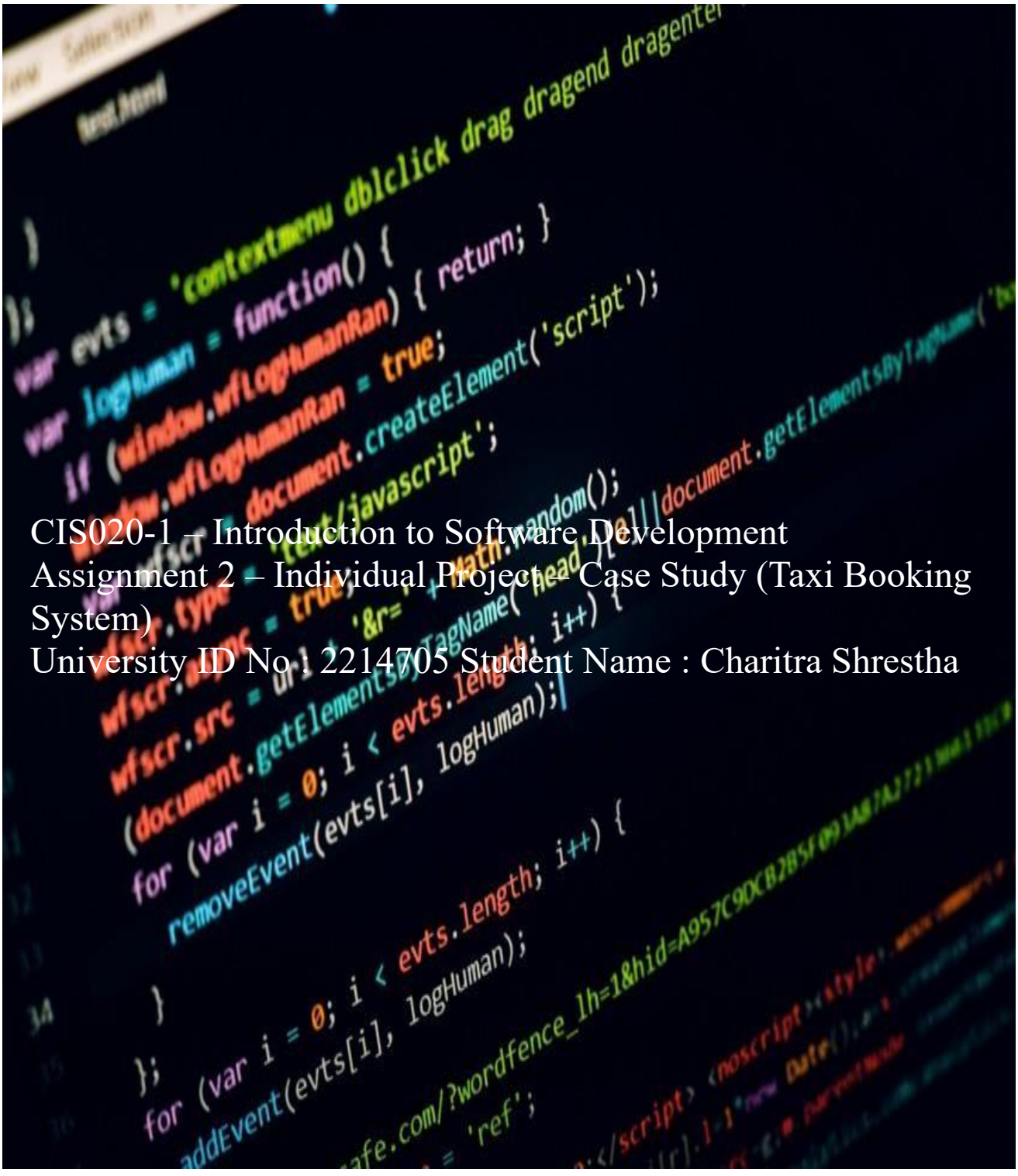
CIS020-1 – Introduction to Software Development
Assignment 2 – Individual Project – Case Study (Taxi Booking System)
University ID No : 2214705 Student Name : Charitra Shrestha



CIS020-1 – Introduction to Software Development
Assignment 2 – Individual Project – Case Study (Taxi Booking System)
University ID No : 2214705 Student Name : Charitra Shrestha

Contents

University ID No: 2214705 Student Name : Charitra Shrestha

Introduction/Overview

The online taxi booking system was created to make taxi journeys easier and more automated for clients. Customers and drivers may easily interact with this system, which was developed in Python as a semester assignment. This system consists of a desktop application that assists customers in making travel requests, an admin that accepts rides and assigns drivers, drivers that make travel plans and finish them on time, and a database that stores information about both customers and drivers.

The project started with a thorough investigation into the features and design of the current taxi booking systems. There were difficulties throughout the development process. Upon commencing the coding of multiple components, including the registration, booking, and assign system, I had several challenges. Problem-solving and debugging became essential components of the voyage. But I overcame these obstacles and made sure that the program's needs were fulfilled by being persistent and dedicated to producing a desired result.

Overall, I've learned a lot from working on this project. It gave me new perspectives on the complexities of creating user-friendly interfaces and improved my technical proficiency in Python. The necessity of flexibility and problem-solving in the field of software development was highlighted by the iterative research, design, and coding process. I can see how my skills have evolved as I think back on finishing this task, and I enjoy conquering obstacles. With an

emphasis on user-centric design and effective system functionality, the skills I've gained from this project will surely help me grow as a developer going forward.

Task Description

A taxi company wants to offer an online booking tool so that users can schedule trips and drivers can view their forthcoming schedules. Data must be kept in an external file, such as a database or text file. At the very least, client information (name, address, phone number, email, and mode of payment) and travel data (location, time, and date of pickup and drop-off) would need to be recorded and accessible. You will also need to keep driver information (license plate number and name). Every journey needs to have a single driver assigned to it (and a driver's allotted trips cannot overlap).

Customers who haven't registered yet must do so by entering their name, address, phone number, email address, and payment method. A registered customer should be able to schedule a trip, which includes providing the drop-off address and time in addition to the pickup address. Ideally, this can be done through a graphical user interface.Additionally, users ought to be able to see and cancel their trips as needed.These functions are exclusively accessible upon the customer's sign-in (i.e., registration required). The administrator of the taxi company must confirm each new trip reservation and appoint a driver. For every confirmed reservation, a single driver must be assigned. Taxi drivers should be able to see a list of their upcoming trips after logging in.

Project Plan/Schedule

Week No.	Tasks	Priority
1.	Requirements Gathering	MUST
2.	System Design, such as the Register and Login GUIs, etc.	MUST
3.	Create a database structure to hold client, driver, reservation, and other pertinent data.	MUST
4.	Design of MiddleWare	MUST
5.	Connect the GUI to the database.	MUST
6.	Application Examination	MUST
7.	Create documentation and Video Presentation	MUST

Table 1: Project Plan

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 TBS	SHOULD
15.	A taxi driver must be able to view his bookings	MUST

Table 2: Functional Requirements

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.	

Table 3:Non-Functional Requirement

Usability Requirements

Reg.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	
6.	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.	

Table 4:Usability Requirements

*MOSCOW Notation:

M = MUST

S = SHOULD

C = COULD

W = WON'T

UML Diagrams

Uses Case Diagrams(s)

Sea Level Diagram

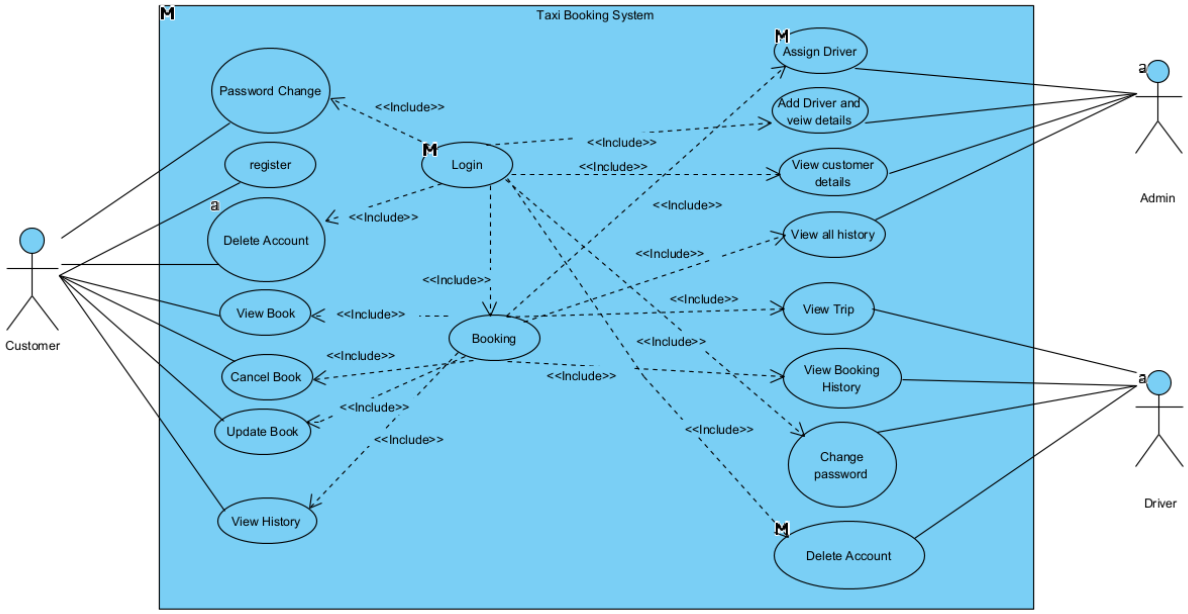


Figure 1: Sea level Diagram of the taxi booking system

Fish Level Diagram

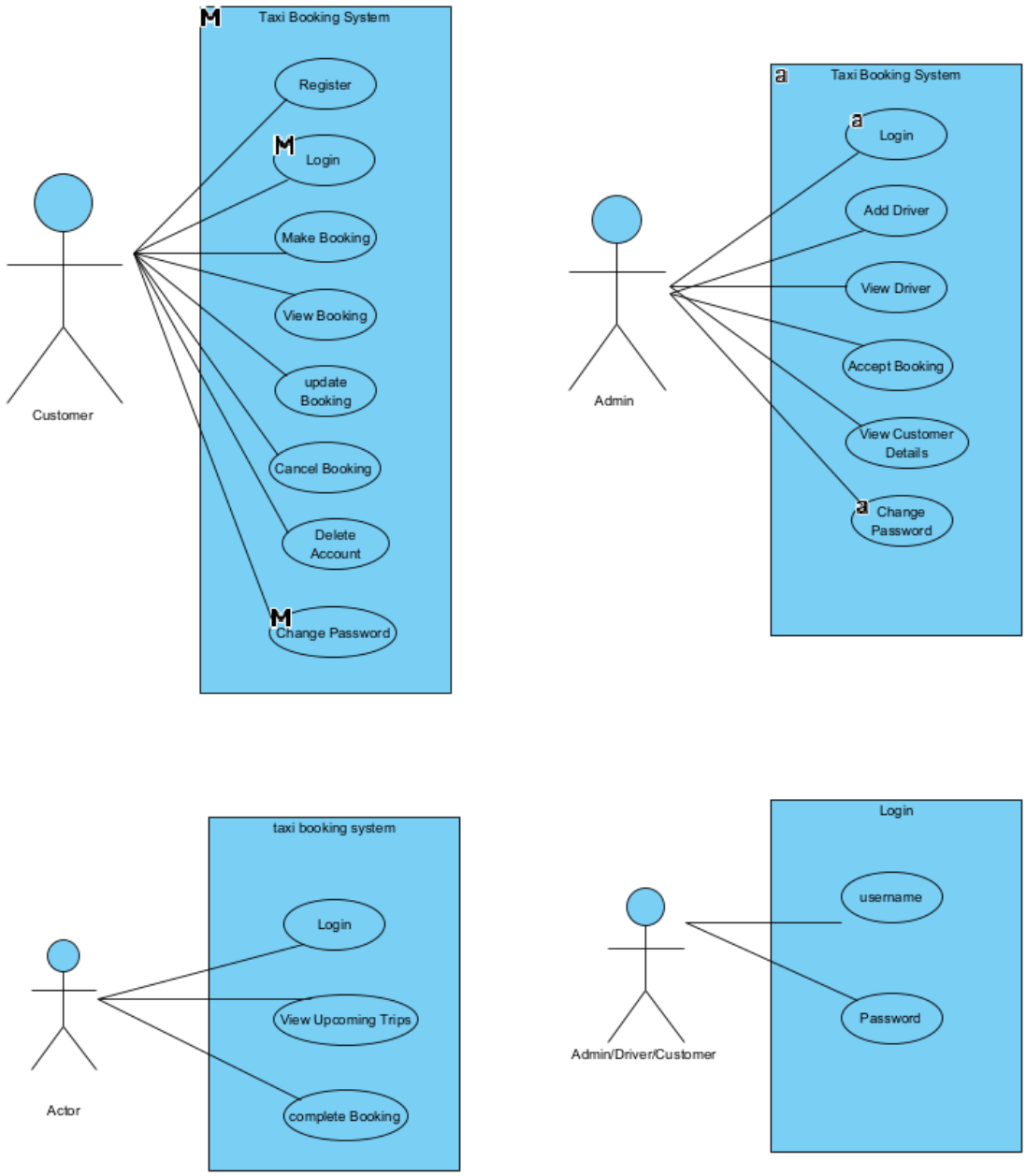


Figure 2:Fish level Diagram of taxi booking system

The sea level and fish level diagram for the taxi booking system, which has three actors—the customer, the administrator, and the driver—is shown in the following

figure. First, in order for the client to access their dashboard, they need to register if they don't already have an account. The customer will log in to access their dashboard following a successful registration. Customers will have access to several features on their dashboard, including the ability to make and view reservations. Additionally, the client has the option to make any required adjustments, such updating or canceling their reservation. Additionally, users can remove their accounts and modify their passwords. In contrast, managers must log in order to view the dashboard. The admin will be in charge of registering drivers, accessing client information, and allocating drivers to specific reservations. Furthermore, the driver will have the ability to log in to their system once they have been officially registered by admin. They can review the allocated travels, check booking history, reset their password, and deactivate their account after checking in.

Use Case Specification/Description

For Customer:

Login into the System

The customer has to be able to log in to the system. In order to log in, the user must first create an account by providing the system with accurate information such as their name, address, phone number, email address, and so forth.

Book the trip

They can schedule the trip by entering the correct information (pickup and dropoff addresses, dates, and times) on the booking panel after logging into the system.

View the booking

After booking the trip, the customer can check both their booking and the completed trips.

Update booking

The customer has the ability to modify the trip details and cancel the reservation they made.

For Driver

Login to the system

To view their journey, the driver needs log into the system.

View booking

Drivers have access to the booking that the admin has allotted to them.

For Admin

Login to the System

To assist the driver, the administrator needs to sign in to the system using a valid username and password.

Assign the driver

The only drivers that the admin can assign are those who are available at that moment.

Activity Diagram(s)

Activity diagram of Admins

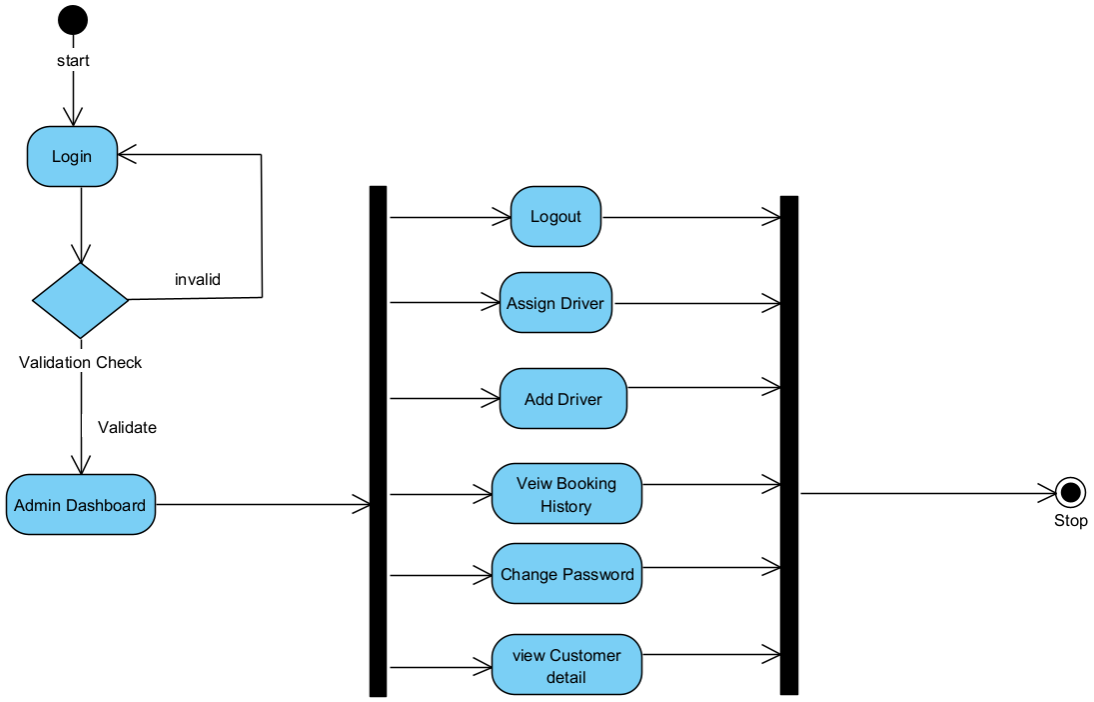


Figure 3:Admin Activity Diagram

The activities of an admin in a taxi booking system are depicted in an activity diagram. Starting with the login interface, the admin logs in to the system using a working email address and password. From there, they may view the customer request trips on the admin dashboard. They have to Assign the driver to the customer who are available at that time. The driver's can also be added by the admin in his company. He gets access to the booking history, where clients make reservations, designate drivers, and track when drivers finish their excursions.

Admins have the option to alter the password as well. He has access to customer details that have been registered in the system.

Activity diagram of Customers

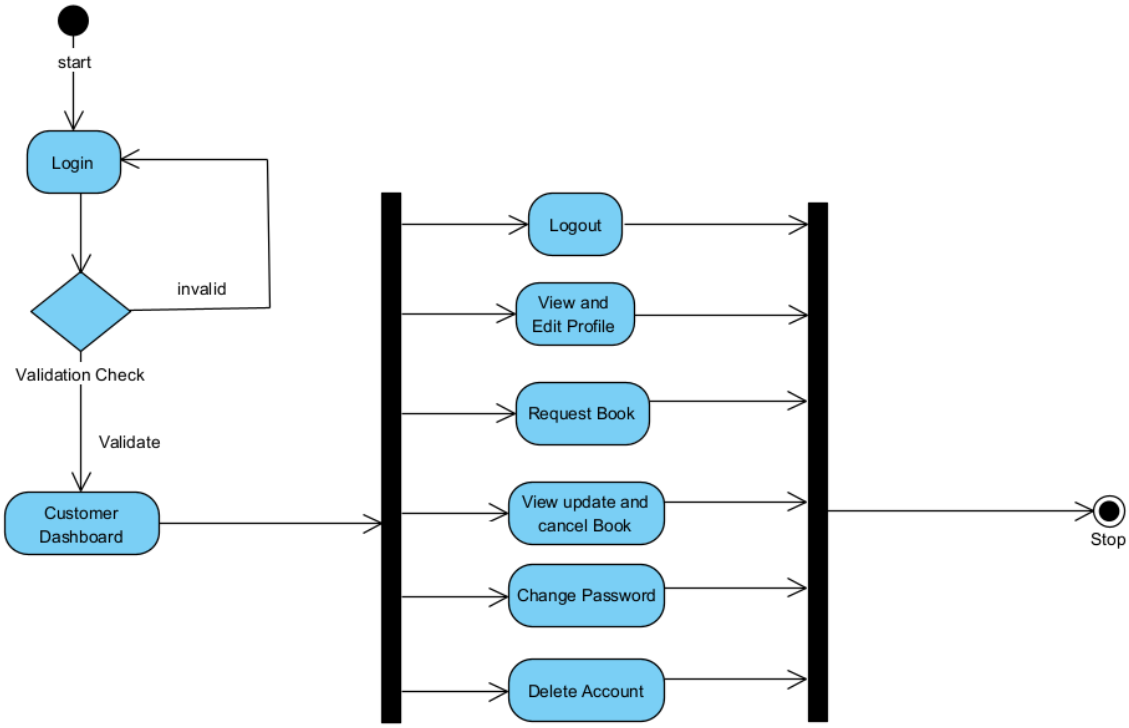


Figure 4:Activity Diagram of Customer

One kind of flowchart that illustrates the processes an customer takes to book a taxi is an activity diagram of the customer in the taxi booking system. The login interface is where it all begins, where users can log in using their verified email address and password. Upon logging in, customers are directed to the Customer Dashboard, where they can check and amend their profile should any errors have occurred during registration. Customers can make a reservation by entering the time and address for their preferred pick-up, then choosing the drop-off location and date. The consumer making a booking request is depicted in the diagram. After

that, the system will allow them to view the trip they requested. They can then modify or cancel their reservation whenever they'd like, as well as change their password. They can remove their account and log out of the system if they decide they don't need this app.

Activity diagram of Drivers

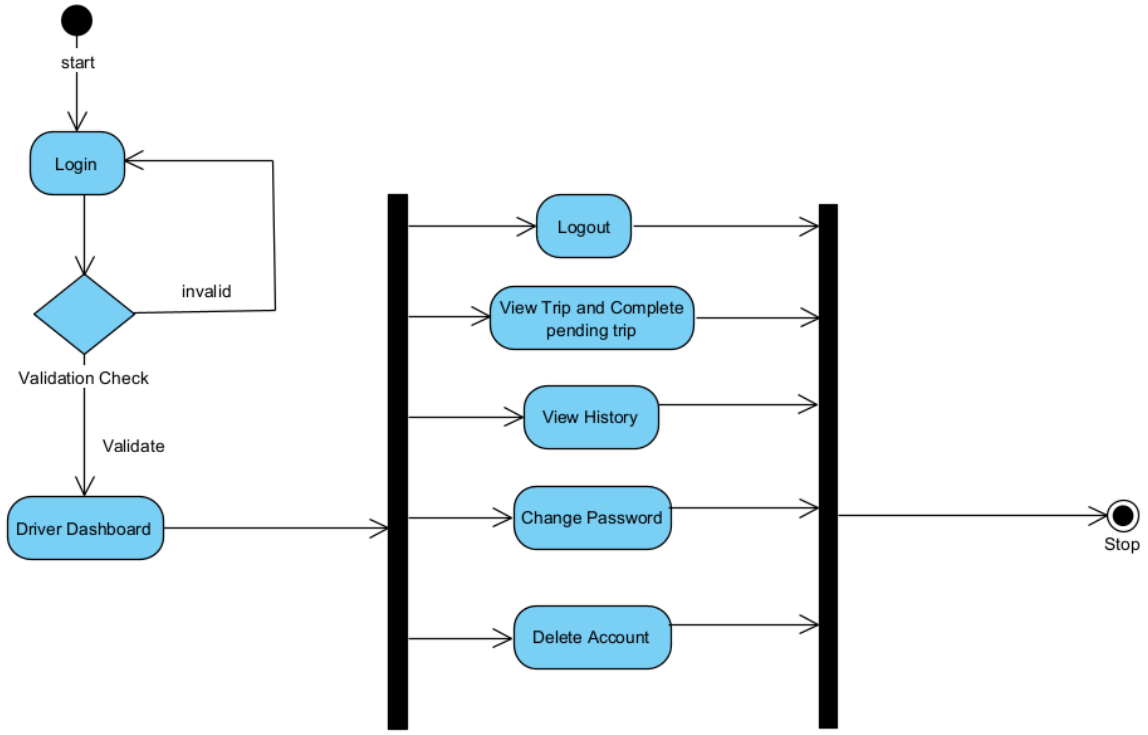


Figure 5:Activity Diagram of Driver

Similar to the customer, the driver must first login. Upon validation, they are then taken to the driver dashboard, where they may examine the booking that the admin has allocated to them. They ought to finish the journey that was allocated to him. They can see the history of his completed trips once the vacation is over. They

have the option to deactivate their accounts and change their passwords. They can quickly log off of the system after finishing their task.

Class Diagram(s)

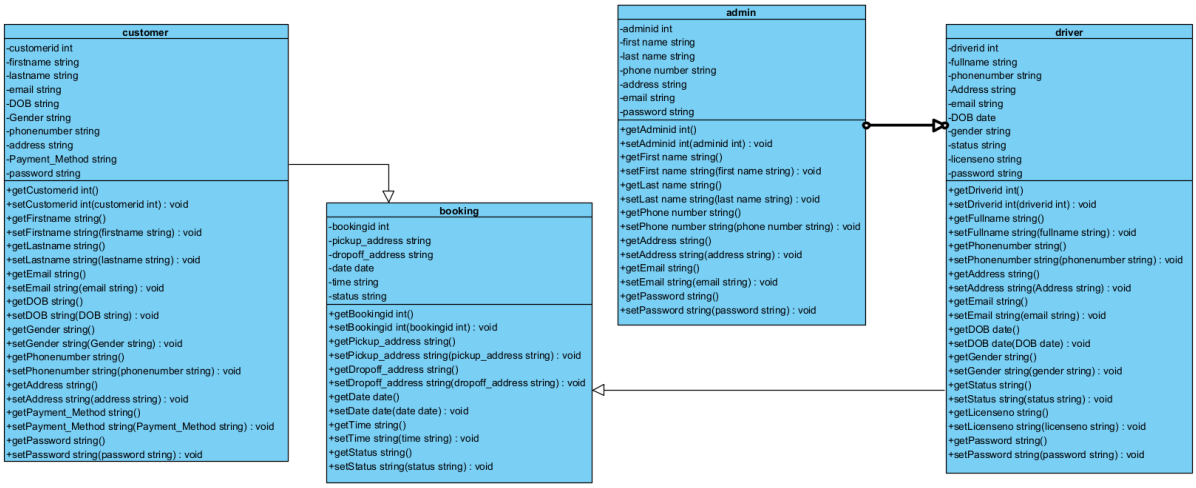


Figure 6:Class diagram of taxi booking system

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved. (Contributor, n.d.)

Customer: This class represents a customer of the taxi booking system. Its

attributes are customerid, customer first name, customer last name, Email address, Date of birth, Gender, Phone number, Address, Payment_ Method and password. It operate request , update and cancel booking.

Booking: This class represents a booking made by a customer for a trip. Its attributes include bookingid, pick-up address and drop-off address, time, date, and status of the booking.

Driver: This class represents a driver who works for the taxi booking system. Its attributes include the driver id, driver's name,phonenumber, address, Gender, status, email, password, date of birth and license plate number of the taxi. Its operation includes View trip and completing a trip.

Admin: This class represents an admin of the taxi booking system. Its attributes include adminid, firstname, lastname, email, password. Its operation includes Add the driver , assign driver to the trip and manage the driver.

Classes(omitting methods/ opearation)

1. Customer(customerid, customer first name, customer last name, Email address, Date of birth, Gender, Phone number, Address, Payment_ Method, password.)
2. Booking(bookingid, pick-up address and drop-off address, time, date, and status)
3. Driver(driverid, driver's name, phonenumber, address, Gender, status, email, password, date of birth, license plate number)
4. Admin(adminid, firstname, lastname, email, password)

Database Design

Logical Database Design

Figure 7: Database Design

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model. This mapping can be understood as the result of trying to achieve two distinct sets of goals: (i) representation goal: preserving the ability to capture and distinguish all valid states of the conceptual schema; (ii) data management goals: addressing issues related to the ease and cost of querying the logical schema, as well as costs of storage and constraint maintenance. This entry focuses mostly on the mapping of (Extended) Entity-Relationship (EER) diagrams to relational databases. (Alexander Borgida, 2009)

Entity Relationship Model(ERM)

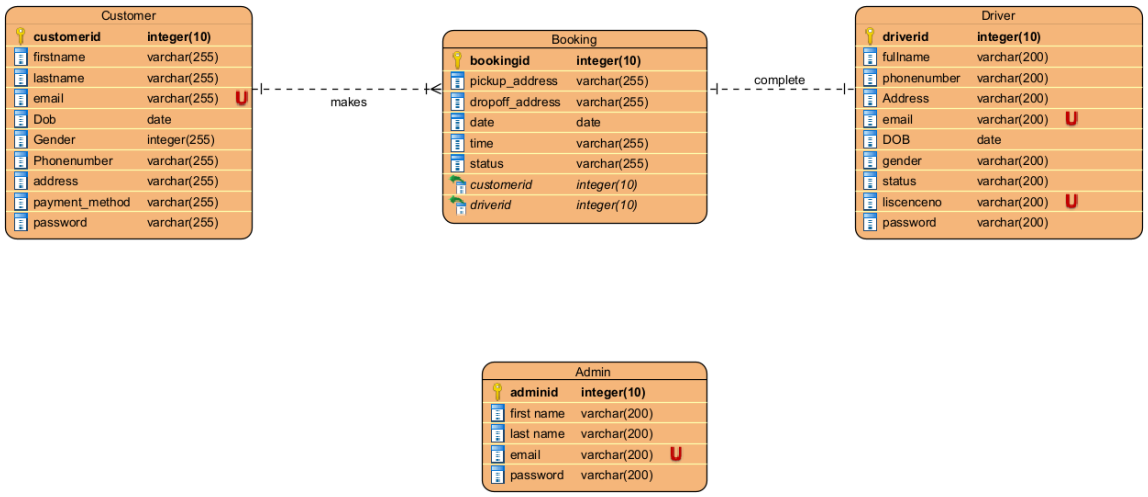


Figure 8:Erm of taxi booking system

The ER diagram of a taxi booking system illustrates the entities, attributes, and relationship between them. In this ERM, each entity is represented by a rectangle, and each relationship is represented by a line connecting the entities. The entity's attributes are listed within the rectangle.

- The customer entity has one-to- many relationships with the booking entity since each customer can make multiple booking.
- The booking entity has one-to-one relationships with driver enity since each booking can assign only one driver at a time.

List of Entities

- **o** customer (**customerid**, customer first name, customer last name, Email address, Date of birth, Gender, Phone number, Address, Payment_ Method and password.)
- **o** booking (**bookingid**, pick-up address and drop-off address, time, date, status, customerid*, driverid*)
- **o** driver (**driverid**, driver's name, phonenumber, address, Gender, status, email, password, date of birth, license plate number)
- **o** admin (**adminid**, firstname, lastname, email, password)

Primary Key = Bold and Underlined

Foreign Key = *

Skeleton Tables (with Primary Keys and Foreign Keys)

- **o** customer (**customerid**, customer first name, customer last name, Email address, Date of birth, Gender, Phone number, Address, Payment_ Method and password.)
- **o** booking (**bookingid**, pick-up address and drop-off address, time, date, status, customerid*, driverid*)
- **o** driver (**driverid**, driver's name, phonenumber, address, Gender, status, email, password, date of birth, license plate number)
- **o** admin (**adminid**, firstname, lastname, email, password)

Primary Key = Bold and Underlined

Foreign Key = *

Data Dictionary

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
First name	Varchar(255)	255		No			First name of customer
Last name	Varchar(255)	255		No			Last name of customer
Email	Varchar(255)	255		No		Must be email format containing an @ and a '.' Regex expression used	Email of customer
address	Varchar(255)	255		No			address of customer
phonenumber	Varchar(255)	255		No			Phone number of customer

Gender	Varchar(255)	255		No			Gender of customer
DOB	date			No			Date of birth of customer
Payment method	Varchar(255)	255		No			Payment method of customer
Password	Varchar(255)	255		No			Password of customer

Indexes

Keyname	Type	Unique	Column	Null
Primary	BTREE	Yes	customerid	No

Figure 9:data Dictionary of Customer

Booking							
Description: Customer Details							
Field Name	Datatype	Length	Index	Null	Default	Validation rule	Description
Bookingid (primary)	Int(10)unsigned	10	PK	No			Autoincremented Uniquely identifies every Booking made
Pickup_address	Varchar(255)	255		No			Pickup_address of customer
Dropoff_address	Varchar(255)	255		No			Dropoff address of customer
Date	date			No			Booking date of customer
time	Varchar(255)	255		No			Time for pick up
status	Varchar(255)	255		No			Status of booking

Indexes

Keyname	Type	Unique	Column	Null
Primary	BTREE	Yes	bookingid	No
FOREIGN	BTREE	YES	driverid	No
FOREIGN	BTREE	YES	customerid	No

Figure 10:data Dictionary of Booking

Driver							
Description: Driver Details							
Field Name	Datatype	Length	Index	Null	Default	Validation rule	Description
driverid (primary)	Int(10)unsigned	10	PK	No			Autoincremented Uniquely identifies every Driver
fullname	Varchar(255)	255		No			Name of driver
Email	Varchar(255)	255		No		Must be email format containing an @ and a '.' Regex expression used	Email of driver
address	Varchar(255)	255		No			address of driver
phonenumber	Varchar(255)	255		No			Phone number of driver
Gender	Varchar(255)	255		No			Gender of driver
DOB	date			No			Date of birth of driver

status	Varchar(255)	255		No			Status of driver
liscenceno	Varchar(255)	255		No			Liscenceno of taxi of driver
Password	Varchar(255)	255		No			Password of Password

Indexes

Keyname	Type	Unique	Column	Null
Primary	BTREE	Yes	driverid	No

Figure 11:data Dictionary of Driver

Admin							
Description: Admin Details							
Field Name	Datatype	Length	Index	Null	Default	Validation rule	Description
adminid (primary)	Int(10)unsigned	10	PK	No			Autoincremented Uniquely identifies every Admin
firstname	Varchar(255)	255		No			First of admin
lastname	Varchar(255)	255		No			Last of admin
email	Varchar(255)	255		No		Must be email format containing an @ and a '.' Regex expression used	Email of Admin
password	Varchar(255)	255		No			Passsword of admin

Indexes

Keyname	Type	Unique	Column	Null
Primary	BTREE	Yes	adminid	No

Figure 12:Data Dictionary of Admin

User Interface Design

1. Login GUI

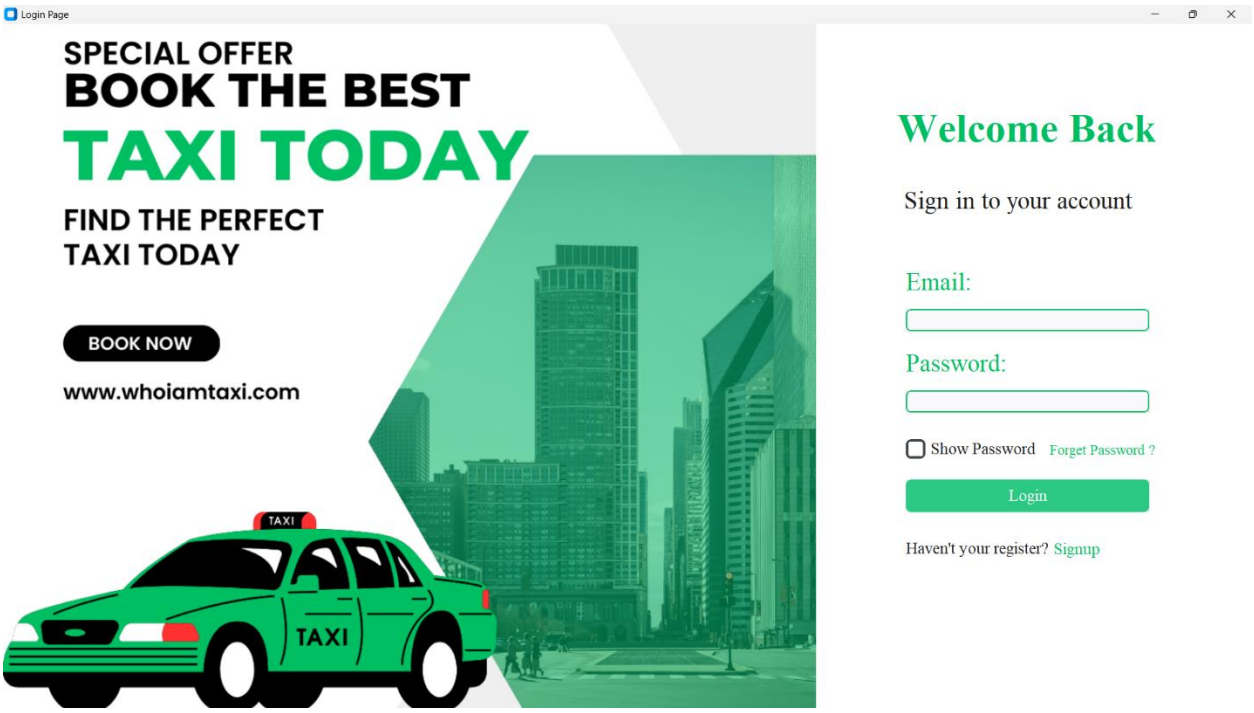


Figure 13:Login GUI

A desktop screen that allows users to log into the system is called a login page for a taxi booking system. The user can log in by entering their password and email address, both of which must be current. Once a valid email address and password have been entered, the user is directed to their respective dashboards: the customer, driver, and admin.

2. Forget Password GUI

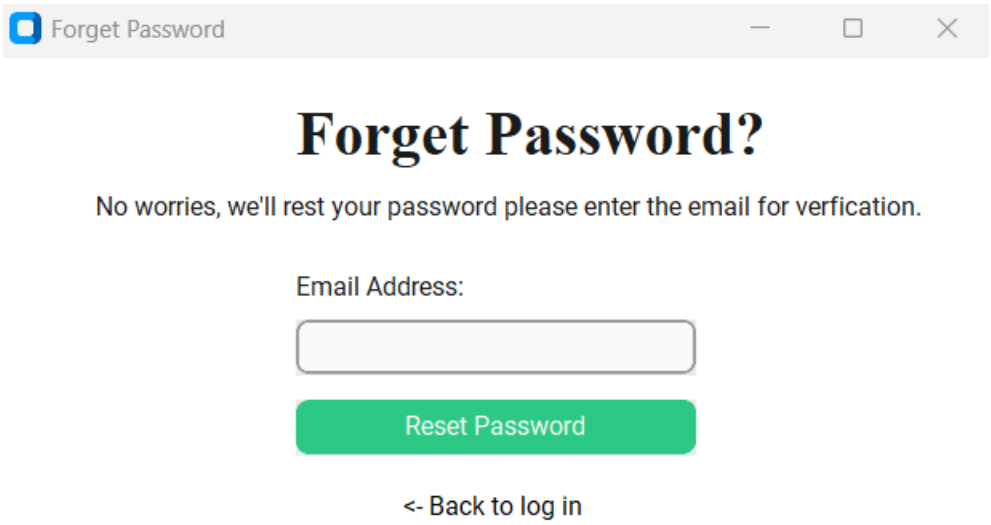


Figure 14:Forget password GUI

The user can reset their password using the aforesaid GUI if they misplace it or forget it. Verifying the email address provided upon registration is the only method by which a user can reset their password if they lose it.

3. Create new password

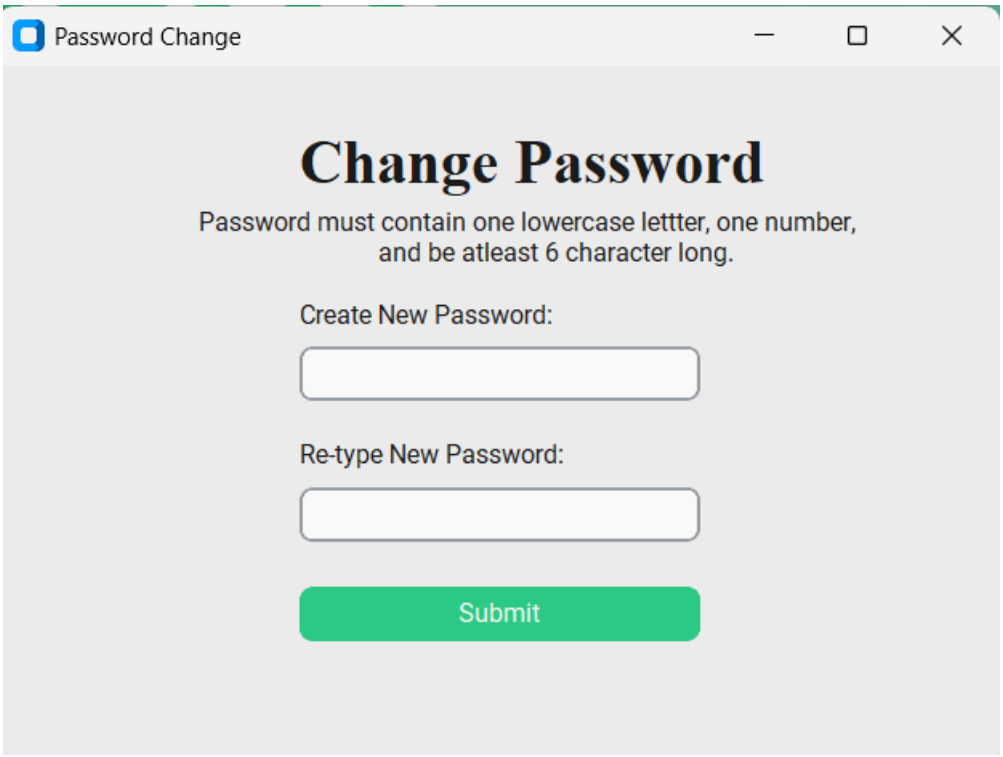


Figure 15:Create new password GUI

When you try to forget the password and you have confirmed your email address with the system, this GUI appears.

4. Customer Registation Page

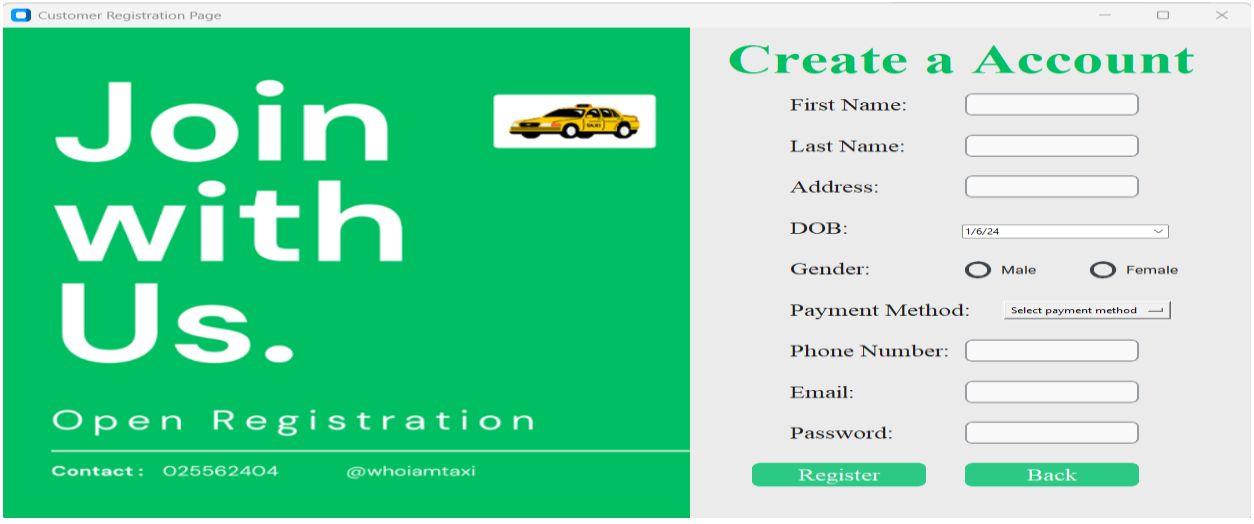


Figure 16: Customer Registration Page

The signup page for customers who haven't created an account yet is shown by the above GUI. When registering, the customer must enter their first and last names, address, gender, date of birth, payment method, phone number, email address, and password in the correct order. We must click the register button on the registration page in order to register or establish an account.

5. Customer Dashboard

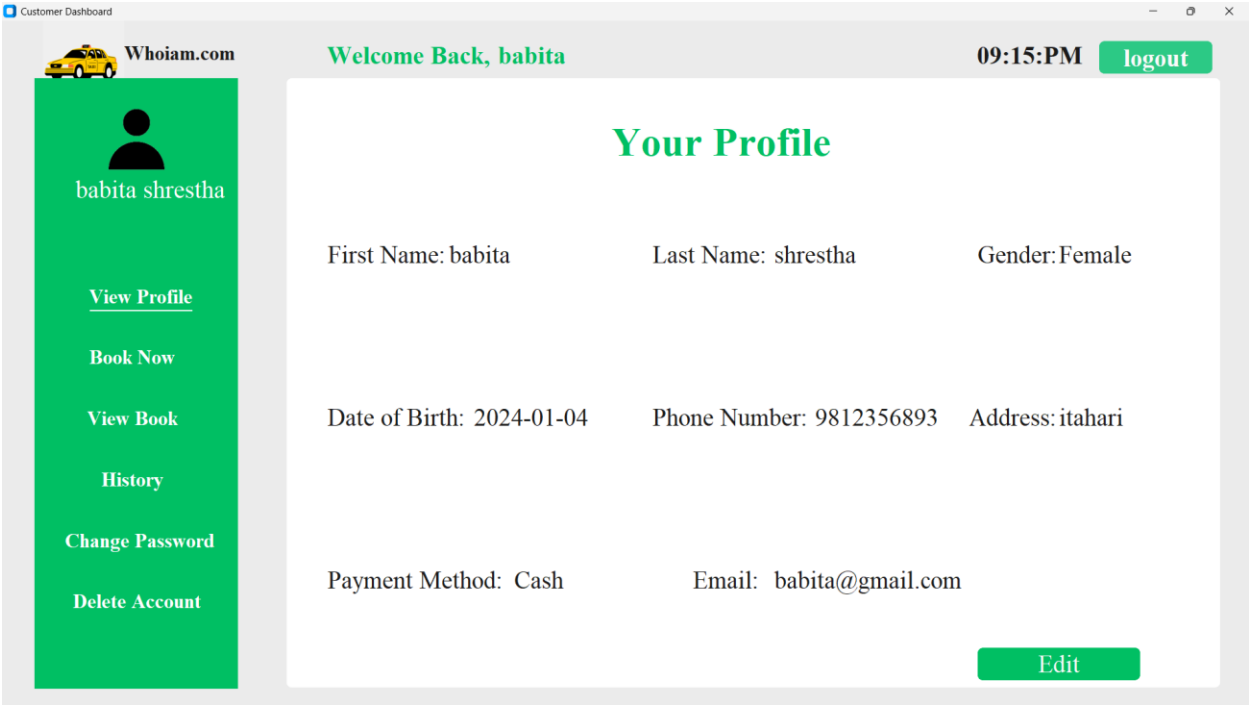


Figure 17: Customer Dashboard

This graphic depicts the Customer Dashboard that appears after a customer logs in to the system using a working password and email address. Initially, the name of my company appears in the upper left corner of the website, followed by the time in the right corner and the customer profile, which they may access and modify.

6. Edit profile page

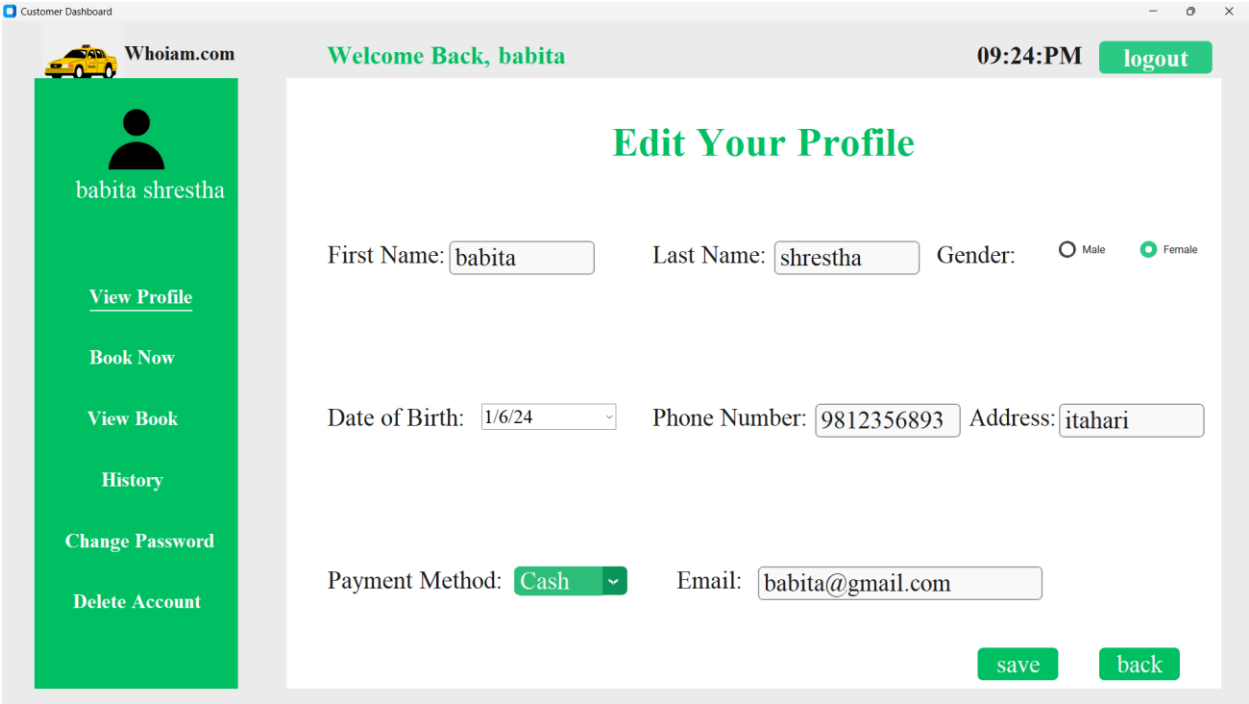


Figure 18:Edit Profile Page

When we click the edit button on the view profile, the Edit profile GUI shown above appears. If the user so chooses, this GUI assists in updating the profile information. To change information such as first and last names, gender, date of birth, phone number, address, payment method, and email, the customer must input the data correctly. In order to update the profile, the customer needs to press the save button.

7. Request Booking Page

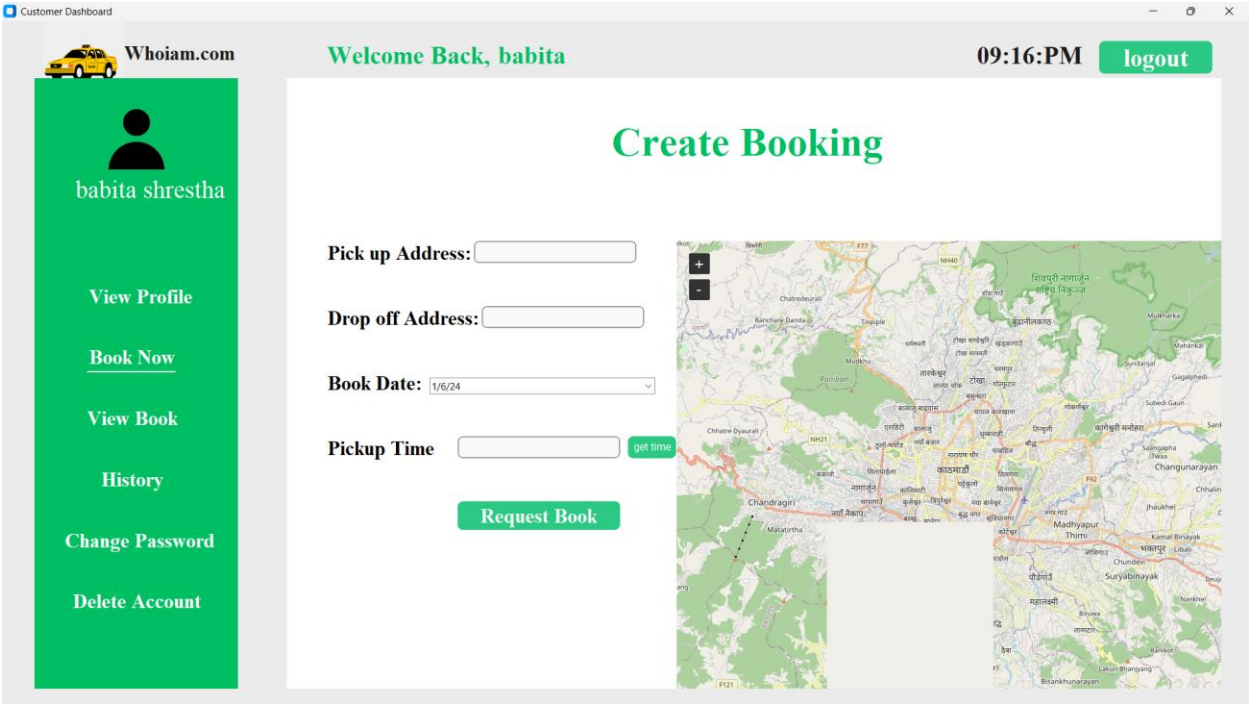


Figure 19: Booking Request Page

When a customer clicks the "book now" button, they can access this page. Customers can use this page to request a book by entering accurate information such as the booking date, time, pick-up address, and drop-off address. By clicking the "Request Book" button, the consumer can submit a request and view a map of their location.

8. View Book

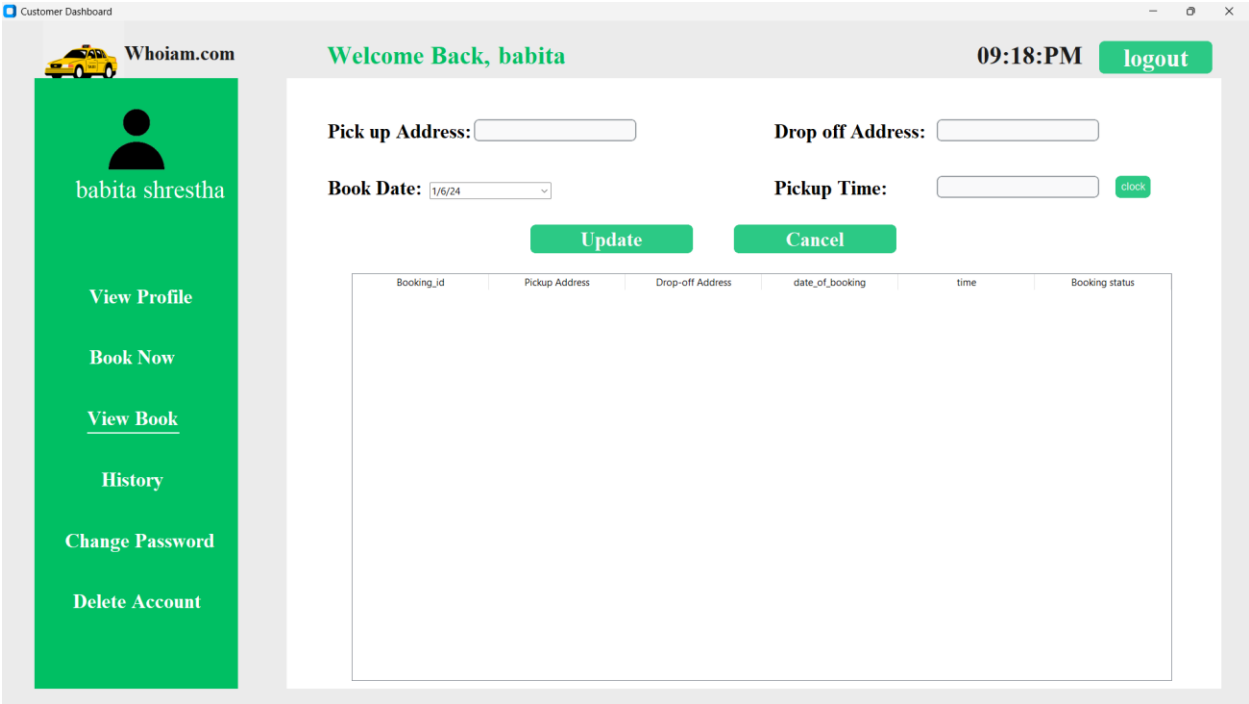


Figure 20: Customer View Booking

When a customerclicks the "View Book" button, the above GUI appears. Customers can access, update, and cancel their books at any time using this GUI. To amend or cancel a book, the customer must first pick the row from the table . Once that row has been chosen, they can update or cancel the booking by clicking the update or cancel button.

9. Change Password

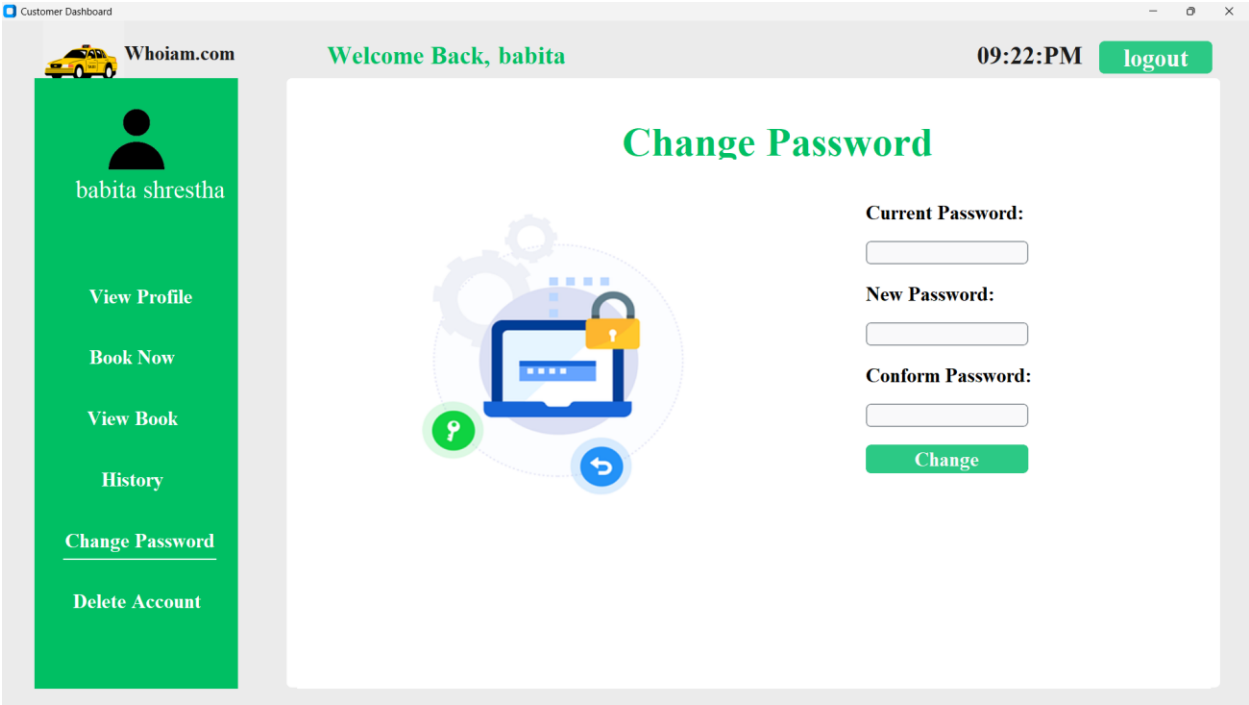


Figure 21: Change Password Page

A customer and driver sees the GUI shown above when they click the "change Password" button. By entering their old password, new password, and confirmation, they can modify their passwords using this GUI. If the previous password matches, the password will be changed.

10. Admin Dashboard

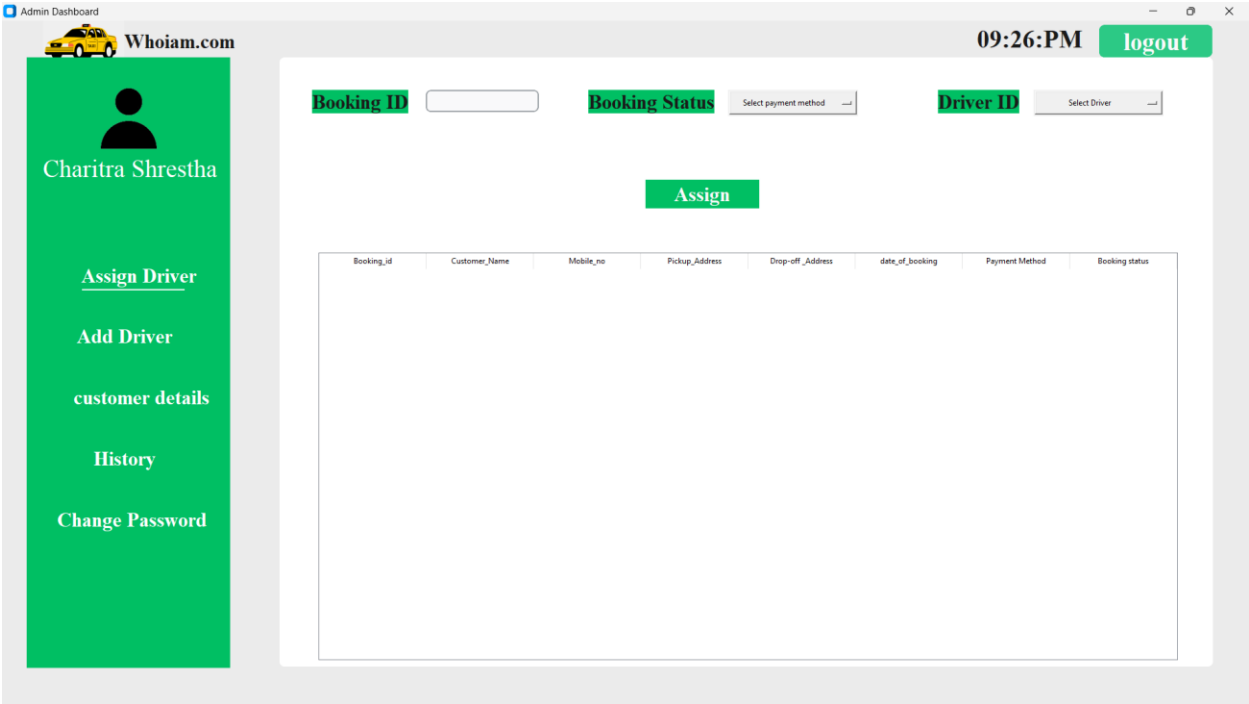


Figure 22: Admin Dashboard

After an administrator logs in to the system using a working email address and password, this guide is accessible. The Admin Dashboard initially has an Assign Driver Page. The Admin Dashboard opens with the Assign Driver Page visible. The pending trip will be listed in a table in this gui, and the administrator will assign the driver to each booking by choosing a row from the table.

11. Driver Registration Page

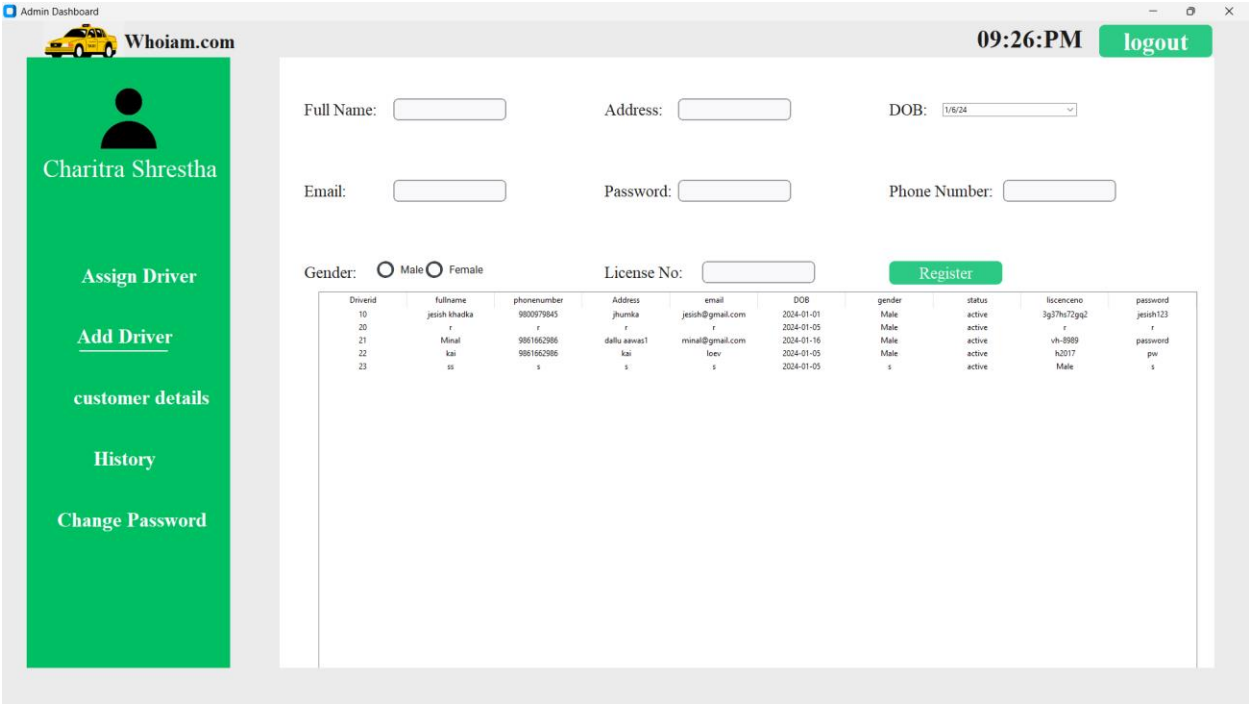


Figure 23: Driver Registration Page

Using this user interface, an administrator registers a driver with the database. The GUI appeared when the administrator clicked the ADD Driver button. The admin can register the driver by providing details such complete name, address, DOB, email, password, phone number, gender, and license number. The Register Driver is also viewable by the Admin.

12. Customer Details

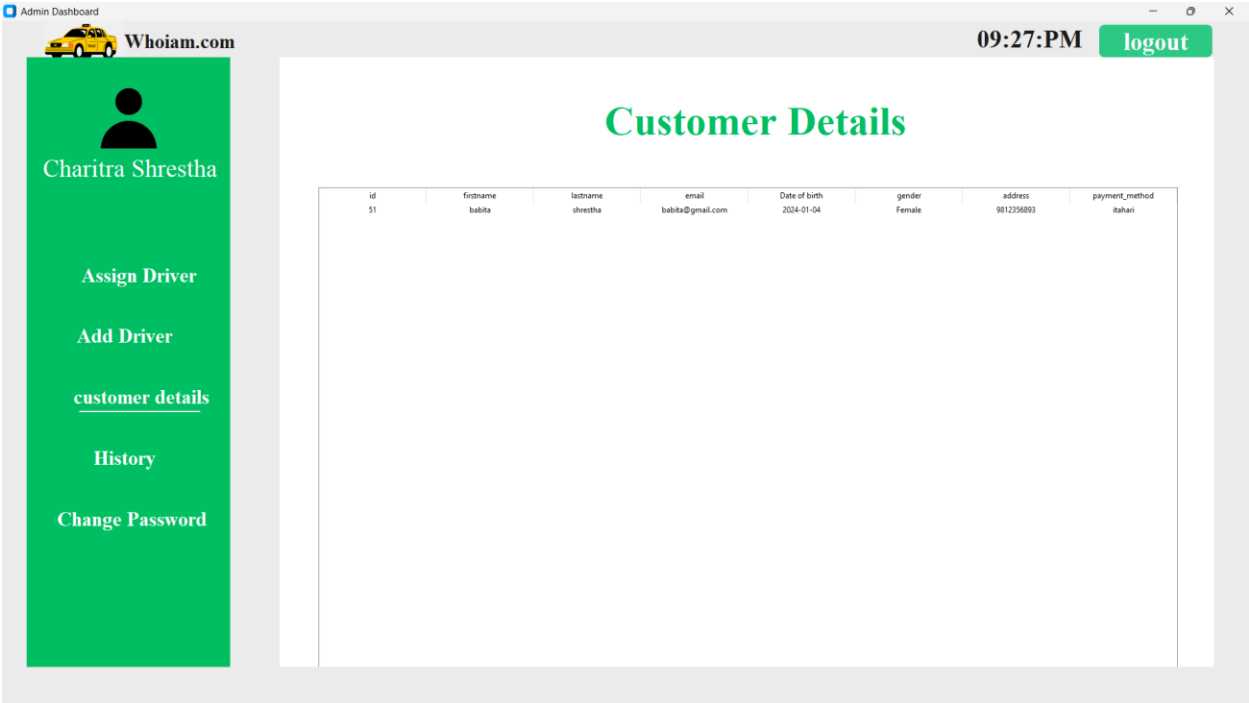


Figure 24: customer Details Page

When the admin clicked on the customer details button, the GUI shown above appeared. Then, when a customer creates an account in this system, the administrator can view the customer's details.

13.Booking History

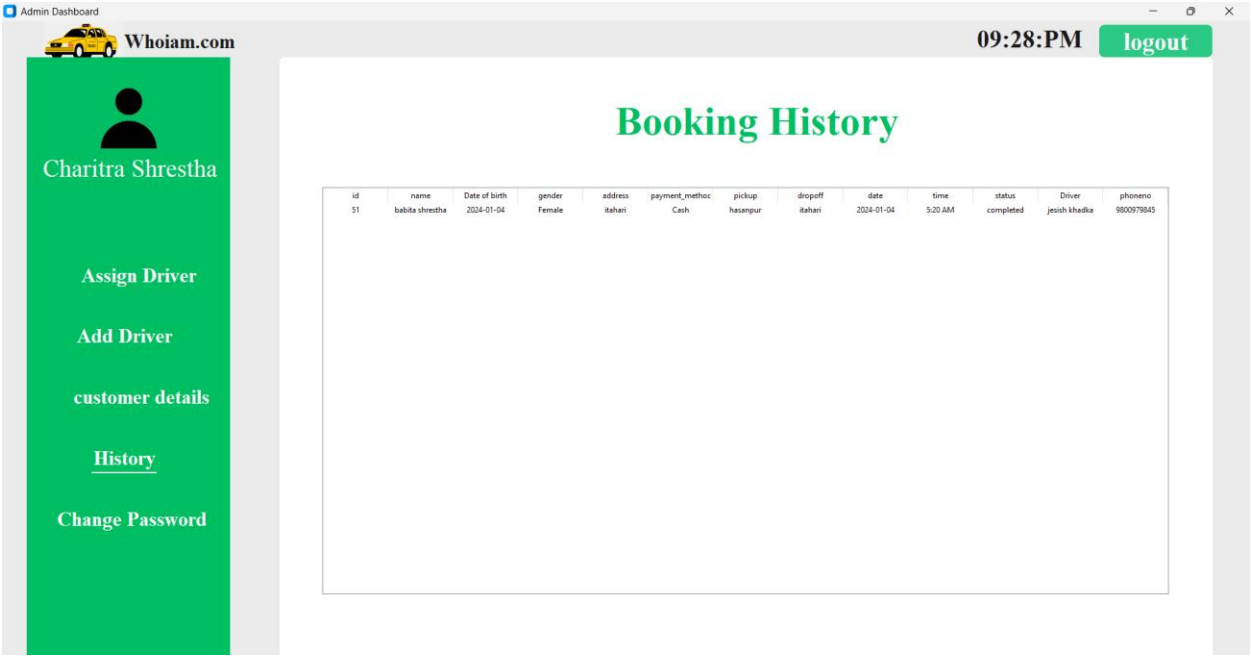


Figure 25: booking hsitory of Customer

When an administrator clicks the History button on the Admin Dashboard, the GUI shown above appears. The administrator can see the customer's booking history and the driver who was assigned to them.

14. View Trip(Driver Dashboard)

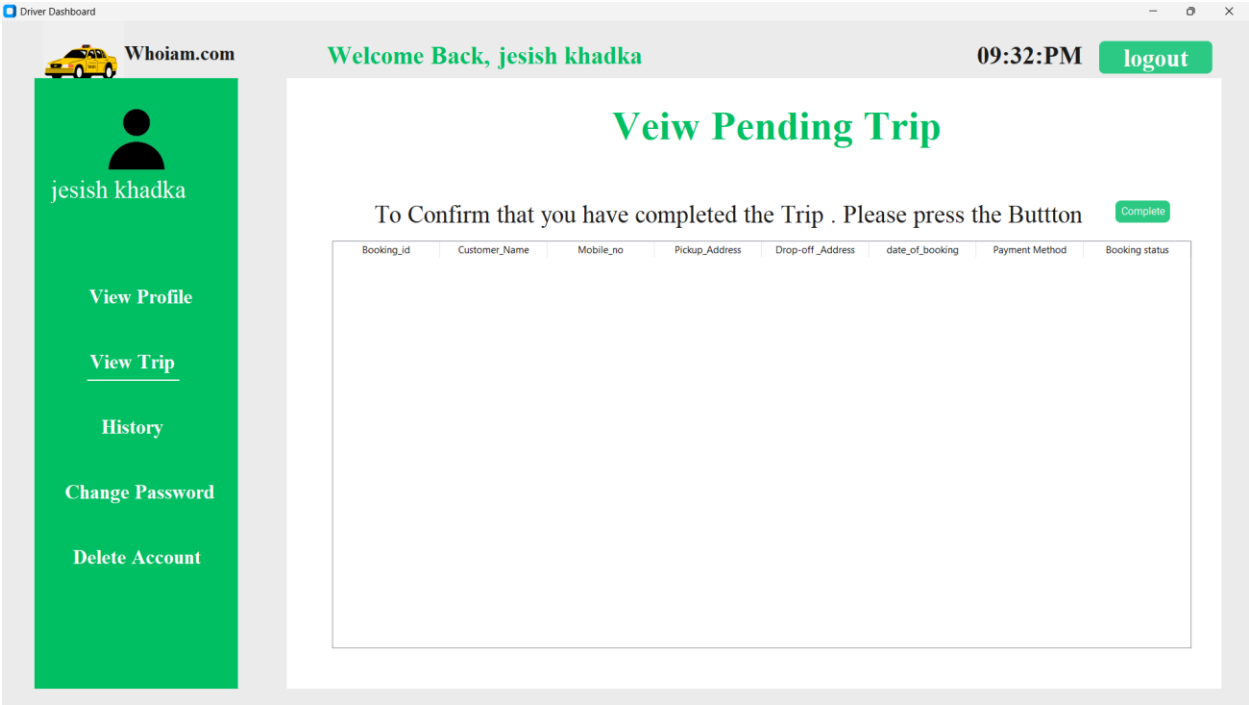


Figure 26: Veiw Trip of Customer Page

When a driver logs onto the system, the GUI shown above appears. The driver can view the allocated bookings that have been made specifically for him in this GUI. The driver can end the journey by clicking button on the above GUI after the trip is finished.

15. Delete Account

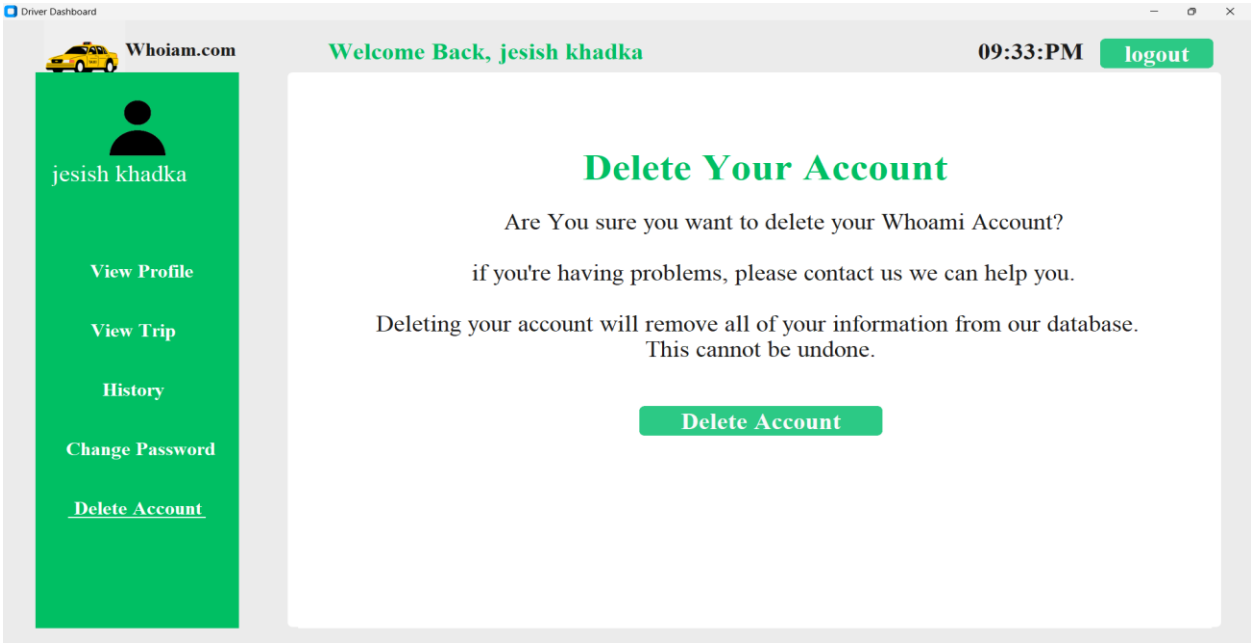


Figure 27: Delete Account Page

The Gui becomes visible on both the driver's and customer's dashboards when clicking the "Delete Account" button. Customers and drivers can remove their existing accounts from the system using this GUI.

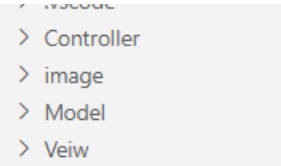
Implementation

The assignment involved developing a taxi booking system using the Python programming language. Giving customers an expedient and reasonably priced travel experience is the aim of this system. The admin will assign the driver to finish the trip based on booking requests from clients. A streamlined code editor with support for development tasks like task scheduling, version control, and debugging, Visual Studio Code was used to create this system. It is an extremely

capable source code editor that works with almost all programming languages and is compatible with Windows, Mac OS , and Linux desktop operating systems.This system manages the data and is in charge of keeping everything organized, accessible, and stored. Wampserver is used to create the database which store all of the data in this system, while MySQL is utilized as the backend.


```
import mysql.connector as c
class Database:
    message = ""
    try:
        @staticmethod
        def Connect():
            connection = c.connect(
                host='localhost',
                user = 'root',
                password = 'root',
                database = 'taxi_booking_system',
            )
            if connection.is_connected():
                Database.message = "Connected"
                return connection
            connection.close()
    except Exception as e:
        print(e)
```

The MVC (Model, View, Controller) architecture is the foundation of this project.It is a paradigm used frequently in software design to implement controlling logic, data, and user interfaces. It facilitates coding and makes it easier to read and comprehend.



This project made use of object-oriented programming, or oop, since it offered a methodical approach to managing several components. To create this system, I used Python libraries like Custometkinter, ttk, tkinter, PIL, and for connecting to the mysql I was used python mysql connector libaray.The rationale behind customtkinter is its abundance of style properties, which contribute to a streamlined and visually appealing user interface. While building this system, I ran into a number of issues, such as the other GUI was not opening when the function was called, the table was not refreshing while updating and deleting, the files were not importing from other folders, and i faced errors when connecting to the database. However, I was able to resolve some of these issues with the assistance of the internet, while others were resolved with the assistance of my mentor, videos, and senior. Since I had studied Python programming in high school, it was easy for me to code in the language. However, developing this system was a totally new experience that taught me even more about Python. It took me seven or eight hours a day to do this project. I learned about the oops notion, which was helpful as it was new to me. Even though I had to overcome new obstacles, it was an enjoyable and novel experience doing this project.

Testing

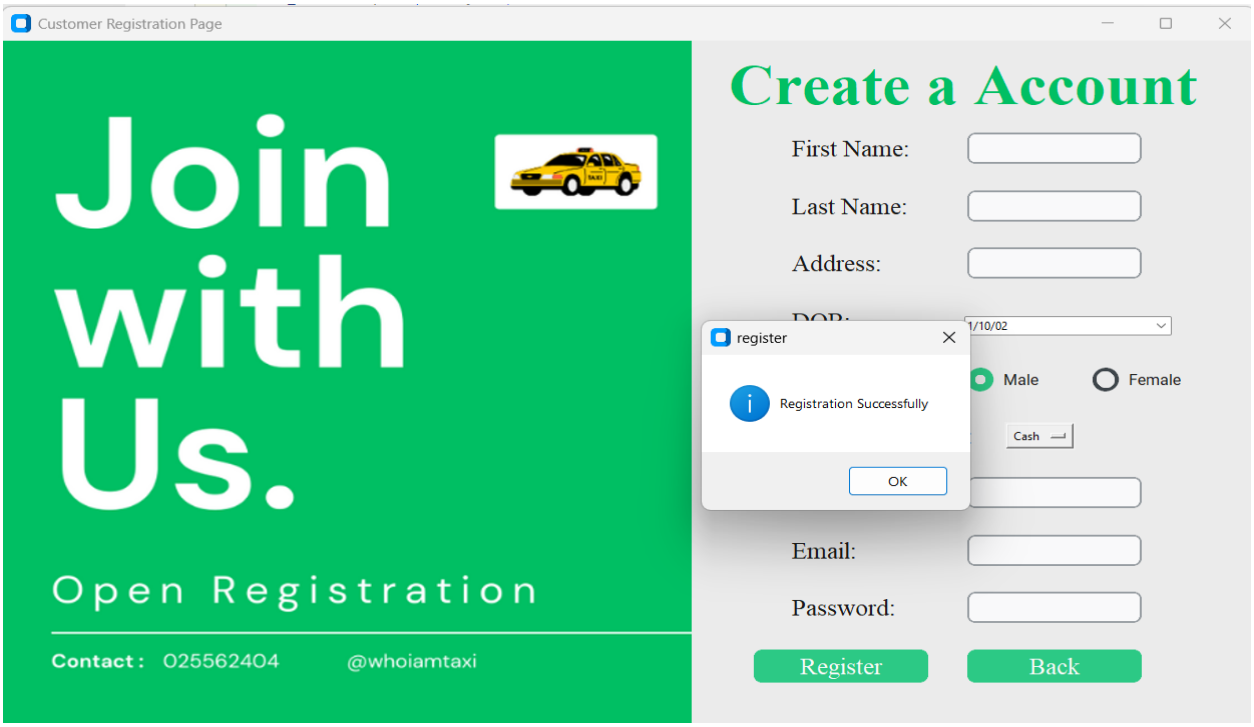


Figure 28:try to create a new account

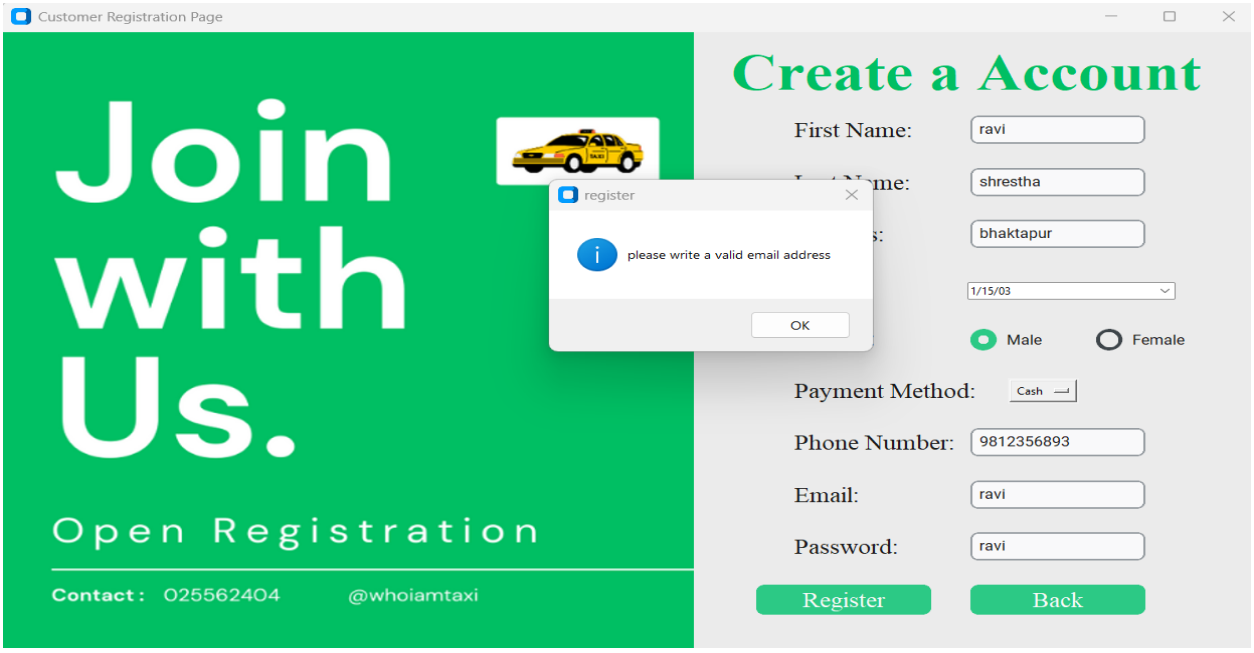


Figure 29: try to insert email in other form

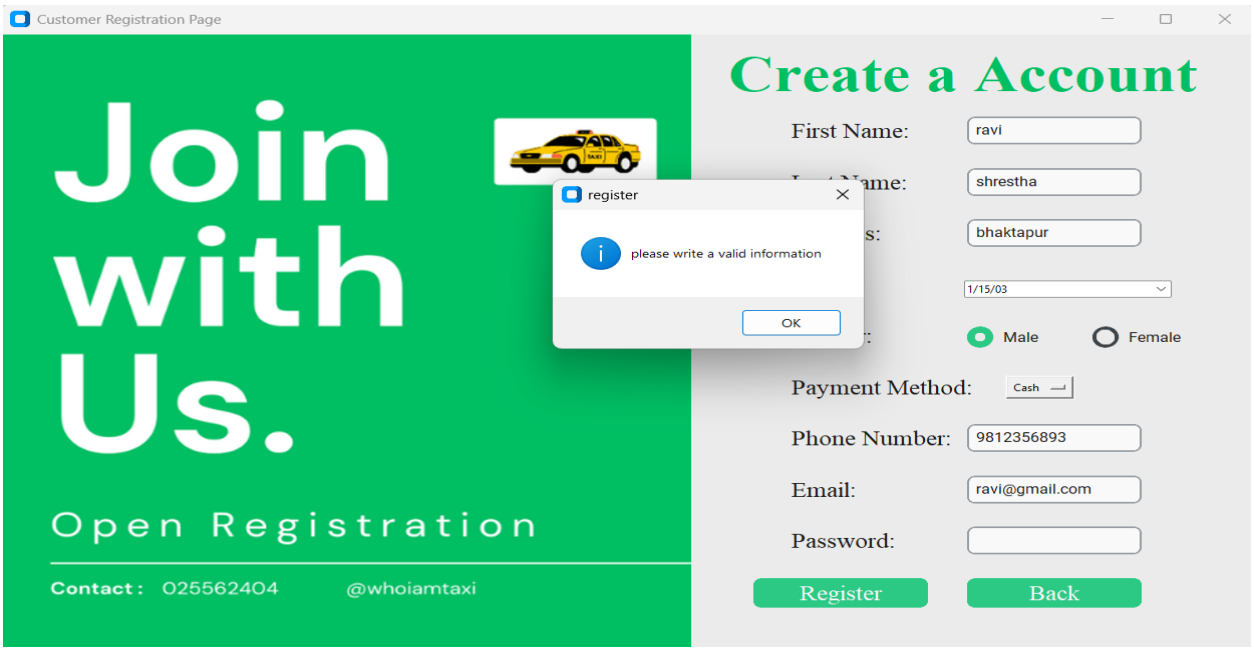


Figure 30: Trying not insert the password in registration form

Test Number	**Test date**	**Purpose of test**	Input data or action	Expected result	**Actual result**
1	7th jan 2023	Register an Account Sucessfully	Input all the field required and clicked register button.	A pop-up message that confirms the account has been registered.	A message that shows an account has be registered into system successfully.
2.	7th jan 2023	Register an Account Sucessfully	Try to enter a invalid email	A pop-up message that give error essage that email is invalid	A pop-up message that please enter the valid information
3.	7th jan 2023	Register an Account Sucessfully	Try to click the button without entering the password	A pop-up message that please enter the valid information	A pop-up message that please enter the valid information

Table 5: Testing table for Registration

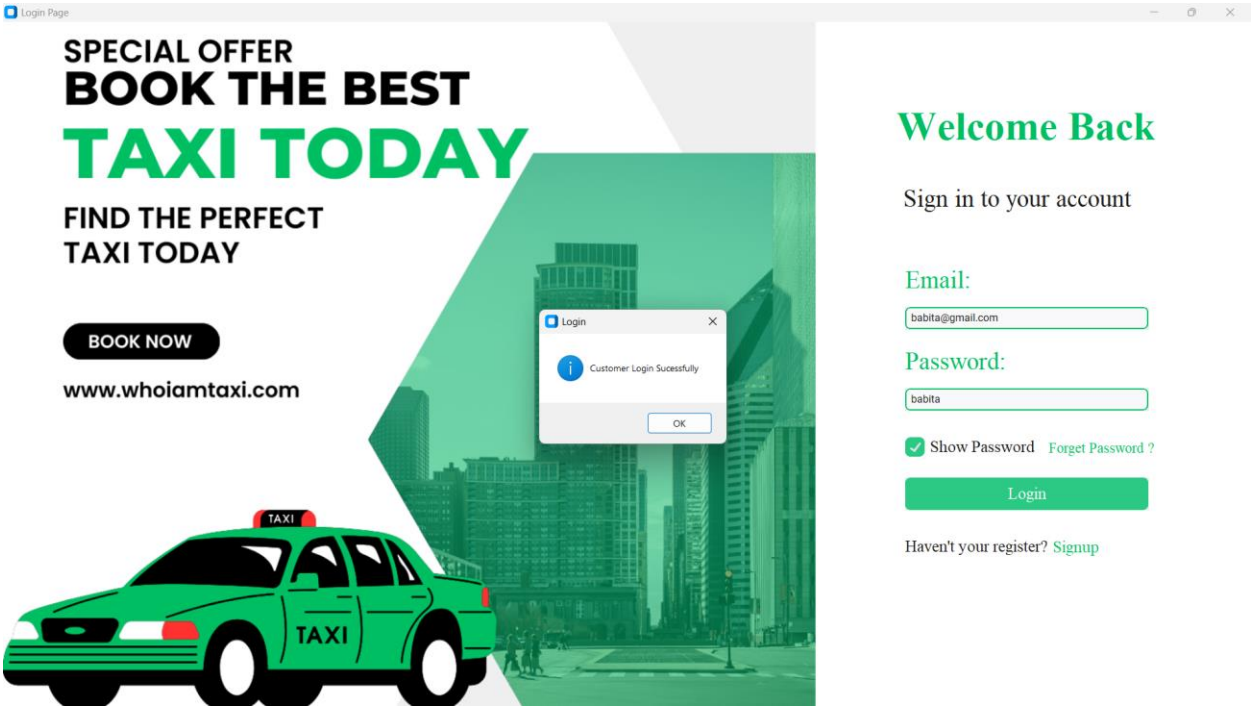


Figure 31: Try to login in the system

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
4.	7th jan 2023	Login Sucessful	Input all the field required an clicked login button	A pop-up message that confirm login sucessfully	A message that show a login sucessfully into the system.

Table 6: testing table for login

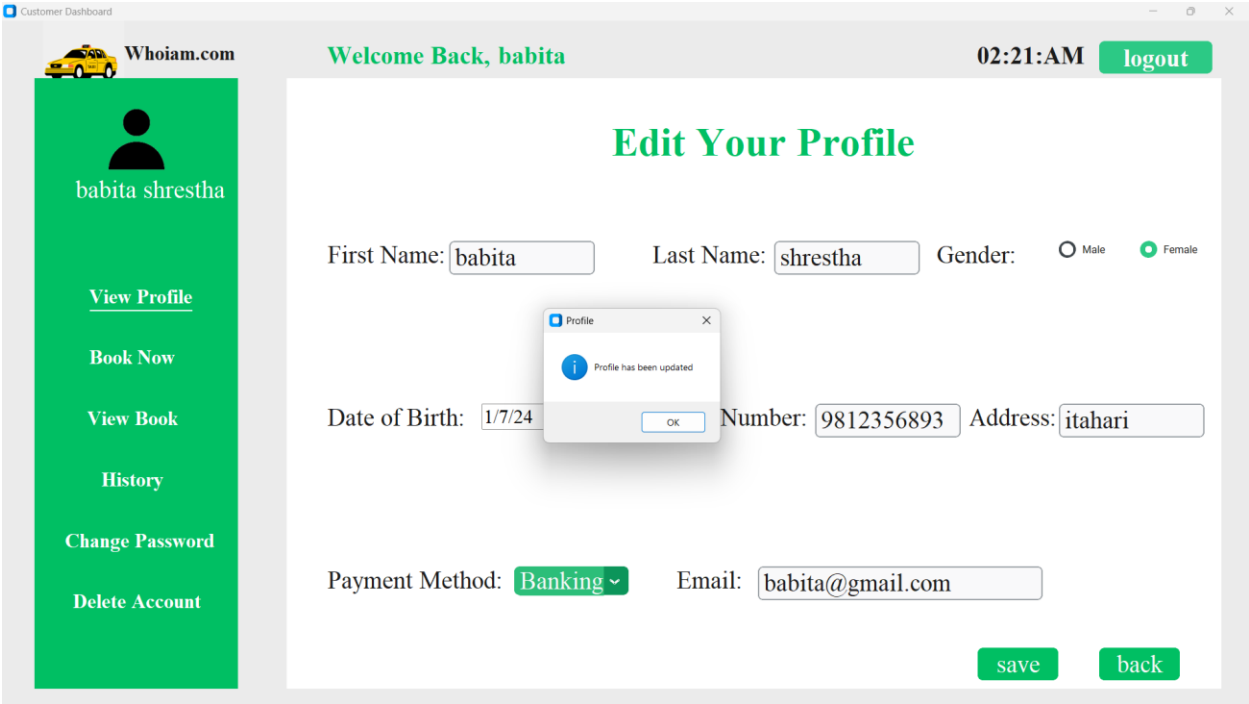


Figure 32: try to edit profile

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
5.	7th jan 2023	Profile Updated	Input all the edited field requied and clicked save button	A pop-up message that confirms profit has been updated	A message that show a Profile has been updated

Table 7: Testing table for Profile Updated

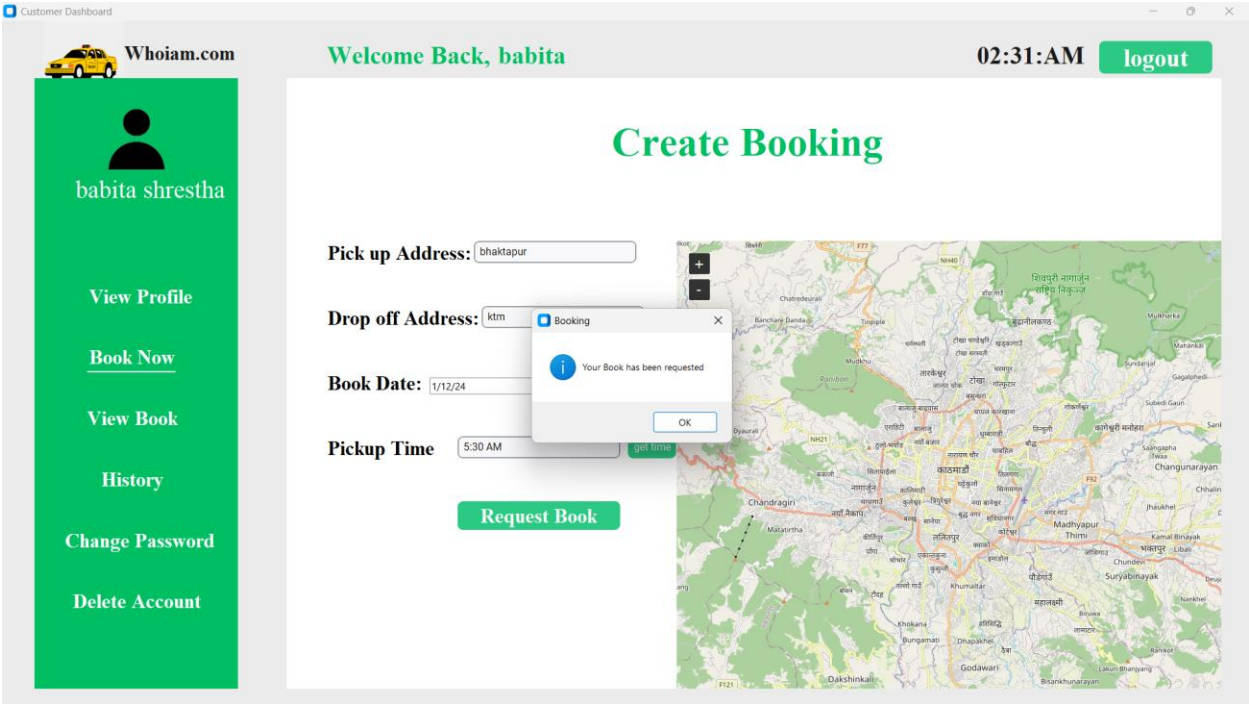


Figure 33: Try to request Book

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
6.	7th jan 2023	Book a trip successfully.	Input all the field required and clicked request booking button.	A pop-up message that confirms booking trip successfully.	A message that shows a customer successfully booked trip into the system .

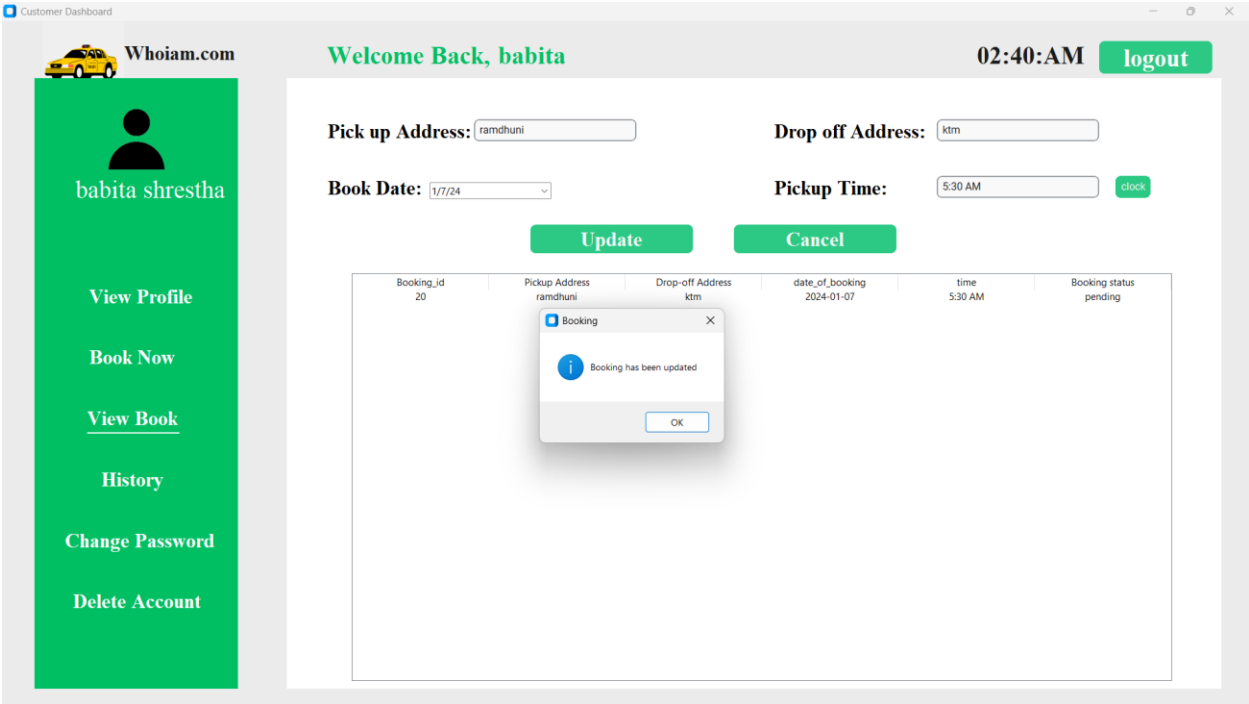


Figure 34: try to edit the booking

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
6.	7th jan 2023	Update the booking trip successfully	Input all the field required and clicked Update button	A pop-up message that confirms booking trip successfully.	A message that shows a customer successfully booked trip into the system .

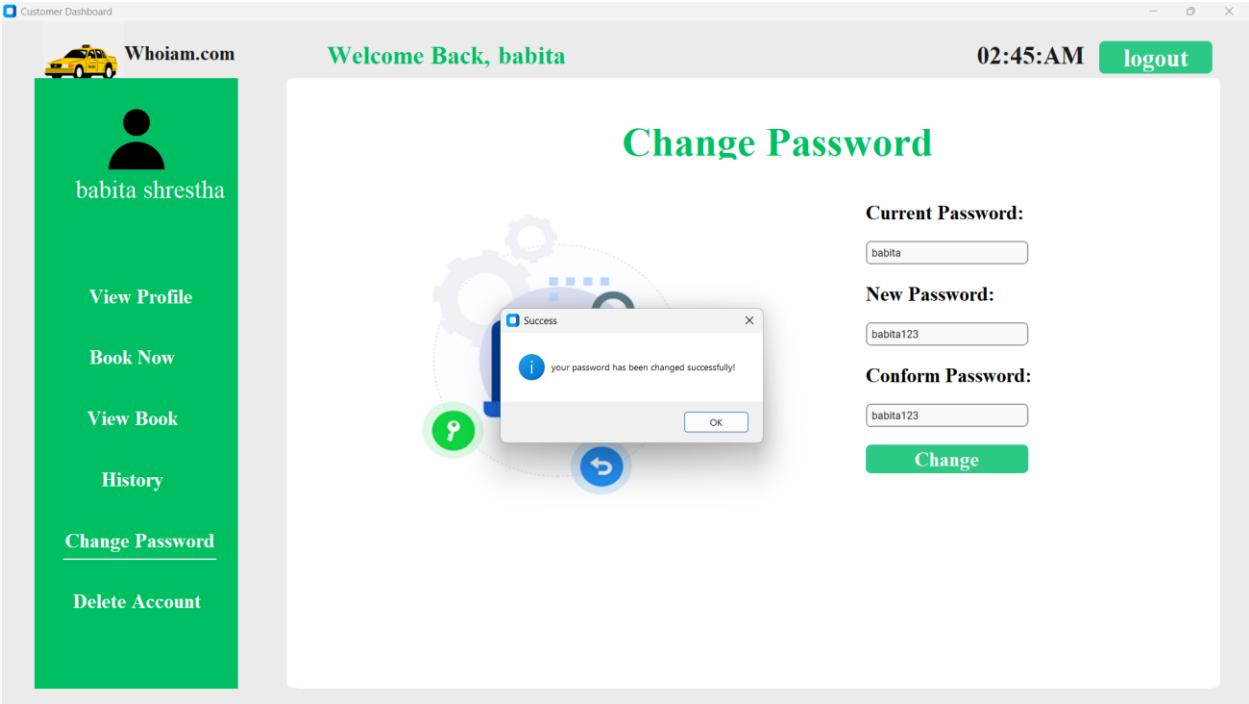


Figure 35: try to change the password

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
7.	7th jan 2023	Change the password successfully	Input all the field required and clicked Change button	A pop-up message that confirms the password has been change successfully.	A pop-up message that confirms the password has been change successfully.

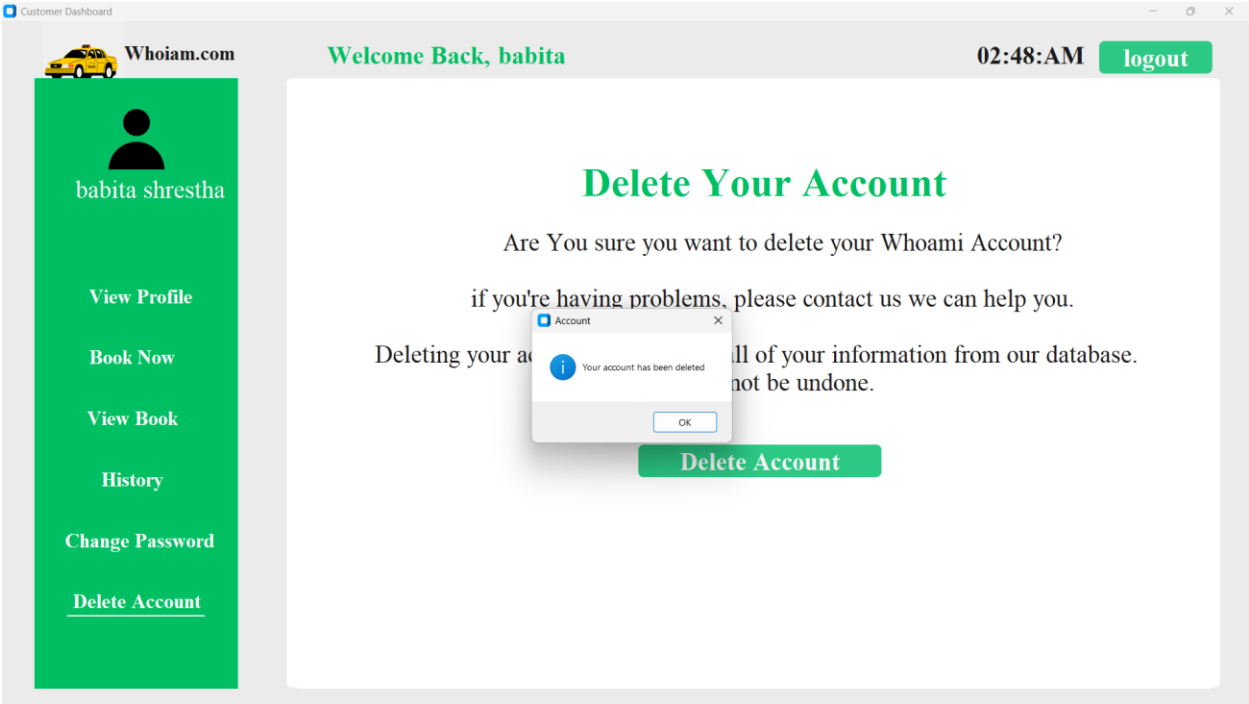


Figure 36: Try to delete the Account

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
8.	7th jan 2023	Delete the account successfully	Delete the account by clicking the delete account button	A pop-up message that confirms account has been delete successfully.	A pop-up message that confirms account has been delete successfully.

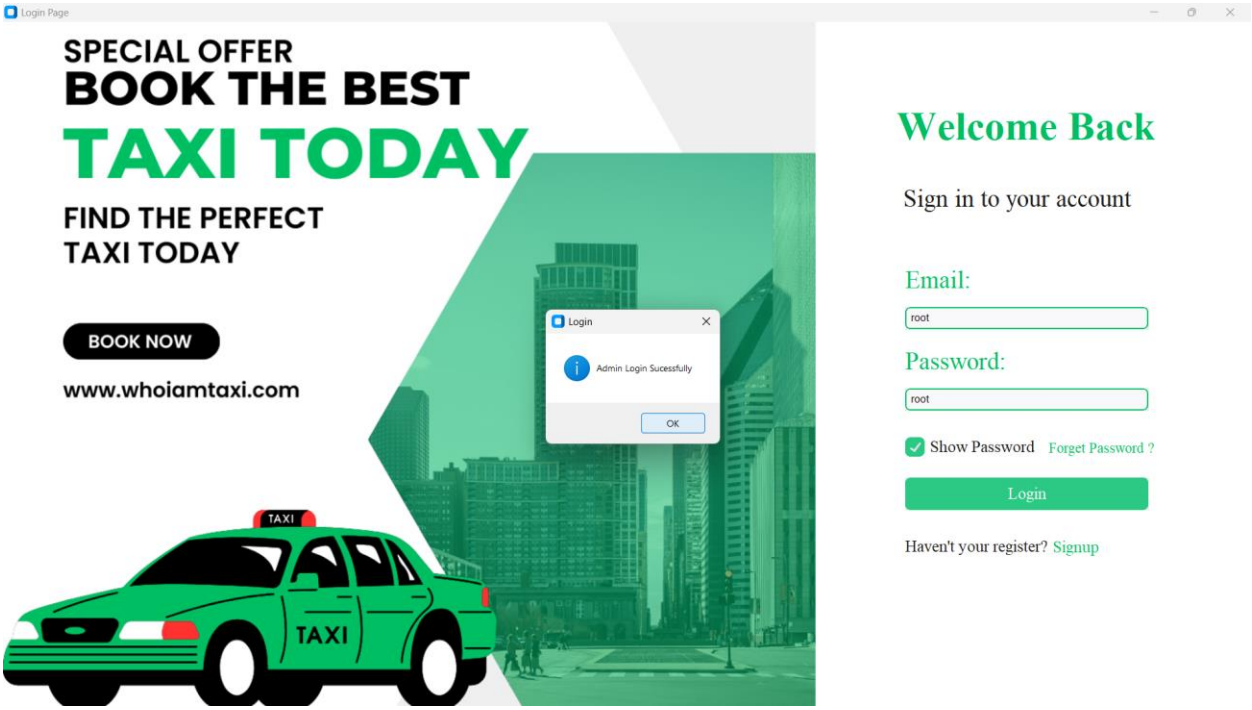


Figure 37: try to login by admin

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
9.	7th jan 2023	Admin Login Sucessful	Input all the field required an clicked Admin login button	A pop-up message that confirm Admin login sucessfully	A message that show a admin login sucessfully into the system.

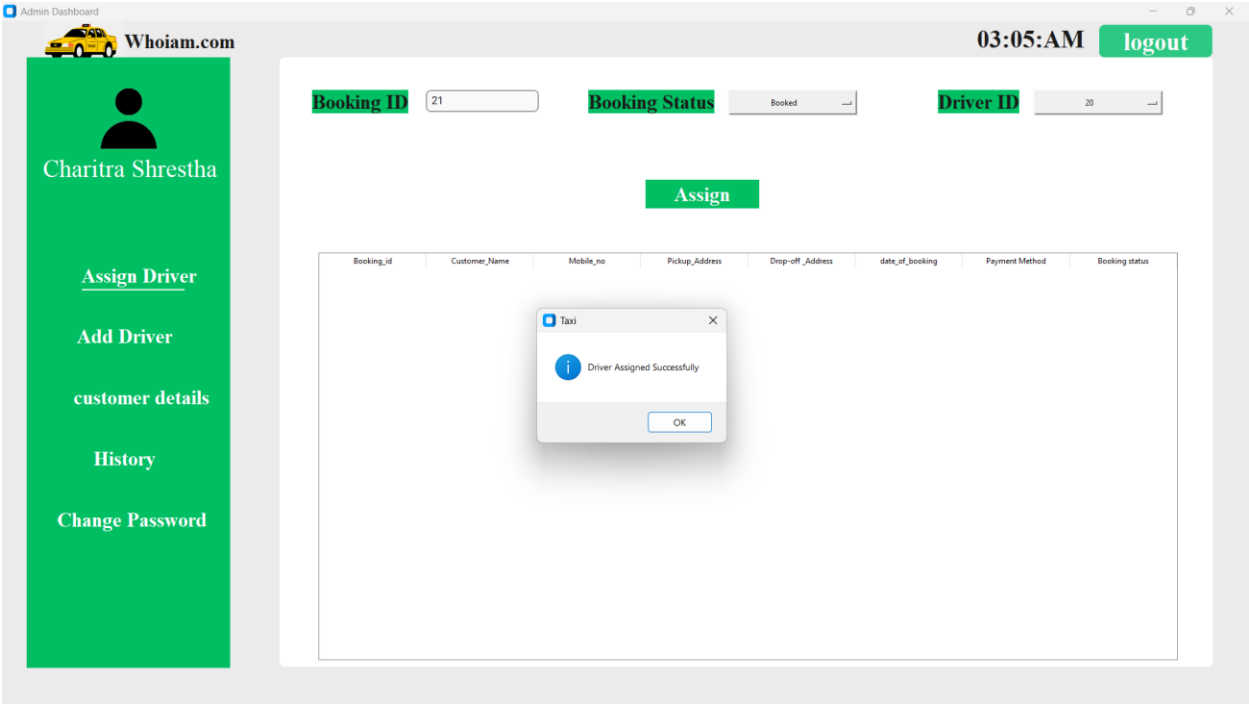


Figure 38: try to accpet or assigned booking by admin

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
10.	7th jan 2023	Admin assigning driver to trip	Input all the field required an clicked Assigned button	A pop-up message that confirm Driver assigned sucessfully	A message that show a driver assigned sucessfully

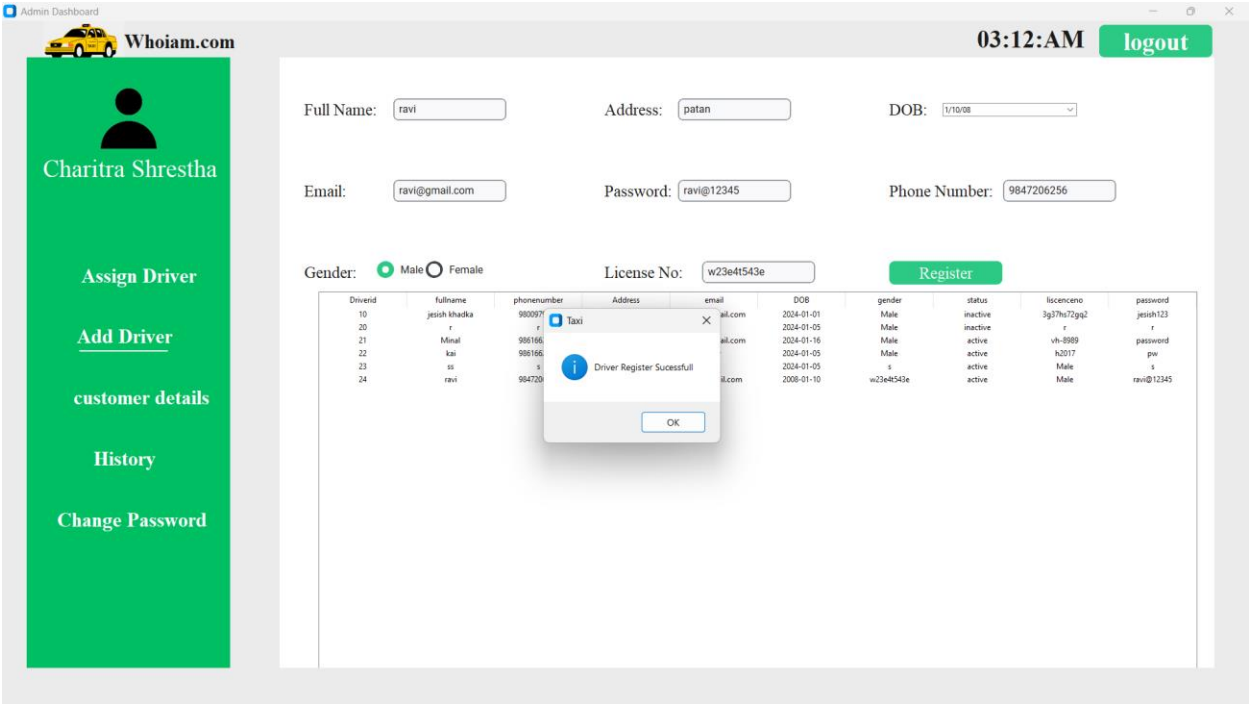


Figure 39: try to add new driver to the system

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
11.	7th jan 2023	Admin adding driver	Input all the field required an clicked Register button	A pop-up message that confirm Admin add the driver sucessfully	A message that show a driver register sucessfully

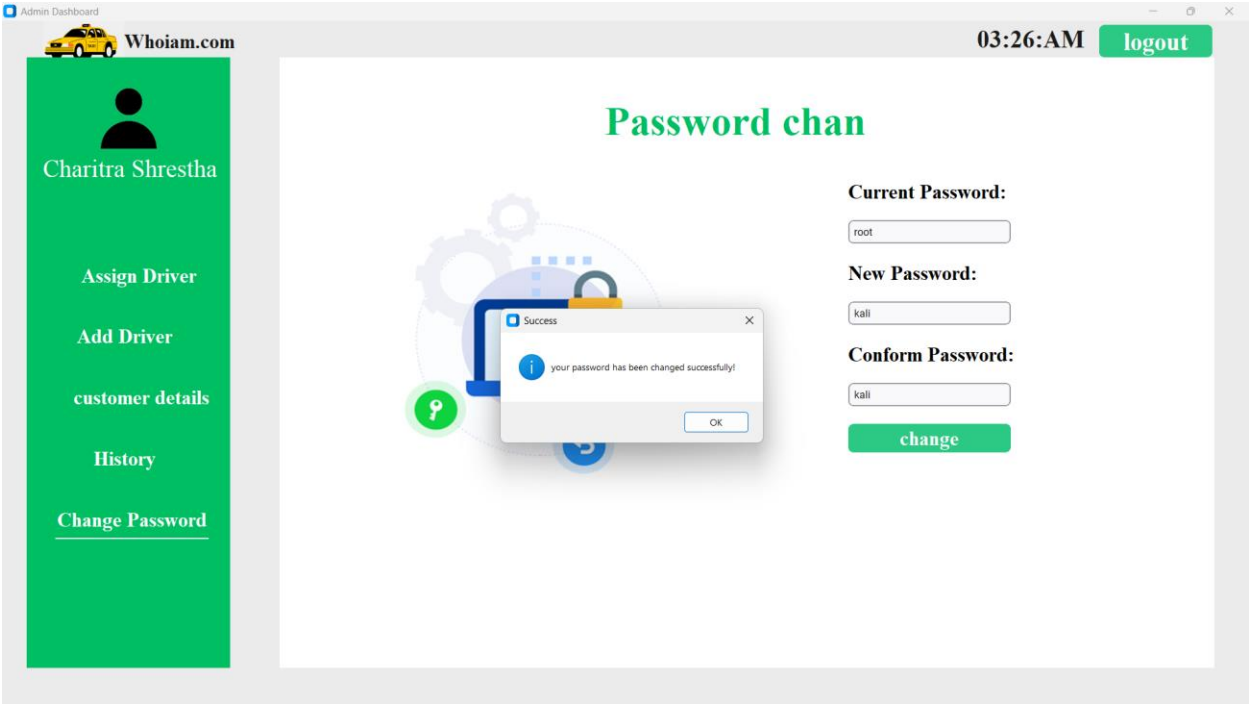


Figure 40: Try to Chnage the password

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
12.	7th jan 2023	Change the password of admin successfully	Input all the field required and clicked Change button	A pop-up message that confirms admin password has change sucessfully	A pop-up message that confirms the password has been change successfully.

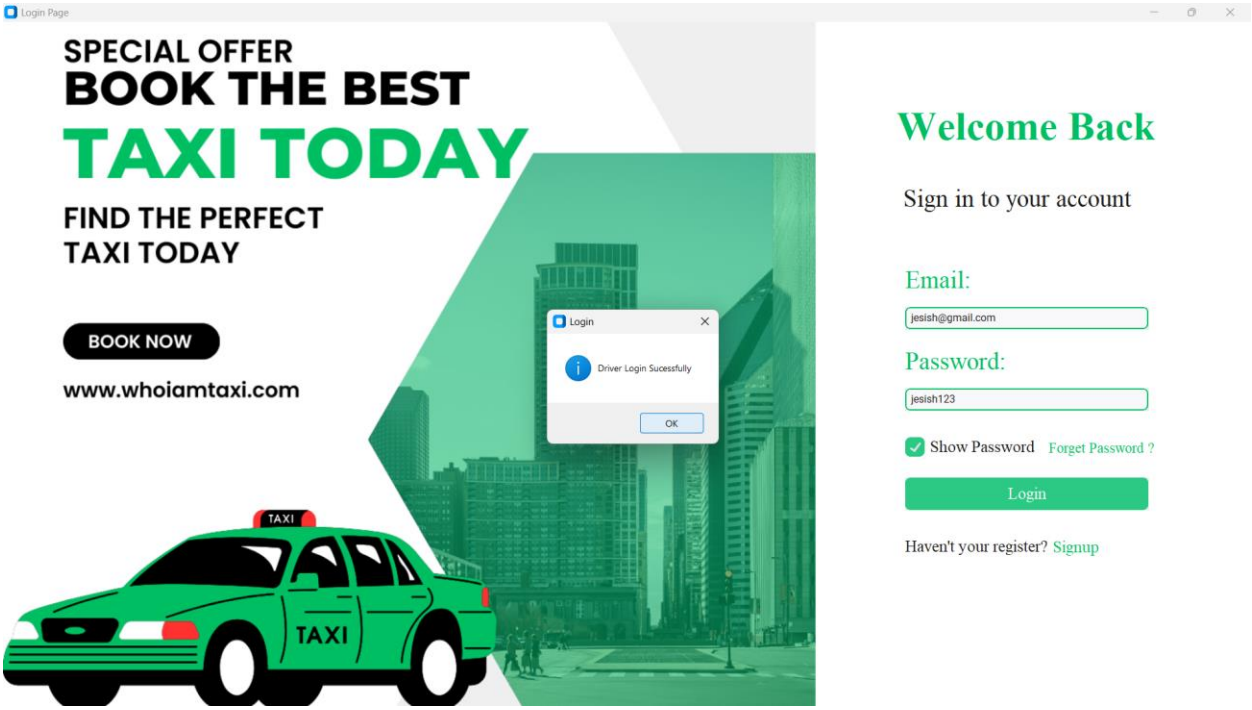


Figure 41: Try to Login by Driver

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
13.	7th jan 2023	Driver Login Sucessful	Input all the field required an clicked login button	A pop-up message that confirm Driver login sucessfully	A message that show a Driver login sucessfully into the system.

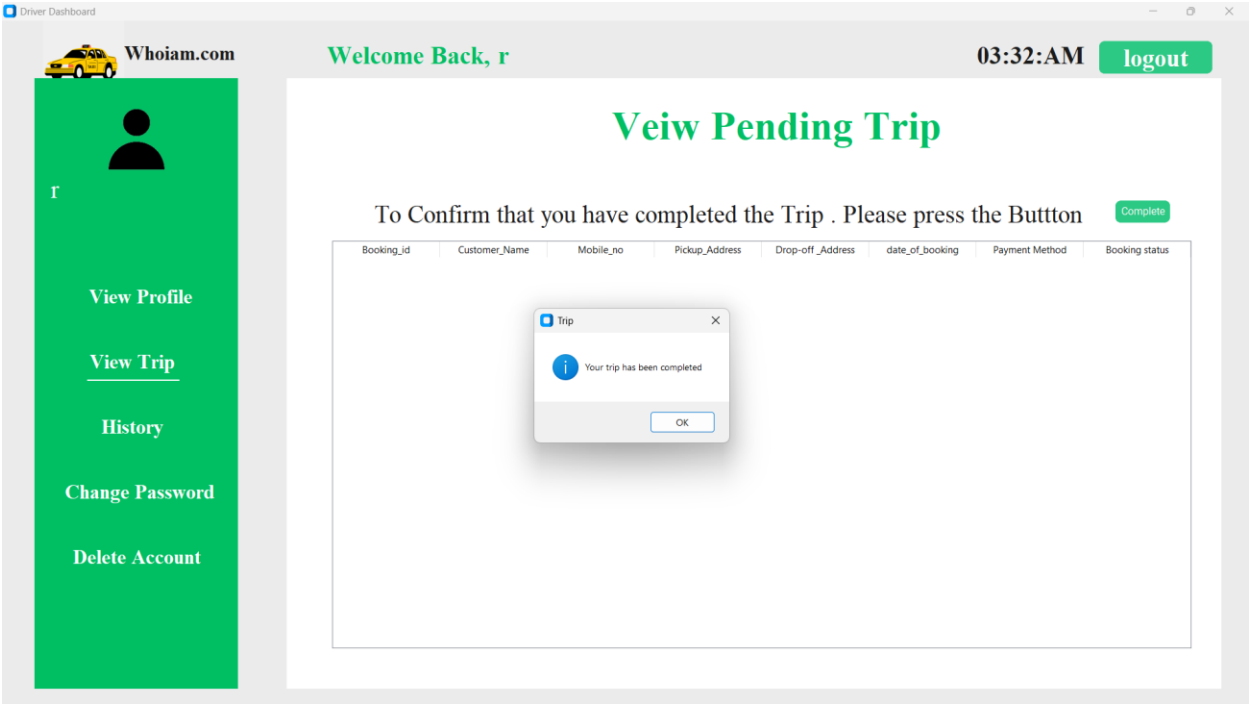


Figure 42: Try to complete the trip by driver

Test No	Test Date	Purpose of test	Input data or action	Expected result	Actual result
13.	7th jan 2023	Driver completed the trip sucessfully	Input all the field required and clicked trip completed button.	A pop-up message that confirms Driver completed trip successfully.	A message that shows a driver successfully completed the trip.

Discussion/Reflection/Critical Analysis

I knew how to program in Python from my undergraduate days, so this project was rather easy. But when I used the customtkinter library, there were some problems. I managed to resolve these issues with the assistance of the internet and my mentor. Despite not using customtkinter library, I used Python languages. From then on, I started working on these projects, getting to know Synatx gradually before starting to develop the application. The database needed to be created, and the tables and attributes needed to be defined. For the customer, driver, booking, and admin, I had to develop a table. Writing SQL queries wasn't any different for me because I've done it before; join queries, however, present a problem. I tried to tackle that problem for hours on end, but I was unsuccessful. My friend then assisted me in resolving the issue. I managed to complete the assignment on time. I had trouble connecting to the database, but I still adore them. On the other side, I enjoyed creating the GUI. I was able to gain more knowledge about Python programming languages through this project. I had a difficult time creating the user interface because I lacked a proper design from the start. To finish the entire assignment, I tried numerous ways and strategies for hours. I worked really hard and was really dedicated to finishing my tasks on time.

Conclusion

In summary, the project to develop a Python-based taxi booking system was successful in providing a user-friendly interface for effective taxi administration and booking. The system had important features like user registration, travel

booking and it fulfilled all criteria despite development hurdles. The project is successful because the graphical user interface makes the product more accessible to drivers and customers. This experience has shown how to use Python practically to create scalable and successful solutions for real-world problems, and it has given me important insights and abilities for my future software development attempts.

References

Alexander Borgida, M. A. C. &. A. H. F. L., 2009. Logical Database Design: from Conceptual to Logical Schema. *Encyclopedia of Database Systems.*

Contributor, T., n.d. *Application development and design.* [Online]
Available at: https://www.techtarget.com/searchapparchitecture/definition/class-diagram

APPENDIX

	customerid	firstname	lastname	email	DOB	Gender	phonenumber	address	Payment_Method	password
Edit Copy Delete	53	ravi	shrestha	chari@gmail.com	2002-01-10	Male	9812356893	ktm	Cash	chari123
Edit Copy Delete	54	minal	pariyar	minal@gmail.com	2004-01-15	Male	9800879845	kathmandu	Cash	minal@123
Edit Copy Delete	55	pratik	tamang	pratik@gmail.com	2003-01-02	Male	9834206292	ilam	Banking	pratik@123
Edit Copy Delete	56	sparsha	tandukar	sparsha@gmail.com	2099-01-07	Male	98333622926	patan	Esewa	sparsha@123
Edit Copy Delete	57	takri	byanjankar	takri@gmail.com	2000-01-07	Male	987763229362	patan	Banking	takri@123

Check all With selected: Edit Copy Delete Export

	bookingid	pickup_address	dropoff_address	date	time	status	customerid	driverid
Edit Copy Delete	21	KTM	ktm	2024-01-15	1:0 AM	completed	53	10
Edit Copy Delete	22	pokhara	ktm	2024-01-08	7:0 AM	pending	54	NULL
Edit Copy Delete	23	patan	pokhara	2024-01-15	5:30 AM	booked	56	24
Edit Copy Delete	24	hell	heaven	2024-01-23	12:0 AM	pending	57	NULL

Check all With selected: Edit Copy Delete Export

	adminid	first name	last name	email	password
Edit Copy Delete	1	Charitra	Shrestha	root	kali

Check all With selected: Edit Copy Delete Export

	driverid	fullname	phonenumber	Address	email	DOB	gender	status	liscenceno	password
Edit Copy Delete	10	jesish khadka	9800979845	jhumka	jesish@gmail.com	2024-01-01	Male	inactive	3g37hs72gq2	jesish123
Edit Copy Delete	21	Minal	9861662986	dallu aawas1	minal@gmail.com	2024-01-16	Male	active	vh-8989	password
Edit Copy Delete	22	kai	9861662986	kai	loev	2024-01-05	Male	active	h2017	pw
Edit Copy Delete	24	ravi	9847206256	patan	ravi@gmail.com	2008-01-10	w23e4t543e	active	Male	ravi@12345

LoginView.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from tkinter import messagebox
from PIL import ImageTk, Image
from Controller import CustomerController as cusdb , DriverController as drivedb ,
AdminController as admindb
from Model import LoginModel as loginmd
import tkinter as tk
import customtkinter as CT
import Veiw.VerifyEmailView as VerifyEmailView , Veiw.RegistrationView as
RegistrationView , GobalVariable, CustomerDashboard, DriverDashboardBoard
,AdminDashboard

# create the class
class LoginPage():
    # create the constructor
    def __init__(self,master):
```

```
        self.master = master
        self.master.title('Login Page')#titlename
        self.master.config(background="white")#background
        CT.set_default_color_theme("green")

        # decleare the font for text
        my_font = CT.CTkFont(family="Times", size=50,weight="bold")

        #create Background Image
        self.background_image = CT.CTkImage(Image.open('D:\\Code\\Python\\python
project\\TaxBookingSystem\\image\\Home.png'), size=(1000,850))
        self.img = CT.CTkLabel(self.master,image=self.background_image,
text="").place(x=0,y=0)

        # Create CTkLabels
        self.welcome = CT.CTkLabel(self.master,text="Welcome Back",
font=my_font,text_color="#00BF63", fg_color="white")
        my_font.configure(family="TImes")
        self.welcome.place(x=1100,y=100)

        self.txt = CT.CTkLabel(self.master,text=" Sign in to your account", font=
CT.CTkFont(family="Times", size=30),fg_color="white")
        self.txt.place(x=1100,y=200)

        self.CTkLabel_username = CT.CTkLabel(self.master, text="Email:", font=
CT.CTkFont(family="Times",size=30),text_color="#00BF63",fg_color="white").place(x=111
0,y=300)
        self.CTkLabel_password = CT.CTkLabel(self.master,
text="Password:",font=CT.CTkFont(family="Times",size=30),text_color="#00BF63",
fg_color="white").place(x=1110,y=400)

        self.register = CT.CTkLabel(self.master,text="Haven't your register?",
font=CT.CTkFont(family="Times",size=20), bg_color="white").place(x=1110,y=630)

        # Create CTkentry widgets for username and password
        self.CTkentry_username = CT.CTkEntry(self.master,
textvariable=CT.StringVar(),width=300, border_color="#00BF63")
        self.CTkentry_username.place(x=1110,y=350)
        self.show = CT.StringVar()
        self.CTkentry_password = CT.CTkEntry(self.master,
textvariable=  self.show,width=300 , show='*',border_color="#00BF63", border_width=2)
        self.CTkentry_password.place(x=1110,y=450) # Show * for passwords
        self.sh = CT.IntVar(value=0)
        def my_show():
            if(  self.sh.get()==1):
```

```
                self.CTkentry_password.configure(show='') # display the chars
            else:
                self.CTkentry_password.configure(show='*')# hide the chars using mask
        # create check box
        self.remember = CT.CTkCheckBox(self.master,text="Show Password",
bg_color="white", font=CT.CTkFont(family="Times",
size=20),variable=self.sh,onvalue=1,offvalue=0,command=my_show).place(x=1110,y=510)

        # Create login button
        self.button_login = CT.CTkButton(self.master, text="Login",
command=self.on_login, width=300, height=40, text_color="white",
font=CT.CTkFont(family="Times", size=20),hover=False).place(x=1110,y=560)
        self.button_SignUp = CT.CTkButton(self.master, text="Signup",
command=self.on_SignUp,fg_color='white', text_color='#00BF63',
bg_color="white",hover=False,
width=30,font=CT.CTkFont(family="Times",size=20)).place(x=1285,y=630)
        self.button_Forget = CT.CTkButton(self.master, text="Forget Password ?",
command=self.on_forget, fg_color="white", bg_color="white", text_color="#00BF63",
font=CT.CTkFont(family="Times", size=18), hover=False).place(x=1280,y=510)

        # create the signup function
    def on_SignUp(self):
        # self.master.destroy()
        reg = CT.CTkToplevel()
        RegistrationView.RegistrationPage(reg)
        reg.mainloop()

    # create the forget function
    def on_forget(self):
        reg = CT.CTkToplevel()
        VerifyEmailView.VerifyEmail(reg)
        reg.mainloop()

    # create the login function
    def on_login(self):
        username = self.CTkentry_username.get()
        password = self.CTkentry_password.get()
        login = loginmd.Login(_email=username, _password=password)

        customer = cusdb.CustomerDatabase()
        cus = customer.CustomerLogin(login)
        driver  = drivedb.DriverDatabase()
        dri = driver.DriverLogin(login)
        admin = admindb.AdminDatabase()
        ad = admin.AdminLogin(login)
```

```
        if cus!=None:
            messagebox.showinfo('Login','Customer Login
Sucessfully',parent=self.master)
            GobalVariable.Customer = cus
            self.master.destroy()
            reg = CT.CTkToplevel()
            CustomerDashboard.Dashboard(reg)
            reg.after(0,lambda:reg.state('zoomed'))
            reg.mainloop()

        elif dri!=None:
            messagebox.showinfo('Login','Driver Login
Sucessfully',parent=self.master)
            GobalVariable.Driver = dri
            self.master.destroy()
            reg = CT.CTkToplevel()
            DriverDashboardBoard.Dashboard(reg)
            reg.after(0,lambda:reg.state('zoomed'))
            reg.mainloop()

        elif ad!=None:
            messagebox.showinfo('Login','Admin Login Sucessfully',parent=self.master)
            GobalVariable.Admin  = ad
            self.master.destroy()
            reg = CT.CTkToplevel()
            AdminDashboard.Dashboard(reg)
            reg.after(0,lambda:reg.state('zoomed'))
            reg.mainloop()
        elif username=='' or password=='':
            messagebox.showwarning('Login','please write both email and
password',parent=self.master)
        elif username=='' and password =='':
            messagebox.showwarning('Login','please write email and
password',parent=self.master)
        else:
            messagebox.showerror('Login','Incorrect email and
password',parent=self.master)
#create the name function
if __name__ == "__main__":

    app = CT.CTk()
    app.after(0,lambda:app.state('zoomed'))
    LoginPage(app)
    # CT.deactivate_automatic_dpi_awareness()
```

```
    app.mainloop()
```

NewPasswordView.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from Controller import CustomerController
from Model import LoginModel
from tkinter import messagebox
import customtkinter as CT
import GobalVariable
import tkinter as tk
import Veiw.LoginView as LoginView
class PasswordCreation():
    def __init__(self, master):
        self.master = master
        self.master.title("Password Change")
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        width = 500
        height =350
        x = (screen_width - width) // 2
        y = (screen_height - height) // 2
        self.master.geometry(f"{width}x{height}+{x}+{y}")
        # self.master.geometry('500x350')
        CT.set_default_color_theme("green")
        self.master.attributes('-topmost',True)
        self.create_widget()
    def create_widget(self):
        # Label and Entry for Create New Password
        self.topic = CT.CTkLabel(self.master, text='Change
Password',font=CT.CTkFont(family='Times', size=30, weight="bold")).place(x=150, y=30)

        self.requied = self.topic = CT.CTkLabel(self.master, text='''Password must
contain one lowercase lettter, one number,
        and be atleast 6 character long.''').place(x=100, y=70)
        self.create_password_label = CT.CTkLabel(self.master, text="Create New
Password:")
        self.create_password_label.place(x=150,y=110)
```

```
        self.create_password_entry = CT.CTkEntry(self.master,
show="*",textvariable=CT.StringVar(), width=200)  # Entry widget with '*' to hide the
password
        self.create_password_entry.place(x=150,y=140)

        # Label and Entry for Re-type New Password
        self.retype_password_label = CT.CTkLabel(self.master, text="Re-type New
Password:")
        self.retype_password_label.place(x=150, y=180)

        self.retype_password_entry = CT.CTkEntry(self.master,
show="*",textvariable=CT.StringVar(), width=200)
        self.retype_password_entry.place(x=150, y=210)

        # Submit Button
        self.submit_button = CT.CTkButton(self.master, text="Submit",
command=self.create_password, width=200)
        self.submit_button.place(x=150, y=260)

    def create_password(self):
        new_password = self.create_password_entry.get()
        retype_password = self.retype_password_entry.get()

        if new_password == retype_password:
            try:
                gmail = GobalVariable.email
                send = LoginModel.Login(_email=gmail,_password=new_password)

                var = CustomerController.CustomerDatabase()
                var._ChangePassword(send)
                if var:
                    messagebox.showinfo("Password Created", "Your password has been
changed",parent=self.master)
                    self.master.destroy()
                    # LoginVeiw.LoginPage()

            except Exception as e:
                print(e)

        else:
            messagebox.showerror("Password Error", "Passwords do not match. Please
try again.",parent=self.master)

if __name__ == "__main__":
```

```
    app = CT.CTk()
    PasswordCreation(app)
    app.mainloop()
```

VerifyEmailView.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from tkinter import messagebox
from Model.VerifyModel import Email
from Controller.CustomerController import CustomerDatabase
import tkinter as tk
import GobalVariable
import customtkinter as CT
import Veiw.NewPasswordView as NewPasswordView

class VerifyEmail():
    verifyemail = str
    def __init__(self, master):

        self.master = master
        self.master.title("Forget Password")
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        width = 500
        height =300
        x = (screen_width - width) // 2
        y = (screen_height - height) // 2
        self.master.geometry(f"{width}x{height}+{x}+{y}")
        CT.set_default_color_theme("green")
        self.master.config(background="white")
        self.master.attributes('-topmost',True)
        # Label and Entry for Create New Password

        self.topic = CT.CTkLabel(self.master,text='Forget
Password?',bg_color="white", font=CT.CTkFont(family='Times', size=30, weight="bold"))
        self.topic.place(x=150, y=20)

        self.worries = CT.CTkLabel(self.master,text="No worries, we'll rest your
password please enter the email for verfication.",bg_color="white").place(x=50,y=60)

        self.emailaddress_label = CT.CTkLabel(self.master,
        text="Email Address:",bg_color="white")
```

```
        self.emailaddress_label.place(x=150, y=100)

        self.emailaddress_entry =
CT.CTkEntry(self.master,textvariable=CT.StringVar(), width=200)  # Entry widget with
'*' to hide the password
        self.emailaddress_entry.place(x=150, y=130)

        # Submit Button
        self.submit_button = CT.CTkButton(self.master, text="Reset Password",
command=self.emailaddress, width=200)
        self.submit_button.place(x=150, y=170)
        self.back_btn = CT.CTkButton(self.master, text="<- Back to log in
",fg_color="white",text_color="black",bg_color="white",hover=False,
command=self.back, width=200)
        self.back_btn.place(x=150, y=210)


    def emailaddress(self):
        verifyemail = self.emailaddress_entry.get()
        verify = Email(_email=verifyemail)
        emaill = CustomerDatabase()
        check = emaill._isValidEmail(verify)

        if check==True:
            GobalVariable.email = self.emailaddress_entry.get()
            messagebox.showinfo("verify", "Email has been verify
successful!",parent=self.master)
            self.master.destroy()
            reg = CT.CTkToplevel()
            NewPasswordView.PasswordCreation(reg)
            reg.mainloop()

        else:
            messagebox.showerror("Error", "Email do not match. Please try
again.",parent=self.master)
    def back(self):
        self.master.destroy()

if __name__ == "__main__":
    app = CT.CTk()
    VerifyEmail(app)
    app.mainloop()
```

CustomerDashboard.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from PIL import Image
from tkinter import ttk
from time import strftime
from tkcalendar import DateEntry
from Controller import CustomerController
from Controller.DataBaseConnection import Database
from tkinter import messagebox
import LoginView
import customtkinter as Ct
import tkinter as tkk
import tkintermapview

import GobalVariable


class Dashboard():
    def __init__(self,master):
        self.master = master
        self.master.title('Customer Dashboard')
        Ct.set_default_color_theme("green")
        self.master.attributes('-topmost',True)
        self.company=Ct.CTkLabel(self.master,text="Whoiam.com",font=Ct.CTkFont(family
="Times",size=25, weight='bold'))
        self.company.place(x=150,y=25)
        self.cmpic = Ct.CTkImage(Image.open('D:\\Code\\Python\\python
project\\TaxBookingSystem\\image\\Green White Simple Open Registration Facebook Post
(3).png'), size=(100,100))
        self.cmimg = Ct.CTkLabel(self.master,image=self.cmpic,
text="").place(x=50,y=-5)

        self.welcome=Ct.CTkLabel(self.master,text="Welcome
Back,",font=Ct.CTkFont(family="Times",size=30, weight='bold'),text_color='#00BF63')
        self.welcome.place(x=400,y=25)
        self.user=Ct.CTkLabel(self.master,text="",font=Ct.CTkFont(family="Times",size
=30, weight='bold'),text_color='#00BF63')
        self.user.place(x=610,y=25)
        self.user.configure(text=GobalVariable.Customer[1])

        self.lbl = Ct.CTkLabel(self.master,font=Ct.CTkFont(family="Times",size=30,
weight='bold'))
        self.lbl.place(x=1200,y=25)
        self.lbl2 =
Ct.CTkButton(self.master,text="logout",font=Ct.CTkFont(family="Times",size=30,
weight='bold'),command=self.logout)
```

```
        self.lbl2.place(x=1350,y=25)
        self.time()
        self.connection = Database.Connect()
        self.options_frame = Ct.CTkFrame(self.master,
fg_color='#00BF63',bg_color='#00BF63')

        self.profil = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\images-removebg-preview.png'), size=(150,150))
        self.img = Ct.CTkLabel(self.options_frame,image=self.profil,
text="").place(x=50,y=0)

        self.name = Ct.CTkLabel(self.options_frame,text="",
font=Ct.CTkFont(family='Times',size=30), text_color='white')
        self.name.place(x=50,y=120)
        self.name.configure(text=GobalVariable.Customer[1] + ' ' +
GobalVariable.Customer[2])

        self.dash = Ct.CTkButton(self.options_frame,text="View Profile",
fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False,command=lambda: self.indicate(self.proview, self.profile_page))
        self.dash.place(x=60,y=250)

        self.proview = Ct.CTkFrame(self.options_frame,
fg_color='white',width=127,height=3)
        self.proview.place(x=68,y=285)

        self.book = Ct.CTkButton(self.options_frame, text='Book
Now',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda: self.indicate(self.book_indicate,
self.book_page))
        self.book.place(x=50,y=325)

        self.book_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=110,height=3)
        self.book_indicate.place(x=65,y=360)

        self.view = Ct.CTkButton(self.options_frame, text='View
Book',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda: self.indicate(self.view_indicate,self.view_page))
        self.view.place(x=50, y=400)
```

```
        self.view_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=115,height=3)
        self.view_indicate.place(x=65,y=435)

        self.history = Ct.CTkButton(self.options_frame,
text='History',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.history_indicate,self.history_page))
        self.history.place(x=50, y=475)

        self.history_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=80,height=3)
        self.history_indicate.place(x=80,y=510)

        self.change_password = Ct.CTkButton(self.options_frame, text='Change
Password',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.change_indicate,self.change_page))
        self.change_password.place(x=30, y=550)

        self.change_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=190,height=3)
        self.change_indicate.place(x=35,y=590)

        self.delete = Ct.CTkButton(self.options_frame, text='Delete
Account',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.delete_indicate,self.delete_page))
        self.delete.place(x=40, y=625)

        self.delete_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=170,height=3)
        self.delete_indicate.place(x=40,y=660)

        self.options_frame.place(x=40,y=70)
        self.options_frame.pack_propagate(False)
        self.options_frame.configure(width=250,height=750)

        self.main_frame = Ct.CTkFrame(self.master, width=1150, height=750)
        self.main_frame.configure(fg_color='white')
        self.profile_lbl=Ct.CTkLabel(self.main_frame,text="Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
```

```
        self.profile_lbl.place(x=400,y=50)
        self.copy(self.main_frame)
        self.update_btn =
Ct.CTkButton(self.main_frame,text="Edit",command=self.dulicate,font=
Ct.CTkFont(family="Times", size=30),width=200,fg_color='#00BF63')
        self.update_btn.place(x=850,y=700)
        self.main_frame.place(x=350,y=70)
        self.main_frame.pack_propagate(False)
        # self.main_frame.configure(height=750, width=1150)

    def profile_page(self):
        ''' The gui for the view profile frame, it shows the details of the customer
who has login '''

        self.home_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.home_frame.configure(fg_color='white')
        Ct.set_default_color_theme("green")
        self.profile_lbl=Ct.CTkLabel(self.home_frame,text="Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.profile_lbl.place(x=400,y=50)

        self.copy(self.main_frame)
        self.update_btn =
Ct.CTkButton(self.home_frame,text="Edit",command=self.dulicate,font=
Ct.CTkFont(family="Times", size=30),width=200,fg_color='#00BF63')
        self.update_btn.place(x=850,y=700)
        self.home_frame.pack(side=Ct.LEFT)

        self.home_frame.pack(side=Ct.RIGHT)



    def book_page(self):
        self.book_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.booklb = Ct.CTkLabel(self.book_frame, text='Create
Booking',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        self.book_frame.configure(fg_color= 'white')
        Ct.set_default_color_theme("green")

        map_widget = tkintermapview.TkinterMapView(self.book_frame, width=850,
height=700, corner_radius=0)
        map_widget.set_address("kathmandu, nepal")
        map_widget.place(x=600,y=250)
        self.pick=Ct.CTkLabel(self.book_frame, text='Pick up
Address:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
```

```
        self.pick.place(x=50,y=200)
        self.pick_entry=Ct.CTkEntry(self.book_frame,width=200)
        self.pick_entry.place(x=230,y=200)

        self.drop=Ct.CTkLabel(self.book_frame, text='Drop off
Address:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.drop.place(x=50,y=280)
        self.drop_entry=Ct.CTkEntry(self.book_frame,width=200)
        self.drop_entry.place(x=240,y=280)

        self.date=Ct.CTkLabel(self.book_frame, text='Book
Date:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.date.place(x=50,y=360)
        self.date_entry=DateEntry(self.book_frame,width=40)
        self.date_entry.place(x=220,y=460)

        self.time_pick=Ct.CTkLabel(self.book_frame, text='Pickup
Time',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.time_pick.place(x=50,y=440)

        self.custime = Ct.StringVar()

        self.time_pick_entry=Ct.CTkEntry(self.book_frame,width=200,textvariable=self.
custime)
        self.time_pick_entry.place(x=210,y=440)


        self.time_entry=Ct.CTkButton(self.book_frame, text="get
time",width=10,command=self.picktime)
        self.time_entry.place(x=420,y=440)

        self.book_frame.pack(side=Ct.LEFT)

        self.request = Ct.CTkButton(self.book_frame, text="Request
Book",width=200,font=Ct.CTkFont(family="Times",size=25,
weight='bold'),command=self.book_request)
        self.request.place(x=210,y= 520)

    def picktime(self):
        import tkinter as tk
        from tktimepicker import AnalogPicker, AnalogThemes

        self.root = tk.Tk()
        self.root.attributes('-topmost',True)
        self.time_picker = AnalogPicker(self.root)
```

```
        self.time_picker.pack(expand=True, fill="both")

        theme = AnalogThemes(self.time_picker)
        theme.setDracula()
        self.request = Ct.CTkButton(self.root,
text="set",width=10,font=Ct.CTkFont(family="Times",size=20,
weight='bold'),command=lambda: self.settime(self.time_picker.time()))
        self.request.place(x=250,y= 200)
        self.root.mainloop()

    def settime(self,time):
            self.custime.set("{}:{} {}".format(*time))

            self.root.destroy()

    def view_page(self):
        self.view_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.view_frame.configure(fg_color= 'white')

        self.id = Ct.StringVar()
        self.Bookid = Ct.CTkEntry(self.view_frame)
        self.Bookid.place_forget()
        self.pick=Ct.CTkLabel(self.view_frame, text='Pick up
Address:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.pick.place(x=50,y=50)
        self.pick_entry=Ct.CTkEntry(self.view_frame,width=200)
        self.pick_entry.place(x=230,y=50)

        self.drop=Ct.CTkLabel(self.view_frame, text='Drop off
Address:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.drop.place(x=600,y=50)
        self.drop_entry=Ct.CTkEntry(self.view_frame,width=200)
        self.drop_entry.place(x=800,y=50)

        self.date=Ct.CTkLabel(self.view_frame, text='Book
Date:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.date.place(x=50,y=120)
        self.date_entry=DateEntry(self.view_frame,width=20,height=50)
        self.date_entry.place(x=220,y=160)

        self.time_pick=Ct.CTkLabel(self.view_frame, text='Pickup
Time:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.time_pick.place(x=600,y=120)

        self.custime = Ct.StringVar()
```

```
        self.time_pick_entry=Ct.CTkEntry(self.view_frame,width=200,textvariable=self.
custime)
        self.time_pick_entry.place(x=800,y=120)


        self.time_entry=Ct.CTkButton(self.view_frame,
text="clock",width=10,command=self.picktime)
        self.time_entry.place(x=1020,y=120)

        self.update=Ct.CTkButton(self.view_frame,
text="Update",width=200,font=Ct.CTkFont(family="Times",size=25,
weight='bold'),command=self.update_book)
        self.update.place(x=300,y=180)

        self.cancel_btn=Ct.CTkButton(self.view_frame,
text="Cancel",width=200,font=Ct.CTkFont(family="Times",size=25,
weight='bold'),command=self.cancel_book)
        self.cancel_btn.place(x=550,y=180)

        self.view_frame.pack(side=Ct.LEFT)

        # Create the Treeview
        column = ("Booking_id", "Pickup Address", "Drop-off
Address","date_of_booking","time","Booking status")
        self.view_booking = ttk.Treeview(self.view_frame, columns=column,
show="headings", height=30)
        self.view_booking.bind("<<TreeviewSelect>>", self.selectedRow)

        for col in column:
            self.view_booking.heading(col, text=col, anchor="center")
            self.view_booking.column(col, anchor="center", width=210)
            self.veiw_book()
        self.view_booking.place(x=100,y=300)
        self.view_frame.pack()

    def history_page(self):
        self.history_frame = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.history_frame.configure(fg_color= 'white')
        # Create the Treeview
        self.booklb = Ct.CTkLabel(self.history_frame, text='Booking
History',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
```

```
        column = ("id","Pickup Address", "Drop-off
Address","date_of_booking","Booking status","Driver
Name","Driverid","Phone_no","liscence no")
        self.history_booking = ttk.Treeview(self.history_frame, columns=column,
show="headings", height=30)

        for col in column:
            self.history_booking.heading(col, text=col, anchor="center")
            self.history_booking.column(col, anchor="center", width=150)
            self.view_detail()
        self.history_booking.place(x=50,y=200)
        self.history_frame.pack()

    def change_page(self):
        self.change_frame = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.change_frame.configure(fg_color= 'white')

        self.cmpic = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\Screenshot 2023-12-30 153422.png'), size=(600,500))
        self.booklb = Ct.CTkLabel(self.change_frame, text='Change
Password',font=Ct.CTkFont(family="Times",size=50,
weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        self.cmimg = Ct.CTkLabel(self.change_frame,image=self.cmpic,
text="").place(x=50,y=100)
        self.new_pass=Ct.CTkLabel(self.change_frame, text='Current
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.new_pass.place(x=700,y=150)
        self.new_pass_entry=Ct.CTkEntry(self.change_frame,height=30,width=200)
        self.new_pass_entry.place(x=700,y=200)

        self.password=Ct.CTkLabel(self.change_frame, text='New
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.password.place(x=700,y=250) # Place password
        self.pass_entry=Ct.CTkEntry(self.change_frame,height=30,width=200)
        self.pass_entry.place(x=700,y=300)

        self.password2=Ct.CTkLabel(self.change_frame, text="Conform
Password:",font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.password2.place(x=700,y=350)
        self.pass_entry2=Ct.CTkEntry(self.change_frame,height=30,width=200)
        self.pass_entry2.place(x=700,y=400)

        self.con_button=Ct.CTkButton(self.change_frame,text="Change",font=Ct.CTkFont(
family="Times",size=25, weight='bold'),width=200,command=self.change)
```

```
        self.con_button.place(x=700,y=450)



        self.change_frame.pack()

    def delete_page(self):
        self.delete_fram = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.delete_fram.configure(fg_color= 'white')
        self.booklb = Ct.CTkLabel(self.delete_fram, text='Delete Your
Account',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=350,y=100)
        self.booklb = Ct.CTkLabel(self.delete_fram, text='''
                    Are You sure you want to delete your Whoami Account?

                if you're having problems, please contact us we can help you.

            Deleting your account will remove all of your information from our
database.
            This cannot be undone.''',font=Ct.CTkFont(family="Times",size=30))
        self.booklb.place(x=0,y=150)


        self.con_button=Ct.CTkButton(self.delete_fram,text="Delete
Account",font=Ct.CTkFont(family="Times",size=30,
weight='bold'),width=300,command=self.delete_acc)
        self.con_button.place(x=420,y=450,)



        self.delete_fram.pack()


    def delete_frame(self):
        for frame in self.main_frame.winfo_children():
            frame.destroy()

    def remove(self):
        self.proview.configure(fg_color='#00BF63')
        self.view_indicate.configure(fg_color='#00BF63')
        self.book_indicate.configure(fg_color='#00BF63')
        self.change_indicate.configure(fg_color='#00BF63')
        self.delete_indicate.configure(fg_color='#00BF63')
        self.history_indicate.configure(fg_color='#00BF63')

    def indicate(self,lb, page):
```

```
        self.remove()
        lb.configure(fg_color='white')
        self.delete_frame()
        page()

    def dulicate(self):
        self.delete_frame()
        self.update_page()

    def copy(self,frame):
        self.view_profile()
        for i in self.profile:

            self.view_first = Ct.CTkLabel(frame,text="First
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_first.place(x=50,y=200)

            self.view_firstname =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_firstname.place(x=200,y=200)
            self.view_firstname.configure(text=i[0])

            self.view_last = Ct.CTkLabel(frame,text="Last
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_last.place(x=450,y=200)

            self.view_lastname =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_lastname.place(x=600,y=200)
            self.view_lastname.configure(text=i[1])

            self.view_gender =
Ct.CTkLabel(frame,text="Gender:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gender.place(x=850,y=200)

            self.view_genderdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_genderdata.place(x=950,y=200)
            self.view_genderdata.configure(text=i[4])

            self.view_Dob = Ct.CTkLabel(frame,text="Date of
Birth:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_Dob.place(x=50,y=400)
```

```
            self.view_dobdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_dobdata.place(x=230,y=400)
            self.view_dobdata.configure(text=i[3])

            self.number = Ct.CTkLabel(frame,text="Phone
Number:",font=Ct.CTkFont(family="Times",size=30,))
            self.number.place(x=450,y=400)

            self.view_numberdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_numberdata.place(x=650,y=400)
            self.view_numberdata.configure(text=i[5])

            self.view_add =
Ct.CTkLabel(frame,text="Address:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_add.place(x=840,y=400)

            self.view_adddata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_adddata.place(x=950,y=400)
            self.view_adddata.configure(text=i[6])

            self.view_pay = Ct.CTkLabel(frame,text="Payment
Method:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_pay.place(x=50,y=600)


            self.view_paydata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_paydata.place(x=280,y=600)
            self.view_paydata.configure(text=i[7])

            self.view_gmail =
Ct.CTkLabel(frame,text="Email:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gmail.place(x=500,y=600)

            self.view_gmaildata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gmaildata.place(x=600,y=600)
            self.view_gmaildata.configure(text=i[2])

    def update_page(self):
        self.update_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
```

```
        self.update_frame.configure(fg_color= 'white')
        Ct.set_default_color_theme("green")

        for i in self.profile:
            self.profile_lbl=Ct.CTkLabel(self.update_frame,text="Edit Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
            self.profile_lbl.place(x=400,y=50)

            self.update_first = Ct.CTkLabel(self.update_frame,text="First
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_first.place(x=50,y=200)

            self.edit_first = Ct.StringVar()
            self.edit_first.set(i[0])
            self.update_firstname =
Ct.CTkEntry(self.update_frame,textvariable=self.edit_first,font=Ct.CTkFont(family="Ti
mes",size=30,),width=180)
            self.update_firstname.place(x=200,y=200)

            self.edit_last = Ct.StringVar()
            self.edit_last.set(i[1])
            self.update_last = Ct.CTkLabel(self.update_frame,text="Last
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_last.place(x=450,y=200)

            self.update_lastname =
Ct.CTkEntry(self.update_frame,textvariable=self.edit_last,font=Ct.CTkFont(family="Tim
es",size=30,),width=180)
            self.update_lastname.place(x=600,y=200)

            self.update_gender =
Ct.CTkLabel(self.update_frame,text="Gender:",font=Ct.CTkFont(family="Times",size=30,)
)
            self.update_gender.place(x=800,y=200)

            self.edit_gender = Ct.StringVar()
            self.edit_gender.set(i[4])
            self.radio_btn =Ct.CTkRadioButton(self.update_frame, text="Male",
variable=self.edit_gender, value="Male")
            self.radio_btn.place(x=950, y=200)
            self.radio_btn1 = Ct.CTkRadioButton(self.update_frame, text="Female",
variable=self.edit_gender, value="Female")
            self.radio_btn1.place(x=1050, y=200)
```

```
            self.update_Dob = Ct.CTkLabel(self.update_frame,text="Date of
Birth:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_Dob.place(x=50,y=400)

            self.edit_date = Ct.StringVar()
            self.edit_date.set(i[3])
            self.update_dobdata =
DateEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),textvariable=sel
f.edit_date,selectmode='day')
            self.update_dobdata.place(x=300,y=500)

            self.update_phone = Ct.CTkLabel(self.update_frame,text="Phone
Number:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_phone.place(x=450,y=400)

            self.edit_phone = Ct.StringVar()
            self.edit_phone.set(i[5])

            self.update_phonedata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=180,text
variable=self.edit_phone)
            self.update_phonedata.place(x=650,y=400)

            self.update_add =
Ct.CTkLabel(self.update_frame,text="Address:",font=Ct.CTkFont(family="Times",size=30)
)
            self.update_add.place(x=840,y=400)

            self.edit_add = Ct.StringVar()
            self.edit_add.set(i[6])
            self.update_adddata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=180,text
variable=self.edit_add)
            self.update_adddata.place(x=950,y=400)

            self.update_pay = Ct.CTkLabel(self.update_frame,text="Payment
Method:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_pay.place(x=50,y=600)

            self.edit_pay = Ct.StringVar()
            self.edit_pay.set(i[7])
            self.update_paydata =
Ct.CTkOptionMenu(self.update_frame,font=Ct.CTkFont(family="Times",size=30),variable=s
elf.edit_pay, values=["Cash", "Banking", "Esewa"])
            self.update_paydata.place(x=280,y=600)
```

```
            self.update_mail =
Ct.CTkLabel(self.update_frame,text="Email:",font=Ct.CTkFont(family="Times",size=30))
            self.update_mail.place(x=480,y=600)

            self.mail = Ct.StringVar()
            self.mail.set(i[2])

            self.update_maildata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=350,text
variable=self.mail)
            self.update_maildata.place(x=580,y=600)

            self.update_btn =
Ct.CTkButton(self.main_frame,text="save",command=self.edit_profile,font=
Ct.CTkFont(family="Times", size=30),width=100,fg_color='#00BF63')
            self.update_btn.place(x=850,y=700)

            self.update_btn =
Ct.CTkButton(self.main_frame,text="back",command=self.back,font=
Ct.CTkFont(family="Times", size=30),width=100,fg_color='#00BF63')
            self.update_btn.place(x=1000,y=700)

            self.update_frame.pack(side=Ct.RIGHT)
            self.update_frame.pack_propagate(False)

    def back(self):
        self.delete_frame()
        self.profile_page()

    def time(self):
        self.string = strftime('%I:%M:%p')
        self.lbl.configure(text=self.string)
        self.lbl.after(1000, self.time)

    def book_request(self):
        try:
            self.id = 0
            self.Cusid = GobalVariable.Customer[0]
            cursor = self.connection.cursor()
            query  = f"INSERT INTO `booking`(`bookingid`, `pickup_address`,
`dropoff_address`, `date`, `time`, `status`, `customerid`) VALUES
('{self.id}','{self.pick_entry.get()}','{self.drop_entry.get()}','{self.date_entry.ge
t_date()}','{self.custime.get()}','pending','{self.Cusid}')"
            cursor.execute(query)
```

```
            self.connection.commit()
            self.veiw_book()
            messagebox.showinfo("Booking", "Your Book has been
requested",parent=self.master)
            self.pick_entry.delete(0,'end')
            self.drop_entry.delete(0,'end')
            self.time_pick_entry.delete(0,'end')
        except Exception as e:
            messagebox.showerror("Booking", "Something Error",parent=self.master)


    def edit_profile(self):

        try:
            customerid = GobalVariable.Customer[0]
            cursor =self.connection.cursor()
            query = f"UPDATE `customer` SET
`firstname`='{self.edit_first.get()}',`lastname`='{self.edit_last.get()}',`email`='{s
elf.mail.get()}',`DOB`='{self.update_dobdata.get_date()}',`Gender`='{self.edit_gender
.get()}',`phonenumber`='{self.edit_phone.get()}',`address`='{self.edit_add.get()}',`P
ayment_Method`='{self.edit_pay.get()}' WHERE customerid={customerid}"
            cursor.execute(query)
            # Commit the transaction
            self.connection.commit()
            self.view_profile()
            messagebox.showinfo("Profile", "Profile has been
updated",parent=self.master)
        except Exception as e:
            messagebox.showerror("Profile", f"Update Failure:
{e}",parent=self.master)

    def veiw_book(self):
        self.cust_id=GobalVariable.Customer[0]
        try:
            cursor =self.connection.cursor()
            query = f"SELECT `bookingid`, `pickup_address`, `dropoff_address`,
`date`, `time`, `status` FROM `booking` WHERE `customerid` ={self.cust_id} and
`status` = 'pending' "
            cursor.execute(query)
            rows = cursor.fetchall()

            for item in self.view_booking.get_children():
                self.view_booking.delete(item)

            for row in rows:
```

```
                self.view_booking.insert(parent='', index='end', values=(row[0],
row[1], row[2], row[3], row[4],row[5]))

        except Exception as err:
            print(f"Error: {err}")

    def selectedRow(self,event):
        selected_item = self.view_booking.focus()
        values = self.view_booking.item(selected_item, "values")

        if values:
            self.id.set(values[0])

            self.pick_entry.delete(0, "end")
            self.pick_entry.insert(0, values[1])

            self.drop_entry.delete(0, "end")
            self.drop_entry.insert(0, values[2])

            self.date_entry.delete(0, "end")
            self.date_entry.insert(0, values[3])

            self.custime.set(values[4])
    def update_book(self):
        try:
            cursor =self.connection.cursor()
            query = f"UPDATE booking SET  `pickup_address`='{self.pick_entry.get()}',
`dropoff_address`='{self.drop_entry.get()}',
`date`='{self.date_entry.get_date()}',`time`='{self.custime.get()}' WHERE
`bookingid`={self.id.get()}"

            cursor.execute(query)
            # Commit the transaction
            self.connection.commit()
            self.veiw_book()
            messagebox.showinfo("Booking", "Booking has been
updated",parent=self.master)
        except Exception as err:
            messagebox.showerror("Booking", f"Update Failure:
{err}",parent=self.master)

    def cancel_book(self):
        try:
            cursor =self.connection.cursor()
            query = f"DELETE  FROM booking WHERE bookingid={self.id.get()}"
```

```
            cursor.execute(query)

            self.connection.commit()
            self.veiw_book()
            messagebox.showinfo("booking", "Your booking has been
cancel",parent=self.master)
        except Exception as err:
            messagebox.showerror("booking", f" Failure: {err}",parent=self.master)

    def view_detail(self):
        self.cust_id=GobalVariable.Customer[0]
        try:
            cursor =self.connection.cursor()
            query = f'''SELECT
customer.customerid,booking.pickup_address,booking.dropoff_address,booking.date,booki
ng.status,driver.fullname,driver.driverid,driver.phonenumber,driver.liscenceno
            FROM customer
            JOIN booking
            ON customer.customerid =booking.customerid
            JOIN driver
            ON booking.driverid =driver.driverid
            where customer.customerid={self.cust_id}'''
            cursor.execute(query)
            rows = cursor.fetchall()

            for item in self.history_booking.get_children():
                self.history_booking.delete(item)

            for row in rows:
                self.history_booking.insert(parent='', index='end', values=(row[0],
row[1], row[2], row[3], row[4],row[5],row[6],row[7]))

        except Exception as err:
            print(f"Error: {err}")

    def view_profile(self):
        self.cust_id=GobalVariable.Customer[0]
        try:
            cursor =self.connection.cursor()
            query = f'''SELECT `firstname`, `lastname`, `email`, `DOB`, `Gender`,
`phonenumber`, `address`, `Payment_Method`FROM `customer` WHERE
customerid={self.cust_id} '''
            cursor.execute(query)
            self.profile = cursor.fetchall()
```

```
        except Exception as err:
            print(f"Error: {err}")

    def delete_acc(self):
        self.cust_id=GobalVariable.Customer[0]
        try:
            cursor =self.connection.cursor()
            query1 = f"DELETE  FROM booking WHERE customerid={self.cust_id}"
            cursor.execute(query1)
            self.connection.commit()
            query = f"DELETE  FROM customer WHERE customerid={self.cust_id}"
            cursor.execute(query)
            self.connection.commit()
            value = messagebox.askyesno("Account","Do you want to delete your
Account?",parent=self.master)
            if value:
                messagebox.showinfo("Account", "Your account has been
deleted",parent=self.master)
                app = Ct.CTkToplevel()
                self.master.destroy()
                LoginView.LoginPage(app)
                app.after(0,lambda:app.state('zoomed'))
                app.lift()
                app.mainloop()


        except Exception as err:
            messagebox.showerror("Taxi", f" Failure: {err}",parent=self.master)

    def change(self):

            old_password=GobalVariable.Customer[9]
            customer_id = GobalVariable.Customer[0]

            current_password=self.new_pass_entry.get()
            new_password = self.pass_entry.get()
            re_password = self.pass_entry2.get()
            try:
                if current_password!= old_password:
                    messagebox.showerror("password","Please check the Old password")

                elif new_password ==re_password:
                    cursor =self.connection.cursor()
                    query = f"UPDATE `customer` SET `password`='{new_password}' WHERE
customerid = '{customer_id}'"
```

```
                    cursor.execute(query)
                    self.connection.commit()
                    messagebox.showinfo("Success", "your password has been changed
successfully!",parent=self.master)

            except Exception as err:
                messagebox.showerror("Error", err)
    def logout(self):
        self.master.destroy()
        app = Ct.CTkToplevel()
        app.after(0,lambda:app.state('zoomed'))
        LoginView.LoginPage(app)
        app.mainloop()

if __name__ == '__main__':
    apps = Ct.CTk()
    apps.after(0,lambda:apps.state('zoomed'))
    Dashboard(apps)
    apps.mainloop()
```

DriverDashboard.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from PIL import Image
import customtkinter as Ct
import tkinter as tkk
from tkinter import ttk
import tkintermapview
from time import strftime
from tkcalendar import DateEntry
from Controller.DataBaseConnection import Database
import GobalVariable
import LoginView
from tkinter import messagebox

class Dashboard():
    def __init__(self,master):
        self.master = master
        self.master.title('Driver Dashboard')
        Ct.set_default_color_theme("green")
        self.master.attributes('-topmost',True)
```

```
        self.company=Ct.CTkLabel(self.master,text="Whoiam.com",font=Ct.CTkFont(family
="Times",size=25, weight='bold'))
        self.company.place(x=150,y=25)
        self.connection = Database.Connect()
        self.cmpic = Ct.CTkImage(Image.open('D:\\Code\\Python\\python
project\\TaxBookingSystem\\image\\Green White Simple Open Registration Facebook Post
(3).png'), size=(100,100))
        self.cmimg = Ct.CTkLabel(self.master,image=self.cmpic,
text="").place(x=50,y=-5)
        self.id = Ct.StringVar()
        book_id = Ct.CTkEntry(self.master,textvariable=self.id)
        book_id.place_forget()
        self.welcome=Ct.CTkLabel(self.master,text="Welcome
Back,",font=Ct.CTkFont(family="Times",size=30, weight='bold'),text_color='#00BF63')
        self.welcome.place(x=400,y=25)
        self.user=Ct.CTkLabel(self.master,text="",font=Ct.CTkFont(family="Times",size
=30, weight='bold'),text_color='#00BF63')
        self.user.place(x=610,y=25)

        self.user.configure(text=GobalVariable.Driver[1])
        self.lbl =
Ct.CTkLabel(self.master,text="",font=Ct.CTkFont(family="Times",size=30,
weight='bold'))
        self.lbl.place(x=1200,y=25)
        self.lbl2 =
Ct.CTkButton(self.master,text="logout",font=Ct.CTkFont(family="Times",size=30,
weight='bold'),command=self.logout)
        self.lbl2.place(x=1350,y=25)

        self.options_frame = Ct.CTkFrame(self.master,
fg_color='#00BF63',bg_color='#00BF63')

        self.profile = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\images-removebg-preview.png'), size=(150,150))
        self.img = Ct.CTkLabel(self.options_frame,image=self.profile,
text="").place(x=50,y=0)
        self.time()
        self.name = Ct.CTkLabel(self.options_frame,text="",
font=Ct.CTkFont(family='Times',size=30), text_color='white')
        self.name.place(x=20,y=120)
        self.name.configure(text=GobalVariable.Driver[1])

        self.dash = Ct.CTkButton(self.options_frame,text="View Profile",
fg_color='#00BF63', border_width=0,
```

```
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False,command=lambda: self.indicate(self.proveiw, self.profile_page))
        self.dash.place(x=60,y=250)

        self.proveiw = Ct.CTkFrame(self.options_frame,
fg_color='white',width=127,height=3)
        self.proveiw.place(x=68,y=285)

        self.veiw = Ct.CTkButton(self.options_frame, text='View
Trip',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda: self.indicate(self.veiw_indicate,self.veiw_page))
        self.veiw.place(x=50, y=330)

        self.veiw_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=115,height=3)
        self.veiw_indicate.place(x=65,y=370)

        self.history = Ct.CTkButton(self.options_frame,
text='History',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.history_indicate,self.history_page))
        self.history.place(x=50, y=410)

        self.history_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=80,height=3)
        self.history_indicate.place(x=80,y=448)

        self.change_password = Ct.CTkButton(self.options_frame, text='Change
Password',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.change_indicate,self.change_page))
        self.change_password.place(x=30, y=490)

        self.change_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=190,height=3)
        self.change_indicate.place(x=35,y=525)

        self.delete = Ct.CTkButton(self.options_frame, text='Delete
Account',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.delete_indicate,self.delete_page))
```

```
        self.delete.place(x=40, y=570)

        self.delete_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=170,height=3)
        self.delete_indicate.place(x=40,y=600)

        self.options_frame.place(x=40,y=70)
        self.options_frame.pack_propagate(False)
        self.options_frame.configure(width=250,height=750)

        self.main_frame = Ct.CTkFrame(self.master, width=1150, height=750)
        self.main_frame.configure(fg_color='white')
        self.profile_lbl=Ct.CTkLabel(self.main_frame,text="Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.profile_lbl.place(x=400,y=50)
        self.copy(self.main_frame)
        self.update_btn =
Ct.CTkButton(self.main_frame,text="Edit",command=self.dulicate,font=
Ct.CTkFont(family="Times", size=30),width=200,fg_color='#00BF63')
        self.update_btn.place(x=850,y=600)
        self.main_frame.place(x=350,y=70)
        self.main_frame.pack_propagate(False)
        # self.main_frame.configure(height=750, width=1150)

    def profile_page(self):
        ''' The gui for the veiw profile frame, it shows the details of the customer
who has login '''

        self.home_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.home_frame.configure(fg_color='white')
        Ct.set_default_color_theme("green")
        self.profile_lbl=Ct.CTkLabel(self.home_frame,text="Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.profile_lbl.place(x=400,y=50)
        self.copy(self.main_frame)
        self.update_btn =
Ct.CTkButton(self.home_frame,text="Edit",command=self.dulicate,font=
Ct.CTkFont(family="Times", size=30),width=200,fg_color='#00BF63')
        self.update_btn.place(x=850,y=600)
        self.home_frame.pack(side=Ct.LEFT)

        self.home_frame.pack(side=Ct.RIGHT)


    def picktime(self):
        import tkinter as tk
```

```
        from tktimepicker import AnalogPicker

        self.root = tk.Tk()
        self.root.attributes('-topmost',True)
        self.time_picker = AnalogPicker(self.root)
        self.time_picker.pack(expand=True, fill="both")

        # theme = AnalogThemes(time_picker)
        # theme.setDracula()
        self.request = Ct.CTkButton(self.root,
text="set",width=10,font=Ct.CTkFont(family="Times",size=20,
weight='bold'),command=self.settime)
        self.request.place(x=250,y= 200)
        self.root.mainloop()

    def settime(self):
            time = self.time_picker.time()
            self.custime.set(time)
            self.root.destroy()

    def veiw_page(self):
        self.veiw_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.veiw_frame.configure(fg_color= 'white')
        self.profile_lbl=Ct.CTkLabel(self.veiw_frame,text="Veiw Pending
Trip",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.profile_lbl.place(x=400,y=30)
        self.click = Ct.CTkLabel(self.veiw_frame,text=
        ' To Confirm that you have completed the Trip . Please press the
Buttton',font= Ct.CTkFont(family="Times", size=30))
        self.click.place(x=100,y=150)
        self.update=Ct.CTkButton(self.veiw_frame,
text="Complete",width=20,command=self.make_complete)
        self.update.place(x=1020,y=150)
        self.veiw_frame.pack(side=Ct.LEFT)

        # Create the Treeview
        column = ("Booking_id", "Customer_Name","Mobile_no", "Pickup_Address", "Drop-
off _Address","date_of_booking","Payment Method","Booking status")
        self.veiw_booking = ttk.Treeview(self.veiw_frame, columns=column,
show="headings", height=30)
        self.veiw_booking.bind("<<TreeviewSelect>>",self.selectedRow)

        for col in column:
            self.veiw_booking.heading(col, text=col, anchor="center")
            self.veiw_booking.column(col, anchor="center", width=165)
```

```
            self.viewbooking()


        self.veiw_booking.place(x=70,y=250)
        self.veiw_frame.pack()

    def history_page(self):
        self.history_frame = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.history_frame.configure(fg_color= 'white')
        # Create the Treeview
        self.profile_lbl=Ct.CTkLabel(self.history_frame,text="Booking
History",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.profile_lbl.place(x=400,y=30)
        column = ("Booking_id", "Customer_Name","Mobile_no", "Pickup_Address", "Drop-
off _Address","date_of_booking","Payment Method","Booking status")

        self.veiw_booking = ttk.Treeview(self.history_frame, columns=column,
show="headings", height=30)

        for col in column:
            self.veiw_booking.heading(col, text=col, anchor="center")
            self.veiw_booking.column(col, anchor="center", width=165)
            self.viewhistory()

        self.veiw_booking.place(x=50,y=200)
        self.history_frame.pack()

    def change_page(self):
        self.chan = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.chan.configure(fg_color= 'white')

        self.cmpic = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\Screenshot 2023-12-30 153422.png'), size=(600,500))
        self.booklb = Ct.CTkLabel(self.chan, text='Password
change',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        self.cmimg = Ct.CTkLabel(self.chan,image=self.cmpic,
text="").place(x=50,y=100)
        self.new_pass=Ct.CTkLabel(self.chan, text='Current
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.new_pass.place(x=700,y=150)
        self.new_pass_entry=Ct.CTkEntry(self.chan,height=30,width=200)
        self.new_pass_entry.place(x=700,y=200)

        self.password=Ct.CTkLabel(self.chan, text='New
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
```

```
        self.password.place(x=700,y=250) # Place password
        self.pass_entry=Ct.CTkEntry(self.chan,height=30,width=200)
        self.pass_entry.place(x=700,y=300)

        self.password2=Ct.CTkLabel(self.chan, text="Conform
Password:",font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.password2.place(x=700,y=350)
        self.pass_entry2=Ct.CTkEntry(self.chan,height=30,width=200)
        self.pass_entry2.place(x=700,y=400)

        self.con_button=Ct.CTkButton(self.chan,text="change",font=Ct.CTkFont(family="
Times",size=25, weight='bold'),width=200,command=self.change)
        self.con_button.place(x=700,y=450)
        self.chan.pack()


    def delete_page(self):
        self.delete_fram = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.delete_fram.configure(fg_color= 'white')
        self.booklb = Ct.CTkLabel(self.delete_fram, text='Delete Your
Account',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=350,y=100)
        self.booklb = Ct.CTkLabel(self.delete_fram, text='''
                    Are You sure you want to delete your Whoami Account?

                if you're having problems, please contact us we can help you.

            Deleting your account will remove all of your information from our
database.
            This cannot be undone.''',font=Ct.CTkFont(family="Times",size=30))
        self.booklb.place(x=0,y=150)


        self.con_button=Ct.CTkButton(self.delete_fram,text="Delete
Account",font=Ct.CTkFont(family="Times",size=30,
weight='bold'),width=300,command=self.delete_acc)
        self.con_button.place(x=420,y=450,)



        self.delete_fram.pack()


    def delete_frame(self):
        for frame in self.main_frame.winfo_children():
            frame.destroy()
```

```
    def remove(self):
        self.proveiw.configure(fg_color='#00BF63')
        self.veiw_indicate.configure(fg_color='#00BF63')
        self.change_indicate.configure(fg_color='#00BF63')
        self.delete_indicate.configure(fg_color='#00BF63')
        self.history_indicate.configure(fg_color='#00BF63')

    def indicate(self,lb, page):
        self.remove()
        lb.configure(fg_color='white')
        self.delete_frame()
        page()

    def dulicate(self):
        self.delete_frame()
        self.update_page()

    def copy(self,frame):
        self.view_profile()
        for i in self.driprofile:

            self.view_first = Ct.CTkLabel(frame,text="Full
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_first.place(x=50,y=200)

            self.view_firstname =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_firstname.place(x=200,y=200)
            self.view_firstname.configure(text=i[0])

            self.view_gender =
Ct.CTkLabel(frame,text="Gender:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gender.place(x=850,y=200)

            self.view_genderdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_genderdata.place(x=950,y=200)
            self.view_genderdata.configure(text=i[5])

            self.view_Dob = Ct.CTkLabel(frame,text="Date of
Birth:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_Dob.place(x=50,y=400)
```

```
            self.view_dobdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_dobdata.place(x=230,y=400)
            self.view_dobdata.configure(text=i[4])

            self.number = Ct.CTkLabel(frame,text="Phone
Number:",font=Ct.CTkFont(family="Times",size=30,))
            self.number.place(x=450,y=400)

            self.view_numberdata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_numberdata.place(x=650,y=400)
            self.view_numberdata.configure(text=i[1])

            self.view_add =
Ct.CTkLabel(frame,text="Address:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_add.place(x=840,y=400)

            self.view_adddata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_adddata.place(x=950,y=400)
            self.view_adddata.configure(text=i[2])

            self.view_pay = Ct.CTkLabel(frame,text="Liscence
No:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_pay.place(x=50,y=600)


            self.view_paydata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_paydata.place(x=280,y=600)
            self.view_paydata.configure(text=i[6])

            self.view_gmail =
Ct.CTkLabel(frame,text="Email:",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gmail.place(x=500,y=600)

            self.view_gmaildata =
Ct.CTkLabel(frame,text="",font=Ct.CTkFont(family="Times",size=30,))
            self.view_gmaildata.place(x=600,y=600)
            self.view_gmaildata.configure(text=i[3])


    def update_page(self):
        self.update_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
```

```
        self.update_frame.configure(fg_color= 'white')
        Ct.set_default_color_theme("green")

        for i in self.driprofile:
            self.profile_lbl=Ct.CTkLabel(self.update_frame,text="Edit Your
Profile",font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
            self.profile_lbl.place(x=400,y=50)

            self.update_first = Ct.CTkLabel(self.update_frame,text="Full
Name:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_first.place(x=50,y=200)

            self.edit_first = Ct.StringVar()
            self.edit_first.set(i[0])
            self.update_firstname =
Ct.CTkEntry(self.update_frame,textvariable=self.edit_first,font=Ct.CTkFont(family="Ti
mes",size=30,),width=180)
            self.update_firstname.place(x=200,y=200)

            self.update_gender =
Ct.CTkLabel(self.update_frame,text="Gender:",font=Ct.CTkFont(family="Times",size=30,)
)
            self.update_gender.place(x=800,y=200)

            self.edit_gender = Ct.StringVar()
            self.edit_gender.set(i[5])
            self.radio_btn =Ct.CTkRadioButton(self.update_frame, text="Male",
variable=self.edit_gender, value="Male")
            self.radio_btn.place(x=950, y=200)
            self.radio_btn1 = Ct.CTkRadioButton(self.update_frame, text="Female",
variable=self.edit_gender, value="Female")
            self.radio_btn1.place(x=1050, y=200)


            self.update_Dob = Ct.CTkLabel(self.update_frame,text="Date of
Birth:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_Dob.place(x=50,y=400)

            self.edit_date = Ct.StringVar()
            self.edit_date.set(i[4])
            self.update_dobdata =
DateEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),textvariable=sel
f.edit_date,selectmode='day')
            self.update_dobdata.place(x=300,y=500)
```

```
            self.update_phone = Ct.CTkLabel(self.update_frame,text="Phone
Number:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_phone.place(x=450,y=400)

            self.edit_phone = Ct.StringVar()
            self.edit_phone.set(i[1])

            self.update_phonedata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=180,text
variable=self.edit_phone)
            self.update_phonedata.place(x=650,y=400)

            self.update_add =
Ct.CTkLabel(self.update_frame,text="Address:",font=Ct.CTkFont(family="Times",size=30)
)
            self.update_add.place(x=840,y=400)

            self.edit_add = Ct.StringVar()
            self.edit_add.set(i[2])
            self.update_adddata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=180,text
variable=self.edit_add)
            self.update_adddata.place(x=950,y=400)

            self.update_pay = Ct.CTkLabel(self.update_frame,text="Liscence
No:",font=Ct.CTkFont(family="Times",size=30,))
            self.update_pay.place(x=50,y=600)

            self.edit_pay = Ct.StringVar()
            self.edit_pay.set(i[6])
            self.update_paydata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30),textvariable=se
lf.edit_pay)
            self.update_paydata.place(x=280,y=600)

            self.update_mail =
Ct.CTkLabel(self.update_frame,text="Email:",font=Ct.CTkFont(family="Times",size=30))
            self.update_mail.place(x=480,y=600)

            self.mail = Ct.StringVar()
            self.mail.set(i[3])

            self.update_maildata =
Ct.CTkEntry(self.update_frame,font=Ct.CTkFont(family="Times",size=30,),width=350,text
variable=self.mail)
```

```
            self.update_maildata.place(x=580,y=600)

            self.update_btn =
Ct.CTkButton(self.main_frame,text="save",command=self.edit_profile,font=
Ct.CTkFont(family="Times", size=30),width=100,fg_color='#00BF63')
            self.update_btn.place(x=850,y=700)

            self.update_btn =
Ct.CTkButton(self.main_frame,text="back",command=self.back,font=
Ct.CTkFont(family="Times", size=30),width=100,fg_color='#00BF63')
            self.update_btn.place(x=1000,y=700)

            self.update_frame.pack(side=Ct.RIGHT)
            self.update_frame.pack_propagate(False)

    def back(self):
        self.delete_frame()
        self.profile_page()


    def time(self):
        self.string = strftime('%I:%M:%p')
        self.lbl.configure(text=self.string)
        self.lbl.after(1000, self.time)

    def view_profile(self):
        self.cust_id=GobalVariable.Driver[0]
        try:
            cursor =self.connection.cursor()
            query = f'''SELECT `fullname`, `phonenumber`, `Address`, `email`, `DOB`,
`gender`,`liscenceno` FROM `driver` WHERE driverid={self.cust_id} '''
            cursor.execute(query)
            self.driprofile = cursor.fetchall()

        except Exception as err:
            print(f"Error: {err}")

    def edit_profile(self):

        try:
            customerid = GobalVariable.Driver[0]
            cursor =self.connection.cursor()
            query = f"UPDATE `driver` SET
`fullname`='{self.edit_first.get()}',`email`='{self.mail.get()}',`DOB`='{self.update_
dobdata.get_date()}',`gender`='{self.edit_gender.get()}',`phonenumber`='{self.edit_ph
```

```
one.get()}',`Address`='{self.edit_add.get()}',`liscenceno`='{self.edit_pay.get()}'
WHERE driverid={customerid}"
            cursor.execute(query)
            # Commit the transaction
            self.connection.commit()
            self.view_profile()
            messagebox.showinfo("Taxi", "Your profile has been
Update",parent=self.master)
        except Exception as e:
            messagebox.showerror("Taxi", f"Update Failure: {e}",parent=self.master)

    def delete_acc(self):
        self.cust_id=GobalVariable.Driver[0]
        try:
            cursor =self.connection.cursor()
            query = f"DELETE  FROM driver WHERE driverid={self.cust_id}"
            cursor.execute(query)


            self.connection.commit()
            value = messagebox.askyesno("Taxi","Do you want to delete your
Account?",parent=self.master)
            if value:
                messagebox.showinfo("Taxi", "Your account has been
deleted",parent=self.master)
                app = Ct.CTkToplevel()
                self.master.destroy()
                import LoginView
                LoginView.LoginPage(app)
                app.after(0,lambda:app.state('zoomed'))
                app.lift()
                app.mainloop()


        except Exception as err:
            messagebox.showerror("Taxi", f" Failure: {err}",parent=self.master)

    def change(self):

        old_password=GobalVariable.Driver[9]
        customer_id = GobalVariable.Driver[0]
        current_password=self.new_pass_entry.get()
        new_password = self.pass_entry.get()
        re_password = self.pass_entry2.get()
        try:
            if current_password!= old_password:
```

```
                messagebox.showerror("password","Please check the Old password")

            elif new_password ==re_password:
                cursor =self.connection.cursor()
                query = f"UPDATE `driver` SET `password`='{new_password}' WHERE
driverid = '{customer_id}'"
                cursor.execute(query)
                self.connection.commit()
                messagebox.showinfo("Success", "your password has been changed
successfully!",parent=self.master)


        except Exception as err:
            messagebox.showerror("Error", err)
    def viewbooking(self):
        try:
            driverid = GobalVariable.Driver[0]
            cursor =  self.connection.cursor()
            query = f'''SELECT
booking.bookingid,customer.firstname,customer.phonenumber,booking.pickup_address,book
ing.dropoff_address,booking.date,customer.Payment_Method,booking.status
            FROM customer
            JOIN booking
            ON customer.customerid = booking.customerid
            where booking.status="booked" and booking.driverid = "{driverid}" '''
            cursor.execute(query)
            rows = cursor.fetchmany(size=10)  # Adjust the size as needed

            for item in self.veiw_booking.get_children():
                self.veiw_booking.delete(item)

            for row in rows:
                self.veiw_booking.insert(parent='', index='end',values=(row[0],
row[1], row[2], row[3], row[4], row[5],row[6], row[7]))

        except Exception as err:
            print(f"Error: {err}")
            messagebox.showerror("Taxi", f"Error fetching bookings:
{err}",parent=self.master)


    def viewhistory(self):
        try:
            driverid = GobalVariable.Driver[0]
            cursor =  self.connection.cursor()
```

```
            query = f'''SELECT
booking.bookingid,customer.firstname,customer.phonenumber,booking.pickup_address,book
ing.dropoff_address,booking.date,customer.Payment_Method,booking.status
            FROM customer
            JOIN booking
            ON customer.customerid = booking.customerid
            where booking.status="completed" and booking.driverid = "{driverid}" '''
            cursor.execute(query)
            rows = cursor.fetchall()

            for item in self.veiw_booking.get_children():
                self.veiw_booking.delete(item)

            for row in rows:
                self.veiw_booking.insert(parent='', index='end',values=(row[0],
row[1], row[2], row[3], row[4], row[5],row[6], row[7]))

        except Exception as err:
            print(f"Error: {err}")
            messagebox.showerror("Taxi", f"Error fetching bookings:
{err}",parent=self.master)

    def make_complete(self):
        try:
            value = self.id.get()
            if value=='':
                messagebox.showinfo("Taxi", "please selected the
row",parent=self.master)
            else:
                driverid = GobalVariable.Driver[0]
                cursor = self.connection.cursor()
                query = f"UPDATE booking SET `status`='completed' WHERE
`bookingid`={self.id.get()}"
                query1 = f"UPDATE driver SET `status`='active' where
`driverid`='{driverid}'"
                cursor.execute(query)
                cursor.execute(query1)
                # Commit the transaction
                self.connection.commit()
                self.viewbooking()

                messagebox.showinfo("Trip", "Your trip has been
completed",parent=self.master)
        except Exception as err:
```

```
            messagebox.showerror("Trip", f"Failure: {err}",parent=self.master)

    def selectedRow(self,event):
        selected_item = self.veiw_booking.focus()
        values = self.veiw_booking.item(selected_item, "values")

        if values:
            self.id.set(values[0])

    def logout(self):
        self.master.destroy()
        app = Ct.CTkToplevel()
        app.after(0,lambda:app.state('zoomed'))
        LoginView.LoginPage(app)
        app.mainloop()

if __name__ == '__main__':
    apps = Ct.CTk()
    apps.after(0,lambda:apps.state('zoomed'))
    Dashboard(apps)
    apps.mainloop()
```

AdminDashboard.py

```
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from PIL import Image
import customtkinter as Ct
import tkinter as tk
from tkinter import ttk
from tkcalendar import DateEntry
from tkinter import messagebox
import GobalVariable
import CustomerDashboard
from time import strftime
import LoginView
from Controller.DriverController import DriverDatabase
from Controller.DataBaseConnection import Database

class Dashboard():
    def __init__(self,master):
        self.master = master
        self.master.title('Admin Dashboard')
        self.connection = Database.Connect()
```

```
        Ct.set_default_color_theme("green")
        # self.master.configure(fg_color='white')

        self.company=Ct.CTkLabel(self.master,text="Whoiam.com",font=Ct.CTkFont(family
="Times",size=25, weight='bold'))
        self.company.place(x=150,y=10)
        self.cmpic = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\Green White Simple Open Registration Facebook Post
(3).png'), size=(100,100))
        self.cmimg = Ct.CTkLabel(self.master,image=self.cmpic,
text="").place(x=50,y=-30)

        self.options_frame = Ct.CTkFrame(self.master,
fg_color='#00BF63',bg_color='#00BF63')
        # self.master.iconbitmap(self.cmpic)
        self.lbl = Ct.CTkLabel(self.master,font=Ct.CTkFont(family="Times",size=30,
weight='bold'))
        self.lbl.place(x=1200,y=5)
        self.lbl2 =
Ct.CTkButton(self.master,text="logout",font=Ct.CTkFont(family="Times",size=30,
weight='bold'),command=self.logout)
        self.lbl2.place(x=1350,y=5)
        self.time()
        self.profile = Ct.CTkImage(Image.open('image\\images-removebg-preview.png'),
size=(150,150))
        self.img = Ct.CTkLabel(self.options_frame,image=self.profile,
text="").place(x=50,y=0)

        self.name = Ct.CTkLabel(self.options_frame,text="Charitra Shrestha",
font=Ct.CTkFont(family='Times',size=30), text_color='white')
        self.name.place(x=20,y=120)

        self.dash = Ct.CTkButton(self.options_frame,text="Assign Driver",
fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False,command=lambda: self.indicate(self.proveiw, self.profile_page))
        self.dash.place(x=60,y=250)

        self.proveiw = Ct.CTkFrame(self.options_frame,
fg_color='white',width=127,height=3)
        self.proveiw.place(x=68,y=285)

        self.book = Ct.CTkButton(self.options_frame, text='Add
Driver',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
```

```
white',hover=False, command=lambda: self.indicate(self.book_indicate,
self.driver_regi))
        self.book.place(x=50,y=325)

        self.book_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=110,height=3)
        self.book_indicate.place(x=65,y=360)

        self.veiw = Ct.CTkButton(self.options_frame, text='customer
details',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.veiw_indicate,self.cust_details))
        self.veiw.place(x=50, y=400)

        self.veiw_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=150,height=3)
        self.veiw_indicate.place(x=65,y=435)

        self.history = Ct.CTkButton(self.options_frame,
text='History',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.history_indicate,self.history_page))
        self.history.place(x=50, y=475)

        self.history_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=80,height=3)
        self.history_indicate.place(x=80,y=510)

        self.change_password = Ct.CTkButton(self.options_frame, text='Change
Password',fg_color='#00BF63', border_width=0,
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'),text_color='
white',hover=False, command=lambda:
self.indicate(self.change_indicate,self.change_page))
        self.change_password.place(x=30, y=550)

        self.change_indicate = Ct.CTkFrame(self.options_frame,
fg_color='#00BF63',width=190,height=3)
        self.change_indicate.place(x=35,y=590)

        self.options_frame.pack(side=Ct.LEFT,padx=30)
        self.options_frame.pack_propagate(False)
        self.options_frame.configure(width=250,height=750)
```

```
        self.main_frame = Ct.CTkFrame(self.master, width=1150, height=750)
        self.main_frame.configure(fg_color='white')
        self.booking_id = Ct.CTkLabel(self.main_frame, text="Booking ID",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.booking_id.place(x=40,y=40)
        self.bookingid = Ct.StringVar()
        self.booking_id_entry =
Ct.CTkEntry(self.main_frame,textvariable=self.bookingid)
        self.booking_id_entry.place(x=180, y=40)

        self.booking_status = Ct.CTkLabel(self.main_frame, text="Booking Status",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.booking_status.place(x=380,y=40)
        self.var1=Ct.StringVar()
        self.var1.set("Select payment method")

        self.status=tk.OptionMenu(self.main_frame,self.var1,"Booked", "Pending")
        self.status.place(x=690,y=50, width=200, height=40)

        self.veiw_driver()
        self.driver=Ct.CTkLabel(self.main_frame, text="Driver ID",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.driver.place(x=810, y=40)
        self.var2=Ct.StringVar()
        self.var2.set("Select Driver")
        self.driverr = ["driver unavailable"]
        if len(self.mylist)!=0:
            self.driver_option=tk.OptionMenu(self.main_frame,self.var2,*self.mylist)
            self.driver_option.place(x=1160,y=50, width=200, height=40)
        else:
            self.driver_option=tk.OptionMenu(self.main_frame,self.var2,*self.driverr)
            self.driver_option.place(x=1160,y=50, width=200, height=40)

        self.assign=Ct.CTkButton(self.main_frame, text="Assign", fg_color='#00BF63',
bg_color='#00BF63',
font=Ct.CTkFont(family='Times',size=25,weight='bold'),command=self.assign_driver)
        self.assign.place(x=450, y=150)

        column = ("Booking_id", "Customer_Name","Mobile_no", "Pickup_Address", "Drop-
off _Address","date_of_booking","Payment Method","Booking status")
        self.view_booking = ttk.Treeview(self.main_frame, columns=column,
show="headings", height=30)
```

```
        self.view_booking.bind("<<TreeviewSelect>>", self.selectedRow)

        for col in column:
            self.view_booking.heading(col, text=col, anchor="center")
            self.view_booking.column(col, anchor="center", width=165)
            self.veiw_customer()

        self.view_booking.place(x=60,y=300)


        self.main_frame.pack(side=Ct.LEFT,padx=30)
        self.main_frame.pack_propagate(False)
        # self.main_frame.configure(height=750, width=1150)

    def profile_page(self):
        ''' The gui for the veiw profile frame, it shows the details of the customer
whio has login '''
        self.home_frame = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.home_frame.configure(fg_color='white')
        self.booking_id = Ct.CTkLabel(self.home_frame, text="Booking ID",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.booking_id.place(x=40,y=40)
        self.bookingid = Ct.StringVar()
        self.booking_id_entry =
Ct.CTkEntry(self.home_frame,textvariable=self.bookingid)
        self.booking_id_entry.place(x=180, y=40)

        self.booking_status = Ct.CTkLabel(self.home_frame, text="Booking Status",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.booking_status.place(x=380,y=40)
        self.var1=Ct.StringVar()
        self.var1.set("Select payment method")

        self.status=tk.OptionMenu(self.home_frame,self.var1,"Booked", "Pending")
        self.status.place(x=690,y=50, width=200, height=40)

        self.veiw_driver()
        self.driver=Ct.CTkLabel(self.home_frame, text="Driver ID",
fg_color='#00BF63',
bg_color='#00BF63',font=Ct.CTkFont(family='Times',size=25,weight='bold'))
        self.driver.place(x=810, y=40)
        self.var2=Ct.StringVar(self.home_frame)
        self.var2.set("Select Driver")
```

```
        self.driver_option=tk.OptionMenu(self.home_frame,self.var2,*self.mylist)
        self.driver_option.place(x=1160,y=50, width=200, height=40)

        self.assign=Ct.CTkButton(self.home_frame, text="Assign", fg_color='#00BF63',
bg_color='#00BF63',
font=Ct.CTkFont(family='Times',size=25,weight='bold'),command=self.assign_driver)
        self.assign.place(x=450, y=150)

        column = ("Booking_id", "Customer_Name","Mobile_no", "Pickup_Address", "Drop-
off _Address","date_of_booking","Payment Method","Booking status")
        self.view_booking = ttk.Treeview(self.home_frame, columns=column,
show="headings", height=30)
        self.view_booking.bind("<<TreeviewSelect>>", self.selectedRow)

        for col in column:
            self.view_booking.heading(col, text=col, anchor="center")
            self.view_booking.column(col, anchor="center", width=165)
            self.veiw_customer()

        self.view_booking.place(x=60,y=300)
        self.home_frame.pack(side=Ct.LEFT)


    def driver_regi(self):
        self.driver_reg = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.full_name_CTkLabel =Ct.CTkLabel(self.driver_reg, text="Full
Name:",font=Ct.CTkFont(family="Times",size=20))
        self.driver_reg.configure(fg_color='white')
        self.full_name_CTkLabel.place(x=30,y=50)
        self.full_name_CTkentry
=Ct.CTkEntry(self.driver_reg,textvariable=Ct.StringVar())
        self.full_name_CTkentry.place(x=140,y=50)

        self.address_CTkLabel =Ct.CTkLabel(self.driver_reg,
text="Address:",font=Ct.CTkFont(family="Times",size=20))
        self.address_CTkLabel.place(x=400,y=50)
        self.address_CTkentry
=Ct.CTkEntry(self.driver_reg,textvariable=Ct.StringVar())
        self.address_CTkentry.place(x=490,y=50)

        self.date_CTklabel =Ct.CTkLabel(self.driver_reg,text =
'DOB:',font=Ct.CTkFont(family="Times",size=20))
        self.date_CTklabel.place(x=750,y=50)
        self.date_CTkEntry = DateEntry(self.driver_reg,width=30,selectmode='day')
        self.date_CTkEntry.place(x=1020,y=70)
```

```
        self.email_CTkLabel =Ct.CTkLabel(self.driver_reg,
text="Email:",font=Ct.CTkFont(family="Times",size=20))
        self.email_CTkLabel.place(x=30,y=150)
        self.email_CTkentry =Ct.CTkEntry(self.driver_reg,textvariable=Ct.StringVar())
        self.email_CTkentry.place(x=140,y=150)

        self.pass_CTkLabel =Ct.CTkLabel(self.driver_reg,
text="Password:",font=Ct.CTkFont(family="Times",size=20))
        self.pass_CTkLabel.place(x=400,y=150)
        self.pass_CTkentry =Ct.CTkEntry(self.driver_reg,textvariable=Ct.StringVar())
        self.pass_CTkentry.place(x=490,y=150)

        self.phone_CTkLabel =Ct.CTkLabel(self.driver_reg, text="Phone
Number:",font=Ct.CTkFont(family="Times",size=20))
        self.phone_CTkLabel.place(x=750,y=150)
        self.phone_CTkentry =Ct.CTkEntry(self.driver_reg,textvariable=Ct.StringVar())
        self.phone_CTkentry.place(x=890,y=150)

        self.gender_CTkLabel =Ct.CTkLabel(self.driver_reg,
text="Gender:",font=Ct.CTkFont(family="Times",size=20))
        self.gender_CTkLabel.place(x=30,y=250)
        self.var_gender =Ct.StringVar()
        self.radio_btn =Ct.CTkRadioButton(self.driver_reg, text="Male",
variable=self.var_gender, value="Male")
        self.radio_btn.place(x=120, y=250)
        self.radio_btn1 =Ct.CTkRadioButton(self.driver_reg, text="Female",
variable=self.var_gender, value="Female")
        self.radio_btn1.place(x=180, y=250)

        self.license=Ct.CTkLabel(self.driver_reg,text='License
No:',font=Ct.CTkFont(family="Times",size=20))
        self.license.place(x=400,y=250)
        self.license_entry=Ct.CTkEntry(self.driver_reg, textvariable=Ct.StringVar())
        self.license_entry.place(x=520,y=250)

        self.register = Ct.CTkButton(self.driver_reg, text="Register",
font=Ct.CTkFont(family="Times",size=20),command=self.driver_register)
        self.register.place(x=750, y=250)

        column = ("Driverid", "fullname", "phonenumber", "Address", "email", "DOB",
"gender", "status", "liscenceno", "password")
        self.view_driver = ttk.Treeview(self.driver_reg, columns=column,
show="headings", height=30)
        self.driver_view()
        for col in column:
```

```
            self.view_driver.heading(col, text=col, anchor="center")
            self.view_driver.column(col, anchor="center", width=135)


        self.view_driver.place(x=60,y=360)

        self.driver_reg.pack(side=Ct.LEFT)

    def cust_details(self):

        self.cus_det = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.cus_det.configure(fg_color='white')
        self.booklb = Ct.CTkLabel(self.cus_det, text='Customer
Details',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        column = ("id","firstname", "lastname","email","Date of
birth","gender","address","payment_method")
        self.view_customer = ttk.Treeview(self.cus_det, columns=column,
show="headings", height=40)

        for col in column:
            self.view_customer.heading(col, text=col, anchor="center")
            self.view_customer.column(col, anchor="center", width=165)
            self.details_customer()
        self.view_customer.place(x=60,y=200)
        self.cus_det.pack()

    def time(self):
        self.string = strftime('%I:%M:%p')
        self.lbl.configure(text=self.string)
        self.lbl.after(1000, self.time)

    def history_page(self):
        self.history_frame = Ct.CTkFrame(self.main_frame,width=1125, height=750)
        self.history_frame.configure(fg_color= 'white')
        # Create the Treeview
        self.booklb = Ct.CTkLabel(self.history_frame, text='Booking
History',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        column = ("id","name","Date of
birth","gender","address","payment_method","pickup","dropoff","date","time","status",
"Driver","phoneno")
        self.history_booking = ttk.Treeview(self.history_frame, columns=column,
show="headings", height=30)

        for col in column:
```

```
            self.history_booking.heading(col, text=col, anchor="center")
            self.history_booking.column(col, anchor="center", width=100)
            self.historyy()

        self.history_booking.place(x=50,y=200)
        self.history_frame.pack()

    def change_page(self):
        self.chan = Ct.CTkFrame(self.main_frame,width=1150, height=750)
        self.chan.configure(fg_color= 'white')

        self.cmpic = Ct.CTkImage(Image.open('D:\Code\Python\python
project\TaxBookingSystem\image\Screenshot 2023-12-30 153422.png'), size=(600,500))
        self.booklb = Ct.CTkLabel(self.chan, text='Password
chan',font=Ct.CTkFont(family="Times",size=50, weight='bold'),text_color='#00BF63')
        self.booklb.place(x=400,y=50)
        self.cmimg = Ct.CTkLabel(self.chan,image=self.cmpic,
text="").place(x=50,y=100)
        self.new_pass=Ct.CTkLabel(self.chan, text='Current
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.new_pass.place(x=700,y=150)
        self.new_pass_entry=Ct.CTkEntry(self.chan,height=30,width=200)
        self.new_pass_entry.place(x=700,y=200)

        self.password=Ct.CTkLabel(self.chan, text='New
Password:',font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.password.place(x=700,y=250) # Place password
        self.pass_entry=Ct.CTkEntry(self.chan,height=30,width=200)
        self.pass_entry.place(x=700,y=300)

        self.password2=Ct.CTkLabel(self.chan, text="Conform
Password:",font=Ct.CTkFont(family="Times",size=25, weight='bold'),text_color='black')
        self.password2.place(x=700,y=350)
        self.pass_entry2=Ct.CTkEntry(self.chan,height=30,width=200)
        self.pass_entry2.place(x=700,y=400)

        self.con_button=Ct.CTkButton(self.chan,text="change",font=Ct.CTkFont(family="
Times",size=25, weight='bold'),width=200,command=self.change)
        self.con_button.place(x=700,y=450)
        self.chan.pack()

    def delete_frame(self):
        for frame in self.main_frame.winfo_children():
            frame.destroy()
```

```
    def remove(self):
        self.proveiw.configure(fg_color='#00BF63')
        self.veiw_indicate.configure(fg_color='#00BF63')
        self.book_indicate.configure(fg_color='#00BF63')
        self.change_indicate.configure(fg_color='#00BF63')
        self.history_indicate.configure(fg_color='#00BF63')

    def indicate(self,lb, page):
        self.remove()
        lb.configure(fg_color='white')
        self.delete_frame()
        page()

    def change(self):

        old_password=GobalVariable.Admin[4]
        customer_id = GobalVariable.Admin[0]
        current_password=self.new_pass_entry.get()
        new_password = self.pass_entry.get()
        re_password = self.pass_entry2.get()
        try:
            if current_password!= old_password:
                messagebox.showerror("password","Please check the old
password",parent=self.master)

            elif new_password ==re_password:
                cursor =self.connection.cursor()
                query = f"UPDATE `admin` SET `password`='{new_password}' WHERE
adminid = '{customer_id}'"
                cursor.execute(query)
                self.connection.commit()
                messagebox.showinfo("Success", "your password has been changed
successfully!",parent=self.master)
        except Exception as err:
            messagebox.showerror("Error", err)

    def driver_register(self):
        try:
            self.gender = "Male" if self.var_gender.get() == 'Male' else "Female"
            cursor =  self.connection.cursor()
            query  = f"INSERT INTO `driver`(`driverid`,fullname,`phonenumber`,
`Address`, `email`, `DOB`,`gender`,`liscenceno`, `password`) VALUES
('0','{self.full_name_CTkentry.get()}','{self.phone_CTkentry.get()}','{self.address_C
Tkentry.get()}','{self.email_CTkentry.get()}','{self.date_CTkEntry.get_date()}','{sel
f.license_entry.get()}','{self.gender}','{self.pass_CTkentry.get()}')"
```

```
            cursor.execute(query)
            self.connection.commit()
            self.driver_view()
            messagebox.showinfo("Taxi","Driver Register
Sucessfull",parent=self.master)

        except Exception as e:
            messagebox.showinfo("Taxi","{e}")

    def driver_view(self):
        try:
            cursor =  self.connection.cursor()
            query = "SELECT * from driver"
            cursor.execute(query)
            rows = cursor.fetchall()

            for item in self.view_driver.get_children():
                self.view_driver.delete(item)

            for row in rows:
                self.view_driver.insert(parent='', index='end',values=(row[0],
row[1], row[2], row[3], row[4], row[5],row[6], row[7], row[8],row[9]))

        except Exception as err:
            print(f"Error: {err}")
            messagebox.showerror("Taxi", f"Error fetching bookings:
{err}",parent=self.master)

    def delete_acc(self):
        self.cust_id=GobalVariable.Admin[0]
        try:
            cursor =self.connection.cursor()
            query = f"DELETE  FROM admin WHERE adminid={self.cust_id}"
            cursor.execute(query)
            self.connection.commit()
            value = messagebox.askyesno("Taxi","Do you want to delete your
Account?",parent=self.master)
            if value:
                messagebox.showinfo("Taxi", "Your account has been
deleted",parent=self.master)
                app = Ct.CTkToplevel()
                self.master.destroy()
                import LoginView
                LoginView.LoginPage(app)
                app.after(0,lambda:app.state('zoomed'))
```

```
                app.lift()
                app.mainloop()


        except Exception as err:
            messagebox.showerror("Taxi", f" Failure: {err}",parent=self.master)

    def veiw_customer(self):
        try:
            cursor =self.connection.cursor()
            query = f'''SELECT
booking.bookingid,customer.firstname,customer.phonenumber,booking.pickup_address,book
ing.dropoff_address,booking.date,customer.Payment_Method,booking.status
            FROM customer
            JOIN booking
            ON customer.customerid = booking.customerid
            where booking.status = 'pending' '''
            cursor.execute(query)
            rows = cursor.fetchall()

            for item in self.view_booking.get_children():
                self.view_booking.delete(item)

            for row in rows:
                self.view_booking.insert(parent='', index='end', values=(row[0],
row[1], row[2], row[3], row[4],row[5],row[6],row[7]))

        except Exception as err:
            print(f"Error: {err}")


    def selectedRow(self,event):
        selected_item = self.view_booking.focus()
        values = self.view_booking.item(selected_item, "values")


        if values:
            self.bookingid.set(values[0])

            self.var1.set(values[7])

    def veiw_driver(self):
        try:
            with self.connection.cursor() as cursor:
                query = "SELECT `driverid` from driver where `status`='active' "
                cursor.execute(query)
```

```
                self.mydata = cursor.fetchall()
                self.mylist = [r for r, in self.mydata]

        except Exception as err:
            print(f"Error: {err}")
            messagebox.showerror("Taxi", f"Error fetching bookings:
{err}",parent=self.master)

    def assign_driver(self):
        try:
            cursor = self.connection.cursor()
            query = f"UPDATE `booking` SET
`status`='booked',`driverid`={self.var2.get()} WHERE `bookingid`
={self.bookingid.get()}"
            result =cursor.execute(query)
            query1 = f"UPDATE `driver` SET status='inactive' where
`driverid`={self.var2.get()}"
            result1 = cursor.execute(query1)
            self.connection.commit()
            self.veiw_driver()
            self.veiw_customer()
            messagebox.showinfo("Taxi", "Driver Assigned
Successfully",parent=self.master)
        except Exception as err:
            print(f"Error: {err}")
            messagebox.showerror("Taxi", f"Assigned Failure: {err}")



    def details_customer(self):
        try:
            with self.connection.cursor() as cursor:
                query = "SELECT * from customer "
                cursor.execute(query)
                rows = cursor.fetchall()

            for item in self.view_customer.get_children():
                self.view_customer.delete(item)

            for row in rows:
                self.view_customer.insert(parent='', index='end', values=(row[0],
row[1], row[2], row[3], row[4],row[5],row[6],row[7]))

        except Exception as err:
            print(f"Error: {err}")
```

```
    def historyy(self):
        try:
            with self.connection.cursor() as cursor:
                query = ''' SELECT customer.customerid, CONCAT(customer.firstname,'
',customer.lastname),customer.DOB,customer.Gender,customer.address,customer.Payment_M
ethod,booking.pickup_address,booking.dropoff_address,booking.date,booking.time,bookin
g.status,driver.fullname,driver.phonenumber FROM customer JOIN booking ON
customer.customerid = booking.customerid JOIN driver ON booking.driverid =
driver.driverid '''
                cursor.execute(query)
                rows = cursor.fetchall()

            for item in self.history_booking.get_children():
                self.history_booking.delete(item)

            for row in rows:
                self.history_booking.insert(parent='', index='end', values=(row[0],
row[1], row[2], row[3],
row[4],row[5],row[6],row[7],row[8],row[9],row[10],row[11],row[12]))

        except Exception as err:
            print(f"Error: {err}")

    def logout(self):
        self.master.destroy()
        app = Ct.CTkToplevel()
        app.after(0,lambda:app.state('zoomed'))
        LoginView.LoginPage(app)
        app.mainloop()


if __name__ == '__main__':
    app = Ct.CTk()
    Dashboard(app)
    app.after(0,lambda:app.state('zoomed'))
    app.mainloop()
```

RegistrationView.py

```
from tkcalendar import DateEntry
```

```
from tkinter import messagebox
import sys
sys.path.append("D:\Code\Python\python project\TaxBookingSystem")
from Model import CusRegistrationModel as cusmodel
from Controller import CustomerController as cusdb
from PIL import ImageTk, Image
import tkinter as tk
from tkinter import ttk
import valid

import customtkinter as CT

import LoginView
#create a registration class
class RegistrationPage():
    # create the constructor
    def __init__(self,master):
        self.master = master
        self.master.title("Customer Registration Page")#titlename
        self.master.geometry("1000x600")
        CT.set_default_color_theme("green")
        #set the window at top of the page
        self.master.attributes('-topmost',True)

        #create the background Image
        self.background_image = CT.CTkImage(Image.open('image\\registration.png'),
size=(550,600))

        self.img = CT.CTkLabel(self.master,image=self.background_image,
text="").place(x=0,y=0)

        self.id = 0
        #create the Label and Entry  for registration Page
        self.create =  CT.CTkLabel(self.master, text="Create a Account",
font=CT.CTkFont(family="Times",size=50, weight='bold'), text_color='#00BF63'
).place(x=580,y=10)
        self.first_name_CTkLabel = CT.CTkLabel(self.master, text="First
Name:",font=CT.CTkFont(family="Times",size=20))
        self.first_name_CTkLabel.place(x=630,y=80)
        self.first_name_CTkentry =
CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.first_name_CTkentry.place(x=770,y=80)

        self.last_name_CTkLabel = CT.CTkLabel(self.master, text="Last
Name:",font=CT.CTkFont(family="Times",size=20))
```

```
        self.last_name_CTkLabel.place(x=630,y=130)
        self.last_name_CTkentry =
CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.last_name_CTkentry.place(x=770,y=130)

        self.email_CTkLabel = CT.CTkLabel(self.master,
text="Email:",font=CT.CTkFont(family="Times",size=20))
        self.email_CTkLabel.place(x=630,y=430)
        self.email_CTkentry = CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.email_CTkentry.place(x=770,y=430)

        self.address_CTkLabel = CT.CTkLabel(self.master,
text="Address:",font=CT.CTkFont(family="Times",size=20))
        self.address_CTkLabel.place(x=630,y=180)
        self.address_CTkentry = CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.address_CTkentry.place(x=770,y=180)

        self.date_CTklabel = CT.CTkLabel(self.master,text =
'DOB:',font=CT.CTkFont(family="Times",size=20))
        self.date_CTklabel.place(x=630,y=230)
        self.date_CTkEntry = DateEntry(self.master,width=30,selectmode='day')
        self.date_CTkEntry.place(x=960,y=300)

        self.gender_CTkLabel = CT.CTkLabel(self.master,
text="Gender:",font=CT.CTkFont(family="Times",size=20))
        self.gender_CTkLabel.place(x=630,y=280)
        self.var_gender = CT.StringVar()
        self.radio_btn =CT.CTkRadioButton(self.master, text="Male",
variable=self.var_gender, value="Male")
        self.radio_btn.place(x=770, y=285)
        self.radio_btn1 = CT.CTkRadioButton(self.master, text="Female",
variable=self.var_gender, value="Female")
        self.radio_btn1.place(x=870, y=285)

        self.pay1=CT.CTkLabel(self.master,text='Payment
Method:',font=CT.CTkFont(family="Times",size=20))
        self.pay1.place(x=630,y=330)

        self.var=CT.StringVar()
        self.var.set("Select payment method")

        self.pay=tk.OptionMenu(self.master,self.var,"Cash", "Banking", "Esewa")
        self.pay.place(x=1000,y=415)
```

```
        self.phone_CTkLabel = CT.CTkLabel(self.master, text="Phone
Number:",font=CT.CTkFont(family="Times",size=20))
        self.phone_CTkLabel.place(x=630,y=380)
        self.phone_CTkentry = CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.phone_CTkentry.place(x=770,y=380)

        self.pass_CTkLabel = CT.CTkLabel(self.master,
text="Password:",font=CT.CTkFont(family="Times",size=20))
        self.pass_CTkLabel.place(x=630,y=480)
        self.pass_CTkentry = CT.CTkEntry(self.master,textvariable=CT.StringVar())
        self.pass_CTkentry.place(x=770,y=480)
        # Repeat this pattern for the remaining fields

        #create the buttons
        register_CTkbutton = CT.CTkButton(self.master, text="Register",
font=CT.CTkFont(family="Times", size=20), command=self.register)
        register_CTkbutton.place(x=600,y=530)
        back_CTkbutton = CT.CTkButton(self.master,
text="Back",font=CT.CTkFont(family="Times", size=20), command=self.Back)
        back_CTkbutton.place(x=770,y=530)

        #create the register function
    def register(self):
        self.gender = "Male" if self.var_gender.get() == 'Male' else "Female"
        self.pay_method = self.var.get()
        if self.first_name_CTkentry.get()!='' and
self.last_name_CTkentry.get()!=''and self.var_gender.get()!=''and self.var.get!=''
and self.phone_CTkentry.get()!='' and self.email_CTkentry.get()!='' and
self.address_CTkentry.get()!='' and self.date_CTkEntry.get_date()!='' and
self.pass_CTkentry.get()!='':
            if valid.checkemail(self.email_CTkentry.get())==True:

                cus = cusmodel.Customer(self.id,
self.first_name_CTkentry.get(),self.last_name_CTkentry.get(),self.phone_CTkentry.get(
),self.pay_method,self.gender,self.address_CTkentry.get(),self.date_CTkEntry.get_date
(),self.email_CTkentry.get(),self.pass_CTkentry.get())
                reg = cusdb.CustomerDatabase()
                reg._CustomerRegister(cus)
                if reg:
                    self.first_name_CTkentry.delete(0,'end')
                    self.last_name_CTkentry.delete(0,'end')
                    self.email_CTkentry.delete(0,'end')
                    self.phone_CTkentry.delete(0,'end')
                    self.address_CTkentry.delete(0,'end')
                    self.pass_CTkentry.delete(0,'end')
```

```
                    messagebox.showinfo('register','Registration
Successfully',parent=self.master)
                    self.master.destroy()
                else:
                    messagebox.showerror("register","Register
Failure",parent=self.master)
            else:
                messagebox.showinfo("register","please write a valid email
address",parent=self.master)
        else:
                messagebox.showinfo("register","please write a valid
information",parent=self.master)
    def Back(self):
        self.master.destroy()
if __name__ == "__main__":

    app = CT.CTk()
    RegistrationPage(app)
    # app.eval('tk::PlaceWindow . top')
    app.mainloop()
```

GobalVariable.py

```
Customer = None
Driver = None
Admin = None
email = None
```

Valid.py

```
import re

def checkemail(email):
    regex = re.compile(r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-
z]{2,})+')
    if re.fullmatch(regex, email):
        emailResult=True

    else:
        emailResult=False

    return emailResult
```

```
def checkphone(mobile):
    regex=re.compile("^(?:0|\+?977)\s?(?:\d\s?){9,11}$")
    if re.fullmatch(regex, mobile):
        mobileResult=True
    else:
        mobileResult=False

    return mobileResult

def checkcredit(credit):
    regex=re.compile("(?:[0-9]{4}-){3}[0-9]{4}|[0-9]{16}")
    if re.fullmatch(regex, credit):
        creditResult=True
    else:
        creditResult=False
    return creditResult


def namevalidation(name):
    regex=re.compile("^([a-zA-Z]{2,}\s[a-zA-Z]{1,}'?-?[a-zA-Z]{2,}\s?([a-zA-
Z]{1,})?)")
    if re.fullmatch(regex, name):
        nameResult=True
    else:
        nameResult=False
    return nameResult


def passwordvalidation(password):
    regex=re.compile("^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$")
    if re.fullmatch(regex, password):
        passwordResult=True
    else:
        passwordResult=False
    return passwordResult
```

CusRegistartionModel.py

```
class Customer:
    #constructor
    def
__init__(self,_cusid=None,_first=None,_last=None,_phone=None,_payment_method=None,_Ge
nder=None,_address=None,_dob=None,_email=None,_password=None):
        self.cusid = _cusid
        self.first = _first
        self.last = _last
        self.phone = _phone
        self.payment = _payment_method
        self.Gender = _Gender
        self.address = _address
        self.dob = _dob
        self.email = _email
        self.password = _password

    #getters
    def getId(self):
        return self.cusid

    def getFirst(self):
        return self.first

    def getLast(self):
        return self.last

    def getPhone(self):
        return self.phone

    def getPayment(self):
        return self.payment

    def getGender(self):
        return self.Gender

    def getAddress(self):
        return self.address

    def getDOB(self):
        return self.dob

    def getEmail(self):
        return self.email

    def getPassword(self):
```

```
        return self.password

    #setters

    def setId(self, Uid):
        self.cusid = Uid

    def setFirst(self,first ):
        self.first = first

    def setLast(self, last):
        self.last = last

    def setPhone(self, phone):
        self.phone = phone

    def setPayment(self, payment):
        self.payment = payment

    def setGender(self, gender):
        self.Gender = gender


    def setAddress(self, Address):
        self.address = Address

    def setDOB(self, DOB):
        self.dob = DOB

    def setEmail(self, email):
        self.email = email

    def setPassword(self, password):
        self.password = password
```

LoginModel.py

```
class Login:
    def __init__(self,_email=None,_password=None):
        self.email = _email
        self.password = _password

    #getters
```

```
    def getEmail(self):
        return self.email

    def getPassword(self):
        return self.password

    #setters
    def setEmail(self, email):
        self.email = email

    def setPassword(self, password):
        self.password = password
```

VerifyModel.py

```
class Email:
    def __init__(self,_email):
        self.email = _email
    #getters
    def getEmail(self):
        return self.email
    #setters
    def setEmail(self, email):
        self.email = email
```

AdminController.py

```
# import the file and tkinter library
from tkinter import messagebox
from Controller import DataBaseConnection as con
#create a  AdminDatabase Class
class AdminDatabase:

    __connection__ = None
    #create a constructor
    def __init__(self):
        self.__connection__ = con.Database.Connect()
    #create  a Adminlogin Methods
    def AdminLogin(self,LoginAdmin):
        try:
```

```
            cursor = self.__connection__.cursor()
            #write a Select Query
            cursor.execute("SELECT * FROM admin WHERE
email='"+LoginAdmin.getEmail()+"' AND password='"+LoginAdmin.getPassword()+"'")
            record = cursor.fetchone()

        except Exception as e:
            print(e)
        return record
```

CustomerController.py

```
from tkinter import messagebox
from Controller import DataBaseConnection as con
class CustomerDatabase:
    __connection__ = None

    def __init__(self):
        self.__connection__ = con.Database.Connect()

    def _CustomerRegister(self,CusRegister):
        try:
            cursor = self.__connection__.cursor()
            query  = f"INSERT INTO `customer`(`customerid`, `firstname`, `lastname`,
`email`, `DOB`,`Gender`,`phonenumber`, `address`,`Payment_Method`,`password`) VALUES
('{CusRegister.getId()}','{CusRegister.getFirst()}','{CusRegister.getLast()}','{CusRe
gister.getEmail()}','{CusRegister.getDOB()}','{CusRegister.getGender()}','{CusRegiste
r.getPhone()}','{CusRegister.getAddress()}','{CusRegister.getPayment()}','{CusRegiste
r.getPassword()}')"
            cursor.execute(query)
            self.__connection__.commit()
            cursor.close()
            return True
        except Exception as e:
            print(e)
            return False

    def CustomerLogin(self,LoginCustomer):
        try:
            cursor = self.__connection__.cursor()
            cursor.execute("SELECT * FROM customer WHERE
email='"+LoginCustomer.getEmail()+"' AND password='"+LoginCustomer.getPassword()+"'")
            record = cursor.fetchone()
        except Exception as e:
```

```
            print(e)
        return record

    def _isValidEmail(self,email):
        cursor = self.__connection__.cursor()
        cursor.execute("SELECT * FROM customer WHERE email='"+email.getEmail()+"'")
        record = cursor.fetchone()
        if (record!=None):
            return True
        return False

    def _ChangePassword(self,email):
        try:
            cursor = self.__connection__.cursor()
            cursor.execute("UPDATE customer SET password ='"+email.getPassword()+"'
WHERE email='"+email.getEmail()+"'")
            record = cursor.fetchone()
            self.__connection__.commit()
            cursor.close()
            return True
        except Exception as e:
            print(e)
```

DatabaseConnection.py

```
import mysql.connector as c
class Database:
    message = ""
    try:
        @staticmethod
        def Connect():
            connection = c.connect(
                host='localhost',
                user = 'root',
                password = 'root',
                database = 'taxi_booking_system',
            )
            if connection.is_connected():
                Database.message = "Connected"
                return connection
            connection.close()
    except Exception as e:
        print(e)
```

DriverController.py

```
from tkinter import messagebox
from Controller import DataBaseConnection as con

class DriverDatabase:
    __connection__ = None
    def __init__(self):
        self.__connection__ = con.Database.Connect()

    def DriverLogin(self,LoginDriver):
        try:
            cursor = self.__connection__.cursor()
            cursor.execute("SELECT * FROM driver WHERE
email='"+LoginDriver.getEmail()+"' AND password='"+LoginDriver.getPassword()+"'")
            record = cursor.fetchone()

        except Exception as e:
            print(e)

        return record
```