 lol\_string = ''.join(map(str, id))  
 cid = lol\_string[0]  
  
 #passing root in customer dashboard after declaring neccessary window frame  
 Window\_main\_customer = root  
 Window\_main\_customer.title("Taxi Booking System")  
 mywidth = 1150  
 myheight = 600  
 screen\_width = Window\_main\_customer.winfo\_screenwidth()  
 screen\_height = Window\_main\_customer.winfo\_screenheight()  
  
 xCordinate = int((screen\_width / 2) - (mywidth / 2))  
 yCordinate = int((screen\_height / 2) - (myheight / 2))  
 Window\_main\_customer.geometry('{}x{}+{}+{}'.format(mywidth, myheight, xCordinate, yCordinate))  
 Window\_main\_customer.resizable(False, False)  
  
 #call class CUSTOMER\_DASHBOARD and passing frame and customer id  
 window = CUSTOMER(Window\_main\_customer, cid)  
 Window\_main\_customer.mainloop()  
  
 return window  
  
 #checking for driver user  
 if record == None:  
 messagebox.showerror("Error", "Cannot find your account. Please register first")  
 else:  
 id = []  
 global did  
 #getting driver id for specific driver information while login  
 for i in record:  
 j = str(i)  
 id.append(j[0])  
 lol\_string = ''.join(map(str, id))  
 did = lol\_string[0]  
  
 #passing root in driver dashboard after declaring neccessary window frame  
 Window\_main\_driver = self.Window\_main  
 Window\_main\_driver.title("Taxi Booking System")  
 mywidth = 1150  
 myheight = 600  
 screen\_width = Window\_main\_driver.winfo\_screenwidth()  
 screen\_height = Window\_main\_driver.winfo\_screenheight()  
  
 xCordinate = int((screen\_width / 2) - (mywidth / 2))  
 yCordinate = int((screen\_height / 2) - (myheight / 2))  
 Window\_main\_driver.geometry('{}x{}+{}+{}'.format(mywidth, myheight, xCordinate, yCordinate))  
 Window\_main\_driver.resizable(False, False)  
 #call class DRIVER\_DASHBOARD and passing frame and driver id  
 window = DRIVER\_DASHBOARD(Window\_main\_driver, did)  
 Window\_main\_driver.mainloop()  
  
 return window  
  
 #call register page  
 def register\_btn(self):  
 # removing widgit  
 self.frame.destroy()  
 Window\_main\_register = self.Window\_main  
 window = REGISTER(Window\_main\_register)  
 Window\_main\_register.mainloop()  
 return window  
 pass  
  
  
  
  
from Assignment.Classes.Customer.Register import register  
from Assignment.Manager.customer.database import customerinsert  
# create a class for login  
# every frame will work inside this class where tkinter is pass while calling the class LOGIN  
class REGISTER():  
 def \_\_init\_\_(self, master):  
 super().\_\_init\_\_()  
 # passing the master to create a framework  
 self.Window\_main = master  
 #create frame  
 self.frame = Frame(self.Window\_main, bg="#E0E4CC", height=550, width=750)  
 self.frame.place(relx=0.5, rely=0.5, anchor=CENTER)  
  
 # Create a variable to store the password visibility  
 self.show\_password\_var = tk.IntVar()  
  
 #add label and text for registration  
 self.headlbl = Label(self.frame, text="Register", font=("San Francisco", 26), bg="#E0E4CC")  
 self.namelbl = Label(self.frame, text="Name", font=("San Francisco", 14), bg="#E0E4CC")  
 self.addresslbl = Label(self.frame, text="Address", font=("San Francisco", 14), bg="#E0E4CC")  
 self.mobilelbl = Label(self.frame, text="Mobile", font=("San Francisco", 14), bg="#E0E4CC")  
 self.emaillbl = Label(self.frame, text="Email", font=("San Francisco", 14), bg="#E0E4CC")  
 self.doblbl = Label(self.frame, text="DOB", font=("San Francisco", 14), bg="#E0E4CC")  
 self.genderlbl = Label(self.frame, text="Gender", font=("San Francisco", 14), bg="#E0E4CC")  
 self.passwordlbl = Label(self.frame, text="Password", font=("San Francisco", 14), bg="#E0E4CC")  
  
 self.nametxt = Entry(self.frame, width=15, font=("San Francisco", 14), )  
 self.addresstxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.mobiletxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.emailtxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.dobtxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.gendertxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.passwordtxt = Entry(self.frame, width=15, show="\*", font=("San Francisco", 14))  
  
 self.loginbtn = Button(self.frame, text="Login", command=self.login)  
 self.registerbtn = Button(self.frame, text="Register", command=self.saveUser)  
 self.driverbtn = Button(self.frame, text="Driver Account", command=self.driver)  
  
 self.headlbl.place(relx=0.5, rely=0.15, anchor=CENTER)  
 self.namelbl.place(relx=0.35, rely=0.35, anchor=CENTER)  
 self.addresslbl.place(relx=0.3655, rely=0.4, anchor=CENTER)  
 self.mobilelbl.place(relx=0.353, rely=0.45, anchor=CENTER)  
 self.emaillbl.place(relx=0.348, rely=0.5, anchor=CENTER)  
 self.doblbl.place(relx=0.345, rely=0.55, anchor=CENTER)  
 self.genderlbl.place(relx=0.358, rely=0.6, anchor=CENTER)  
 self.passwordlbl.place(relx=0.375, rely=0.65, anchor=CENTER)  
  
 self.nametxt.place(relx=0.6, rely=0.35, anchor=CENTER)  
 self.addresstxt.place(relx=0.6, rely=0.4, anchor=CENTER)  
 self.mobiletxt.place(relx=0.6, rely=0.45, anchor=CENTER)  
 self.emailtxt.place(relx=0.6, rely=0.5, anchor=CENTER)  
  
 self.dob\_txt = DateEntry(self.frame,width=12, background='darkblue',foreground='white', borderwidth=2)  
 self.dob\_txt.place(relx=0.6, rely=0.55, width=170, anchor=CENTER)  
 self.dob\_txt.bind("<<DateEntrySelected>>")  
 self.gendertxt.place(relx=0.6, rely=0.6, anchor=CENTER)  
 self.passwordtxt.place(relx=0.6, rely=0.65, anchor=CENTER)  
  
 # Create the "show password" checkbutton  
 self.show\_password\_checkbutton = tk.Checkbutton(self.frame, text="Show Password",  
 variable=self.show\_password\_var,  
 command=self.show\_password)  
 self.show\_password\_checkbutton.place(x=400, y=400, anchor=CENTER)  
  
 self.loginbtn.place(relx=0.9, rely=0.1, anchor=CENTER)  
 self.registerbtn.place(x=480, y=450)  
 self.driverbtn.place(x=10, y=20)  
  
 # function to show password  
 def show\_password(self):  
 if self.show\_password\_var.get() == 1:  
 self.passwordtxt.config(show="")  
 else:  
 self.passwordtxt.config(show="\*")  
  
 #save new register  
 def saveUser(self):  
  
 # Read values from Window  
 # Read value from TextBox  
 name = (self.nametxt.get())  
 address = (self.addresstxt.get())  
 mobile = (self.mobiletxt.get())  
 email = (self.emailtxt.get())  
 dob = (self.dobtxt.get())  
 gender = (self.gendertxt.get())  
 password = (self.passwordtxt.get())  
  
 def validate\_name(name):  
 if len(name) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid name")  
 return False  
 return True  
  
 def validate\_address(address):  
 if len(address) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid address")  
 return False  
 return True  
  
 def validate\_mobile(mobile):  
 num = [mobile]  
 result = customergetAll1(num)  
 if result:  
 messagebox.showerror("Error", "Mobile number already exist")  
 return False  
 else:  
 return True  
  
 def validate\_email(email):  
 # Add your validation code here  
 # For example, you can use a regular expression to check if the email is in the correct format  
 if not re.match(r'[^@]+@[^@]+\.[^@]+', email):  
 messagebox.showerror("Error", "Please enter a valid email address")  
 return False  
 return True  
  
  
 def validate\_gender(gender):  
 # Add your validation code here  
 # For example, you can check if the gender is "male" or "female"  
 if gender.lower() not in ["male", "female"]:  
 messagebox.showerror("Error", "Please enter a valid gender (male or female)")  
 return False  
 return True  
  
 def validate\_password(password):  
 # Add your validation code here  
 # For example, you can check if the password meets certain criteria (e.g., length, complexity)  
 if len(password) < 8:  
 messagebox.showerror("Error", "Password must be at least 8 characters long")  
 return False  
 if not any(char.isdigit() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one digit")  
 return False  
 if not any(char.isupper() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one uppercase letter")  
 return False  
 if not any(char.islower() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one lowercase letter")  
 return False  
 return True  
  
 NAME = validate\_name(name)  
 ADDRESS = validate\_address(address)  
 MOBILE = validate\_mobile(mobile)  
 EMAIL = validate\_email(email)  
 GENDER = validate\_gender(gender)  
 PASSWORD = validate\_password(password)  
  
 if NAME == True and ADDRESS == True and MOBILE == True and EMAIL == True and GENDER == True and PASSWORD == True:  
 c1 = register("", name, address, mobile, email, self.dobtxt.get(), gender, password)  
  
 # Send values to save (database)  
 result = customerinsert(c1)  
 # Display Message  
 if result['status']==True:  
 # self.lblMessage['text']="Save Record"  
 messagebox.showinfo("Done", "Account created, Now you can log in using mobile number as password")  
  
 #switch to login  
 def login(self):  
 self.frame.destroy()  
 window = self.Window\_main  
 c = LOGIN(window)  
 return c  
  
 # switch driver frame  
 def driver(self):  
 self.frame.destroy()  
 window = self.Window\_main  
 c = DRIVER(window)  
 return c  
  
  
  
  
# create a class for login  
# every frame will work inside this class where tkinter is pass while calling the class LOGIN  
class DRIVER():  
 def \_\_init\_\_(self, master):  
 super().\_\_init\_\_()  
 # passing the master to create a framework  
 self.Window\_main = master  
 #create frame  
 self.frame = Frame(self.Window\_main, bg="#E0E4CC", height=550, width=750)  
 self.frame.place(relx=0.5, rely=0.5, anchor=CENTER)  
  
 #add label and text box  
 self.headlbl = Label(self.frame, text="Register", font=("San Francisco", 26), bg="#E0E4CC")  
 self.namelbl = Label(self.frame, text="Name", font=("San Francisco", 14), bg="#E0E4CC")  
 self.addresslbl = Label(self.frame, text="Address", font=("San Francisco", 14), bg="#E0E4CC")  
 self.mobilelbl = Label(self.frame, text="Mobile", font=("San Francisco", 14), bg="#E0E4CC")  
 self.emaillbl = Label(self.frame, text="Email", font=("San Francisco", 14), bg="#E0E4CC")  
 self.doblbl = Label(self.frame, text="Age", font=("San Francisco", 14), bg="#E0E4CC")  
 self.genderlbl = Label(self.frame, text="Gender", font=("San Francisco", 14), bg="#E0E4CC")  
 self.passwordlbl = Label(self.frame, text="Password", font=("San Francisco", 14), bg="#E0E4CC")  
 self.numberplatelbl = Label(self.frame, text="Num. Plate", font=("San Francisco", 14), bg="#E0E4CC")  
  
  
 self.nametxt = Entry(self.frame, width=15, font=("San Francisco", 14), )  
 self.addresstxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.mobiletxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.emailtxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.dobtxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.gendertxt = Entry(self.frame, width=15, font=("San Francisco", 14))  
 self.passwordtxt = Entry(self.frame, width=15, show="\*", font=("San Francisco", 14))  
  
 self.numberplate = ttk.Combobox(self.frame,width=13, font=("San Francisco", 14))  
 self.numberplate['values'] = ['STATE 1','STATE 2','STATE 3','STATE 4','STATE 5','STATE 6','STATE 7']  
  
 self.registerbtn = Button(self.frame, text="Register", command=self.saveUser)  
  
 self.headlbl.place(relx=0.5, rely=0.15, anchor=CENTER)  
 self.namelbl.place(relx=0.35, rely=0.35, anchor=CENTER)  
 self.addresslbl.place(relx=0.3655, rely=0.4, anchor=CENTER)  
 self.mobilelbl.place(relx=0.353, rely=0.45, anchor=CENTER)  
 self.emaillbl.place(relx=0.348, rely=0.5, anchor=CENTER)  
 self.doblbl.place(relx=0.345, rely=0.55, anchor=CENTER)  
 self.genderlbl.place(relx=0.358, rely=0.6, anchor=CENTER)  
 self.passwordlbl.place(relx=0.375, rely=0.65, anchor=CENTER)  
 self.numberplatelbl.place(relx=0.375, rely=0.7, anchor=CENTER)  
  
 self.nametxt.place(relx=0.6, rely=0.35, anchor=CENTER)  
 self.addresstxt.place(relx=0.6, rely=0.4, anchor=CENTER)  
 self.mobiletxt.place(relx=0.6, rely=0.45, anchor=CENTER)  
 self.emailtxt.place(relx=0.6, rely=0.5, anchor=CENTER)  
 self.dobtxt.place(relx=0.6, rely=0.55, anchor=CENTER)  
 self.gendertxt.place(relx=0.6, rely=0.6, anchor=CENTER)  
 self.passwordtxt.place(relx=0.6, rely=0.65, anchor=CENTER)  
 self.numberplate.place(relx=0.6, rely=0.7, anchor=CENTER)  
  
 self.registerbtn.place(x=480, y=450)  
  
 self.loginbtn = Button(self.frame, text="Login", command=self.login)  
 self.loginbtn.place(relx=0.9, rely=0.1, anchor=CENTER)  
 def saveUser(self):  
  
 # Read values from Window  
 # Read value from TextBox  
 name = (self.nametxt.get())  
 address = (self.addresstxt.get())  
 mobile = (self.mobiletxt.get())  
 email = (self.emailtxt.get())  
 dob = (self.dobtxt.get())  
 gender = (self.gendertxt.get())  
 number\_plate = (self.numberplate.get())  
 password = (self.passwordtxt.get())  
 status = "open"  
  
 def validate\_name(name):  
 if len(name) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid name")  
 return False  
 return True  
  
 def validate\_address(address):  
 if len(address) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid address")  
 return False  
 return True  
  
 def validate\_mobile(mobile):  
 num = [mobile]  
 result = customergetAll1(num)  
 if result:  
 messagebox.showerror("Error", "Mobile number already exist")  
 return False  
 else:  
 return True  
  
 def validate\_email(email):  
 # Add your validation code here  
 # For example, you can use a regular expression to check if the email is in the correct format  
 if not re.match(r'[^@]+@[^@]+\.[^@]+', email):  
 messagebox.showerror("Error", "Please enter a valid email address")  
 return False  
 return True  
  
 def validate\_dob(dob):  
 if len(dob) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter a valid date")  
 return False  
 return True  
  
 def validate\_gender(gender):  
 # Add your validation code here  
 # For example, you can check if the gender is "male" or "female"  
 if gender.lower() not in ["male", "female"]:  
 messagebox.showerror("Error", "Please enter a valid gender (male or female)")  
 return False  
 return True  
  
 def validate\_password(password):  
 # Add your validation code here  
 # For example, you can check if the password meets certain criteria (e.g., length, complexity)  
 if len(password) < 8:  
 messagebox.showerror("Error", "Password must be at least 8 characters long")  
 return False  
 if not any(char.isdigit() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one digit")  
 return False  
 if not any(char.isupper() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one uppercase letter")  
 return False  
 if not any(char.islower() for char in password):  
 messagebox.showerror("Error", "Password must contain at least one lowercase letter")  
 return False  
 return True  
  
 NAME = validate\_name(name)  
 ADDRESS = validate\_address(address)  
 MOBILE = validate\_mobile(mobile)  
 EMAIL = validate\_email(email)  
 DOB = validate\_dob(dob)  
 GENDER = validate\_gender(gender)  
 PASSWORD = validate\_password(password)  
  
 if NAME == True and ADDRESS == True and MOBILE == True and EMAIL == True and DOB == True and GENDER == True and PASSWORD == True:  
 c1 = driverregister("", name, address, mobile, email, dob, gender, number\_plate, password, status)  
  
 # Send values to save (database)  
 result = driverinsert(c1)  
 # Display Message  
 if result['status']==True:  
 # self.lblMessage['text']="Save Record"  
 messagebox.showinfo("Done", "Account created, Now you can log in using mobile number as password")  
  
 #switch to login  
 def login(self):  
 self.frame.destroy()  
 window = self.Window\_main  
 c = LOGIN(window)  
 return c  
  
  
#declearing main root  
root = Tk()  
root.title("Taxi Booking System")  
  
mywidth = 1150  
myheight = 600  
#getting device display width and height  
screen\_width = root.winfo\_screenwidth()  
screen\_height = root.winfo\_screenheight()  
  
xCordinate = int((screen\_width / 2) - (mywidth / 2))  
yCordinate = int((screen\_height / 2) - (myheight / 2))  
root.geometry('{}x{}+{}+{}'.format(mywidth, myheight, xCordinate, yCordinate))  
root.resizable(False, False)  
root['background']='#F38630'  
  
#create frame  
frame = Frame(root, bg="#E0E4CC", height=400, width=350)  
frame.place(relx=0.5, rely=0.5, anchor=CENTER)  
  
#call login class  
def login\_main():  
 window = LOGIN(root)  
 return window  
  
#call register class  
def register\_main():  
 window2 = REGISTER(root)  
 return window2  
  
#add label  
headlbl = Label(frame, text="Welcome", font=("San Francisco", 26), bg="#E0E4CC")  
loginbtn = Button(frame, text="Login",width = 20, command=login\_main)  
registerbtn = Button(frame, text="Create a account",width= 20, command=register\_main)  
  
headlbl.place(relx=0.5, rely=0.15, anchor=CENTER)  
loginbtn.place(relx=0.5, rely=0.47, anchor=CENTER)  
registerbtn.place(relx=0.5, rely=0.6, anchor=CENTER)  
  
  
root.mainloop()

The LOGIN class takes a master parameter, which is used to create a frame for the login interface. The frame contains labels, text boxes, and buttons for the user to enter their mobile number, password, and login or register. The script also includes a checkbutton that allows the user to show or hide their password. The script also has function calls to other classes and functions like CUSTOMER, ADMIN, driverregister, searchUser, and register\_btn. These classes and functions are probably used to handle the login process and redirect the user to the appropriate dashboard based on their role. The LOGIN class that is used to search for a user in the system based on the mobile number and password entered by the user in the login interface. The function reads the values entered in the mobile and password text boxes and sends them to the customersearch, adminsearch, and driversearch functions to search for the user. The function also includes validation for the mobile and password fields, to ensure that they are not empty. If the validation passes, the function checks if the user is an admin, customer, or driver and redirects them to the appropriate dashboard by calling the appropriate class (ADMIN, CUSTOMER, DRIVER\_DASHBOARD) and passing the necessary parameters. Additionally, the function stores the customer and driver ids in global variables (cid, did) for use in other parts of the program.Class called DRIVER, which is used to create a registration interface for a driver. The DRIVER class takes a master parameter, which is used to create a frame for the registration interface. The frame contains labels and text boxes for the user to enter their name, address, mobile number, email, date of birth, gender, password, and number plate. The script also includes a ttk Combobox that allows the user to select their number plate state. The script also has a button that allows the user to register by calling the saveUser function.

Customer\_Dashboard.py

from Assignment.Classes.Booking.booking\_classes import booking  
from tkinter import \*  
import messagebox  
from Assignment.Manager.booking.booking\_database import \*  
from Assignment.Manager.customer.database import \*  
from Assignment.Manager.driver.driver\_database import \*  
from tkinter import ttk  
from tkcalendar import DateEntry  
  
  
# create a class for customer dashboard  
# every frame will work inside this class where tkinter is pass while calling the class CUSTOMER  
class CUSTOMER():  
 def \_\_init\_\_(self, master, cid):  
 super().\_\_init\_\_()  
 # passing the master to create a framework  
 self.Window\_main = master  
 #passing the cid receive during login of the customer  
 self.cid = cid  
  
 screen\_width = self.Window\_main.winfo\_screenwidth()  
 screen\_height = self.Window\_main.winfo\_screenheight()  
  
 #creating seperate frames for an individual widgit purpose  
 self.side\_frame = Frame(self.Window\_main, bg="#A7DBD8", height=screen\_height, width=screen\_width)  
 self.side\_frame.pack(side=LEFT, fill=BOTH)  
 self.side\_frame.pack\_propagate(False)  
  
 self.main\_frame = Frame(self.side\_frame, bg="#F38630", height=510, width=990)  
 self.main\_frame.pack(side=RIGHT, fill=BOTH, anchor=SE)  
  
 self.booking\_frame = Frame(self.main\_frame, bg="#E0E4CC", height=450, width=700)  
 self.booking\_frame.place(relx=0.5, rely=0.6, anchor=CENTER)  
  
 #main buttons which will help to call the function as per needed  
 dashboard\_btn = Button(self.side\_frame, text="Dashboard", font=("San Francisco", 20))  
 booking\_btn = Button(self.side\_frame, text="Booking", command=self.Booking\_btn)  
 tripinfo\_btn = Button(self.side\_frame, text="Trip Information", command=self.tripinfo)  
 delete\_profile\_btn = Button(self.side\_frame, text="Profile", command=self.Account)  
 payment\_btn = Button(self.side\_frame, text="Payment", command=self.payment)  
 logout\_btn = Button(self.side\_frame, text="Logout", command=self.btn\_clicked)  
  
 dashboard\_btn.place(x=2, y=50)  
 booking\_btn.place(x=13, y=200)  
 tripinfo\_btn.place(x=13, y=250)  
 delete\_profile\_btn.place(x=13, y=300)  
 payment\_btn.place(x=13, y=350)  
 logout\_btn.place(x=13, y=400)  
  
 def payment(self):  
  
 payment\_lbl = Label(self.booking\_frame, text="Payment",font=("San Francisco", 20))  
 d\_num\_lbl = Label(self.booking\_frame, text="Driver number")  
 p\_option\_lbl = Label(self.booking\_frame, text="Payment Options")  
 total\_lbl = Label(self.booking\_frame, text="Total")  
 pay\_lbl = Label(self.booking\_frame, text="Amount")  
  
 p\_option\_txt = ttk.Combobox(self.booking\_frame)  
 p\_option\_txt['values'] = ['Esawa','Kalthi']  
 total\_txt = Entry(self.booking\_frame)  
 d\_num\_txt = Entry(self.booking\_frame)  
 pay\_txt = Entry(self.booking\_frame)  
  
 payment\_lbl.place(x=70, y=100)  
 d\_num\_lbl.place(x=70, y=150)  
 p\_option\_lbl.place(x=70, y=200)  
 total\_lbl.place(x=70, y=250)  
 pay\_lbl.place(x=70, y=300)  
  
 # payment\_txt.place(x=140, y=300)  
 d\_num\_txt.place(x=160, y=150)  
 p\_option\_txt.place(x=160, y=200)  
 total\_txt.place(x=160, y=250)  
 pay\_txt.place(x=160, y=300)  
  
 pay\_btn = Button(self.booking\_frame, text="Pay", command='')  
 pay\_btn.place(x=270, y=380)  
  
  
#function to redirect in login page  
 def btn\_clicked(self):  
 messagebox.askyesno("Confirm","Do you really want to logout")  
 while True:  
 self.login()  
  
 def login(self):  
 self.Window\_main.destroy()  
 from Assignment.UI import UI  
 return UI  
 pass  
  
#function to delete the widgit while switching between different function operation  
 def delete\_frame(self):  
 for frame in self.booking\_frame.winfo\_children():  
 frame.destroy()  
  
  
#booking button to make a trip order  
 def Booking\_btn(self):  
 #removing widgit  
 self.delete\_frame()  
  
 cid = self.cid  
 #searching the pending date from the database for the further new request of trips  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 cursor = conn.cursor()  
 sql = "SELECT \* FROM booking WHERE cid=%s AND status=%s"  
 CID = cid  
 status = "pending"  
 values = (CID, status)  
 cursor.execute(sql, values)  
 records = cursor.fetchall()  
  
  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if records==None:  
 lblMessage['text'] = "Record not found"  
  
  
 else:  
 #frame for table  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=25, y=20)  
  
 #creating treeview table to show the customer information in tables to understand easily  
 tblpersons = ttk.Treeview(tableframe)  
 # Define Our Columns  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 # Format Our Columns  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 # Create Headings  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
  
 global count  
 count = 0  
 #display record  
 for record in records:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
 tblpersons.pack()  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 #show record in txt,label and remove it before displaying  
 def select\_record(e):  
 # Clear entry boxes  
 date\_txt.delete(0, END)  
 hrs\_txt.delete(0, END)  
 min\_txt.delete(0, END)  
 pick\_up\_txt.delete(0, END)  
 destination\_txt.delete(0, END)  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 date\_txt.insert(0, values[2])  
 time = []  
 for t in values[3]:  
 time.append(t[0])  
  
 hrs\_txt.delete(0, len(hrs\_txt.get()))  
 hrs\_txt.insert(0, time[1])  
 hrs\_txt.insert(0, time[0])  
 min\_txt.delete(0, len(min\_txt.get()))  
 min\_txt.insert(0, time[6])  
 min\_txt.insert(0, time[3])  
  
 pick\_up\_txt.insert(0, values[4])  
 destination\_txt.insert(0, values[5])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 def savebooking():  
 # Read values from Window  
 Cid = self.cid  
 date = date\_txt.get\_date() # Read value from TextBox  
 time = (hrs\_txt.get() + min\_txt.get() + "00")  
 pick\_up = (pick\_up\_txt.get())  
 destination = (destination\_txt.get())  
 status = "pending"  
  
 #validation during booking  
 def validate\_time(time):  
 if len(time)==None:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter time")  
 return False  
 return True  
  
 def validate\_pick\_up(pick\_up):  
 if len(pick\_up) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter pick up address")  
 return False  
 return True  
  
 def validate\_destination(destintaion):  
 if len(destination) == 0:  
 # Display an error message in a message box  
 messagebox.showerror("Error", "Please enter destination address")  
 return False  
 return True  
  
 TIME = validate\_time(time)  
 PICK\_UP = validate\_pick\_up(pick\_up)  
 DESTINATION = validate\_destination(destination)  
  
 if DESTINATION==True:  
 # Send values to search (Middleware)  
 c1 = booking(Cid, date, time, pick\_up, destination, status)  
  
 # Send values to save (Middleware)  
 result = bookinginsert(c1)  
 # Display Message  
  
 if result['status'] == True:  
 messagebox.showinfo("Done", "Booked")  
 lblMessage['text'] = "Save Record"  
 else:  
 lblMessage['text'] = "Error to save"  
  
  
 #function to update the record of booked trip for readjustment  
 def update\_record():  
 # Grab the record number  
 selected = tblpersons.focus()  
 time = (hrs\_txt.get() + min\_txt.get() + "00")  
 cid = self.cid  
 bd = []  
 status = "pending"  
 for b in selection(bid=selection):  
 bd.append(b[0])  
 lol\_string = ''.join(map(str, bd))  
 bid = lol\_string  
  
 # Update record  
 tblpersons.item(selected, text="", values=(cid, bid,  
 date\_txt.get(), time, pick\_up\_txt.get(),  
 destination\_txt.get(),  
 status))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
  
 sql = "UPDATE booking SET date=%s, time=%s, pick\_up=%s, destination=%sWHERE bid=%s"  
 values = (date\_txt.get(), time, pick\_up\_txt.get(),  
 destination\_txt.get(), bid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 # Clear entry boxes  
 date\_txt.delete(0, END)  
 hrs\_txt.delete(0, END)  
 min\_txt.delete(0, END)  
 pick\_up\_txt.delete(0, END)  
 destination\_txt.delete(0, END)  
  
 # Add label  
 date\_lbl = Label(self.booking\_frame, text="Date")  
 time\_lbl = Label(self.booking\_frame, text="Time")  
 hrs\_lbl = Label(self.booking\_frame, text="Hrs")  
 min\_lbl = Label(self.booking\_frame, text="Min")  
 pick\_up\_lbl = Label(self.booking\_frame, text="Pick Up")  
 destination\_lbl = Label(self.booking\_frame, text="Destination")  
  
 # Add Record Entry Boxes  
 #import clander from libery to make date selection  
 date\_txt = DateEntry(self.booking\_frame, width=12, background='darkblue',  
 foreground='white', borderwidth=2)  
 date\_txt.place(x=140, y=300)  
 date\_txt.bind("<<DateEntrySelected>>")  
  
 hrs\_txt = Entry(self.booking\_frame, width=4)  
 min\_txt = Entry(self.booking\_frame, width=5)  
 pick\_up\_txt = Entry(self.booking\_frame)  
 destination\_txt = Entry(self.booking\_frame)  
  
 #button to update and for new trip to book  
 update\_btn = Button(self.booking\_frame, text="Update", command=lambda: [update\_record()])  
 book\_btn = Button(self.booking\_frame, text="Book", command=savebooking)  
  
 date\_lbl.place(x=70, y=300)  
 time\_lbl.place(x=70, y=335)  
 hrs\_lbl.place(x=170, y=335)  
 min\_lbl.place(x=235, y=335)  
 pick\_up\_lbl.place(x=350, y=335)  
 destination\_lbl.place(x=350, y=300)  
  
 hrs\_txt.place(x=140, y=335)  
 min\_txt.place(x=200, y=335)  
 pick\_up\_txt.place(x=450, y=335)  
 destination\_txt.place(x=450, y=300)  
  
 update\_btn.place(x=315, y=380)  
 book\_btn.place(x=270, y=380)  
  
 #this function will provide the overall trip information of the customer  
 def tripinfo(self):  
 self.delete\_frame()  
  
 # Read values  
 cid = [self.cid]  
 result = bookinggetAll1(cid)  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=25, y=20)  
  
 # creating treeview table to show the customer information in tables to understand easily  
 tblpersons = ttk.Treeview(tableframe)  
  
 # Define Our Columns  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 # Format Our Columns  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 # Create Headings  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
 #display result  
 for dt in result:  
 tblpersons.insert(parent='', index='end', values=(dt[0], dt[1], dt[2], dt[3], dt[4], dt[5], dt[6]))  
  
 tblpersons.pack()  
  
 #show record in txt,label and remove it before displaying  
 def select\_record(e):  
 global bid, did  
 # Clear entry boxes  
 date\_lbl1.config(text="")  
 time\_lbl1.config(text="")  
 pick\_up\_lbl1.config(text="")  
 destination\_lbl1.config(text="")  
  
 d\_name\_lbl1.config(text="")  
 d\_mobile\_lbl1.config(text="")  
 d\_age\_lbl1.config(text="")  
 d\_gender\_lbl1.config(text="")  
 d\_num\_plate\_lbl1.config(text="")  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 #get booking id of selected value  
 bid = values[1]  
 #search values in database regarding the booking id  
 record = bookingsearch1(bid)  
  
 #passing driver id retrived from database which was assigh to the customer by admin  
 did = record  
  
 #search driver detail in database using driver id  
 # will display the driver information through reciving driver id from booking database  
 # and passing it to the database table to get driver information  
 record1 = driversearch1(did)  
 d\_values = record1  
  
 #applying condition when cancel button will appear  
 if values[6] == "active":  
 cancel\_btn = Button(self.booking\_frame, text="Cancel",  
 command=lambda: [update\_driver1(), cancel\_booking()])  
 cancel\_btn.place(x=300, y=420)  
  
 if values[6] == "pending":  
 cancel\_btn = Button(self.booking\_frame, text="Cancel",  
 command=lambda: [cancel\_booking1()])  
 cancel\_btn.place(x=300, y=420)  
  
 # outpus to entry boxes  
 date\_lbl1.config(text=values[2])  
 time\_lbl1.config(text=values[3])  
 pick\_up\_lbl1.config(text=values[4])  
 destination\_lbl1.config(text=values[5])  
  
 d\_name\_lbl1.config(text=d\_values[1])  
 d\_mobile\_lbl1.config(text=d\_values[3])  
 d\_age\_lbl1.config(text=d\_values[5])  
 d\_gender\_lbl1.config(text=d\_values[6])  
 d\_num\_plate\_lbl1.config(text=d\_values[7])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 #cancel the trip having status active  
 def cancel\_booking():  
 bookingid = bid  
 record = bookingcancel(bookingid)  
 return record  
  
 #cancel the trip having status pending  
 def cancel\_booking1():  
 bookingid = bid  
 record = bookingcancel1(bookingid)  
 return record  
  
  
 def update\_driver1():  
 #get driver id which is marked as global variable  
 driverid = did  
 #also update the driver status while the trip is cancel  
 conn1 = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 sql = "UPDATE driver SET status='open' WHERE did=%s"  
 values = (driverid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = conn1  
 cursor = conn.cursor(buffered=True)  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 #trip heading  
 active\_lbl = Label(self.booking\_frame, text="Trip", font=("San Francisco", 15))  
 active\_lbl.place(x=340, y=260)  
  
 #add label for trip information  
 date\_lbl = Label(self.booking\_frame, text="Date")  
 time\_lbl = Label(self.booking\_frame, text="Time")  
 pick\_up\_lbl = Label(self.booking\_frame, text="Pick Up")  
 destination\_lbl = Label(self.booking\_frame, text="Destination")  
 driver\_lbl = Label(self.booking\_frame, text="Driver Information")  
  
 #add label for driver information to show to the customer who is their driver and driver related information  
 d\_name\_lbl = Label(self.booking\_frame, text="Name")  
 d\_mobile\_lbl = Label(self.booking\_frame, text="Mobile")  
 d\_age\_lbl = Label(self.booking\_frame, text="Age")  
 d\_gender\_lbl = Label(self.booking\_frame, text="Gender")  
 d\_num\_plate\_lbl = Label(self.booking\_frame, text="Number Plate")  
  
 date\_lbl.place(x=70, y=295)  
 time\_lbl.place(x=70, y=325)  
 pick\_up\_lbl.place(x=70, y=355)  
 destination\_lbl.place(x=70, y=385)  
  
 d\_name\_lbl.place(x=420, y=295)  
 d\_mobile\_lbl.place(x=420, y=325)  
 d\_age\_lbl.place(x=420, y=355)  
 d\_gender\_lbl.place(x=420, y=385)  
 d\_num\_plate\_lbl.place(x=420, y=415)  
  
 driver\_lbl.place(x=430, y=260)  
 lblMessage.place(x=250, y=400)  
  
 # add label for trip information, put data from table  
 date\_lbl1 = Label(self.booking\_frame, text="")  
 time\_lbl1 = Label(self.booking\_frame, text="")  
 pick\_up\_lbl1 = Label(self.booking\_frame, text="")  
 destination\_lbl1 = Label(self.booking\_frame, text="")  
  
 # add label for driver information to show to the customer who is their driver and driver related information  
 #will display the driver information through reciving driver id from booking database  
 #and passing it to the database table to get driver information  
 d\_name\_lbl1 = Label(self.booking\_frame, text="")  
 d\_mobile\_lbl1 = Label(self.booking\_frame, text="")  
 d\_age\_lbl1 = Label(self.booking\_frame, text="")  
 d\_gender\_lbl1 = Label(self.booking\_frame, text="")  
 d\_num\_plate\_lbl1 = Label(self.booking\_frame, text="")  
  
 date\_lbl1.place(x=140, y=295)  
 time\_lbl1.place(x=140, y=325)  
 pick\_up\_lbl1.place(x=140, y=355)  
 destination\_lbl1.place(x=140, y=385)  
  
 d\_name\_lbl1.place(x=520, y=295)  
 d\_mobile\_lbl1.place(x=520, y=325)  
 d\_age\_lbl1.place(x=520, y=355)  
 d\_gender\_lbl1.place(x=520, y=385)  
 d\_num\_plate\_lbl1.place(x=520, y=415)  
  
 pass  
  
 # show only personal infornation  
 def Account(self):  
 self.delete\_frame()  
 # Read values  
 CID = [self.cid]  
 print(CID)  
 #search customer personal information only  
 result = customersearch1(CID)  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=25, y=20)  
  
 tblpersons = ttk.Treeview(tableframe)  
 tblpersons['column'] = ('name', 'address', 'mobile', 'email', 'dob', 'gender', 'password')  
  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("name", width=50, anchor=CENTER)  
 tblpersons.column("address", width=100, anchor=CENTER)  
 tblpersons.column("mobile", width=100, anchor=CENTER)  
 tblpersons.column("email", width=100, anchor=CENTER)  
 tblpersons.column("dob", width=100, anchor=CENTER)  
 tblpersons.column("gender", width=100, anchor=CENTER)  
 tblpersons.column("password", width=100, anchor=CENTER)  
  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("name", text='Name', anchor=CENTER)  
 tblpersons.heading("address", text='Address', anchor=CENTER)  
 tblpersons.heading("mobile", text='Mobile', anchor=CENTER)  
 tblpersons.heading("email", text='Email', anchor=CENTER)  
 tblpersons.heading("dob", text='DOB', anchor=CENTER)  
 tblpersons.heading("gender", text='Gender', anchor=CENTER)  
 tblpersons.heading("password", text='Password', anchor=CENTER)  
  
  
 tblpersons.insert(parent='', index='end',  
 values=(result[1], result[2], result[3], result[4], result[5], result[6], result[7]))  
  
 tblpersons.pack()  
  
 def selection(cid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[0]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 def select\_record(e):  
 # Clear entry boxes  
 nametxt.delete(0, END)  
 addresstxt.delete(0, END)  
 mobiletxt.delete(0, END)  
 emailtxt.delete(0, END)  
 date\_txt.delete(0, END)  
 gendertxt.delete(0, END)  
 passwordtxt.delete(0, END)  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 nametxt.insert(0, values[0])  
 addresstxt.insert(0, values[1])  
 mobiletxt.insert(0, values[2])  
 emailtxt.insert(0, values[3])  
 date\_txt.insert(0, values[4])  
 gendertxt.insert(0, values[5])  
 passwordtxt.insert(0, values[6])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
  
 def deleteAccount():  
 # cd = []  
 # for c in selection(cid=selection):  
 # cd.append(c[0])  
 # lol\_string = ''.join(map(str, cd))  
 # cid = lol\_string # Read value from TextBox  
  
 # selected = tblpersons.focus()  
 # temp = tblpersons.item(selected, 'values')  
 Cid = self.cid  
  
  
 # Send values to search (Middleware)  
 result = customerdelete(Cid)  
 # Display Message  
  
 if result == True:  
 # lblMessage['text'] = "Record Delete"  
 messagebox.showinfo("Done", "Delete")  
  
 else:  
 # lblMessage['text'] = "Unsuccessfull"  
 messagebox.showerror("Error", "Unsuccessfull")  
  
  
 pass  
  
 #update customer data as per input  
 def saveUser():  
 result = customersearch1(CID)  
 # Read values from table  
 selected = tblpersons.focus()  
 cid = result[0]  
  
 #update table  
 tblpersons.item(selected, text="", values=(  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), date\_txt.get(), gendertxt.get(),  
 passwordtxt.get()))  
  
 #display message  
 if result == True:  
 messagebox.showerror("Error", "update unsuccessfull")  
 else:  
 messagebox.showinfo("Done", "Update successfully")  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 # Send values to save (Middleware)  
 sql = "UPDATE customer set name=%s, address=%s, mobile=%s, email=%s, dob=%s, gender=%s, password=%s WHERE cid=%s"  
 values = (  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), date\_txt.get(), gendertxt.get(),  
 passwordtxt.get(), cid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del values, sql  
 return result  
  
  
  
  
 delete\_btn = Button(self.booking\_frame, text="Delete", command=lambda :[deleteAccount(), self.btn\_clicked()])  
 update\_btn = Button(self.booking\_frame, text="Update", command=saveUser)  
  
 delete\_btn.place(x=315, y=400)  
 update\_btn.place(x=260, y=400)  
  
 #add label  
 namelbl = Label(self.booking\_frame, text="Name")  
 addresslbl = Label(self.booking\_frame, text="Address")  
 mobilelbl = Label(self.booking\_frame, text="Mobile")  
 emaillbl = Label(self.booking\_frame, text="Email")  
 doblbl = Label(self.booking\_frame, text="DOB")  
 genderlbl = Label(self.booking\_frame, text="Gender")  
 passwordlbl = Label(self.booking\_frame, text="Password")  
  
 # lblMessage = Label(self.booking\_frame, text='', bg="gray")  
  
 #add text box  
 nametxt = Entry(self.booking\_frame)  
 addresstxt = Entry(self.booking\_frame)  
 mobiletxt = Entry(self.booking\_frame)  
 emailtxt = Entry(self.booking\_frame)  
 date\_txt = DateEntry(self.booking\_frame, width=12, background='darkblue',  
 foreground='white', borderwidth=2)  
  
 date\_txt.bind("<<DateEntrySelected>>")  
 gendertxt = Entry(self.booking\_frame)  
 passwordtxt = Entry(self.booking\_frame)  
  
 namelbl.place(x=40, y=270)  
 addresslbl.place(x=40, y=300)  
 mobilelbl.place(x=250, y=270)  
 emaillbl.place(x=250, y=300)  
 doblbl.place(x=460, y=270)  
 genderlbl.place(x=460, y=300)  
 passwordlbl.place(x=40, y=330)  
  
 lblMessage.place(x=360, y=400)  
  
 nametxt.place(x=100, y=270)  
 addresstxt.place(x=100, y=300)  
 mobiletxt.place(x=310, y=270)  
 emailtxt.place(x=310, y=300)  
 date\_txt.place(x=520, y=270)  
 gendertxt.place(x=520, y=300)  
 passwordtxt.place(x=100, y=330)  
  
 pass

Booking\_btn function is used to display a table of pending bookings for the logged in customer. It first connects to a MySQL database to retrieve all records from the 'booking' table where the customer ID (cid) is the same as the logged in customer's ID and the status is "pending". If no records are found, it displays a message indicating that no records were found. If records are found, it creates a ttk Treeview table to display the records and allows the customer to select a specific booking to view more details or cancel. The function also includes a select\_record function which is used to display more details of the selected booking in entry widgets. This code defines a class called "CUSTOMER" which creates a customer dashboard using the tkinter library. The class takes in a master frame and a customer ID as arguments in the constructor. The class creates several frames, such as "side\_frame", "main\_frame" and "booking\_frame" which are used to display different widgets. The class also defines several methods, such as "Booking\_btn", "tripinfo", "payment", etc. which are called when specific buttons are clicked. The "Booking\_btn" method, for example, retrieves booking records from a MySQL database for the customer with the given ID and displays them in a table. The "tripinfo" method retrieves the trip information for the customer and displays them in a table and also has a cancel button which allows the customer to cancel their trip. The "payment" method is used to make the payment for the trip. The function Account is used to display the account information of the customer, such as name, address, mobile, email, date of birth, gender, and password. The function first calls a delete\_frame method to remove any previously displayed widgets on the frame. It then calls a customersearch1 function to retrieve the customer's account information from the database, passing in the customer's ID as an argument. If the function returns no records, a message "Record not found" is displayed on the frame. If records are found, a table is created using the ttk Treeview widget to display the customer's account information. The table has columns for name, address, mobile, email, date of birth, gender, and password. The table also includes a selection feature, where the customer can select a record and the information will be displayed in entry boxes for editing. There is also a delete button to delete the account.

Admin\_Dashboard.py

from tkinter import \*  
  
from Assignment.Classes.Driver.driver\_class import driverregister  
from Assignment.Classes.Customer.Register import register  
from Assignment.Manager.booking.booking\_database import \*  
from Assignment.Manager.driver.driver\_database import \*  
from Assignment.Manager.customer.database import \*  
from tkinter import ttk  
import messagebox  
from tkcalendar import DateEntry  
  
  
# create a class for admin dashboard  
# every frame will work inside this class where tkinter is pass while calling the class ADMIN  
class ADMIN():  
 def \_\_init\_\_(self, master):  
 super().\_\_init\_\_()  
 # passing the master to create a framework  
 self.Window\_main = master  
  
 screen\_width = self.Window\_main.winfo\_screenwidth()  
 screen\_height = self.Window\_main.winfo\_screenheight()  
  
 #creating seperate frames for an individual widgit purpose  
 self.side\_frame = Frame(self.Window\_main, bg="#A7DBD8", height=screen\_height, width=screen\_width)  
 self.side\_frame.pack(side=LEFT, fill=BOTH)  
 self.side\_frame.pack\_propagate(False)  
  
 self.main\_frame = Frame(self.side\_frame, bg="#F38630", height=510, width=990)  
 self.main\_frame.pack(side=RIGHT, fill=BOTH, anchor=SE)  
  
 self.booking\_frame = Frame(self.main\_frame, bg="#E0E4CC", height=450, width=800)  
 self.booking\_frame.place(relx=0.5, rely=0.6, anchor=CENTER)  
  
 #main buttons which will help to call the function as per needed  
 dashboard\_btn = Button(self.side\_frame, text="Dashboard", font=("San Francisco", 20))  
 confirm\_booking\_btn = Button(self.side\_frame, text="Confirm Booking", command=self.Confirm\_booking\_btn)  
 tripinfo\_btn = Button(self.side\_frame, text="Trip Information", command=self.tripinfo)  
 add\_user\_btn = Button(self.side\_frame, text="Add User", command=self.Add\_account)  
 add\_Driver\_btn = Button(self.side\_frame, text="Add Driver", command=self.Add\_Driver)  
 logout\_btn = Button(self.side\_frame, text="Logout", command=self.btn\_clicked)  
  
 dashboard\_btn.place(x=2, y=50)  
 confirm\_booking\_btn.place(x=13, y=200)  
 tripinfo\_btn.place(x=13, y=250)  
 add\_user\_btn.place(x=13, y=300)  
 add\_Driver\_btn.place(x=13, y=350)  
 logout\_btn.place(x=13, y=400)  
  
 # function to delete the widgit while switching between different function operation  
 def delete\_frame(self):  
 for frame in self.booking\_frame.winfo\_children():  
 frame.destroy()  
  
 # function to redirect in login page  
 def btn\_clicked(self):  
 messagebox.askyesno("Confirm", "Do you really want to logout")  
 while True:  
 self.login()  
  
 def login(self):  
 self.Window\_main.destroy()  
 from Assignment.UI import UI  
 return UI  
 pass  
  
 #function to assign driver to the customer  
 def Confirm\_booking\_btn(self):  
 # removing widgit  
 self.delete\_frame()  
  
 # Send values to search (database) all the pending trips  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 cursor = conn.cursor()  
 sql = "SELECT \* FROM booking WHERE status='Pending'"  
 cursor.execute(sql)  
 records = cursor.fetchall()  
  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if records == None:  
 lblMessage['text'] = "Record not found"  
 else:  
 #frame for table  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=70, y=20)  
  
 #creating treeview table to show the customer information in tables to understand easily  
 tblpersons = ttk.Treeview(tableframe)  
 # Define Our Columns  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 # Format Our Columns  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 # Create Headings  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
  
 global count  
 count = 0  
  
 # for record in records:  
 for record in records:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
  
 tblpersons.pack()  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 #show record in txt,label and remove it before displaying  
 def select\_record(e):  
 global values  
 # Clear entry boxes  
 date\_lbl1.config(text="")  
 time\_lbl1.config(text="")  
 pick\_up\_lbl1.config(text="")  
 destination\_lbl1.config(text="")  
 driver\_txt.delete(0, 'end')  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 date\_lbl1.config(text=values[2])  
 time\_lbl1.config(text=values[3])  
 pick\_up\_lbl1.config(text=values[4])  
 destination\_lbl1.config(text=values[5])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 #function to update the record of driver for readjustment  
 def update\_driver():  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
  
 sql = "UPDATE driver SET status=%s WHERE did=%s"  
 update = "booked"  
 values = (update, driver\_txt.get())  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 #function to update the record of booked trip for readjustment  
 def update\_record():  
 global bid  
 # Grab the record number  
 selected = tblpersons.focus()  
 value = tblpersons.item(selected, 'values')  
 cid = value[0]  
  
 #get bid from selected table which is focus(while clicking)  
 bd = []  
 for b in selection(bid=selection):  
 bd.append(b[0])  
 lol\_string = ''.join(map(str, bd))  
 bid = lol\_string  
  
 # Recive value  
 date = value[2]  
 time = value[3]  
 pick\_up = value[4]  
 destination = value[5]  
 status = "active"  
  
 # Update record in table  
 tblpersons.item(selected, text="", values=(cid, bid,  
 date, time, pick\_up,  
 destination,  
 status))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 sql = "UPDATE booking SET did=%s, status='active' WHERE bid=%s"  
 values = (driver\_txt.get(), bid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 # Clear entry boxes  
 date\_txt.delete(0, END)  
 hrs\_txt.delete(0, END)  
 min\_txt.delete(0, END)  
 pick\_up\_txt.delete(0, END)  
 destination\_txt.delete(0, END)  
  
 #add label  
 date\_lbl = Label(self.booking\_frame, text="Date")  
 time\_lbl = Label(self.booking\_frame, text="Time")  
 pick\_up\_lbl = Label(self.booking\_frame, text="Pick Up")  
 destination\_lbl = Label(self.booking\_frame, text="Destination")  
 driver\_lbl = Label(self.booking\_frame, text="Driver")  
  
 book\_btn = Button(self.booking\_frame, text="Book", command=lambda: [update\_driver(), update\_record()])  
  
 date\_lbl.place(x=70, y=300)  
 time\_lbl.place(x=70, y=335)  
 pick\_up\_lbl.place(x=220, y=335)  
 destination\_lbl.place(x=220, y=300)  
 driver\_lbl.place(x=450, y=300)  
 lblMessage.place(x=250, y=400)  
  
 #add label to show the information while clicking on the date to assign driver  
 date\_lbl1 = Label(self.booking\_frame, text="")  
 time\_lbl1 = Label(self.booking\_frame, text="")  
 pick\_up\_lbl1 = Label(self.booking\_frame, text="")  
 destination\_lbl1 = Label(self.booking\_frame, text="")  
  
 #show available driver by checking their status for their next booking  
 driver = drivergetAllavailable()  
 drivers = []  
 for i in driver:  
 drivers.append(i[0])  
 driver\_txt = ttk.Combobox(self.booking\_frame)  
 driver\_txt['values'] = (drivers)  
  
 date\_lbl1.place(x=140, y=300)  
 time\_lbl1.place(x=140, y=335)  
 pick\_up\_lbl1.place(x=350, y=335)  
 destination\_lbl1.place(x=350, y=300)  
 driver\_txt.place(x=500, y=300)  
  
 book\_btn.place(x=315, y=380)  
  
  
 def tripinfo(self):  
 self.delete\_frame()  
  
 #search all the trip  
 result = bookinggetAll()  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=70, y=20)  
  
 # creating treeview table to show the customer information in tables to understand easily  
 tblpersons = ttk.Treeview(tableframe)  
  
 # Define Our Columns  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 # Format Our Columns  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 # Create Headings  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
 #display result  
 for dt in result:  
 tblpersons.insert(parent='', index='end', values=(dt[0], dt[1], dt[2], dt[3], dt[4], dt[5], dt[6]))  
  
 tblpersons.pack()  
  
 def select\_record(e):  
 global bid, did  
 # global values  
 # Clear entry boxes  
 date\_lbl1.config(text="")  
 time\_lbl1.config(text="")  
 pick\_up\_lbl1.config(text="")  
 destination\_lbl1.config(text="")  
  
 d\_name\_lbl1.config(text="")  
 d\_mobile\_lbl1.config(text="")  
 d\_age\_lbl1.config(text="")  
 d\_gender\_lbl1.config(text="")  
 d\_num\_plate\_lbl1.config(text="")  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 #get booking id of selected value  
 bid = values[1]  
 #search values in database regarding the booking id  
 record = bookingsearch1(bid)  
  
 #passing driver id retrived from database which was assigh to the customer by admin  
 did = record  
  
 #search driver detail in database using driver id  
 # will display the driver information through reciving driver id from booking database  
 # and passing it to the database table to get driver information  
 record1 = driversearch1(did)  
 d\_values = record1  
  
 #applying condition when cancel button will appear  
 if values[6] == "active":  
 cancel\_btn = Button(self.booking\_frame, text="Cancel",  
 command=lambda: [update\_driver1(), cancel\_booking()])  
 cancel\_btn.place(x=300, y=420)  
  
 if values[6] == "pending":  
 cancel\_btn = Button(self.booking\_frame, text="Cancel",  
 command=lambda: [cancel\_booking1()])  
 cancel\_btn.place(x=300, y=420)  
  
 # outpus to entry boxes  
 date\_lbl1.config(text=values[2])  
 time\_lbl1.config(text=values[3])  
 pick\_up\_lbl1.config(text=values[4])  
 destination\_lbl1.config(text=values[5])  
  
 d\_name\_lbl1.config(text=d\_values[1])  
 d\_mobile\_lbl1.config(text=d\_values[3])  
 d\_age\_lbl1.config(text=d\_values[5])  
 d\_gender\_lbl1.config(text=d\_values[6])  
 d\_num\_plate\_lbl1.config(text=d\_values[7])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 #cancel the trip having status active  
 def cancel\_booking():  
 bookingid = bid  
 record = bookingcancel(bookingid)  
 return record  
  
 #cancel the trip having status pending  
 def cancel\_booking1():  
 bookingid = bid  
 record = bookingcancel1(bookingid)  
 return record  
  
  
 def update\_driver1():  
 #get driver id which is marked as global variable  
 driverid = did  
  
 #also update the driver status while the trip is cancel  
 conn1 = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 sql = "UPDATE driver SET status='open' WHERE did=%s"  
 values = (driverid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = conn1  
 cursor = conn.cursor(buffered=True)  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 #trip heading  
 active\_lbl = Label(self.booking\_frame, text="Trip", font=("San Francisco", 15))  
 active\_lbl.place(x=340, y=260)  
  
 #add label for trip information  
 date\_lbl = Label(self.booking\_frame, text="Date")  
 time\_lbl = Label(self.booking\_frame, text="Time")  
 pick\_up\_lbl = Label(self.booking\_frame, text="Pick Up")  
 destination\_lbl = Label(self.booking\_frame, text="Destination")  
 driver\_lbl = Label(self.booking\_frame, text="Driver Information")  
  
 #add label for driver information to show to the customer who is their driver and driver related information  
 d\_name\_lbl = Label(self.booking\_frame, text="Name")  
 d\_mobile\_lbl = Label(self.booking\_frame, text="Mobile")  
 d\_age\_lbl = Label(self.booking\_frame, text="Age")  
 d\_gender\_lbl = Label(self.booking\_frame, text="Gender")  
 d\_num\_plate\_lbl = Label(self.booking\_frame, text="Number Plate")  
  
 date\_lbl.place(x=70, y=295)  
 time\_lbl.place(x=70, y=325)  
 pick\_up\_lbl.place(x=70, y=355)  
 destination\_lbl.place(x=70, y=385)  
  
 d\_name\_lbl.place(x=420, y=295)  
 d\_mobile\_lbl.place(x=420, y=325)  
 d\_age\_lbl.place(x=420, y=355)  
 d\_gender\_lbl.place(x=420, y=385)  
 d\_num\_plate\_lbl.place(x=420, y=415)  
  
 driver\_lbl.place(x=430, y=260)  
 lblMessage.place(x=250, y=400)  
  
 # add label for trip information, put data from table  
 date\_lbl1 = Label(self.booking\_frame, text="")  
 time\_lbl1 = Label(self.booking\_frame, text="")  
 pick\_up\_lbl1 = Label(self.booking\_frame, text="")  
 destination\_lbl1 = Label(self.booking\_frame, text="")  
  
 # add label for driver information to show to the customer who is their driver and driver related information  
 #will display the driver information through reciving driver id from booking database  
 #and passing it to the database table to get driver information  
 d\_name\_lbl1 = Label(self.booking\_frame, text="")  
 d\_mobile\_lbl1 = Label(self.booking\_frame, text="")  
 d\_age\_lbl1 = Label(self.booking\_frame, text="")  
 d\_gender\_lbl1 = Label(self.booking\_frame, text="")  
 d\_num\_plate\_lbl1 = Label(self.booking\_frame, text="")  
  
 date\_lbl1.place(x=140, y=295)  
 time\_lbl1.place(x=140, y=325)  
 pick\_up\_lbl1.place(x=140, y=355)  
 destination\_lbl1.place(x=140, y=385)  
  
 d\_name\_lbl1.place(x=520, y=295)  
 d\_mobile\_lbl1.place(x=520, y=325)  
 d\_age\_lbl1.place(x=520, y=355)  
 d\_gender\_lbl1.place(x=520, y=385)  
 d\_num\_plate\_lbl1.place(x=520, y=415)  
  
 pass  
  
  
 def Add\_account(self):  
  
 self.delete\_frame()  
  
 # get values  
 result = customergetAll()  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
  
 tableframe.place(x=70, y=20)  
  
 tblpersons = ttk.Treeview(tableframe)  
 tblpersons['column'] = ('id','name', 'address', 'mobile', 'email', 'dob', 'gender', 'password')  
  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("id", width=50, anchor=CENTER)  
 tblpersons.column("name", width=50, anchor=CENTER)  
 tblpersons.column("address", width=100, anchor=CENTER)  
 tblpersons.column("mobile", width=100, anchor=CENTER)  
 tblpersons.column("email", width=100, anchor=CENTER)  
 tblpersons.column("dob", width=100, anchor=CENTER)  
 tblpersons.column("gender", width=100, anchor=CENTER)  
 tblpersons.column("password", width=100, anchor=CENTER)  
  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("id", text='ID', anchor=CENTER)  
 tblpersons.heading("name", text='Name', anchor=CENTER)  
 tblpersons.heading("address", text='Address', anchor=CENTER)  
 tblpersons.heading("mobile", text='Mobile', anchor=CENTER)  
 tblpersons.heading("email", text='Email', anchor=CENTER)  
 tblpersons.heading("dob", text='DOB', anchor=CENTER)  
 tblpersons.heading("gender", text='Gender', anchor=CENTER)  
 tblpersons.heading("password", text='Password', anchor=CENTER)  
  
 global count  
 count = 0  
  
 # for record in records:  
 for record in result:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],  
 record[1], record[2], record[3], record[4], record[5], record[6], record[7]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],  
 record[1], record[2], record[3], record[4], record[5], record[6], record[7]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
  
 tblpersons.pack()  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 def select\_record(e):  
 # Clear entry boxes  
 nametxt.delete(0, END)  
 addresstxt.delete(0, END)  
 mobiletxt.delete(0, END)  
 emailtxt.delete(0, END)  
 date\_txt.delete(0, END)  
 gendertxt.delete(0, END)  
 passwordtxt.delete(0, END)  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 nametxt.insert(0, values[1])  
 addresstxt.insert(0, values[2])  
 mobiletxt.insert(0, values[3])  
 emailtxt.insert(0, values[4])  
 date\_txt.insert(0, values[5])  
 gendertxt.insert(0, values[6])  
 passwordtxt.insert(0, values[7])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 def deletecustomer():  
 # get value while click in the table data  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
  
 # getting specific data from table  
 cid = temp[0]  
  
 # Send values to search (Middleware)  
 result = customerdelete(cid)  
 # Display Message  
  
 if result == True:  
 lblMessage['text'] = "Record Delete"  
 else:  
 lblMessage['text'] = "Unsuccessfull"  
  
 pass  
  
 def saveUser():  
 #get value while click in the table data  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
  
 #getting specific data from table  
 Cid = temp[0]  
  
 #update table  
 tblpersons.item(selected, text="", values=(Cid,  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), date\_txt.get(), gendertxt.get(),  
 passwordtxt.get()))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 # Send values to save (Middleware)  
 sql = "UPDATE customer set name=%s, address=%s, mobile=%s, email=%s, dob=%s, gender=%s, password=%s WHERE cid=%s"  
 values = (nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), date\_txt.get(), gendertxt.get(), passwordtxt.get(), Cid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del values, sql  
 return result  
  
 def adduser():  
 c1 = register("", nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), date\_txt.get(), gendertxt.get(), passwordtxt.get())  
  
 # Send values to save (database)  
 result = customerinsert(c1)  
 # Display Message  
 if result['status'] == True:  
 # self.lblMessage['text']="Save Record"  
 messagebox.showinfo("Done", "Account created, Now you can log in using mobile number as password")  
  
 #call above function while clicking  
 delete\_btn = Button(self.booking\_frame, text="Delete", command=deletecustomer)  
 update\_btn = Button(self.booking\_frame, text="Update", command=saveUser)  
 add\_btn = Button(self.booking\_frame, text="Add", command=adduser)  
  
 delete\_btn.place(x=315, y=400)  
 update\_btn.place(x=260, y=400)  
 add\_btn.place(x=370, y=400)  
  
  
 #add label  
 namelbl = Label(self.booking\_frame, text="Name")  
 addresslbl = Label(self.booking\_frame, text="Address")  
 mobilelbl = Label(self.booking\_frame, text="Mobile")  
 emaillbl = Label(self.booking\_frame, text="Email")  
 doblbl = Label(self.booking\_frame, text="DOB")  
 genderlbl = Label(self.booking\_frame, text="Gender")  
 passwordlbl = Label(self.booking\_frame, text="Password")  
  
 # lblMessage = Label(self.booking\_frame, text='', bg="gray")  
  
 #add text box  
 nametxt = Entry(self.booking\_frame)  
 addresstxt = Entry(self.booking\_frame)  
 mobiletxt = Entry(self.booking\_frame)  
 emailtxt = Entry(self.booking\_frame)  
 date\_txt = DateEntry(self.booking\_frame, width=12, background='darkblue',  
 foreground='white', borderwidth=2)  
  
 date\_txt.bind("<<DateEntrySelected>>")  
 gendertxt = Entry(self.booking\_frame)  
 passwordtxt = Entry(self.booking\_frame)  
  
  
 namelbl.place(x=40, y=270)  
 addresslbl.place(x=40, y=300)  
 mobilelbl.place(x=250, y=270)  
 emaillbl.place(x=250, y=300)  
 doblbl.place(x=460, y=270)  
 genderlbl.place(x=460, y=300)  
 passwordlbl.place(x=40, y=330)  
  
 lblMessage.place(x=360, y=400)  
  
 nametxt.place(x=100, y=270)  
 addresstxt.place(x=100, y=300)  
 mobiletxt.place(x=310, y=270)  
 emailtxt.place(x=310, y=300)  
 date\_txt.place(x=520, y=270)  
 gendertxt.place(x=520, y=300)  
 passwordtxt.place(x=100, y=330)  
  
 pass  
  
 def Add\_Driver(self):  
 self.delete\_frame()  
  
 # Read values from database  
 result = drivergetAll()  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
  
 tableframe.place(x=25, y=20)  
  
 tblpersons = ttk.Treeview(tableframe)  
 tblpersons['column'] = ('id','name', 'address', 'mobile', 'email', 'dob', 'gender', 'numberplate', 'password')  
  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("id", width=50, anchor=CENTER)  
 tblpersons.column("name", width=50, anchor=CENTER)  
 tblpersons.column("address", width=100, anchor=CENTER)  
 tblpersons.column("mobile", width=100, anchor=CENTER)  
 tblpersons.column("email", width=100, anchor=CENTER)  
 tblpersons.column("dob", width=100, anchor=CENTER)  
 tblpersons.column("gender", width=100, anchor=CENTER)  
 tblpersons.column("numberplate", width=100, anchor=CENTER)  
 tblpersons.column("password", width=100, anchor=CENTER)  
  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("id", text='ID', anchor=CENTER)  
 tblpersons.heading("name", text='Name', anchor=CENTER)  
 tblpersons.heading("address", text='Address', anchor=CENTER)  
 tblpersons.heading("mobile", text='Mobile', anchor=CENTER)  
 tblpersons.heading("email", text='Email', anchor=CENTER)  
 tblpersons.heading("dob", text='DOB', anchor=CENTER)  
 tblpersons.heading("gender", text='Gender', anchor=CENTER)  
 tblpersons.heading("numberplate", text='Num Plate', anchor=CENTER)  
 tblpersons.heading("password", text='Password', anchor=CENTER)  
  
 global count  
 count = 0  
  
 # for record in records:  
 for record in result:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],  
 record[1], record[2], record[3], record[4], record[5], record[6], record[7], record[8]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],  
 record[1], record[2], record[3], record[4], record[5], record[6], record[7], record[8]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
  
 tblpersons.pack()  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 def select\_record(e):  
 # Clear entry boxes  
 nametxt.delete(0, END)  
 addresstxt.delete(0, END)  
 mobiletxt.delete(0, END)  
 emailtxt.delete(0, END)  
 age\_txt.delete(0, END)  
 gendertxt.delete(0, END)  
 numplatetxt.delete(0, END)  
 passwordtxt.delete(0, END)  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 nametxt.insert(0, values[1])  
 addresstxt.insert(0, values[2])  
 mobiletxt.insert(0, values[3])  
 emailtxt.insert(0, values[4])  
 age\_txt.insert(0, values[5])  
 gendertxt.insert(0, values[6])  
 numplatetxt.insert(0, values[7])  
 passwordtxt.insert(0, values[8])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 def selection(bid):  
  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 def deletedriver():  
 # get value while click in the table data  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
  
 # getting specific data from table  
 did = temp[0]  
  
 # Send values to search (Middleware)  
 result = deleteDriver(did)  
 # Display Message  
  
 if result == True:  
 lblMessage['text'] = "Record Delete"  
 else:  
 lblMessage['text'] = "Unsuccessfull"  
  
 pass  
  
 def saveDriver():  
  
 # Read values from table  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
  
 # getting specific data from table  
 did = temp[0]  
  
 #update table  
 tblpersons.item(selected, text="", values=(did,  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), age\_txt.get(), gendertxt.get(),  
 passwordtxt.get()))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 # Send values to save (Middleware)  
 sql = "UPDATE driver set name=%s, address=%s, mobile=%s, email=%s, age=%s, gender=%s, password=%s WHERE did=%s"  
 values = (  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), age\_txt.get(), gendertxt.get(),  
 passwordtxt.get(), did)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del values, sql  
 return result  
  
 def addDriver():  
 c1 = driverregister("", nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), age\_txt.get(), gendertxt.get(), numplatetxt.get(), passwordtxt.get())  
  
 # Send values to save (database)  
 result = driverinsert(c1)  
 # Display Message  
 if result['status'] == True:  
 # self.lblMessage['text']="Save Record"  
 messagebox.showinfo("Done", "Account created, Now you can log in using mobile number as password")  
  
 delete\_btn = Button(self.booking\_frame, text="Delete", command=deletedriver)  
 update\_btn = Button(self.booking\_frame, text="Update", command=saveDriver)  
 add\_btn = Button(self.booking\_frame, text="Add", command=addDriver)  
  
  
 delete\_btn.place(x=315, y=400)  
 update\_btn.place(x=260, y=400)  
 add\_btn.place(x=370, y=400)  
  
  
 #add label  
 namelbl = Label(self.booking\_frame, text="Name")  
 addresslbl = Label(self.booking\_frame, text="Address")  
 mobilelbl = Label(self.booking\_frame, text="Mobile")  
 emaillbl = Label(self.booking\_frame, text="Email")  
 doblbl = Label(self.booking\_frame, text="DOB")  
 genderlbl = Label(self.booking\_frame, text="Gender")  
 numplatelbl = Label(self.booking\_frame, text="Num Plate")  
 passwordlbl = Label(self.booking\_frame, text="Password")  
  
 lblMessage = Label(self.booking\_frame, text='', bg="gray")  
  
 #add text box  
 nametxt = Entry(self.booking\_frame)  
 addresstxt = Entry(self.booking\_frame)  
 mobiletxt = Entry(self.booking\_frame)  
 emailtxt = Entry(self.booking\_frame)  
 age\_txt = Entry(self.booking\_frame)  
 gendertxt = Entry(self.booking\_frame)  
 numplatetxt = Entry(self.booking\_frame)  
 passwordtxt = Entry(self.booking\_frame)  
  
  
 namelbl.place(x=40, y=270)  
 addresslbl.place(x=40, y=300)  
 mobilelbl.place(x=250, y=270)  
 emaillbl.place(x=250, y=300)  
 doblbl.place(x=460, y=270)  
 genderlbl.place(x=460, y=300)  
 numplatelbl.place(x=250, y=330)  
 passwordlbl.place(x=40, y=330)  
  
 lblMessage.place(x=360, y=400)  
  
 nametxt.place(x=100, y=270)  
 addresstxt.place(x=100, y=300)  
 mobiletxt.place(x=315, y=270)  
 emailtxt.place(x=315, y=300)  
 age\_txt.place(x=520, y=270)  
 gendertxt.place(x=520, y=300)  
 numplatetxt.place(x=315, y=330)  
 passwordtxt.place(x=100, y=330)  
  
 pass

The ADMIN class is a class that defines the main window for the admin user of the system. It has several methods, including:

init: which sets up the layout of the main window, including frames for different widgets and buttons for different actions (e.g. "Dashboard", "Confirm Booking", "Trip Information", "Add User", "Add Driver", "Logout")

delete\_frame: which deletes all widgets in a specific frame

btn\_clicked: which is called when the "Logout" button is clicked, and asks the user if they want to logout, and redirects them to the login page

login: which redirects the user to the login page

Confirm\_booking\_btn: which assigns a driver to a customer based on a pending booking

tripinfo: which displays all the trip information for a customer

Add\_account: which allows the admin to add a new customer account

Add\_Driver: which allows the admin to add a new driver account.

This code is part of a class called ADMIN which handles the functionality of an administrator in a transportation booking system. The tripinfo() function retrieves all the trip information from the database and displays it in a Treeview table. The select\_record() function is bound to the table so that when an item in the table is selected, the function is called, and it retrieves the selected record's values and performs some additional actions such as displaying the driver's information and creating a cancel button if the trip status is "active". The update\_driver1() and cancel\_booking() functions are called when the cancel button is clicked. This code is for the "Add\_account" function in a GUI application. The function starts by deleting the current frame, then it retrieves all the customer records from a database and stores it in the "result" variable. Then, it creates a table view using a ttk Treeview widget and displays all the customer records in the table. The function also includes features such as selecting a record from the table, displaying the selected record's details in the entry boxes, and updating the record. The function also uses global variables such as 'count' and tags such as 'evenrow' and 'oddrow' to format the table view. This code appears to be a function for a GUI application that displays a table of driver records from a database. The function is called "Add\_Driver()" and it starts by deleting the current frame, retrieves all driver records from the database, and then creates a new frame to display the records in a table format using the ttk.Treeview widget. The table has columns for the driver's ID, name, address, mobile, email, date of birth, gender, number plate, and password. The table is populated with the driver records retrieved from the database and even rows are given a different color than odd rows. The function also has functionality to select a specific record, which will then clear any existing values in certain entry fields and insert the values of the selected record into those fields.

Driver\_dashboard.py

from tkinter import \*  
from Assignment.Manager.booking.booking\_database import \*  
from Assignment.Manager.customer.database import \*  
from Assignment.Manager.driver.driver\_database import \*  
from tkinter import ttk  
  
  
  
# create a class for driver dashboard  
# every frame will work inside this class where tkinter is pass while calling the class DRIVER\_DASHBOARD  
class DRIVER\_DASHBOARD():  
 def \_\_init\_\_(self, master, did):  
 super().\_\_init\_\_()  
  
 # passing the master to create a framework  
 self.Window\_main = master  
  
 #passing the cid receive during login of the driver  
 self.did = did  
  
 screen\_width = self.Window\_main.winfo\_screenwidth()  
 screen\_height = self.Window\_main.winfo\_screenheight()  
  
 # creating seperate frames for an individual widgit purpose  
 self.side\_frame = Frame(self.Window\_main, bg="#A7DBD8", height=screen\_height, width=screen\_width)  
 self.side\_frame.pack(side=LEFT, fill=BOTH)  
 self.side\_frame.pack\_propagate(False)  
  
 self.main\_frame = Frame(self.side\_frame, bg="#F38630", height=510, width=990)  
 self.main\_frame.pack(side=RIGHT, fill=BOTH, anchor=SE)  
  
 self.booking\_frame = Frame(self.main\_frame, bg="#E0E4CC", height=450, width=800)  
 self.booking\_frame.place(relx=0.5, rely=0.6, anchor=CENTER)  
  
 #main buttons which will help to call the function as per needed  
 dashboard\_btn = Button(self.side\_frame, text="Dashboard", font=("San Francisco", 20))  
 booking\_btn = Button(self.side\_frame, text="Customer", command=self.View\_Booking\_btn)  
 tripinfo\_btn = Button(self.side\_frame, text="Trip Information", command=self.tripinfo)  
 delete\_profile\_btn = Button(self.side\_frame, text="Profile", command=self.Account)  
 logout\_btn = Button(self.side\_frame, text="Logout", command=self.btn\_clicked)  
  
 dashboard\_btn.place(x=2, y=50)  
 booking\_btn.place(x=13, y=200)  
 tripinfo\_btn.place(x=13, y=250)  
 delete\_profile\_btn.place(x=13, y=300)  
 logout\_btn.place(x=13, y=350)  
  
 # function to redirect in login page  
 def btn\_clicked(self):  
 while True:  
 self.login()  
  
 def login(self):  
 self.Window\_main.destroy()  
 from Assignment.UI import UI  
 return UI  
 pass  
  
 # function to delete the widgit while switching between different function operation  
 def delete\_frame(self):  
 for frame in self.booking\_frame.winfo\_children():  
 frame.destroy()  
  
 # viewbooking button to see a next trip  
 def View\_Booking\_btn(self):  
 self.delete\_frame()  
  
 # Send values to search (Middleware)  
 did = self.did  
  
 #searching the active data from the database  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 cursor = conn.cursor()  
 sql = "SELECT \* FROM booking WHERE did=%s AND status=%s"  
 DID = did  
 status = "active"  
 values = (DID, status)  
 cursor.execute(sql, values)  
 records = cursor.fetchall()  
  
 #searching the ongoingg data from the database  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 cursor = conn.cursor()  
 sql = "SELECT \* FROM booking WHERE did=%s AND status=%s"  
 DID = did  
 status = "ongoing"  
 values = (DID, status)  
 cursor.execute(sql, values)  
 records1 = cursor.fetchall()  
  
  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if records == None:  
 lblMessage['text'] = "Record not found"  
 else:  
 #frame for table  
 tableframe = Frame(self.booking\_frame)  
 tableframe.place(x=70, y=20)  
  
 #creating treeview table to show the customer information in tables to understand easily  
 tblpersons = ttk.Treeview(tableframe)  
 # Define Our Columns  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 # Format Our Columns  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 # Create Headings  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
  
 global count  
 count = 0  
  
 # for record in records:  
 #to display active record at a single time  
 for record in records:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
 #to display ongoing display at a single time  
 for record in records1:  
 if count % 2 == 0:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('evenrow',))  
 else:  
 tblpersons.insert(parent='', index='end', iid=count, text='',  
 values=(record[0],record[1], record[2], record[3], record[4], record[5], record[6]),  
 tags=('oddrow',))  
 # increment counter  
 count += 1  
  
 tblpersons.pack()  
  
 def selection(bid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[1]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 #show record in txt,label and remove it before displaying  
 def select\_record(e):  
 # global values  
 global bid, did  
 # Clear entry boxes  
 date\_lbl1.config(text="")  
 time\_lbl1.config(text="")  
 pick\_up\_lbl1.config(text="")  
 destination\_lbl1.config(text="")  
  
  
 d\_name\_lbl1.config(text="")  
 d\_mobile\_lbl1.config(text="")  
 d\_gender\_lbl1.config(text="")  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 #get booking id of selected value  
 bid = values[0]  
 BID = [bid]  
  
 #search values in database regarding the booking id  
 record1 = customersearch1(BID)  
  
 #passing driver id retrived from database which was assigh to the customer by admin  
 d\_values = record1  
  
 # outpus to entry boxes  
 date\_lbl1.config(text=values[2])  
 time\_lbl1.config(text=values[3])  
 pick\_up\_lbl1.config(text=values[4])  
 destination\_lbl1.config(text=values[5])  
  
 d\_name\_lbl1.config(text=d\_values[1])  
 d\_mobile\_lbl1.config(text=d\_values[3])  
 d\_gender\_lbl1.config(text=d\_values[6])  
  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
  
 def startbooking():  
 # Grab the record number  
 selected = tblpersons.focus()  
  
 cid = self.did  
 bd = []  
 status = "ongoing"  
 for b in selection(bid=selection):  
 bd.append(b[0])  
 lol\_string = ''.join(map(str, bd))  
 bid = lol\_string # Read value from TextBox  
  
 # Update record  
 # tblpersons.item(selected, text="", values=(cid, bid,  
 # date\_txt.get(), time, pick\_up\_txt.get(),  
 # destination\_txt.get(),  
 # status))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
  
 sql = "UPDATE booking SET status=%s WHERE bid=%s"  
 values = (status, bid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 def completebooking():  
 # Grab the record number  
 selected = tblpersons.focus()  
  
 cid = self.did  
 bd = []  
 status = "Complete"  
 for b in selection(bid=selection):  
 bd.append(b[0])  
 lol\_string = ''.join(map(str, bd))  
 bid = lol\_string # Read value from TextBox  
 print(bid)  
 # Update record  
 # tblpersons.item(selected, text="", values=(cid, bid,  
 # date\_txt.get(), time, pick\_up\_txt.get(),  
 # destination\_txt.get(),  
 # status))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='',  
 database='tbs')  
  
 sql = "UPDATE booking SET status=%s WHERE bid=%s"  
 # sql = "UPDATE`booking` SET `cid` = '%s', `date` = '%s', `time` = '%s', `pick\_up` = '%s', `destination` = '%s', `status` = '%s' WHERE bid = '%s'"  
 values = (status, bid)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del sql, values  
 return result  
  
 # Clear entry boxes  
 date\_txt.delete(0, END)  
 hrs\_txt.delete(0, END)  
 min\_txt.delete(0, END)  
 pick\_up\_txt.delete(0, END)  
 destination\_txt.delete(0, END)  
  
 # trip heading  
 active\_lbl = Label(self.booking\_frame, text="Trip", font=("San Francisco", 15))  
 active\_lbl.place(x=340, y=260)  
  
 #add label for trip information  
 date\_lbl = Label(self.booking\_frame, text="Date")  
 time\_lbl = Label(self.booking\_frame, text="Time")  
 pick\_up\_lbl = Label(self.booking\_frame, text="Pick Up")  
 destination\_lbl = Label(self.booking\_frame, text="Destination")  
 driver\_lbl = Label(self.booking\_frame, text="Driver Information")  
  
 #add label for customer information  
 d\_name\_lbl = Label(self.booking\_frame, text="Name")  
 d\_mobile\_lbl = Label(self.booking\_frame, text="Mobile")  
 d\_gender\_lbl = Label(self.booking\_frame, text="Gender")  
  
 lblMessage = Label(self.booking\_frame, text='', bg="gray")  
  
 date\_lbl.place(x=70, y=295)  
 time\_lbl.place(x=70, y=325)  
 pick\_up\_lbl.place(x=70, y=355)  
 destination\_lbl.place(x=70, y=385)  
  
 d\_name\_lbl.place(x=420, y=295)  
 d\_mobile\_lbl.place(x=420, y=325)  
 d\_gender\_lbl.place(x=420, y=355)  
  
 driver\_lbl.place(x=430, y=260)  
 lblMessage.place(x=250, y=400)  
  
 # add label for trip information, put data from table  
 date\_lbl1 = Label(self.booking\_frame, text="")  
 time\_lbl1 = Label(self.booking\_frame, text="")  
 pick\_up\_lbl1 = Label(self.booking\_frame, text="")  
 destination\_lbl1 = Label(self.booking\_frame, text="")  
  
 # add label for customer information to show to the driver who is their customer and customer related information  
 # will display the customer information through reciving customer id from booking database  
 # and passing it to the database table to get customer information  
 d\_name\_lbl1 = Label(self.booking\_frame, text="")  
 d\_mobile\_lbl1 = Label(self.booking\_frame, text="")  
 d\_gender\_lbl1 = Label(self.booking\_frame, text="")  
  
 date\_lbl1.place(x=140, y=295)  
 time\_lbl1.place(x=140, y=325)  
 pick\_up\_lbl1.place(x=140, y=355)  
 destination\_lbl1.place(x=140, y=385)  
  
 d\_name\_lbl1.place(x=520, y=295)  
 d\_mobile\_lbl1.place(x=520, y=325)  
 d\_gender\_lbl1.place(x=520, y=355)  
  
 start\_btn = Button(self.booking\_frame, text="Start",  
 command=lambda: [startbooking()])  
 start\_btn.place(x=300, y=420)  
 complete\_btn = Button(self.booking\_frame, text="Complete",  
 command=lambda: [completebooking()])  
 complete\_btn.place(x=360, y=420)  
  
  
  
 def tripinfo(self):  
 self.delete\_frame()  
 did = [self.did]  
 # Read values from Window  
  
 # Send values to search (database)  
 result = bookinggetAll2(did)  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
  
 tableframe.place(x=70, y=20)  
  
 tblpersons = ttk.Treeview(tableframe)  
 tblpersons['column'] = ('cid', 'bid', 'date', 'time', 'pickup', 'destination', 'status')  
  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("cid", width=50, anchor=CENTER)  
 tblpersons.column("bid", width=100, anchor=CENTER)  
 tblpersons.column("date", width=100, anchor=CENTER)  
 tblpersons.column("time", width=100, anchor=CENTER)  
 tblpersons.column("pickup", width=100, anchor=CENTER)  
 tblpersons.column("destination", width=100, anchor=CENTER)  
 tblpersons.column("status", width=100, anchor=CENTER)  
  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("cid", text='CID', anchor=CENTER)  
 tblpersons.heading("bid", text='BID', anchor=CENTER)  
 tblpersons.heading("date", text='DATE', anchor=CENTER)  
 tblpersons.heading("time", text='TIME', anchor=CENTER)  
 tblpersons.heading("pickup", text='PICK UP', anchor=CENTER)  
 tblpersons.heading("destination", text='DESTINATION', anchor=CENTER)  
 tblpersons.heading("status", text='STATUS', anchor=CENTER)  
  
 for dt in result:  
 tblpersons.insert(parent='', index='end', values=(dt[0], dt[1], dt[2], dt[3], dt[4], dt[5], dt[6]))  
  
  
 tblpersons.pack()  
  
 pass  
  
 pass  
  
  
 def Account(self):  
  
 self.delete\_frame()  
  
 # Read values from Window  
 DID = [self.did]  
  
 # Send values to search (database)  
 result = driversearch1(DID)  
  
 # Display Message  
 lblMessage = Label(self.booking\_frame, text='')  
 lblMessage.place(x=350, y=180)  
  
 if result == None:  
 lblMessage['text'] = "Record not found"  
 else:  
  
 tableframe = Frame(self.booking\_frame)  
  
 tableframe.place(x=15, y=20)  
  
 tblpersons = ttk.Treeview(tableframe)  
 tblpersons['column'] = ('name', 'address', 'mobile', 'email', 'age', 'gender', 'num\_plate', 'password')  
  
 tblpersons.column("#0", width=0, stretch=NO)  
 tblpersons.column("name", width=100, anchor=CENTER)  
 tblpersons.column("address", width=100, anchor=CENTER)  
 tblpersons.column("mobile", width=100, anchor=CENTER)  
 tblpersons.column("email", width=120, anchor=CENTER)  
 tblpersons.column("age", width=50, anchor=CENTER)  
 tblpersons.column("gender", width=100, anchor=CENTER)  
 tblpersons.column("num\_plate", width=100, anchor=CENTER)  
 tblpersons.column("password", width=100, anchor=CENTER)  
  
 tblpersons.heading("#0", text='', anchor=CENTER)  
 tblpersons.heading("name", text='Name', anchor=CENTER)  
 tblpersons.heading("address", text='Address', anchor=CENTER)  
 tblpersons.heading("mobile", text='Mobile', anchor=CENTER)  
 tblpersons.heading("email", text='Email', anchor=CENTER)  
 tblpersons.heading("age", text='AGE', anchor=CENTER)  
 tblpersons.heading("gender", text='Gender', anchor=CENTER)  
 tblpersons.heading("num\_plate", text='Num Plate', anchor=CENTER)  
 tblpersons.heading("password", text='Password', anchor=CENTER)  
  
  
 tblpersons.insert(parent='', index='end',  
 values=(result[1], result[2], result[3], result[4], result[5], result[6], result[7], result[8]))  
  
 tblpersons.pack()  
  
 def selection(cid):  
 selected = tblpersons.focus()  
 temp = tblpersons.item(selected, 'values')  
 return temp[0]  
  
 tblpersons.bind('<ButtonRelease-1>', selection)  
  
 def select\_record(e):  
 # Clear entry boxes  
 nametxt.delete(0, END)  
 addresstxt.delete(0, END)  
 mobiletxt.delete(0, END)  
 emailtxt.delete(0, END)  
 age\_txt.delete(0, END)  
 gendertxt.delete(0, END)  
 num\_platetxt.delete(0, END)  
 passwordtxt.delete(0, END)  
  
 # Grab record Number  
 selected = tblpersons.focus()  
 # Grab record values  
 values = tblpersons.item(selected, 'values')  
  
 # outpus to entry boxes  
 nametxt.insert(0, values[0])  
 addresstxt.insert(0, values[1])  
 mobiletxt.insert(0, values[2])  
 emailtxt.insert(0, values[3])  
 age\_txt.insert(0, values[4])  
 gendertxt.insert(0, values[5])  
 num\_platetxt.insert(0, values[6])  
 passwordtxt.insert(0, values[7])  
  
 tblpersons.bind('<ButtonRelease-1>', select\_record)  
  
 def deleteAccount():  
 cd = []  
 for c in selection(cid=selection):  
 cd.append(c[0])  
 lol\_string = ''.join(map(str, cd))  
 cid = lol\_string # Read value from TextBox  
 print(cid)  
  
 # Send values to search (Middleware)  
 result = customerdelete(cid)  
 # Display Message  
  
 if result == True:  
 lblMessage['text'] = "Record Delete"  
 else:  
 lblMessage['text'] = "Unsuccessfull"  
  
 pass  
  
 def saveUser():  
 result = driversearch1(DID)  
 # Read values from Window  
 selected = tblpersons.focus()  
  
 did = result[0]  
  
 #update table  
 tblpersons.item(selected, text="", values=(  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), age\_txt.get(), gendertxt.get(),  
 passwordtxt.get()))  
  
 # Update the database  
 # Create a database or connect to one that exists  
 conn = mysql.connector.connect(host='localhost', port=3306, user='root', password='', database='tbs')  
 # Send values to save (database)  
 sql = "UPDATE driver set name=%s, address=%s, mobile=%s, email=%s, age=%s, gender=%s, number\_plate=%s, password=%s WHERE did=%s"  
 values = (  
 nametxt.get(), addresstxt.get(), mobiletxt.get(), emailtxt.get(), age\_txt.get(), gendertxt.get(),num\_platetxt.get(),  
 passwordtxt.get(), did)  
 result = False  
 try:  
 # pass #input, process, output  
 conn = connect()  
 cursor = conn.cursor()  
 cursor.execute(sql, values)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 result = True  
 print("Update successfully")  
  
 except:  
 # pass #error message  
 print("Error : ", sys.exc\_info())  
 finally:  
 # pass #Remove all used resources  
 del values, sql  
 return result  
  
 delete\_btn = Button(self.booking\_frame, text="Delete", command=deleteAccount)  
 update\_btn = Button(self.booking\_frame, text="Update", command=saveUser)  
  
 delete\_btn.place(x=315, y=400)  
 update\_btn.place(x=260, y=400)  
  
 namelbl = Label(self.booking\_frame, text="Name")  
 addresslbl = Label(self.booking\_frame, text="Address")  
 mobilelbl = Label(self.booking\_frame, text="Mobile")  
 emaillbl = Label(self.booking\_frame, text="Email")  
 doblbl = Label(self.booking\_frame, text="AGE")  
 genderlbl = Label(self.booking\_frame, text="Gender")  
 num\_platelbl = Label(self.booking\_frame, text="Password")  
 passwordlbl = Label(self.booking\_frame, text="Password")  
  
 # lblMessage = Label(self.booking\_frame, text='', bg="gray")  
  
 nametxt = Entry(self.booking\_frame)  
 addresstxt = Entry(self.booking\_frame)  
 mobiletxt = Entry(self.booking\_frame)  
 emailtxt = Entry(self.booking\_frame)  
  
 gendertxt = Entry(self.booking\_frame)  
 num\_platetxt = Entry(self.booking\_frame)  
 passwordtxt = Entry(self.booking\_frame)  
 age\_txt = Entry(self.booking\_frame)  
  
  
 namelbl.place(x=40, y=270)  
 addresslbl.place(x=40, y=300)  
 mobilelbl.place(x=250, y=270)  
 emaillbl.place(x=250, y=300)  
 doblbl.place(x=460, y=270)  
 genderlbl.place(x=460, y=300)  
 num\_platelbl.place(x=40, y=330)  
 passwordlbl.place(x=250, y=330)  
  
 lblMessage.place(x=360, y=400)  
  
 nametxt.place(x=100, y=270)  
 addresstxt.place(x=100, y=300)  
 mobiletxt.place(x=310, y=270)  
 emailtxt.place(x=310, y=300)  
 age\_txt.place(x=520, y=270)  
 gendertxt.place(x=520, y=300)  
 num\_platetxt.place(x=100, y=330)  
 passwordtxt.place(x=310, y=330)  
  
 pass

This code creates a class for a driver dashboard, which allows the driver to view their upcoming bookings, trip information, and their profile. The class inherits from a parent class and initializes with a master window and a driver ID (did). It creates several frames for different widgets, and several buttons for different functions such as "Dashboard", "Customer", "Trip Information", "Profile" and "Logout". Each button is associated with a specific function when clicked. The function "delete\_frame" is used to delete widgets when switching between different functions. The "View\_Booking\_btn" function retrieves booking data from the database where the driver ID and status match "active" and "ongoing" respectively, and displays the data in a table. This code appears to be a driver dashboard for a transportation booking system (TBS). It has various buttons on the side frame for different functionality such as viewing customer bookings, trip information, and profile. When the "Customer" button is pressed, the View\_Booking\_btn function is called. This function retrieves all "active" and "ongoing" bookings from a MySQL database where the driver ID (did) matches the one passed in the constructor of the class. It then displays the retrieved bookings in a ttk Treeview widget in the booking\_frame. The driver can then select a specific booking and press the "Complete Booking" button to update the status of the booking to "Complete" in the database. There are also functions for displaying trip information and the driver's profile.

"tripinfo" for the DRIVER\_DASHBOARD class. When this method is called, it deletes any previous widgets that may be present in the "booking\_frame" frame. Then, it gets the driver id from the object and sends it to the function "bookinggetAll2", which appears to be a function to get all data from the "booking" table. The function then checks if there is any data returned, and if not, it displays a message saying "Record not found". If there is data returned, it creates a new frame called "tableframe" and places it in the "booking\_frame" frame, then it creates a ttk Treeview widget and adds it to the "tableframe" frame. It then adds columns to the Treeview widget and sets the column headers. It then iterates through the data returned from the "bookinggetAll2" function and adds it to the Treeview widget as rows. Finally, it packs the Treeview widget so that it is displayed on the screen.

The Account function is used to display the driver's account information in a table view. It first calls the delete\_frame function to delete any existing widgets in the booking\_frame. Then it uses the driversearch1 function to search for the driver's account information by passing the driver's ID (DID) as a parameter. If the search does not return any results, it displays a message saying "Record not found". If it does find results, it creates a table view and populates it with the driver's account information. The function also includes several other features such as the ability to select a record in the table and display its information in entry boxes, the ability to delete the driver's account, and the ability to update the driver's account information.