

**CIS020-1 – Introduction to Software Development**

**CIS093-1 – Mathematics and Concepts for Computational Thinking**

**Assignment 2 – Group / Individual Project –CaseStudy**

**(Taxi Booking System)**

**Student Name: Baibhav Paudel - 2146504X - University ID: 012345**

**Student Name: XXXX XXXXX - University ID: 012345**

**OR**

**Student Name: XXXX XXXXX - University ID: 012345**

**Table of Contents**

[Introduction/Overview 3](#__RefHeading___Toc42041_3358107187)

[Task Description 3](#__RefHeading___Toc42043_3358107187)

[Assumptions about the system : 4](#__RefHeading___Toc42045_3358107187)

[Project Plan/Schedule 4](#__RefHeading___Toc42047_3358107187)

[Requirements Analysis 5](#__RefHeading___Toc42153_3358107187)

[Functional Requirements 5](#__RefHeading___Toc42155_3358107187)

[Non-functional Requirements 6](#__RefHeading___Toc42261_3358107187)

[Usability Requirements 6](#__RefHeading___Toc42263_3358107187)

[Design 6](#__RefHeading___Toc42265_3358107187)

[UML Diagrams 6](#__RefHeading___Toc42267_3358107187)

[Uses Case Diagrams(s) 7](#__RefHeading___Toc42269_3358107187)

[10](#__RefHeading___Toc3282_1066940643)

[Class Diagram(s) 12](#__RefHeading___Toc42463_3358107187)

[Database Design 15](#__RefHeading___Toc42475_3358107187)

[15](#__RefHeading___Toc3284_1066940643)

[Logical Database Design 15](#__RefHeading___Toc42477_3358107187)

[Physical Database Design 16](#__RefHeading___Toc42485_3358107187)

[Skeleton Tables (with Primary Keys and Foreign Keys) 16](#__RefHeading___Toc42487_3358107187)

[Data Dictionary 17](#__RefHeading___Toc1517_2908959286)

[User Interface Design 20](#__RefHeading___Toc42491_3358107187)

[Algorithm Design 25](#__RefHeading___Toc42493_3358107187)

[Implementation 28](#__RefHeading___Toc1519_2908959286)

[Testing 45](#__RefHeading___Toc43111_1759767037)

[Test No 45](#__RefHeading___Toc42495_3358107187)

[Test Date 45](#__RefHeading___Toc42497_3358107187)

[Purpose of test 45](#__RefHeading___Toc42499_3358107187)

[Input data or action 45](#__RefHeading___Toc42501_3358107187)

[Expected result 45](#__RefHeading___Toc42503_3358107187)

[Actual result 45](#__RefHeading___Toc42505_3358107187)

[Action if the above are not the same 45](#__RefHeading___Toc42507_3358107187)

[Screenshot of output 45](#__RefHeading___Toc42509_3358107187)

[Discussion / Reflection / Critical Analysis 46](#__RefHeading___Toc43137_1759767037)

[Conclusion 47](#__RefHeading___Toc42058_1066940643)

[References 47](#__RefHeading___Toc42060_1066940643)

[Appendix 48](#__RefHeading___Toc43143_1759767037)

[PACKAGES AND MODULES 48](#__RefHeading___Toc42360_1066940643)

[SOURCE CODE OF TURBO TAXI BOOKING SYSTEM 48](#__RefHeading___Toc42358_1066940643)

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

CIS093-1 – Mathematics and Concepts for Computational Thinking – 2021-2022  
Assignment 2 – Individual Project – Case Study (Taxi Booking System)

# Introduction/Overview

The Taxi Booking System project, together with the project brief and requirements to execute this assignment, was the final task handed to us. Our primary objective in order to finish Assignment 2 was to create a system for booking taxis. In order to book a cab, a user must first register and sign in using login information (username and password), hence a user registration system and a user login system were created. There must be a way for the system user to reserve a taxi for their journey, thus a booking system was developed. A mechanism for admin control was created to allocate cabs to user booking requests. A driver dashboard was established where he or she could monitor assigned bookings, mark them as completed when the journey was finished, and view forthcoming bookings. A database was made and the system was connected to it using a database connector in order to keep all of the user information and details in the system.

# Task Description

The main aim was to create software for a taxi company so that customers could schedule cabs and view their upcoming journeys online using a desktop application's graphical user interface. All the important data regarding user, booking, taxi, and the driver had to be kept in an external database system.

The user must first register as a user with accurate information in order to make a cab reservation. After registering, the user must be able to input the location, date, and time of their pickup as well as the destination of their journey throughout the booking process.

The system must have a capability that allows users to update or cancel their reservations and view forthcoming trips through a graphical user interface. Admin from the business side must

be allowed to approve the user's booking once they submit it and choose a taxi driver to complete the booking. Each driver must have a single forthcoming trip.

## Assumptions about the system :

* The cost of a journey is determined during the booking process based on the distance between the pickup place and the destination.
* Credit card payments will be added in the driver's account balance.
* The driver is paid immediately once the journey is finished.
* Even after cabs have been assigned, reservations can still be canceled, but only with the admin's permission if an issue occurs.
* Taxi assignments cannot be changed.
* The driver keeps his or her earnings throughout the month, but at the end of the month, he or she is required to pay the corporate service fee which has already been computed and is shown to him in his driver dashboard in order to continue using the system the following month.

# Project Plan/Schedule

|  |  |  |
| --- | --- | --- |
| Week No. | Tasks | Priority |
| 1 | → Will familiarize myself with Python and Tkinter.  → Additionally, I'll establish a development environment and list the characteristics that the taxi booking system must have (text editor, version control). | Must |
| 2 | → will utilize Balsamiq to create the user interface for the taxi booking system.  →I'll use Tkinter to create a user interface that is comparable to the design. | Must |
| 3 | → I'll also implement the user registration and login features. | Must |
| 4 | →I'll put the taxi booking feature into action.  →Users will be able to search for and choose a taxi, and I'll also integrate the payment option. | Must |
| 5 | →I'll add the driver management functionality during the fourth week.  → The allocated reservations will be visible to drivers. | Must |
| 6 | → I'll check the taxi booking system's operation, troubleshoot any problems, and solve them.  → Create the project's documentation. | Should |
| 7 | → Once the project is finished, I will present my institution with the taxi reservation system.  → I'll also turn in the project for evaluation. | Must |

# Requirements Analysis

## Functional Requirements

TTBS = Turbo Taxi Booking System

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | A signup system allows customers to register. | MUST |
| 2 | To access the system, users must log in. | MUST |
| 3 | Request booking from booking form. | MUST |
| 4 | User and reservation information is saved to a separate file. | MUST |
| 5 | An administrator must be able to add a cab to a user's reservation. | MUST |
| 6 | All booking requests will be visible to the administrator. | MUST |
| 7 | Bookings must be able to be canceled by admin. | MUST |
| 8 | The user must be able to see information about forthcoming bookings. | MUST |
| 9 | A booking must be able to be cancelled by the user. | MUST |
| 10 | Booking specifics must to be editable by the user. | SHOULD |
| 11 | he ability to log into the system is required for cab drivers. | MUST |
| 12 | An forthcoming journey must be visible to taxi drivers. | MUST |
| 13 | It should be possible for taxi drivers to see their earnings. | SHOULD |

## Non-functional Requirements

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | The system's processing speed should be high. | MUST |
| 2 | Must be screen-adaptive for devices. | SHOULD |
| 3 | System upgrades must be possible. | MUST |
| 4 | must be free of crashes or run-time problems. | MUST |

## Usability Requirements

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Priority\* |
| 1 | System GUI must be user friendly. | MUST |
| 2 | All forms for entering data should be brief, simple to complete, and include entry validation. | MUST |

# Design

## UML Diagrams

### Uses Case Diagrams(s)

Diagram

Description automatically generatedFigure 1: Taxi\_Booking\_System\_Usecase\_diagram

(Visual Paradigm,2021)

“A UML use case diagram is the primary form of system/software requirements for a new software program underdeveloped”.

Use Case Name: Taxi Booking System

* Description: This use case allows a user to book a taxi from their pickup location to a specified destination.
* Actors: Customer, Admin, Driver

Conditions:

* The user must registered and login into the system.
* The user must confrim with their trip details.
* There is at least one taxi available for booking in the system.

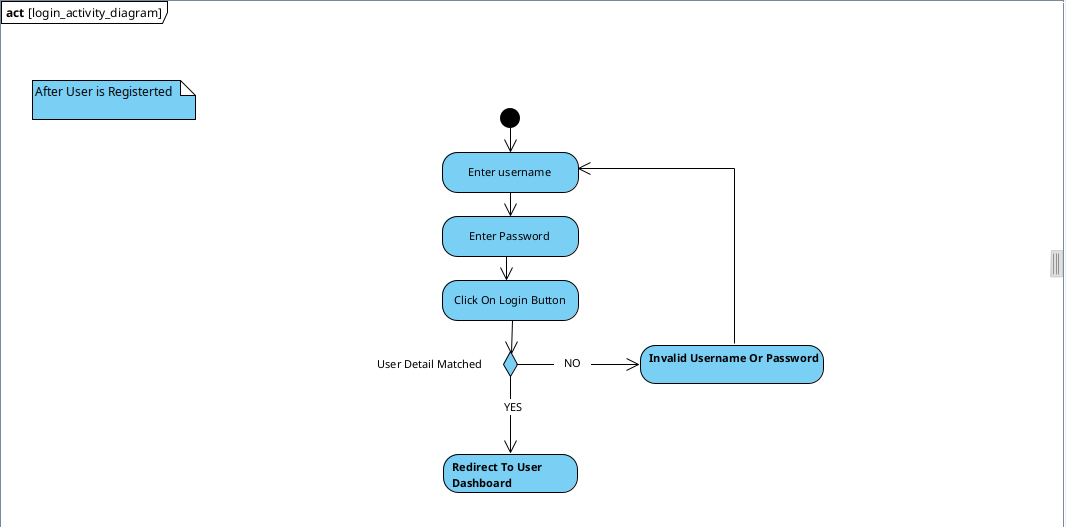
Flow of Events:

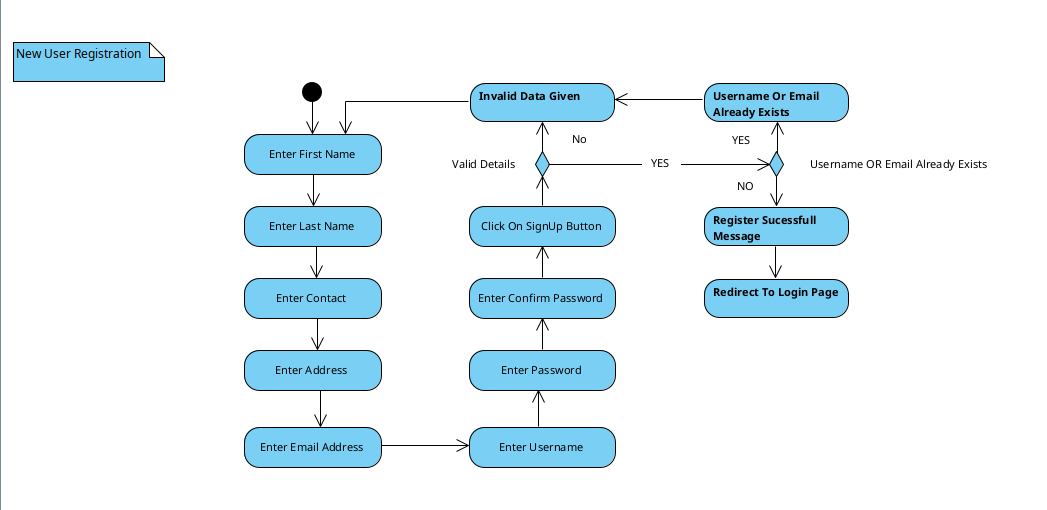
1. The user selects the "Book Taxi" option from the user dashboard.
2. The system allows the user to enter their trip detail through a booking form.
3. The user enters their booking details.
4. In order to allocate a cab to a booking, the admin checks the booking request.
5. The system displays a list of available taxis, along with their driver's name, car model, and estimated fare.
6. The admin assign a taxi from the list.
7. The system confirms the booking and displays the pickup location and estimated arrival time for the taxi.
8. If the user cancels the booking before taxi is assigned, the system will cancel the booking.

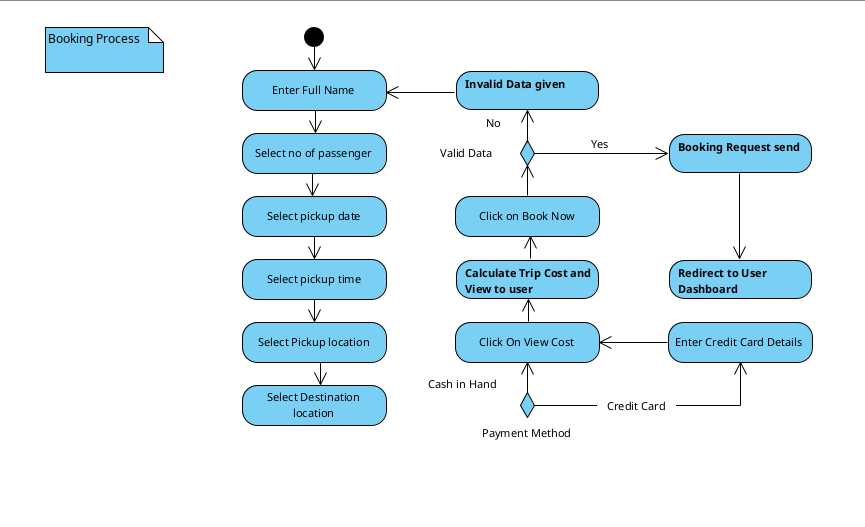
Postconditions:

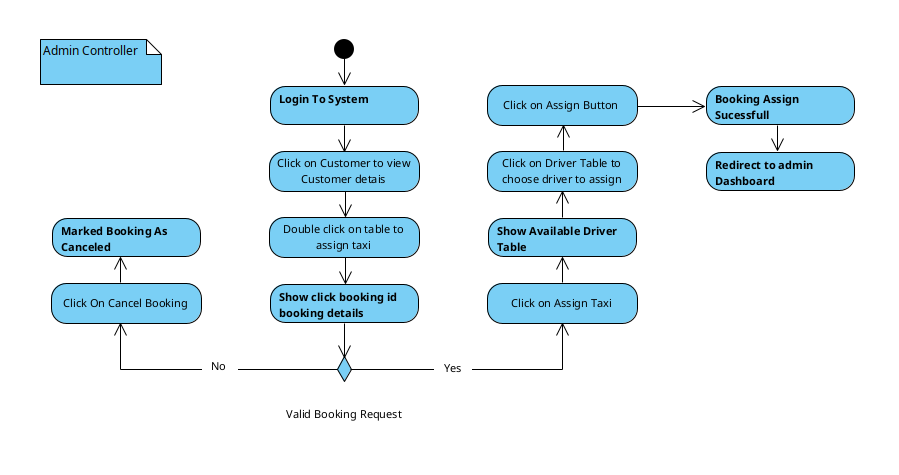
1. The user's booking is recorded in the system.
2. The driver of the selected taxi is notified of the booking and given the pickup location and destination along with all booking details.
3. The user receives updated status that the reservation has been approved and has access to the allocated driver's details.

Activity Diagram(s)

Figure 2: Login\_System\_Activity\_Diagram

Figure 3: Register\_System\_Activity\_Diagram

Figure 4: Booking\_System\_Activity\_Diagram

Figure 5: Admin\_Control\_System\_Activity\_Diagram

### Class Diagram(s)

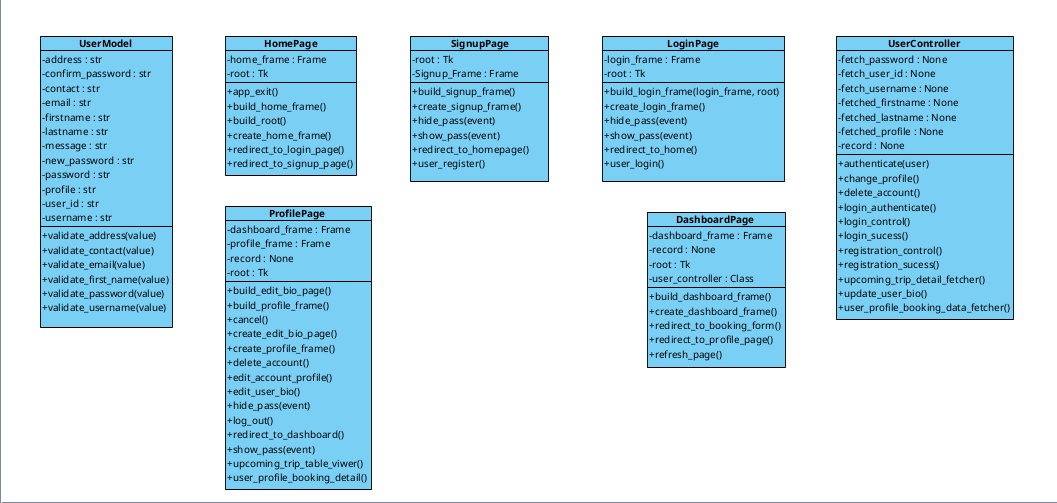
Figure 6: user\_class\_diagram

Figure 7: admin\_class\_diagram

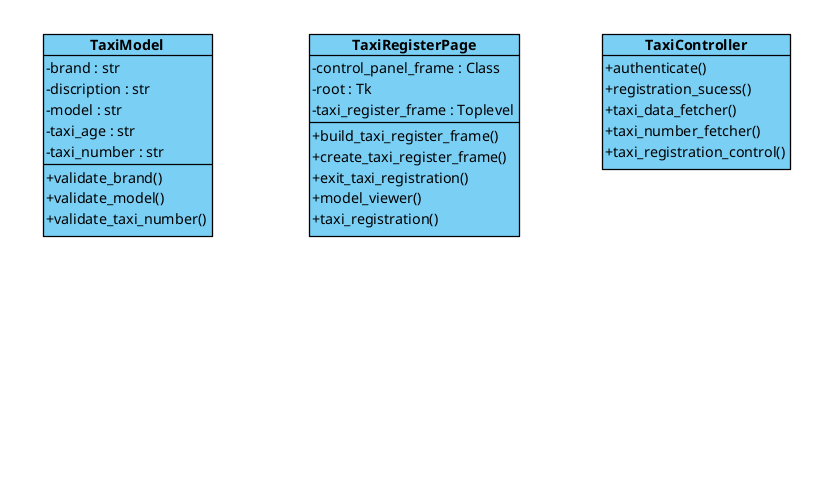
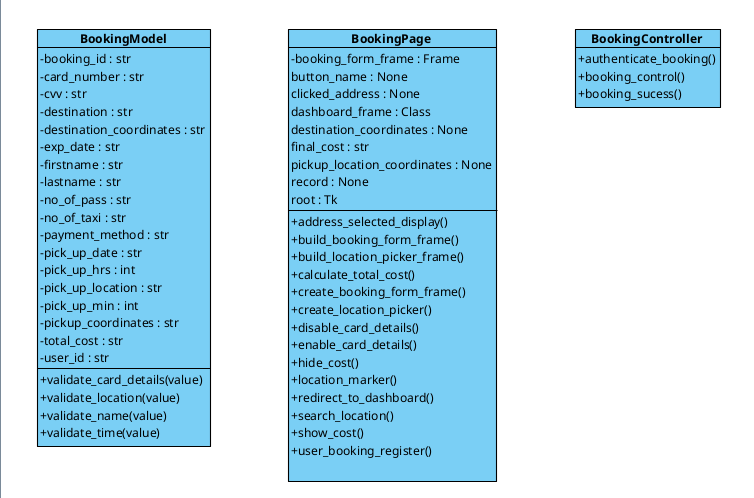
Figure 8: taxi\_class\_diagram

Figure 9: driver\_class\_diagram

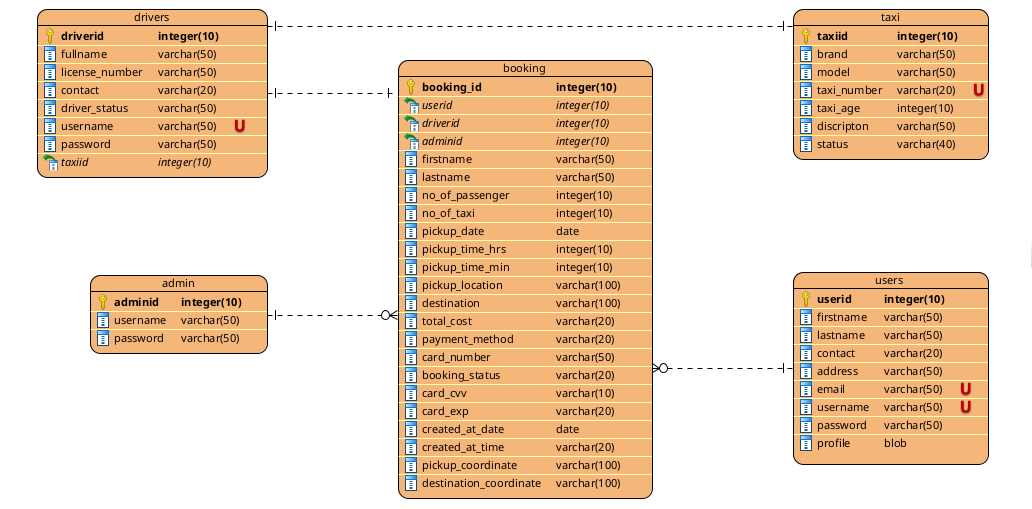
Figure 10: booking\_class\_diagram

(IBM, 2021)

“Class diagrams are fundamental to the object modeling process and model the static structure of a system”.

Class diagrams are the blueprints of a system or subsystem. They can be used to model the objects that make up the system, display the relationships between the objects, and describe the objects' behaviors and the services they provide.

## Database Design

Figure 11: turbo\_tb\_database\_ERD

### Logical Database Design

Diagram

Description automatically generatedFigure 12: databae\_ERM

### Physical Database Design

#### Skeleton Tables (with Primary Keys and Foreign Keys)

Users(**userid**, firstname, lastname, contact, address, email, username,

password, profile)

Booking (**booking\_id**, userid\*, driverid\*, adminid\*, firstname,lastname, no\_of\_passenger, no\_of\_taxi, pickup\_date, pickup\_time\_hrs, pickup\_time\_min, pickup\_location, destination, total\_cost, payment\_method, card\_number, booking\_status, card\_cvv, card\_exp, created\_at\_date, created\_at\_time, pickup\_coordinates, destination\_coordinates)

Taxi (**taxiid**, brand, model, taxi\_number, taxi\_age, discription, status)

driver (**driverid**, fullname, license\_number, contact, driver\_status, username, password, taxiid\*)

Admin (**adminid**, username, password)

#### Data Dictionary

Table 1: user\_data\_dicrionary\_table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Users** | | | | | |
| **Discription :** Users Details | | | | | |
| **Field Name** | **Data Type** | **Length** | **Index** | **Null** | **Discription** |
| **Userid**  **(Primary)** | **int** | **10** | **PK** | **No** | **Auto Increment** |
| **firstname** | **varchar** | **50** |  | **No** | **Firstname of user** |
| **lastname** | **varchar** | **50** |  | **N0** | **Lastname of user** |
| **contact** | **varchar** | **20** |  | **No** | **Contact detail of user** |
| **address** | **varchar** | **50** |  | **No** | **User’s location detail** |
| **email** | **varchar** | **50** | **U** | **No** | **Valid email address of user** |
| **username** | **varchar** | **50** | **U** | **No** | **Username for login** |
| **password** | **varchar** | **50** |  | **No** | **Password for personal security** |
| **profile** | **blob** |  |  | **No** | **User’s profile picture** |

Table 2: booking\_data\_dictionary\_table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Booking** | | | | | |
| **Discription :** Booking details made by user | | | | | |
| **Field Name** | **Data Type** | **Length** | **Index** | **Null** | **Discription** |
| **booking\_id**  **(Primary)** | **int** | **10** | **PK** | **No** | **Auto Increment** |
| **userid** | **int** | **10** | **Fk** | **No** | **User table foreign key** |
| **driverid** | **int** | **10** | **Fk** |  | **drivers table foreign key** |
| **adminid** | **int** | **10** | **Fk** |  | **admin table foreign key** |
| **firstname** | **varchar** | **50** |  | **No** | **User’s first name** |
| **lastname** | **varchar** | **50** |  | **No** | **User’s last name** |
| **no\_of\_passenger** | **int** | **50** |  | **No** | **No of passenger in a trip** |
| **pickup\_date** | **date** | **50** |  | **No** | **Date for the pickup** |
| **pickup\_time\_hrs** | **int** | **10** |  | **No** | **Time for the pickup HRS** |
| **pickup\_time\_min** | **int** | **10** |  | **No** | **Time for the pickup MIN** |
| **pickup\_location** | **varchar** | **100** |  | **No** | **Location to pickup** |
| **destination** | **varchar** | **100** |  | **No** | **Final destination of trip** |
| **total\_cost** | **varchar** | **20** |  | **No** | **Total cost to be paid** |
| **payment\_method** | **varchar** | **20** |  | **No** | **Method of payment** |
| **card\_number** | **varchar** | **50** |  |  | **Credit card number** |
| **card\_cvv** | **varchar** | **20** |  |  | **Credit card cvv** |
| **card\_exp** | **varchar** | **10** |  |  | **Credit card exp date** |
| **created\_at\_date** | **date** |  |  | **No** | **Booking created date** |
| **created\_at\_time** | **varchar** | **20** |  | **No** | **Booking created time** |
| **pickup\_coordinates** | **varchar** | **100** |  | **No** | **Pickup location coordinates** |
| **destination\_coordinates** | **varchar** | **100** |  | **No** | **Destination coordinates** |
| **booking\_status** | **varchar** | **20** |  | **No** | **Status of booking request** |

Table 3: admin\_data\_dictionary\_table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Admin** | | | | | |
| **Discription :** Admin Details | | | | | |
| **Field Name** | **Data Type** | **Length** | **Index** | **Null** | **Discription** |
| **adminid**  **(Primary)** | **int** | **10** | **PK** | **No** | **Auto Increment** |
| **username** | **varchar** | **50** |  | **No** | **Username to login to system** |
| **password** | **varchar** | **50** |  | **N0** | **Account password** |

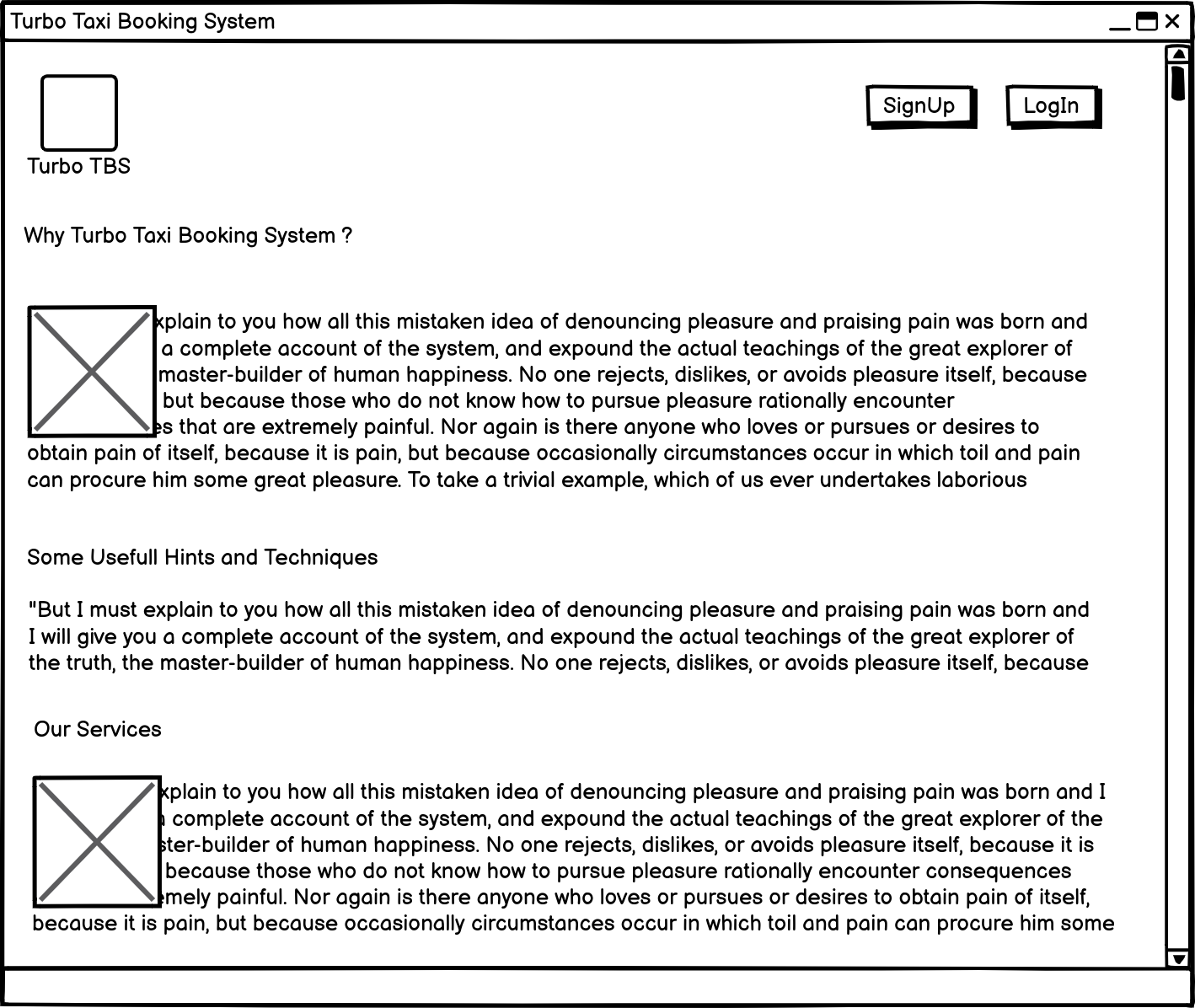
Table 4: taxi\_data\_dictionary\_table

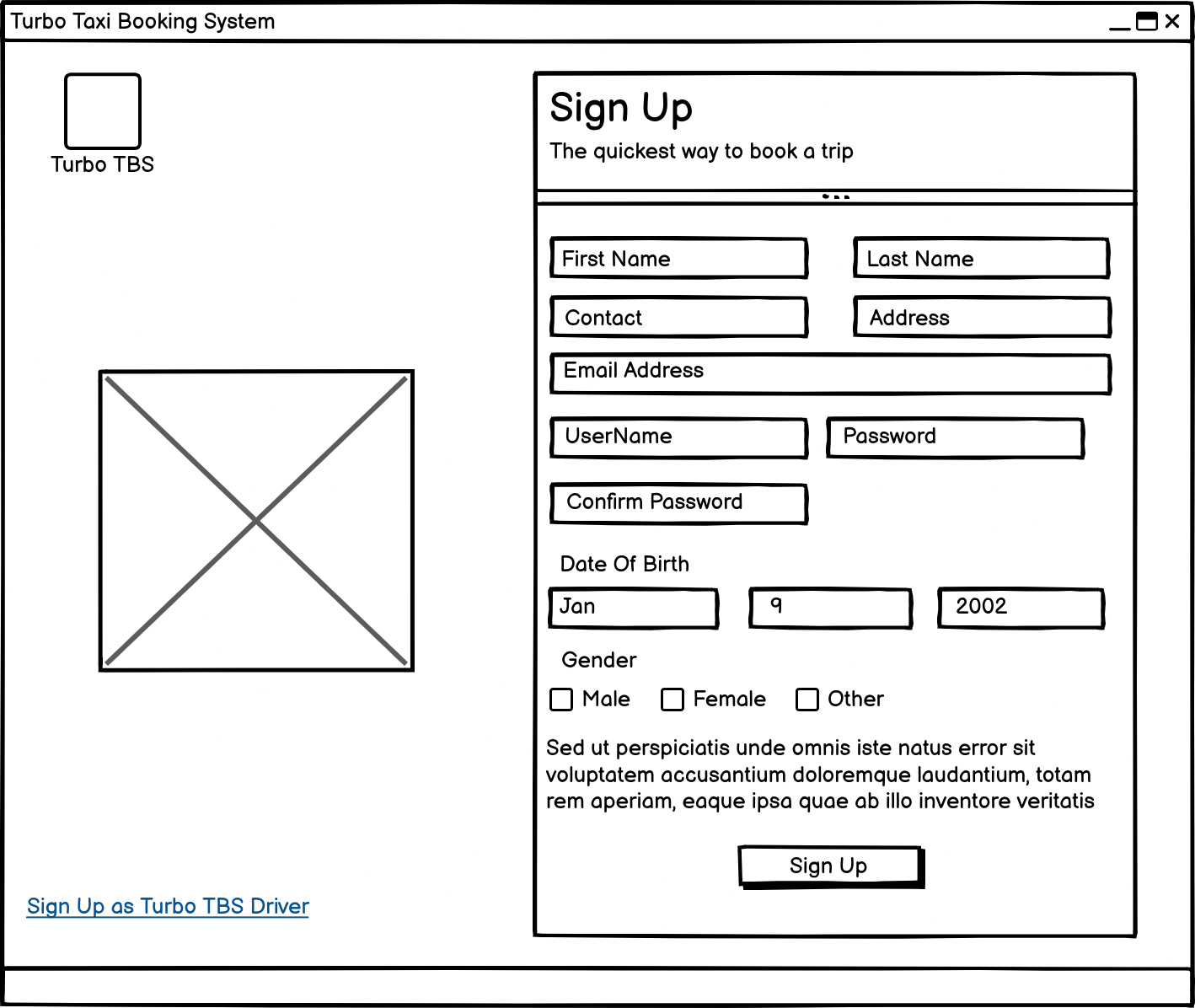
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Taxi** | | | | | |
| **Discription :** Taxi Details | | | | | |
| **Field Name** | **Data Type** | **Length** | **Index** | **Null** | **Discription** |
| **taxiid**  **(Primary)** | **int** | **10** | **PK** | **No** | **Auto Increment** |
| **brand** | **varchar** | **50** |  | **No** | **Brand of a taxi** |
| **model** | **varchar** | **50** |  | **N0** | **Model of a taxi brand** |
| **taxi\_number** | **varchar** | **20** |  | **No** | **Taxi number** |
| **taxi\_age** | **int** | **10** |  | **No** | **Used time of taxi** |
| **discription** | **varchar** | **50** |  | **No** | **Details about taxi** |
| **status** | **varchar** | **40** |  | **No** | **Taxi assign status** |

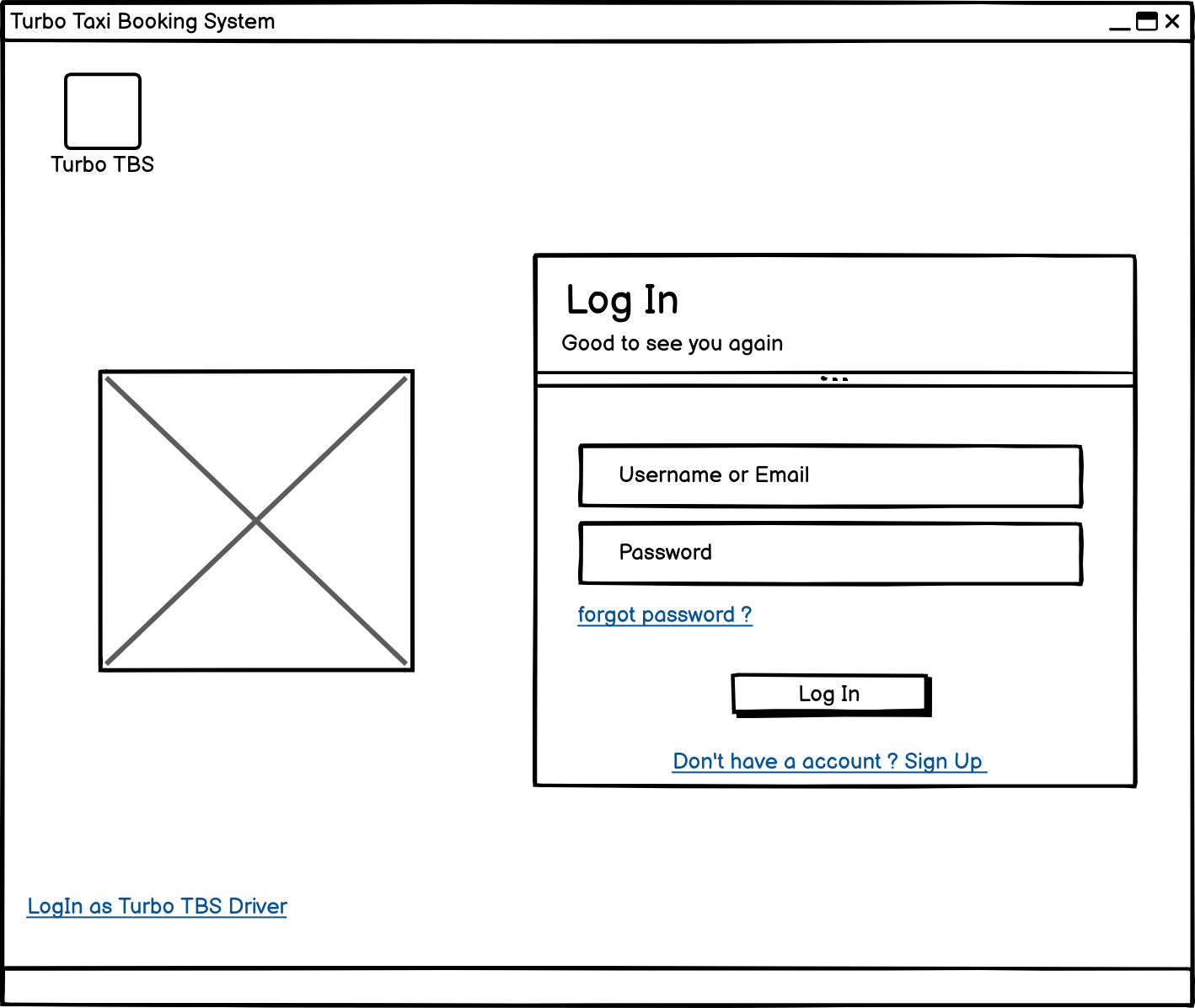
Table 5: drivers\_data\_dictionary\_table

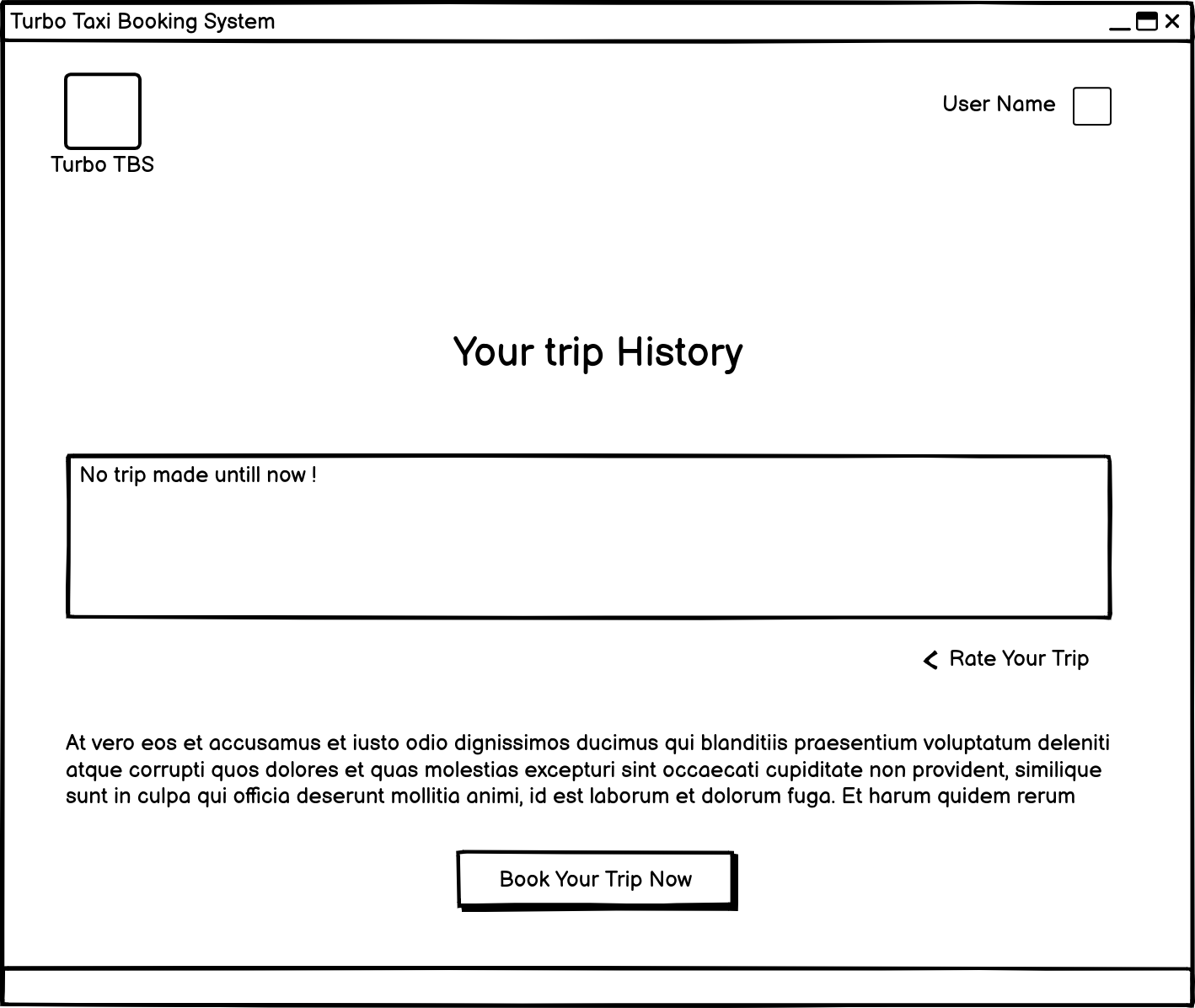
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Drivers** | | | | | |
| **Discription :** Drivers Details | | | | | |
| **Field Name** | **Data Type** | **Length** | **Index** | **Null** | **Discription** |
| **driverid**  **(Primary)** | **int** | **10** | **PK** | **No** | **Auto Increment** |
| **fullname** | **varchar** | **50** |  | **No** | **Firstname of driver** |
| **license\_number** | **varchar** | **50** |  | **N0** | **Lastname of driver** |
| **contact** | **varchar** | **20** |  | **No** | **Contact detail of driver** |
| **driver\_status** | **varchar** | **50** |  | **No** | **Driver’s assign status** |
| **username** | **varchar** | **50** | **U** | **No** | **Username for login** |
| **password** | **varchar** | **50** | **U** | **No** | **Password for personal security** |
| **taxiid** | **int** | **10** | **FK** | **No** | **Foreign key from taxi table** |

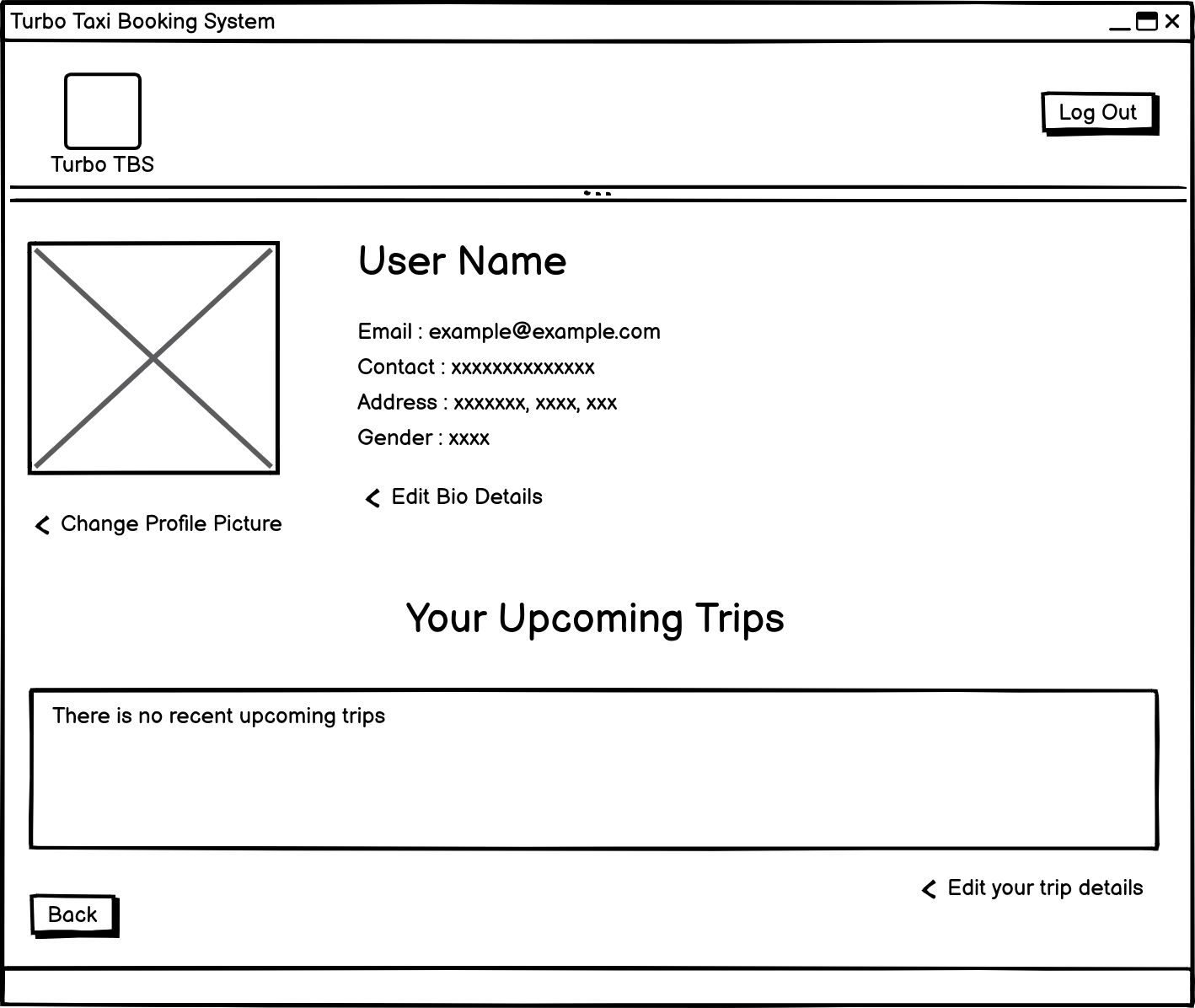
## User Interface Design

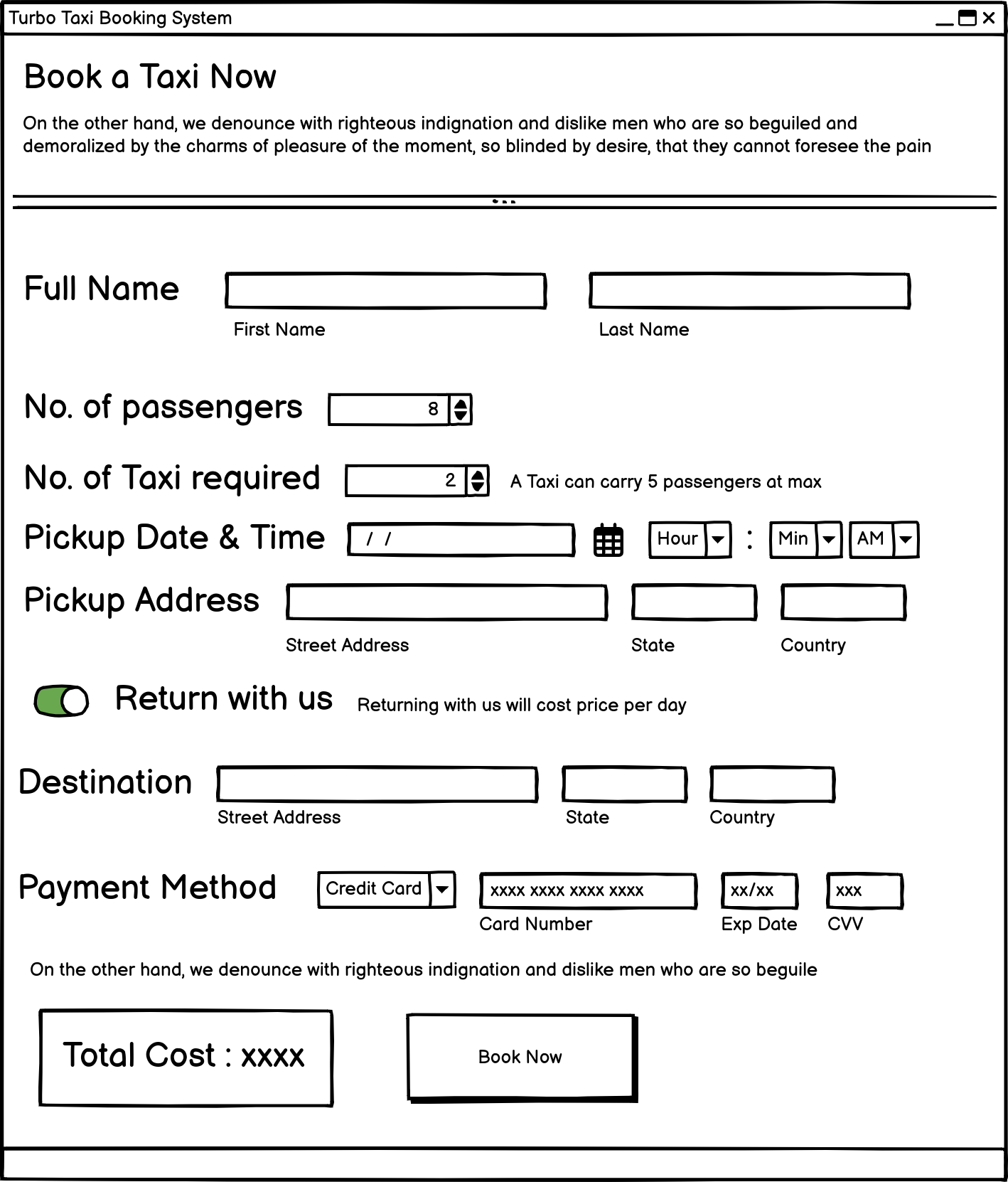
Figure 13: homepage\_interface\_design

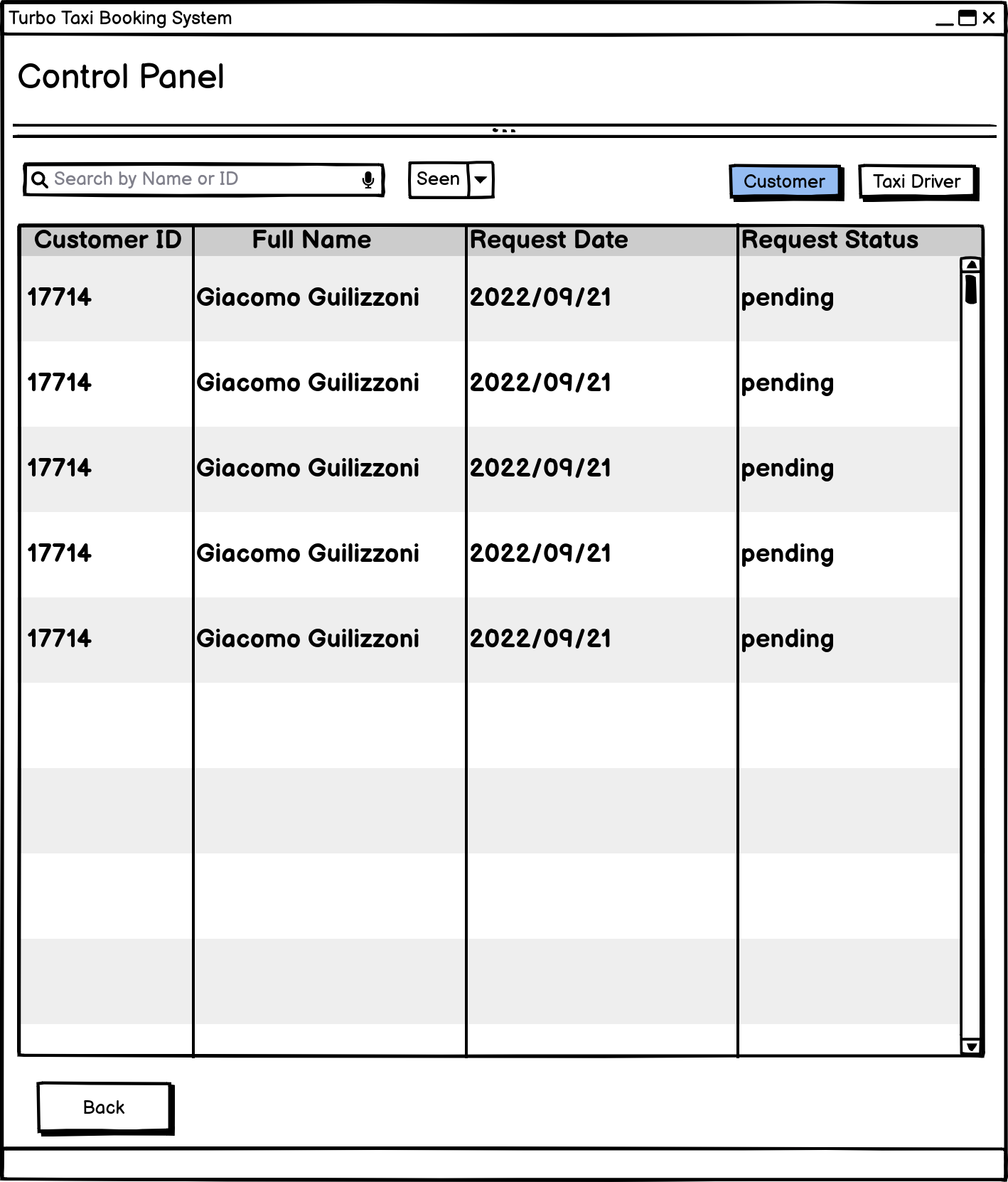
Figure 14: user\_signup\_interface\_design

Figure 15: user\_login\_interface\_design

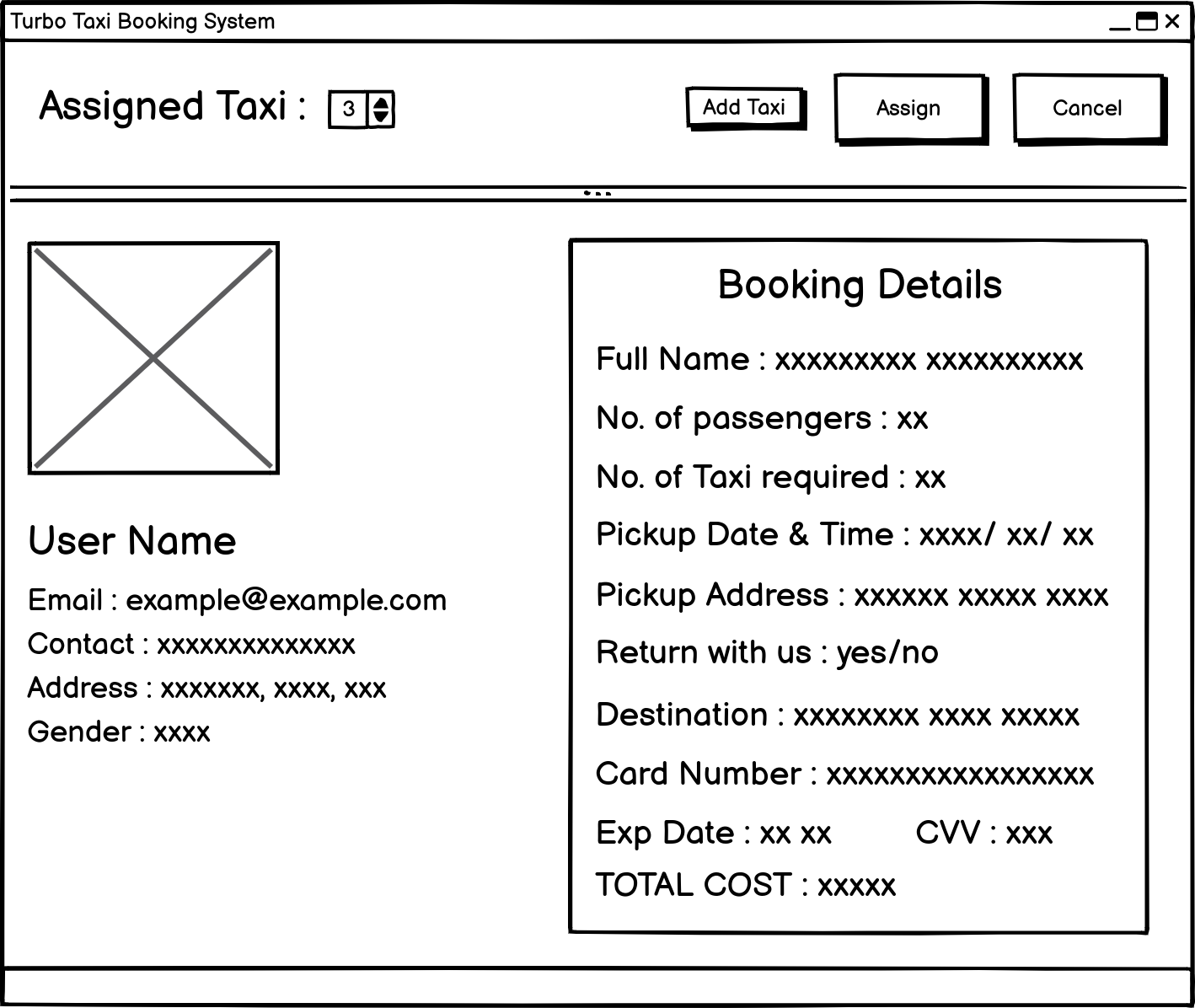
Figure 16: user\_dashboard\_interface\_design

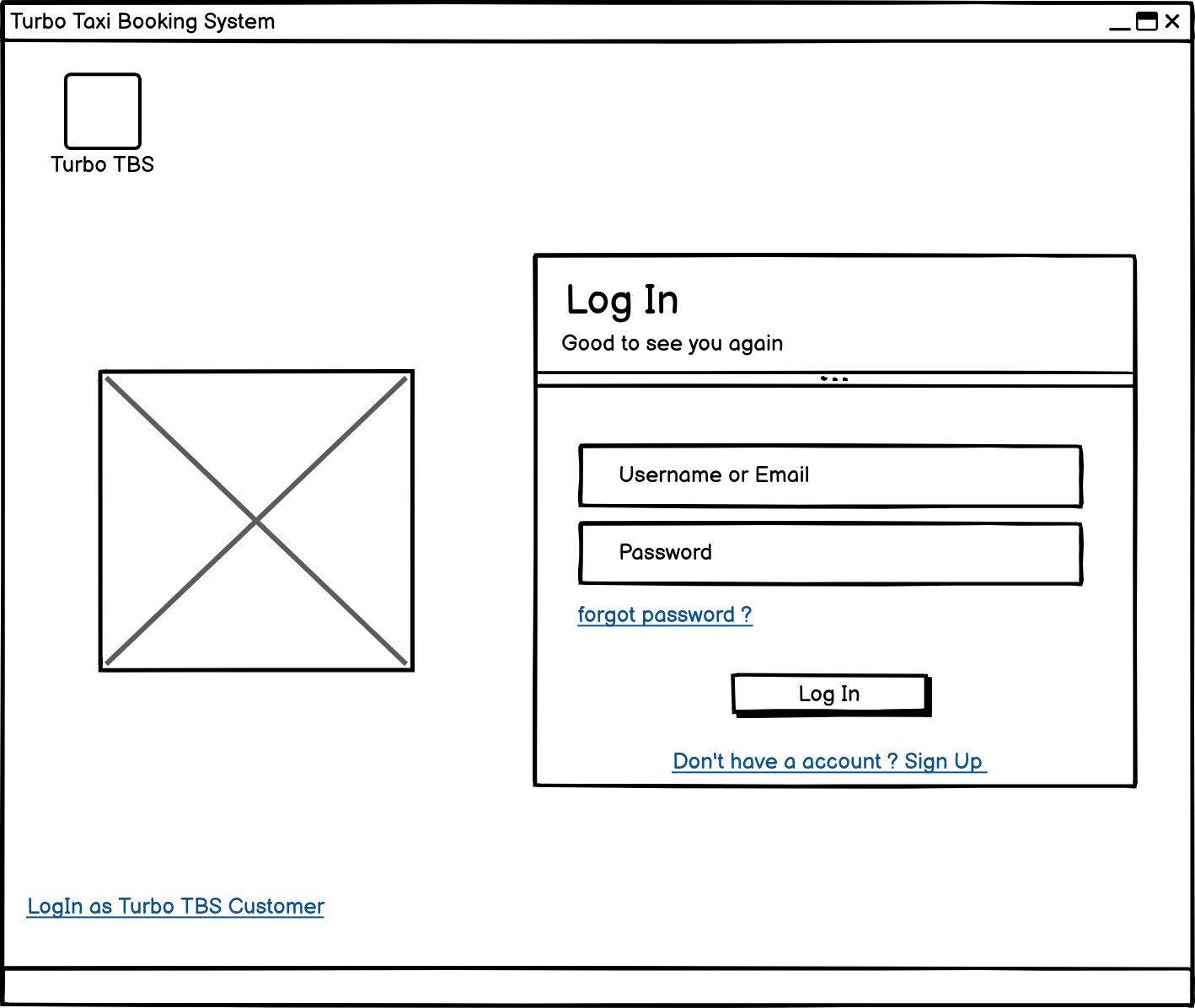
Figure 17: user\_profile\_interface\_design

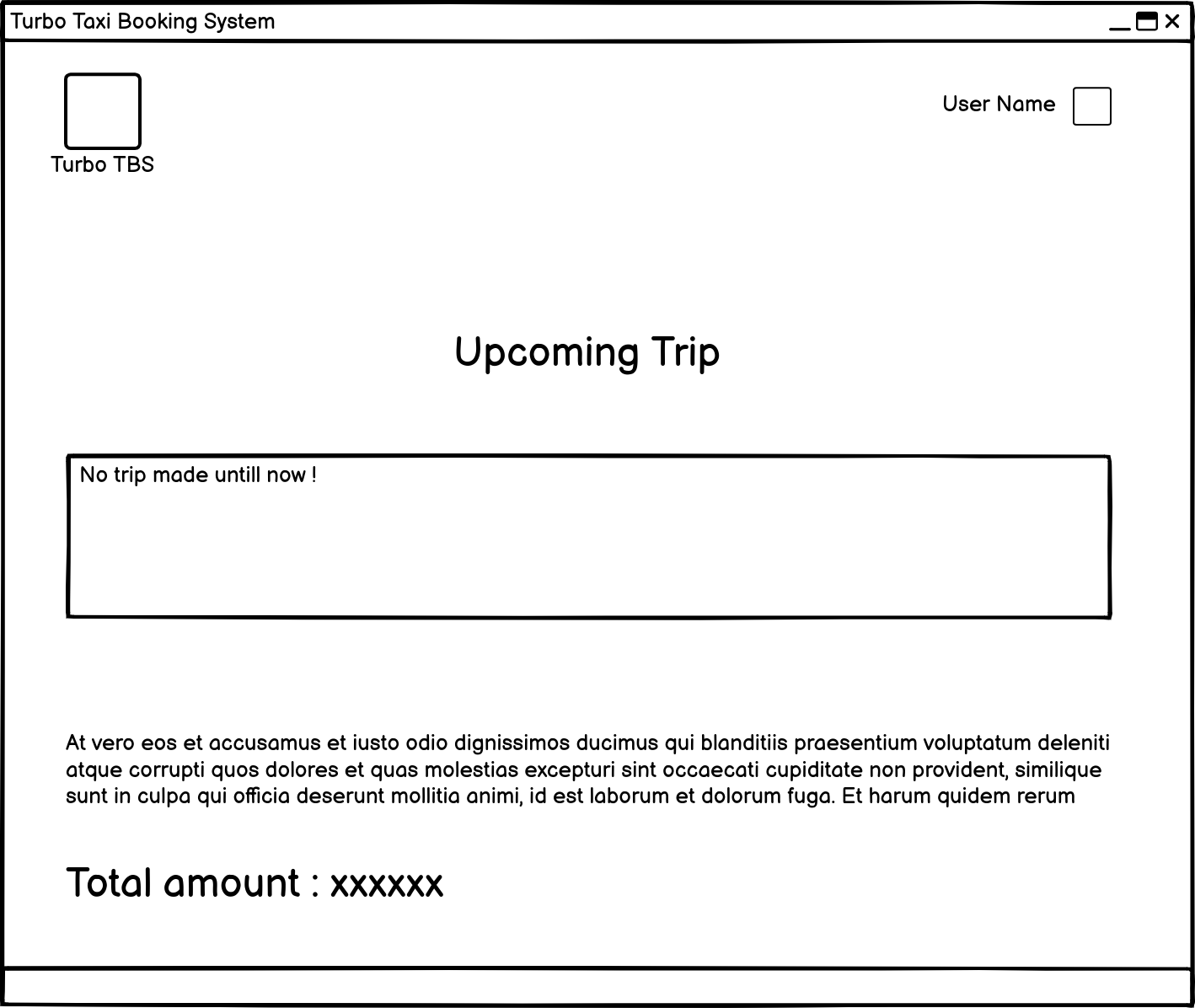
Figure 18: booking\_form\_interface\_design

Figure 19: admin\_interface\_design

## 

Figure 20: booking\_detail\_admin\_view\_interface\_design

Figure 21: driver\_login\_interface\_design

Figure 22: driver\_daashboard\_interface\_design

## Algorithm Design

Flow of the development process

• Flow of the system hierarchy

Taxi booking system

• The system :

Main dashboard : SignUp and SignIn on the main dashboard, with a visible working gallery

1. After clicking SignUp, "SignUp as Customer" will open up and also provide a link for "SignUp as Worker."
2. Both forms will require different information and will be saved on different tables of the database.
3. After clicking Sign In, the Sign In as Customer page will appear, along with a link to Sign In as Worker.
4. The same information will be required for both forms. username, email, and password

type of user

i. Customer ( The one who does the booking action ) :

Working of a customer :

1. Register as a customer

2. Sign up with all the required details.

3. If all the validation is completed, the user will be redirected to the login page.

4. The user will login with the given login email or username and password.

5. The user will be redirected to the user dashboard, which contains a "book now" button, a profile picture, a profile name, and some other details about the system (a hint).

6. Clicking on "Book Now" will redirect the user to the booking form.

7. After entering all of the necessary information, the user can click the Book button, which sends a booking request to the taxi company for approval.

8. This will redirect the user to their dashboard.

9. By clicking on the profile picture or the username on the dashboard, users will be redirected to the profile page.

10. Profile page will include all the previous booking history and ongoing booking information

11. The profile page will allow you to edit the booking details as well as your personal information.

12. The GUI must be good, as each function should change the pages for each different function.

13. Bookings can be canceled from the profile page until the trip date is more than one day away.

14. The completed trip can be seen, and the driver can be rated afterward.

ii . Admin (The one who assign the taxi and driver to the request trip)

Working of a Admin :

1. Log in as an administrator with the provided username and password.

2. Admin should be redirected to the admin dashboard.

3. The portal should contain two options (a taxi driver control panel and a customer booking request control panel).

4. Clicking on "Taxi Driver" (this will bring up a list of all registered taxi drivers as well as all taxi drivers who wanted to register with valid information).

5. It will also provide all the information about the taxi driver after clicking the taxi driver's name on an ID-sized card.

6. It will give the administrator full authority to accept or decline a registration request for a taxi driver after documentation review.

7. Admin will be able to go back to the previous page where they can select between the driver and the customer control panel with a back button.

8. Clicking on "customer" (it will provide a list of all the customers, and you can view their details by clicking their name, which will pop up an ID-type UI) 9. You can view the booking request made by the customer.

10. You can see the drivers that are already booked for a trip and the available taxi drivers that are suitable for the trip.

11. To accept a trip, click on a requested customer name, an Id type UI pops up, and then click on assign taxi after checking the requested booking detail (number of taxi will allow you to input number of taxi vehicle number with add button).

12. The "booking for Customer XXXXX has been completed" message will be shown, and then the pop-up will close on its own.

13. Admin have full authority to accept or decline an offer if it seems reasonable.

14. Admin can ban a registered user or a driver if they are found breaking the rules of an organization.

iii . Driver (The One who are assigned for customer to complete the trip)

Working of Driver :

1. Sign up as a driver

2. Sign up with all the details (full name, address, contact information, gender, date of birth, username, email address, experience, upload of a license photo up to 25 MB, vehicle number, vehicle registration number).

3. If all the validation is completed, the user will be redirected to the login page.

4. The user will login with the given login email or username and password.

5. The user will be redirected to the user dashboard.

6. The driver dashboard contains total income, upcoming trips, some information for easy use, Profile Page

7. By clicking on the profile picture or the username on the dashboard, the user will be redirected to the profile page.

8. The profile page will include all the previous booking history and personal details.

9. After the trip is assigned to the driver, the driver must take the trip as the company requests, but if the driver is not available, the driver can change the status from "available" to "not available."

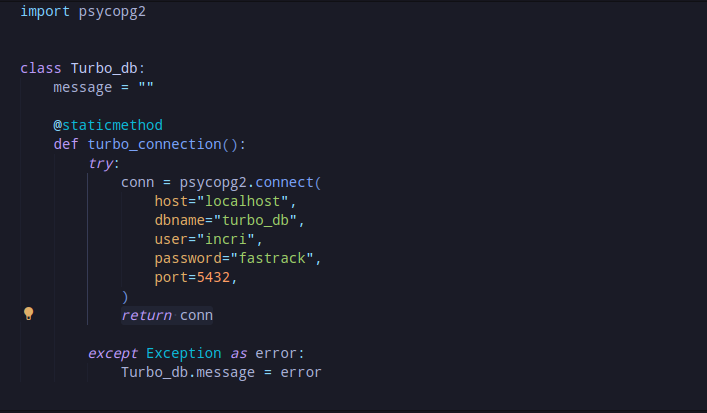
10. Changing the status will affect the admin page, so the filter won't show the driver as an available driver, and assigning the tax trip to such a driver is not allowed.

# Implementation

Using a Tkinter-based GUI, users of the Turbo Cab Booking System can request a taxi and calculating fares for the journey. Users may transmit or request data via a graphical user interface and a client server infrastructure, which allows for easy data sharing. With only one click, the user may use an external database system rather than using user storage to save data in the booking system. In addition, drivers have access to a list of their allocated travels, which are marked as finished when finished.

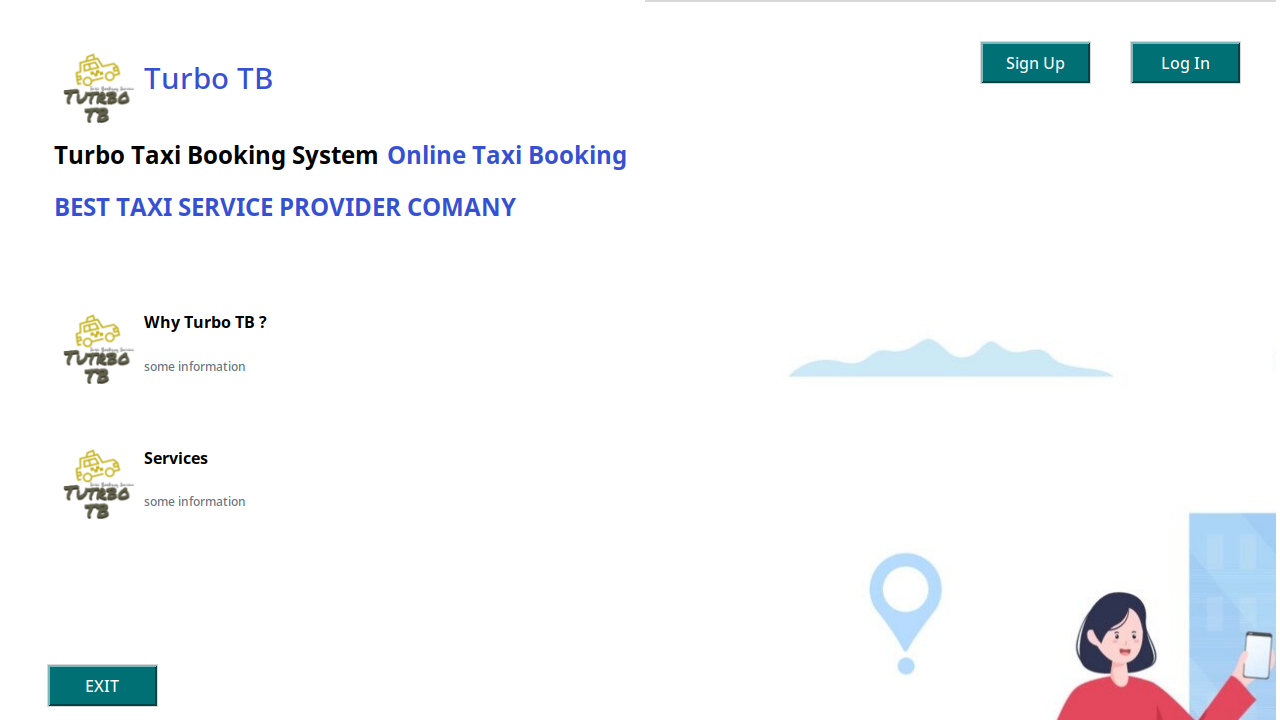
The planning and prototype phases of the development process were the first. All the criteria were initially put down after creating a weekly timetable. Visual Paradigm was used to create ER diagrams for the database design, and Balsamiq was utilized as a prototyping tool for the GUI concept design.

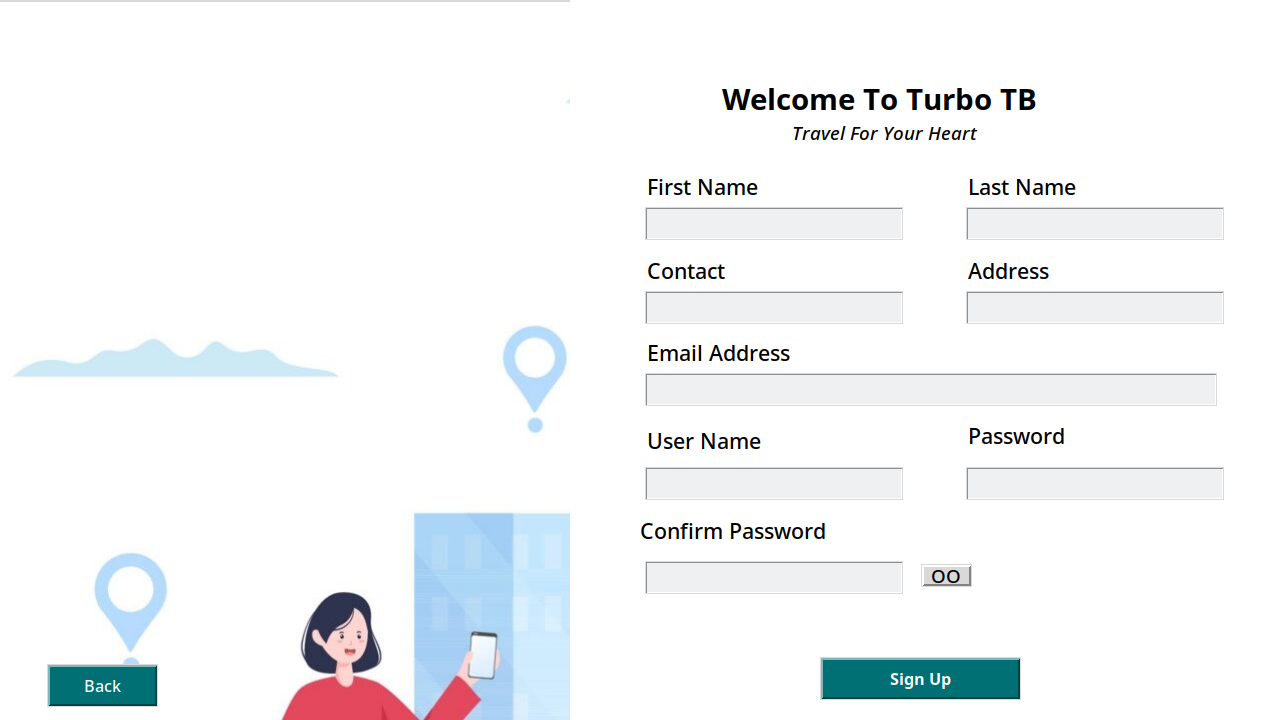
In order to learn and test PostgreSQL as we all know Postgres is a powerful and reliable database that is well-suited to a wide range of applications, Data was stored in a PostgreSQL database, and psycopg2 was utilized as a connector between the system and the database.

Figure 23: database\_connection

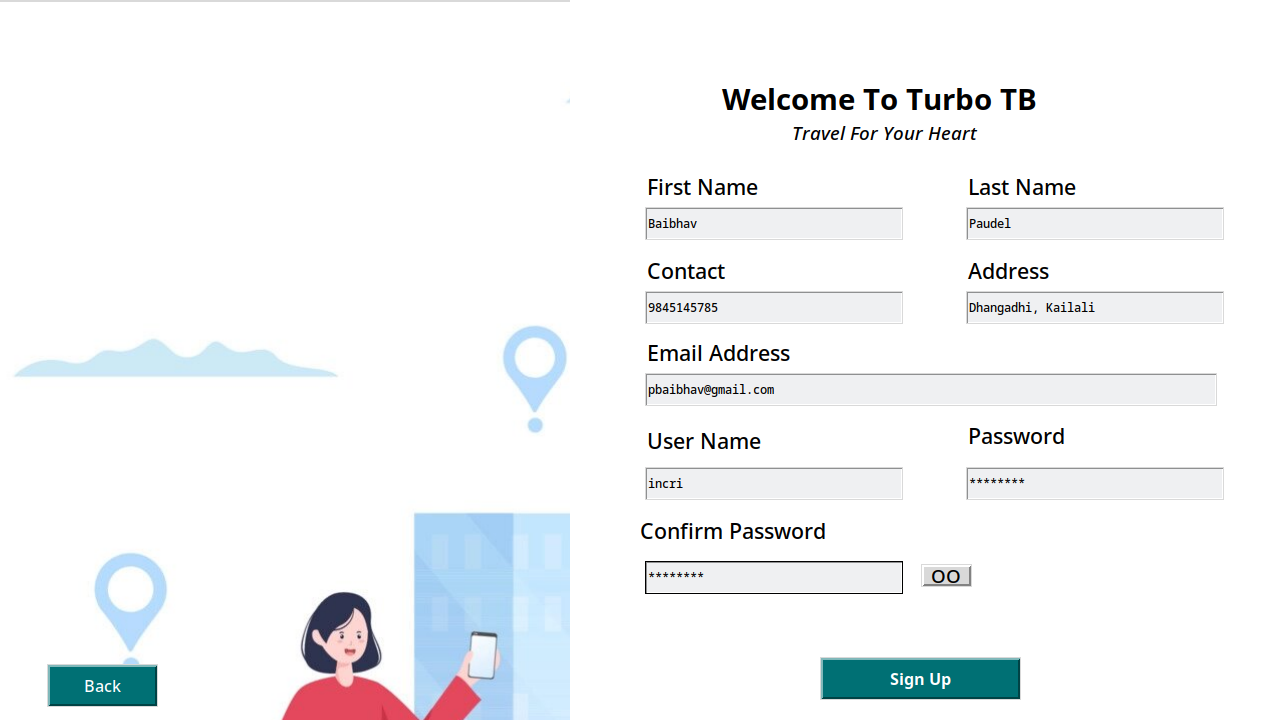
For frontend development, Tkinter, a built-in Python module, was used. The non-visual, or backend, portion of the code was likewise created in Python utilizing the MVC design pattern, where the model manages all of a module's important getter and setter and carries out validation. The frontend, or visual representation, of a program and function that provides data to the model layer's getter for validation is managed by the view layer of a project and The function that establishes a connection to the database system and puts data in the appropriate table according to the SQL command is called by the view layer from controller layer when the validation is finished.

Home page of taxi booking system .

Figure 24: home\_page

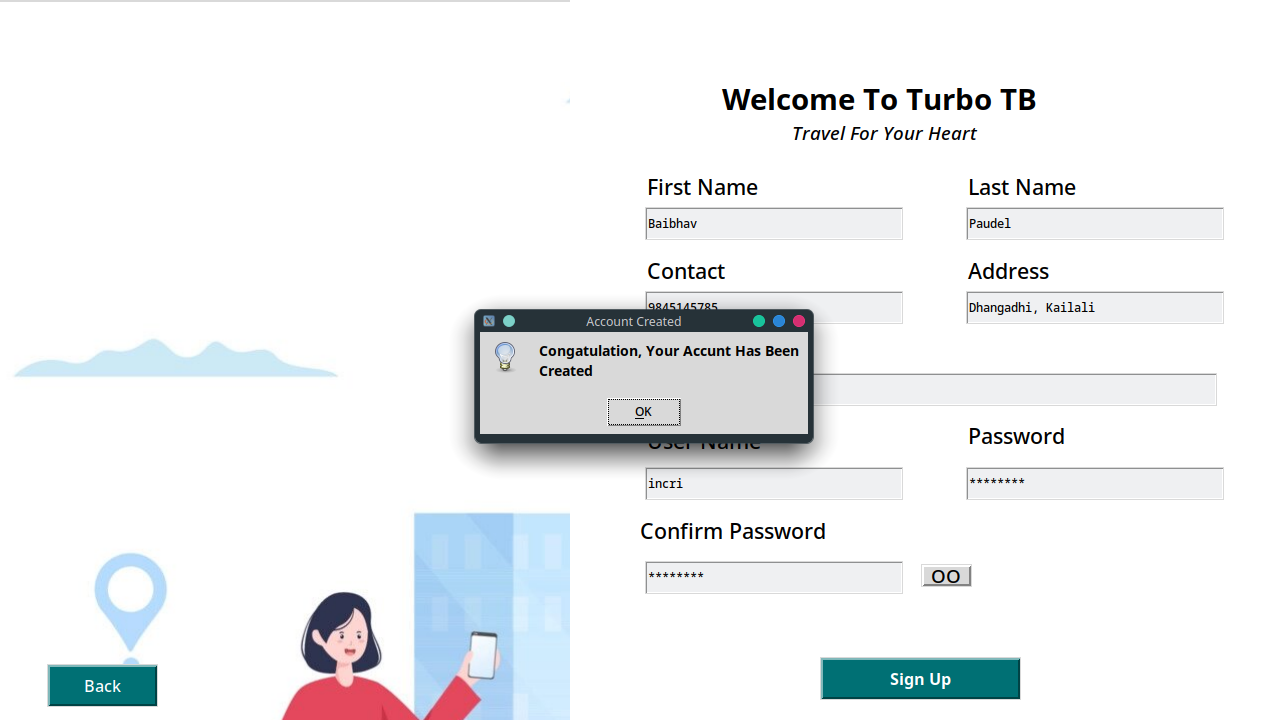
Figure 25: signup\_page

Signup page for the user where unregistered user register for the first time.

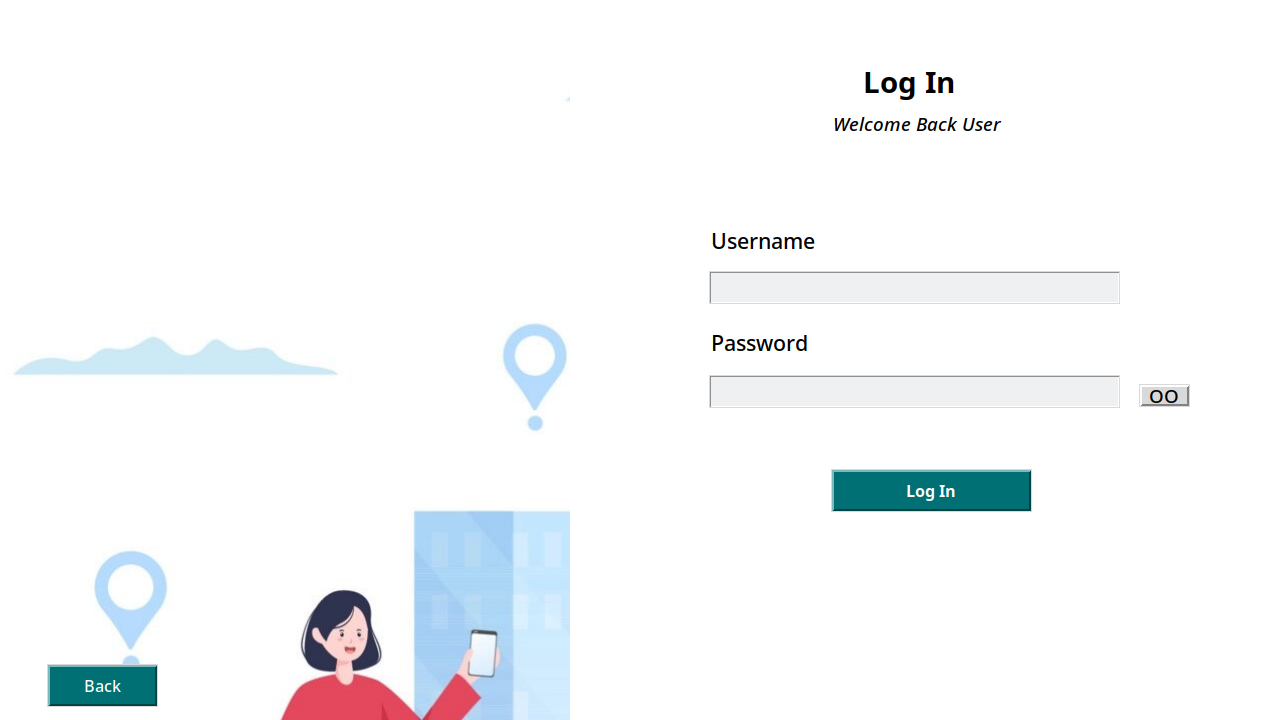
Figure 26: signup\_page\_with data

Registering with valid data.

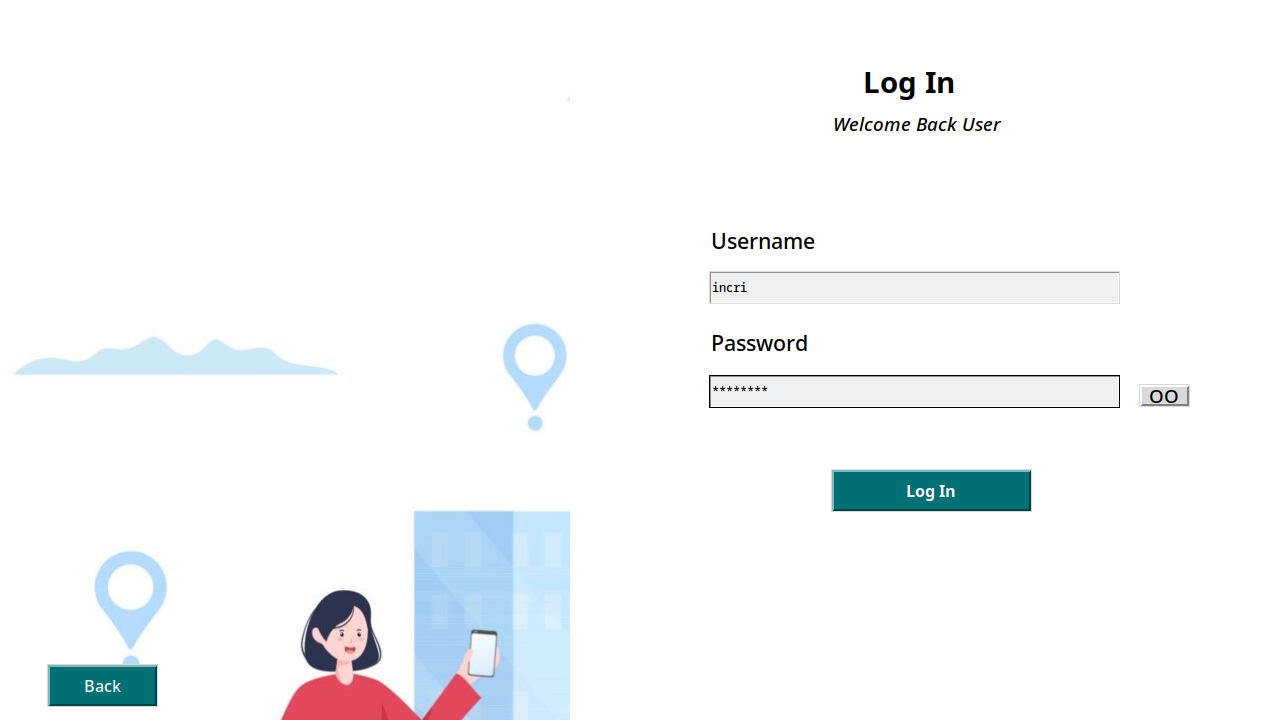
After entering proper information and pressing the signup button, a messagebox with a successful account creation message appears.

Figure 27: after\_clicking\_signup\_button

The login page is used to log into the system if the user is already registered.

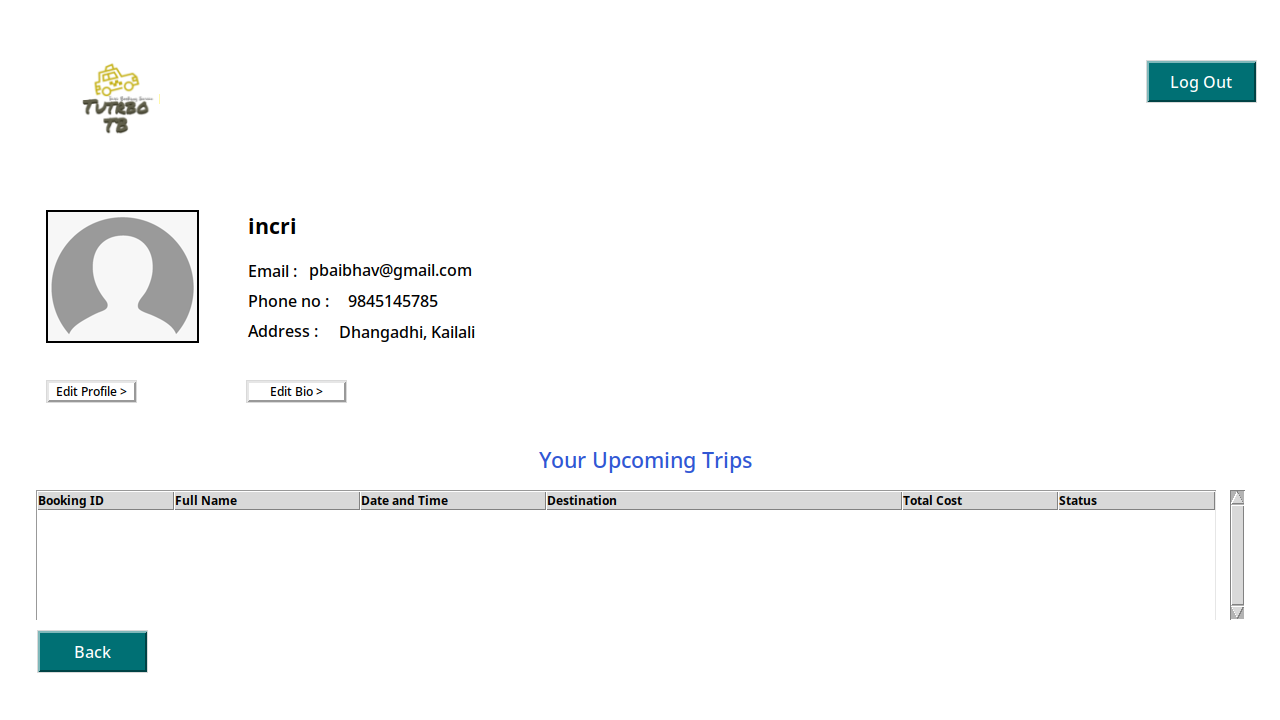
Figure 28: login\_page

Logging in with valid registered data.

Figure 29: login\_page\_with\_data

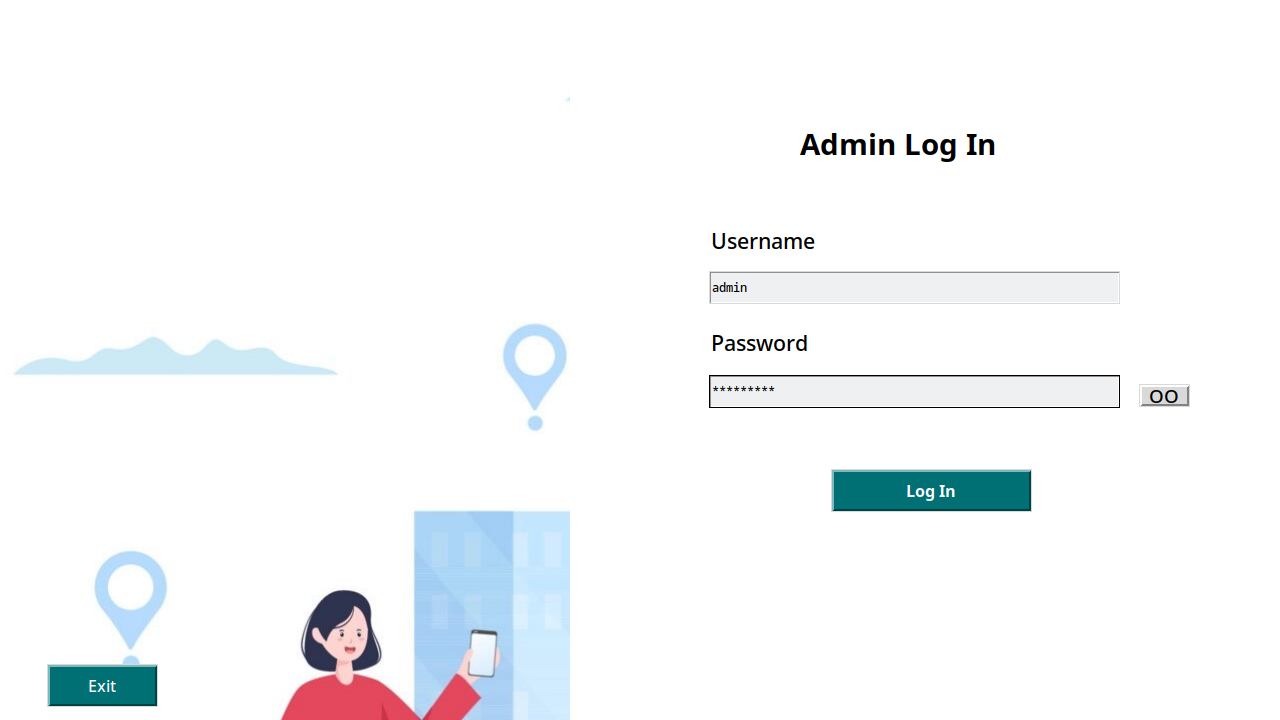
Valid login information enables user dashboard redirection.

Figure 30: user\_dashboard

Figure 31: profile\_page

Users may access their profile pages, which include personal information and information about planned trips.

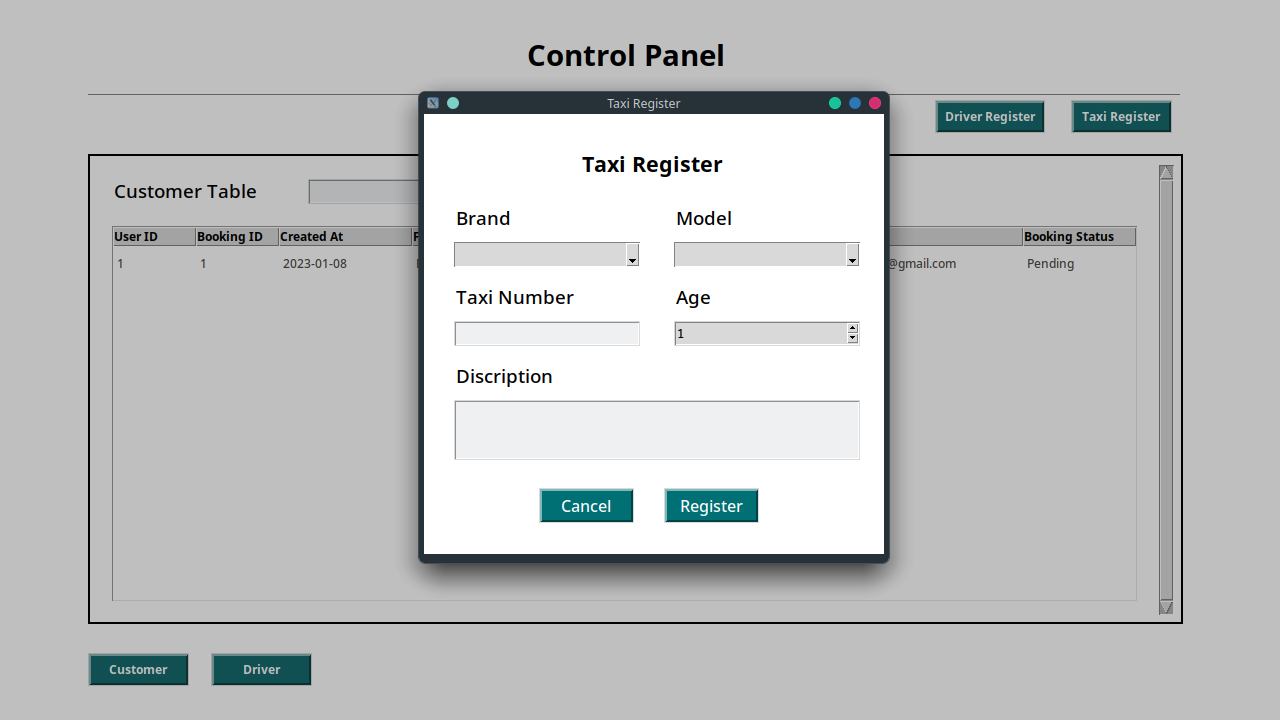
Admin can redirect to the Admin dashboard using valid login information.

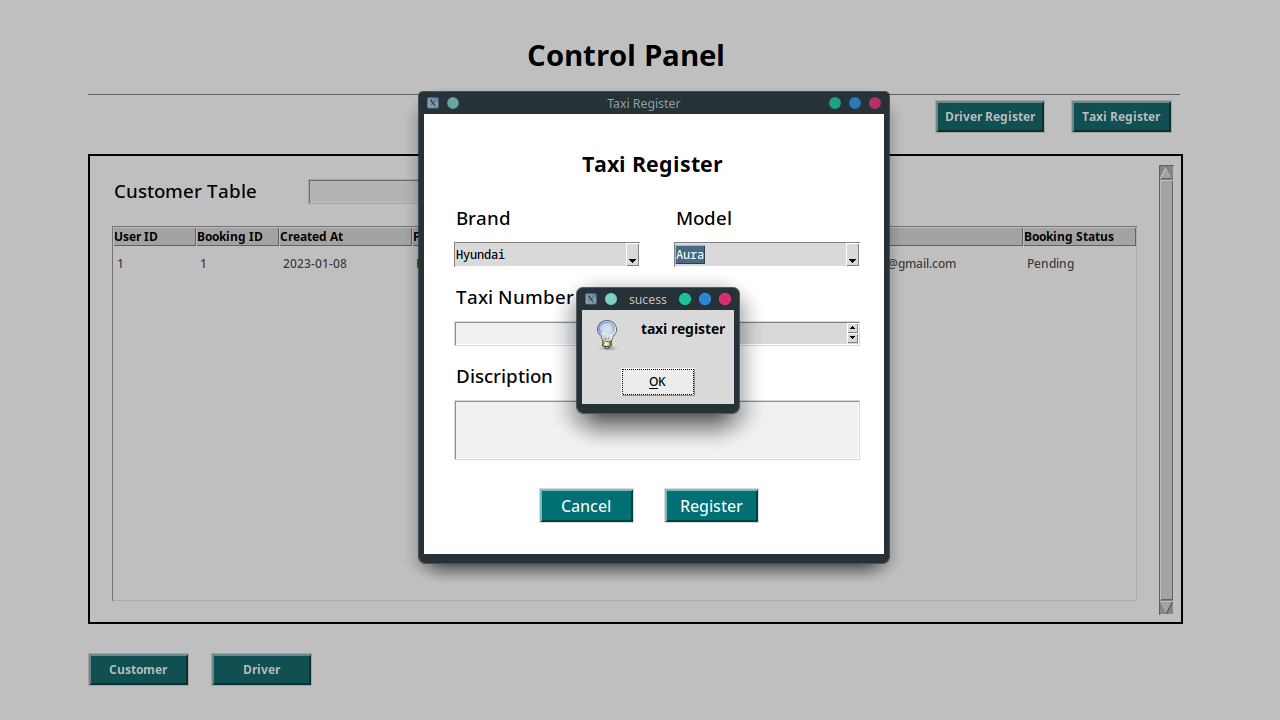
Figure 32: admin\_login

Admin dashboard with search and data filter functionality for seeing booking request information.

Figure 33: admin\_dashboard

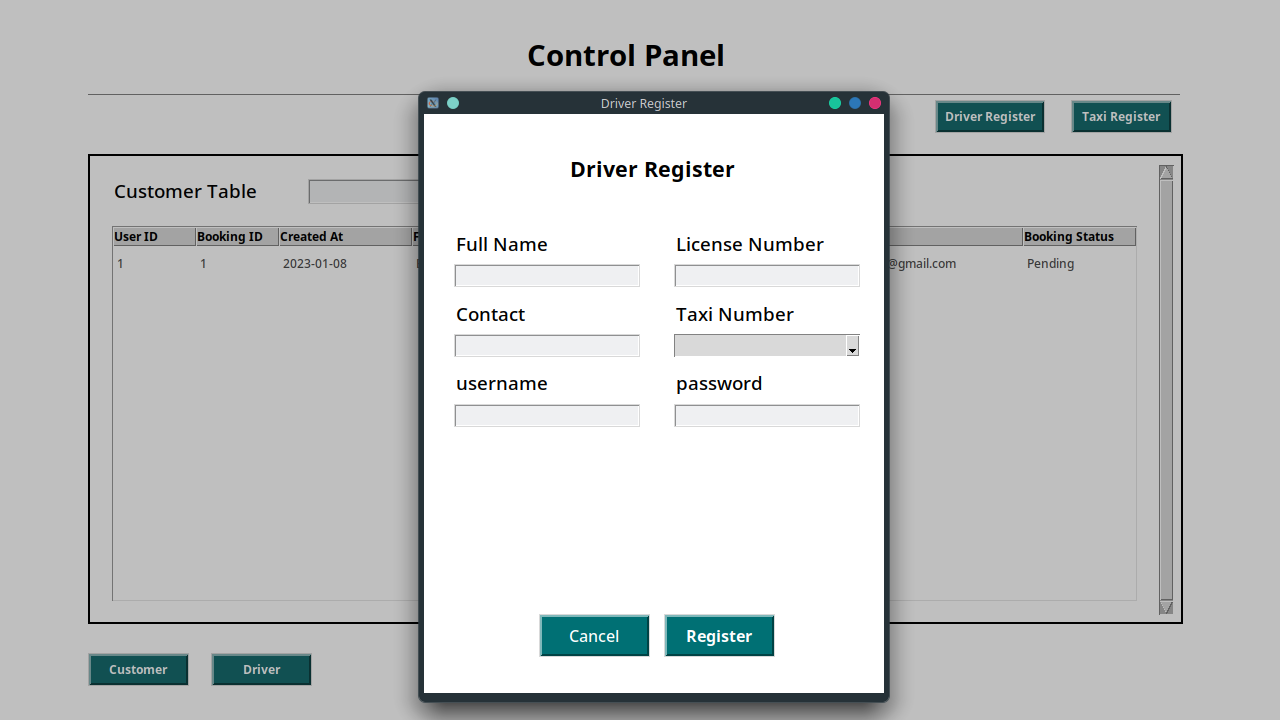
Additionally, the admin dashboard enables admin to register new taxis with relevant information.

Figure 34: register\_taxi

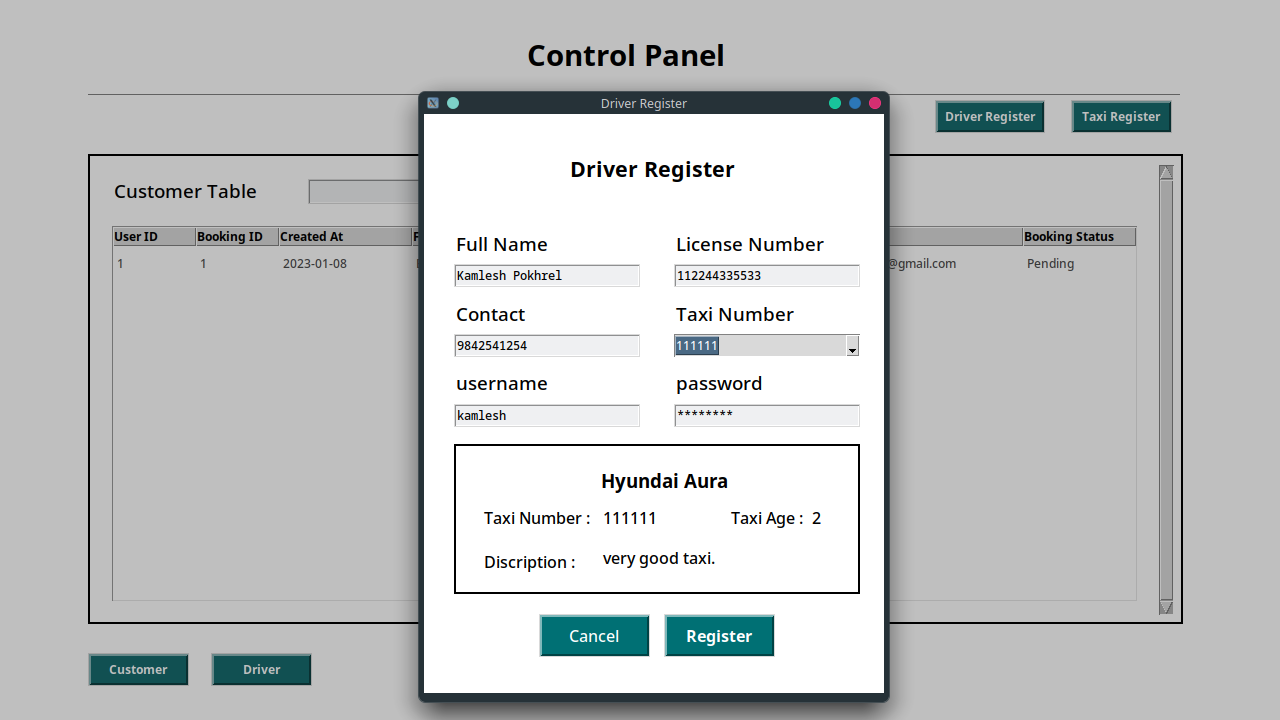
Figure 35: after\_clicking\_on\_taxi\_register

After taxi is registered message box pop up and inserted data get deleted but the toplevel remain there allowiing admin to add more taxi detail without extra clicking work.

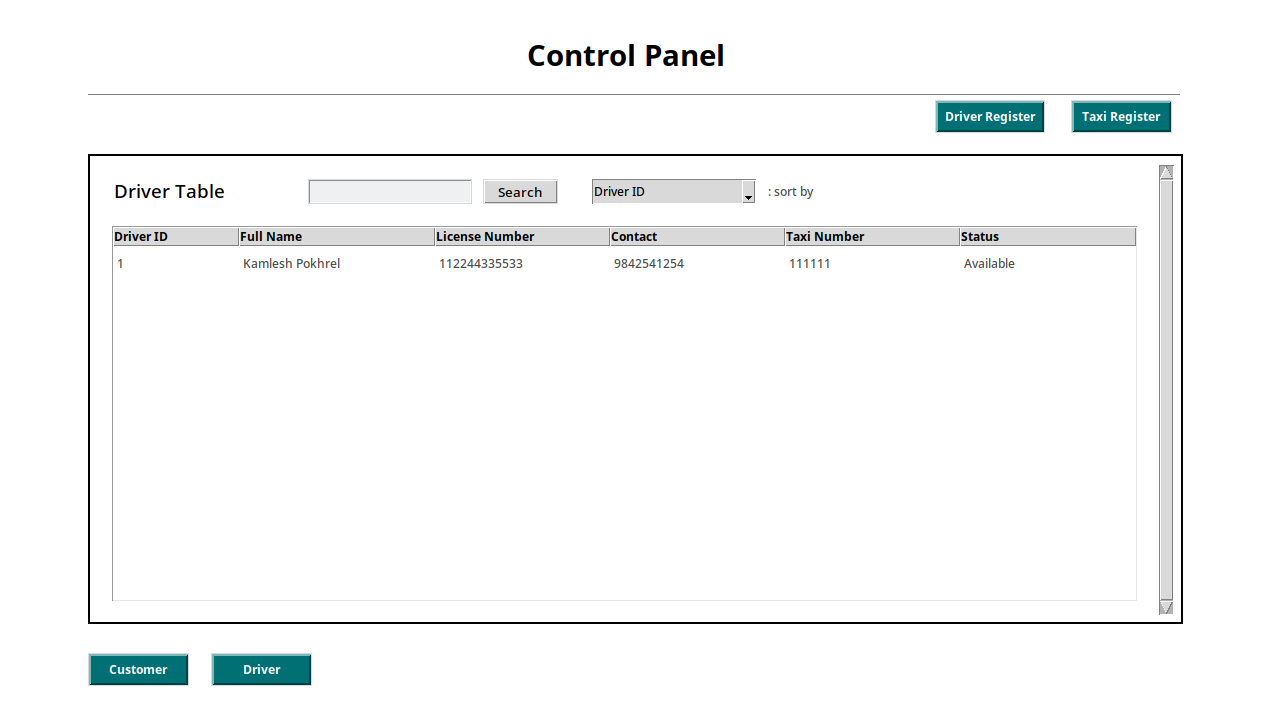
A new driver's personal information can be registered on the admin dashboard as well.

Figure 36: driver\_register

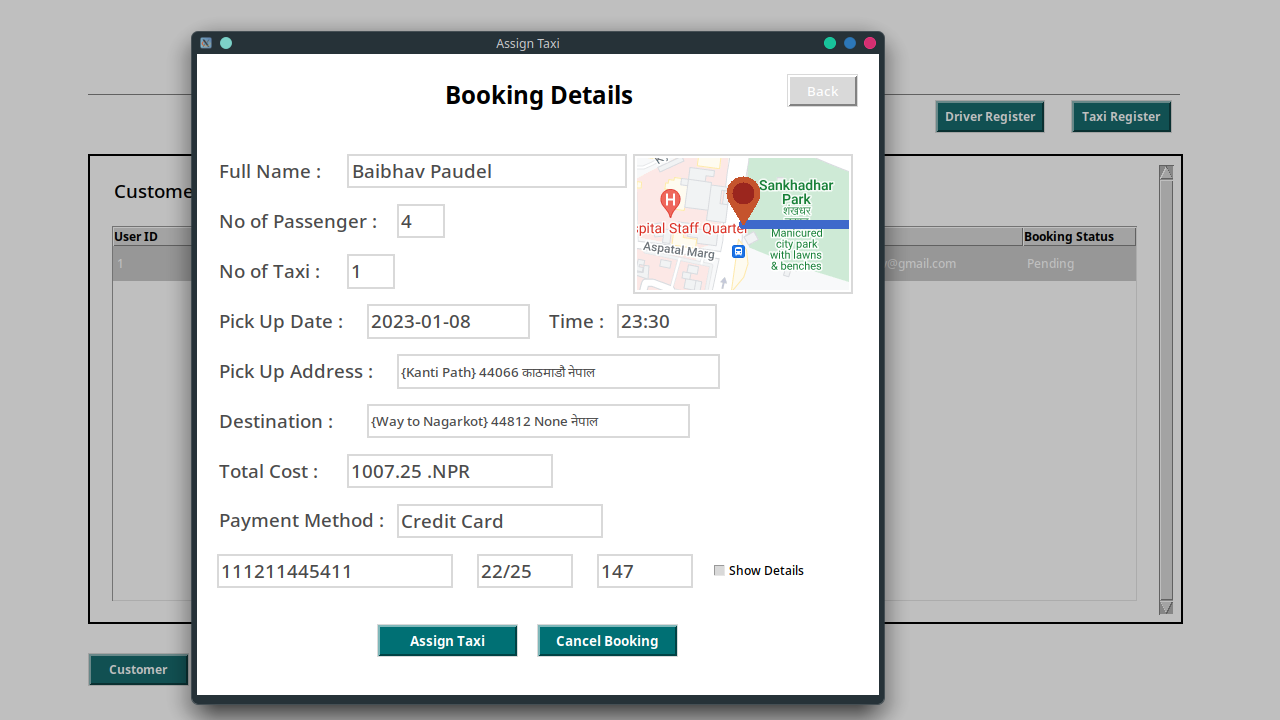
Admin must select a taxi number to provide to the driver of the taxi by using a tkinter combobox, which displays all the available taxi numbers to assign to the driver. After selecting on a taxi number, the details of the taxi are displayed in the frame below.

Figure 37: driver\_register\_with\_data

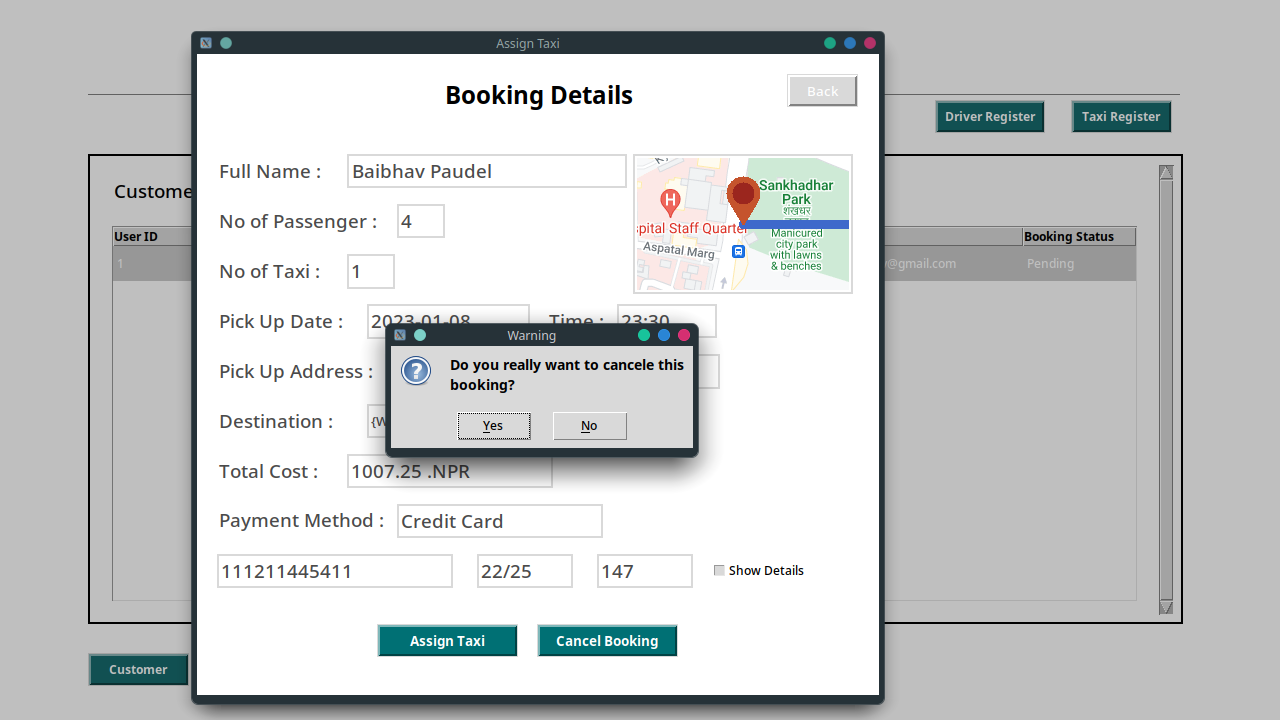
Administrators may view registered drivers and their status on the driver table after registering them.

Figure 38: after\_driver\_is\_registered

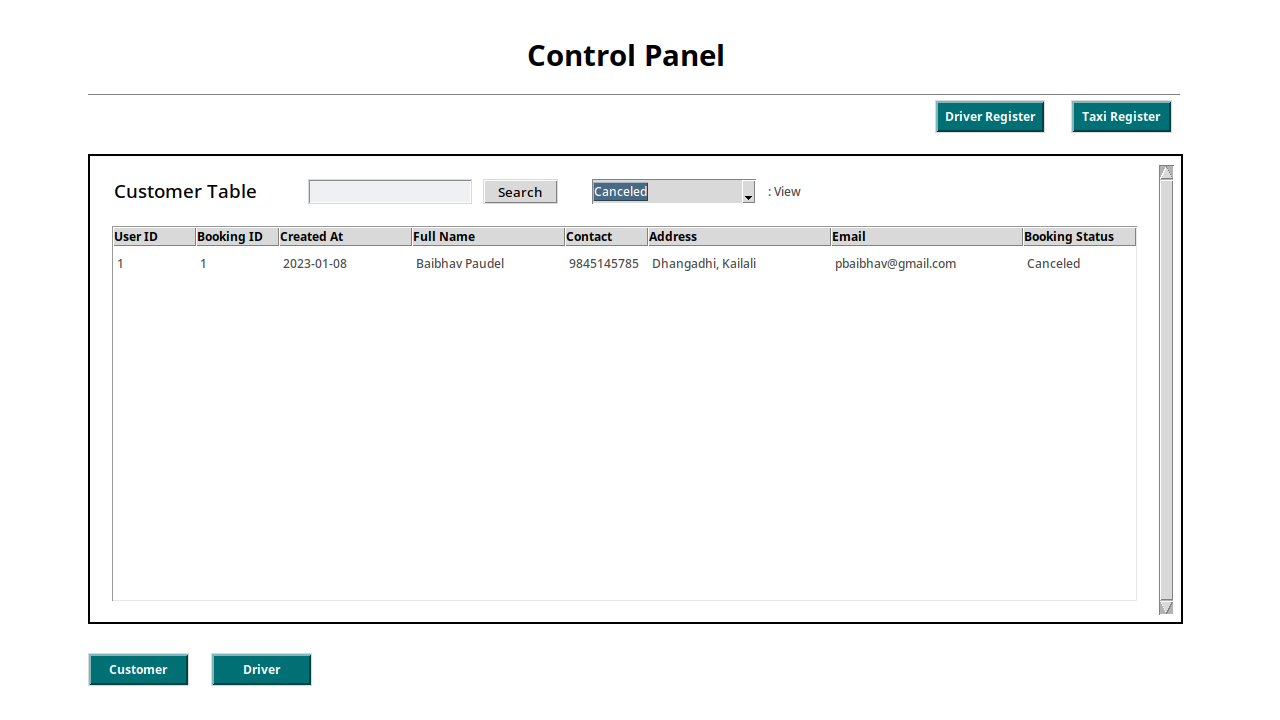
Double clicking on the top level of the booking request will bring up a pop-up with all the booking details, a minimap showing the pickup and drop-off locations, and a button to assign or cancel the service.

Figure 39: double\_clicking\_on\_booking\_request

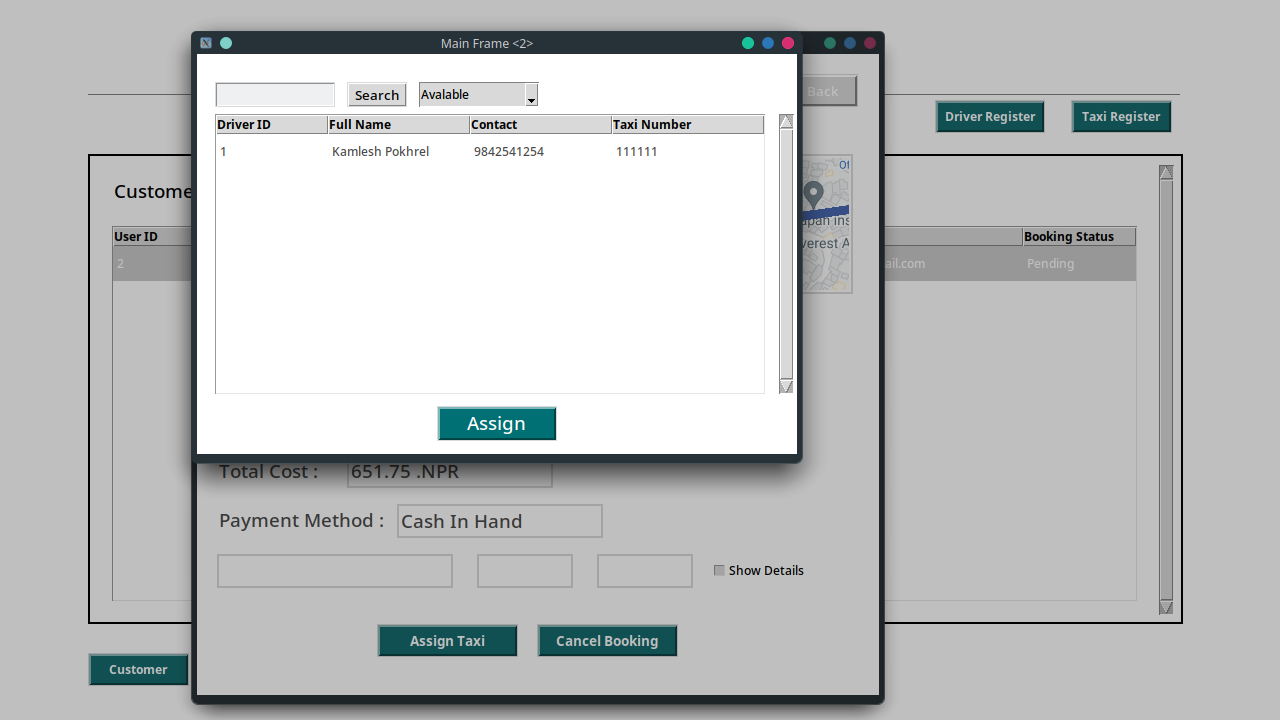
A message box with a security warning appears when a booking request is canceled.

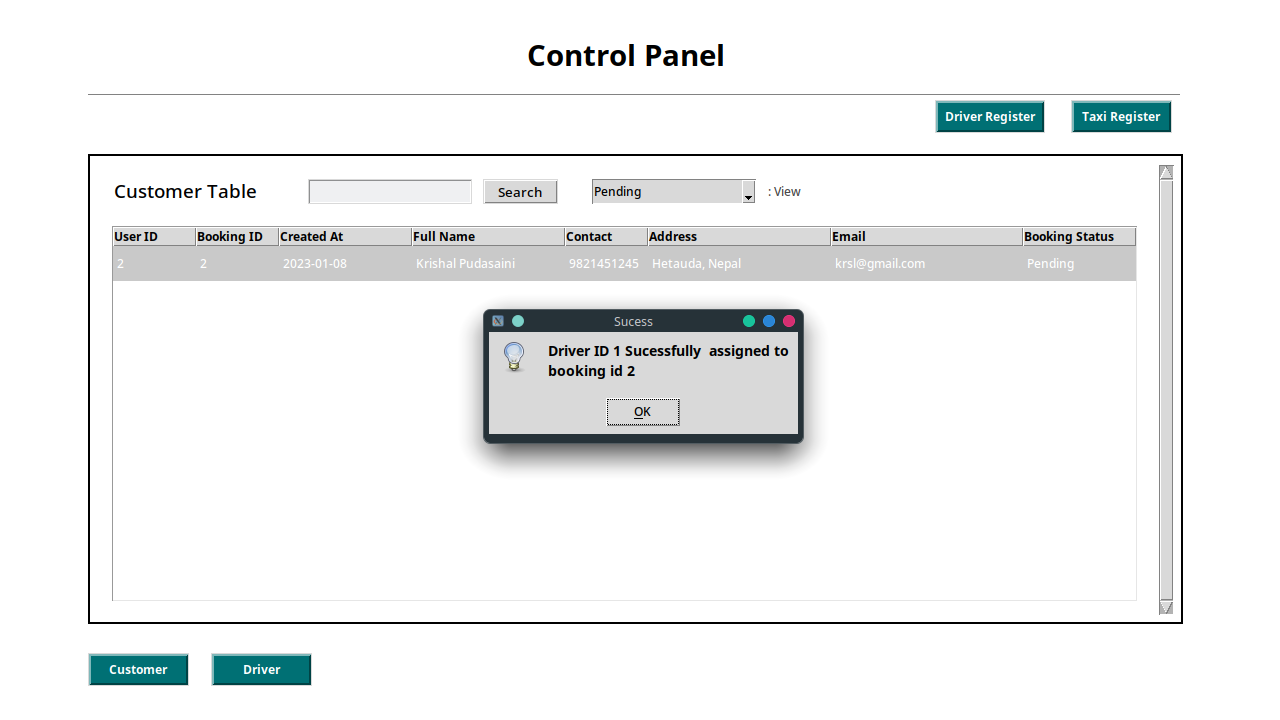
Figure 40: after\_clicking\_on\_cancel\_booking

When a booking request is canceled, it can be kept track of by using the canceled filter, which displays status as canceled.

Figure 41: after\_booking\_is\_canceled

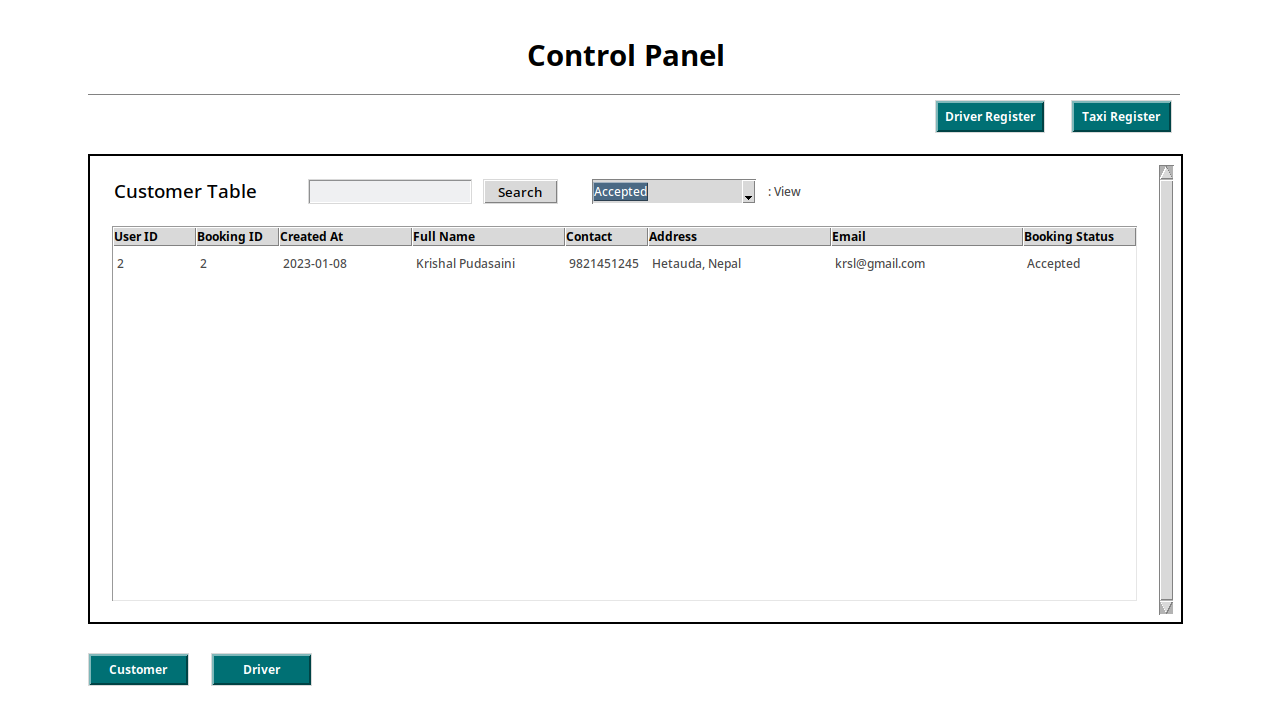
A new top level with a list of all available taxi drivers that may be assigned to the booking request appears when the assign taxi button is pressed.

Figure 42: after\_clicking\_on\_assign\_button

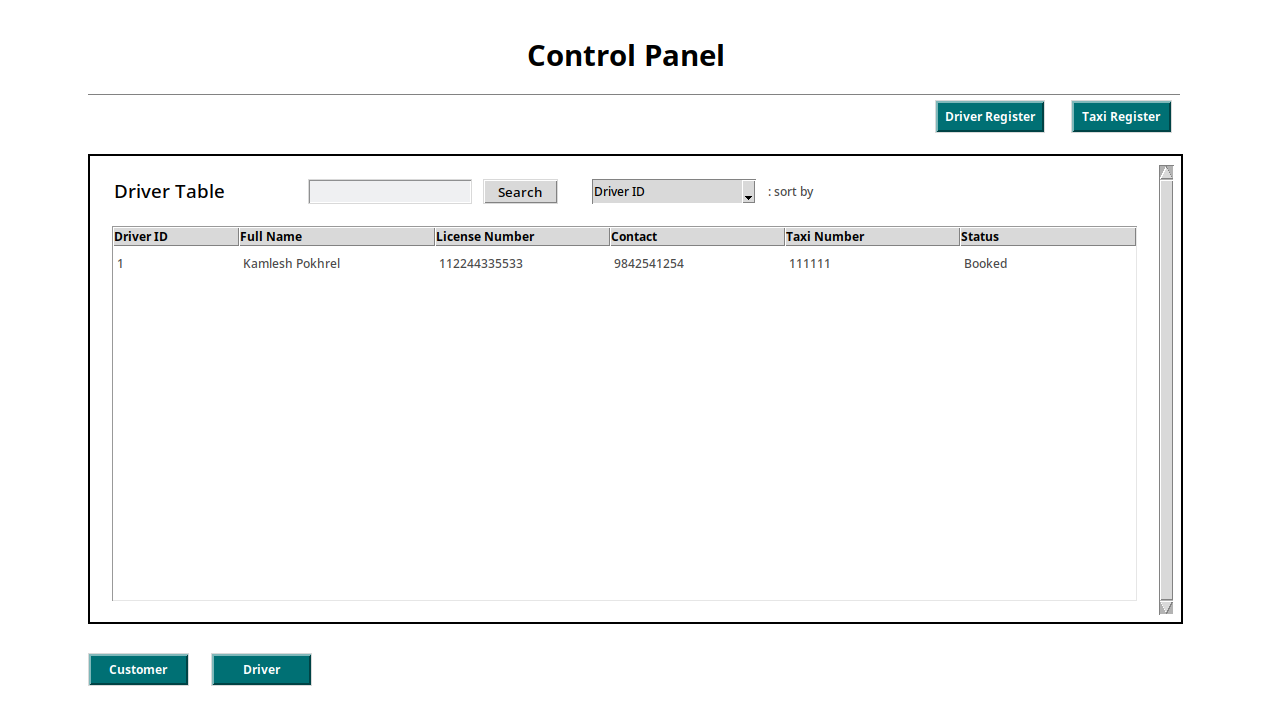
Figure 43: after\_clicking\_on\_register

A success message box appears with the necessary information after selecting the driver to assign to the booking.

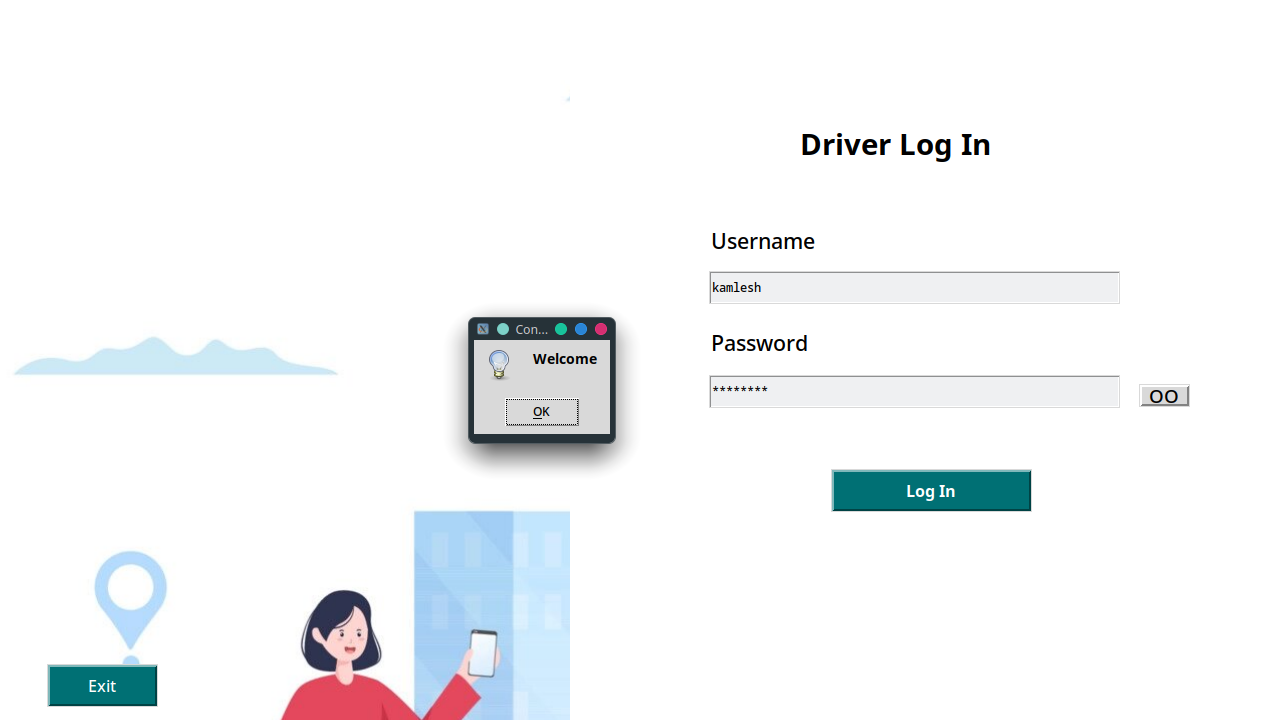
The booking request status changes to accepted when the taxi is assigned, and the Accepted filter may be used to view it.

Figure 44: booking\_detail\_after\_booking\_is\_accepted

Driver status changes to booked once a driver is allocated to a booking in order to prevent the driver from being assigned to any other booking requests until the trip is over.

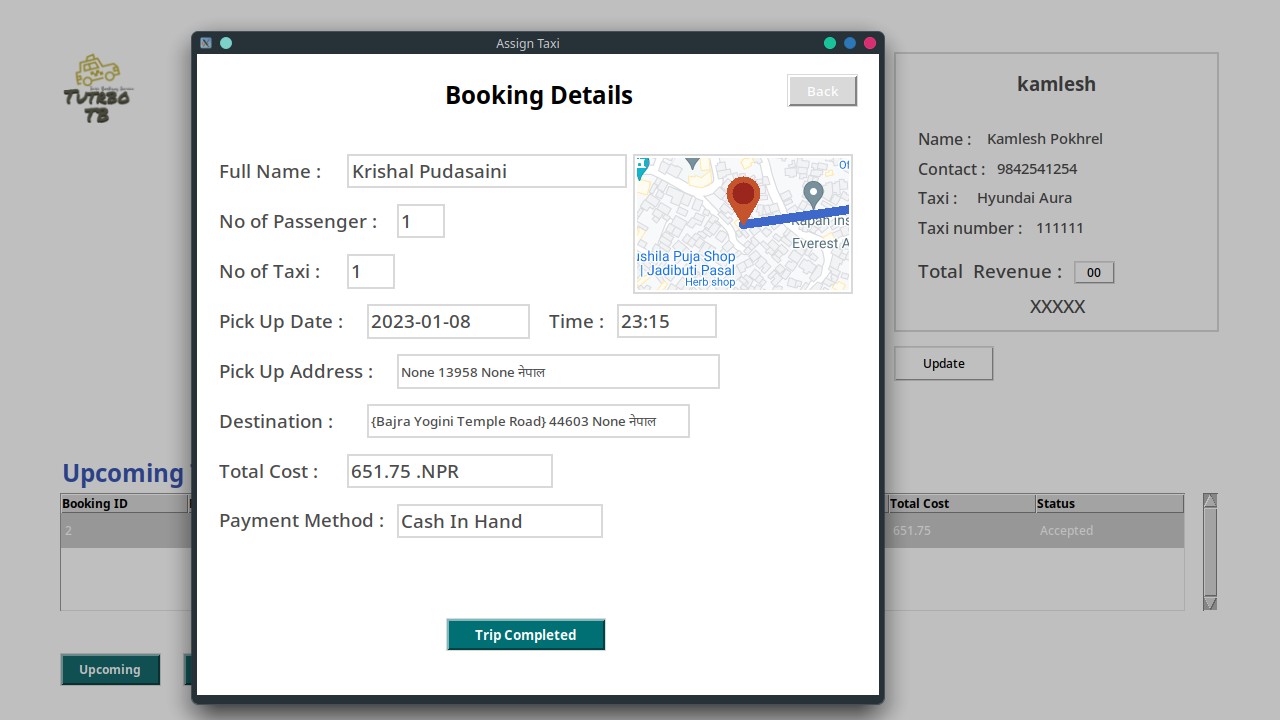
Figure 45: driver\_detail\_after\_booking\_is\_accepted

Driver can access the driver dashboard by using the admin's provided valid login information.

Figure 46: driver\_login\_page

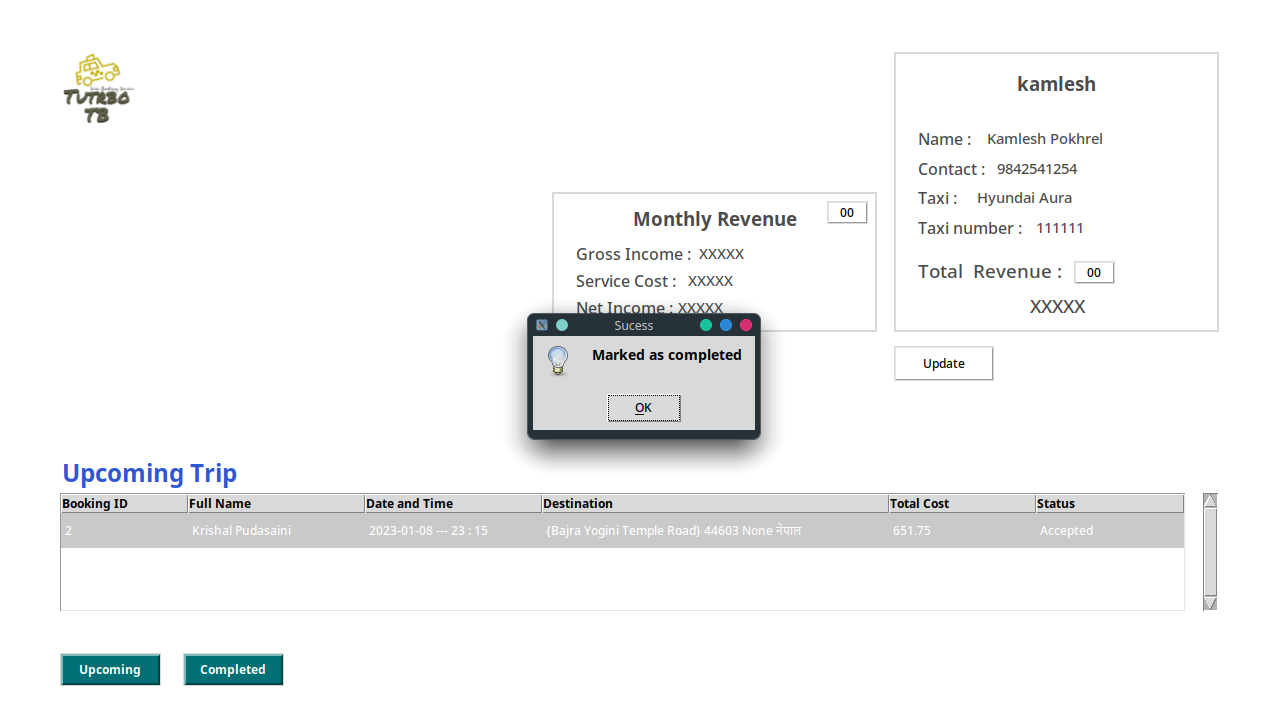
Driver dashboards enable drivers to see the trips that have been allocated to them in the near future.

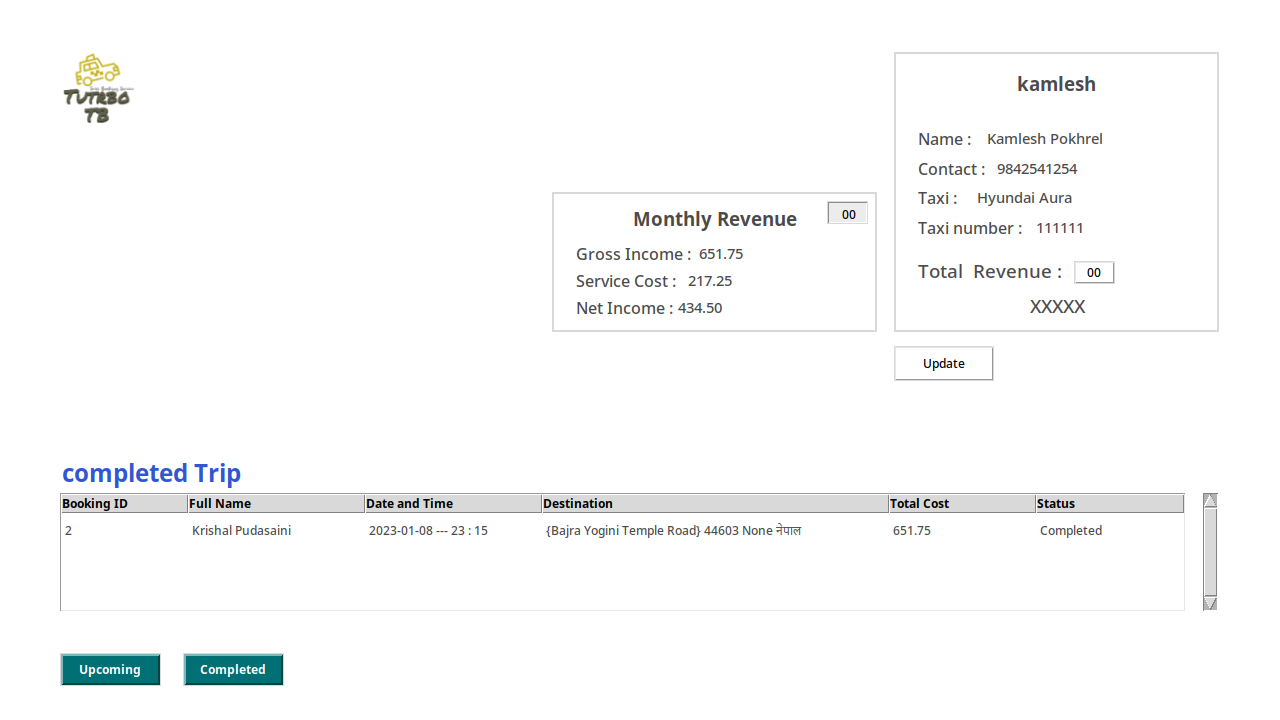
Figure 47: driver\_dashboard

Figure 48: driver\_assigned\_booking\_detail

When you double-click a booking request, the full details of the booking are displayed on the top level along with a trip finished button that enables you to declare a trip complete after the trip date has begun.

The system notifies the driver via a message box that the journey has been designated as completed after clicking on Trip completed.

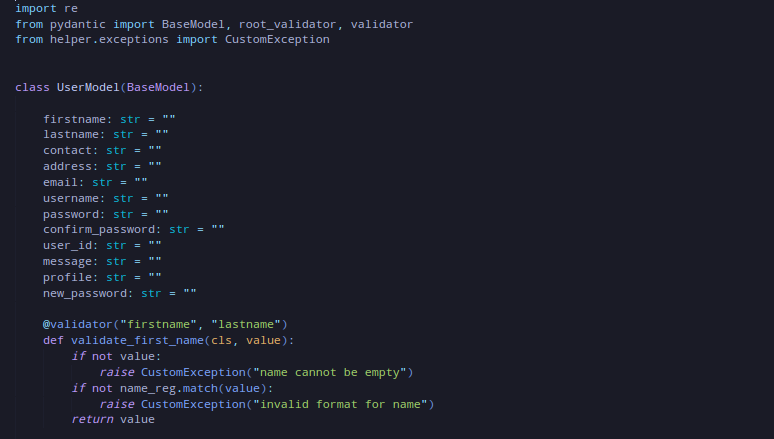
Figure 49: Trip\_marked\_as\_completed

Figure 50: completed\_trip\_details

Completed trip money is contributed to total revenue as well as monthly revenue calculations that calculate gross income, service cost, and net income and are reset every month for fresh month data. Completed trip data is presented in the completed trip table.

(DZone, 2021)

Pydantic, a Python package that offers a set of tools for developing data validation and data parsing code, was utilized to tackle a problem that arose during the development process regarding verifying my user-given data before putting it in a database.

Figure 51: pydantic\_code\_snippets

(Schimansky, T. )

A tile-based interactive map renderer widget for Python, TkinterMapView The Tkinter library was also utilized to resolve my other encounter issue, which included giving the system exact pickup and departure locations so that it could determine the trip's overall cost.

Figure 52: TkinterMapView\_code\_snippets

# Testing

Documentation of each test – one after the other

You need to include:

* Test No
* Test Date
* Purpose of test
* Input data or action
* Expected result
* Actual result
* Action if the above are not the same
* Screenshot of output

# Discussion / Reflection / Critical Analysis

As an individual working on a taxi booking software project, here are some things that have gone well or wrong during the development process:

What went well:

* In order to achieve the project's objectives, I was able to describe them precisely and lay out a plan.
* I was able to effectively research and learn any new technologies or tools necessary for the project.
* I was successful in keeping progress on schedule and meeting project deadlines.

What went wrong:

* I struggled with feeling overwhelmed or burnt out while working on the project alone.
* I had difficulty finding the necessary resources to complete the project.
* You may have encountered technical challenges that were difficult to resolve on your own.

As an individual, I had to cultivate strong time management skills throughout the development process. I had to be mindful of deadlines and ensure that I was making steady progress towards completing my tasks on schedule. I used tools such as project management software to help me track my progress and identify any potential bottlenecks or delays.

Yes, I was able to achieve all the required specifications of the assignment brief. I carefully reviewed the requirements and made sure to incorporate all necessary features into my project.I did encounter a few problems along the way, but I was able to overcome them through careful debugging and troubleshooting. For example, I initially had difficulty implementing the user login system, but I was able to resolve the issue by reviewing the code and consulting online resources for guidance.

I gained a lot of knowledge about creating user interfaces and using databases in Python by working on this assignment. In addition, I learned a lot about utilizing Tkinter to create a useful and attractive taxi booking system. I valued the chance to use my programming knowledge on a practical project, and I believe this experience has helped me become a better programmer. Overall, I believe that working on this assignment has helped me understand software development more thoroughly.

# Conclusion

In conclusion, it was difficult yet satisfying to create a desktop application for a taxi booking system using Python and Tkinter. In order to design a useful and user-friendly application, I had to leverage my understanding of programming languages and software development concepts.I ran across a couple obstacles, but I was able to get beyond them by carefully debugging and troubleshooting. I also had the chance to improve my understanding of Python and tkinter and acquire new abilities.

Overall, I am pleased with the outcome and think this experience has helped me develop as a programmer. I am certain that the Taxi booking system application will be able to satisfy the demands of the intended audience and that it has the potential to be a helpful and handy tool for users.

# References

Visual Paradigm (2021). What is Use Case Diagram. Retrieved from <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>. (Accessed on January 10, 2023)

IBM (2021). Structure Class Diagrams. Retrieved from

<https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams>. (Accessed on January 10, 2023)

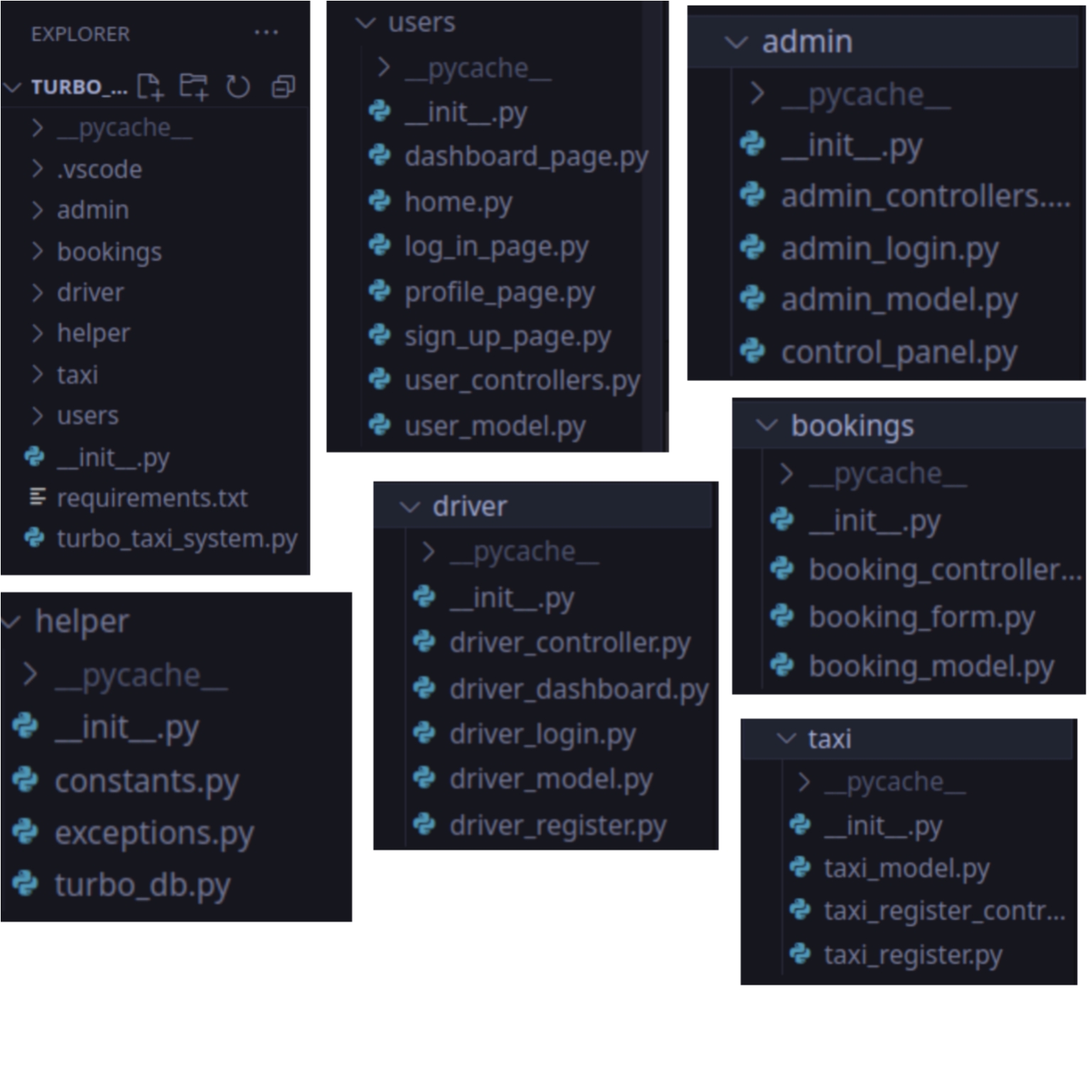
Kumar, R. (2021). What is Pydantic? [online] DZone. Available at: [https://dzone.com/articles/what-is-pydantic-1](https://dzone.com/articles/what-is-pydantic-1" \l ":~:text=Pydantic is a Python library,for JSON encoding and decoding). (Accessed 10 January 2023)

Schimansky, T. (n.d.). TkinterMapView. Retrieved from <https://github.com/TomSchimansky/TkinterMapView>. (Accessed on January 10, 2023)

# Appendix

## PACKAGES AND MODULES

In a Python project, packages and modules are used to organize the code and make it more reusable. All the packages and modules created for this project are listed below:

Figure 53: packages\_and\_modules

## SOURCE CODE OF TURBO TAXI BOOKING SYSTEM

**Package : turbo\_taxi\_booking\_system.py**

code :

from users.home import HomePage

HomePage()

* **Module : helper**

**Package : constants.py**

code :

LOGO\_LOCATION = "/home/vivu/Class Stuff V2/ISD\_Files/TurboTB\_System/img/logo.png"

BG\_LOCATION = "/home/vivu/Class Stuff V2/ISD\_Files/TurboTB\_System/img/background.png"

ADMIN\_LOGO\_LOCATION = (

"/home/vivu/Class Stuff V2/ISD\_Files/TurboTB\_System/img/admin\_logo.png"

)

DRIVER\_LOGO\_LOCATION = (

"/home/vivu/Class Stuff V2/ISD\_Files/TurboTB\_System/img/driver\_logo.png"

)

**Package : exceptions.py**

code :

class CustomException(Exception):

*# raised when empty data*

*pass*

**Package : turbo\_db.py**

code :

import psycopg2

class Turbo\_db:

    message = ""

    @staticmethod

    def turbo\_connection():

*try*:

            conn = psycopg2.connect(

                host="localhost",

                dbname="turbo\_db",

                user="incri",

                password="fastrack",

                port=5432,

            )

*return* conn

*except* Exception *as* error:

            Turbo\_db.message = error

* **Module : users**

**Package : home.py**

code :

import tkinter as tk

from tkinter import messagebox

from helper.constants import (

    LOGO\_LOCATION,

    BG\_LOCATION,

    ADMIN\_LOGO\_LOCATION,

    DRIVER\_LOGO\_LOCATION,

)

from users import sign\_up\_page

from users import log\_in\_page

from admin.admin\_login import LogInPage

from driver.driver\_login import DriverLogInPage

class HomePage:

    def \_\_init\_\_(self):

        self.root = tk.Tk()

        self.build\_root()

        self.create\_home\_frame()

    def build\_root(self):

        self.root.title("Main Frame")

        self.root.resizable(0, 0)

        self.root.configure(background="#FFFFFF")

        self.root.attributes("-fullscreen", True)

    def create\_home\_frame(self):

        self.home\_frame = tk.Frame(self.root)

        self.home\_frame.configure(background="#FFFFFF")

        self.home\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_home\_frame(home\_frame=self.home\_frame, root=self.root)

    @staticmethod

    def build\_home\_frame(home\_frame, root):

        top\_title = tk.Label(home\_frame)

        top\_title.place(relx=0.17, rely=0.068, height=51, width=153)

        top\_title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 22",

            foreground="#3650D3",

            text="""Turbo TB""",

        )

        hp\_sub\_title\_1 = tk.Label(home\_frame)

        hp\_sub\_title\_1.place(relx=0.104, rely=0.163, height=61, width=353)

        hp\_sub\_title\_1.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            text="""Turbo Taxi Booking System""",

        )

        hp\_sub\_title\_2 = tk.Label(home\_frame)

        hp\_sub\_title\_2.place(relx=0.348, rely=0.176, height=41, width=396)

        hp\_sub\_title\_2.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            foreground="#3650D3",

            text="""Online Taxi Booking""",

        )

        hp\_sub\_title\_3 = tk.Label(home\_frame)

        hp\_sub\_title\_3.place(relx=0.104, rely=0.244, height=40, width=597)

        hp\_sub\_title\_3.configure(activebackground="#f9f9f9")

        hp\_sub\_title\_3.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            foreground="#3650D3",

            text="""BEST TAXI SERVICE PROVIDER COMANY""",

        )

        top\_logo\_lable = tk.Label(home\_frame)

        top\_logo\_lable.place(relx=0.110, rely=0.068, height=71, width=80)

        top\_logo\_lable.configure(

            anchor="w",

            compound="left",

            background="#FFFFFF",

        )

        logo\_image = tk.PhotoImage(file=LOGO\_LOCATION)

        top\_logo\_lable.configure(image=logo\_image)

        admin\_logo\_image = tk.PhotoImage(file=ADMIN\_LOGO\_LOCATION)

        sub\_logo\_label\_1 = tk.Button(home\_frame)

        sub\_logo\_label\_1.place(relx=0.110, rely=0.407, height=71, width=80)

        sub\_logo\_label\_1.configure(background="#FFFFFF", relief="flat")

        sub\_logo\_label\_1.configure(

            image=admin\_logo\_image,

            command=lambda: HomePage.redirect\_to\_admin\_page(root),

        )

        info\_level\_1 = tk.Label(home\_frame)

        info\_level\_1.place(relx=0.17, rely=0.393, height=39, width=153)

        info\_level\_1.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            text="""Login As Admin ?""",

        )

        sub\_info\_level\_1 = tk.Label(home\_frame)

        sub\_info\_level\_1.place(relx=0.17, rely=0.448, height=45, width=510)

        sub\_info\_level\_1.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

        )

        driver\_logo\_image = tk.PhotoImage(file=DRIVER\_LOGO\_LOCATION)

        sub\_info\_level\_1.configure(text="click on the admin icon to login as admin.")

        sub\_logo\_label\_2 = tk.Button(home\_frame)

        sub\_logo\_label\_2.place(relx=0.110, rely=0.583, height=71, width=80)

        sub\_logo\_label\_2.configure(background="#FFFFFF", relief="flat")

        sub\_logo\_label\_2.configure(

            image=driver\_logo\_image,

            command=lambda: HomePage.redirect\_to\_driver\_page(root),

        )

        info\_level\_2 = tk.Label(home\_frame)

        info\_level\_2.place(relx=0.17, rely=0.57, height=39, width=153)

        info\_level\_2.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            text="""Login As Driver ?""",

        )

        sub\_info\_level\_2 = tk.Label(home\_frame)

        sub\_info\_level\_2.place(relx=0.17, rely=0.624, height=45, width=510)

        sub\_info\_level\_2.configure(

            anchor="w",

            foreground="#637381",

            background="#FFFFFF",

            compound="left",

            text="click on the admin icon to login as driver.",

        )

        homebg\_lable = tk.Label(home\_frame)

        homebg\_lable.place(relx=0.538, rely=0.0, height=770, width=664)

        homebg\_lable.configure(anchor="w", compound="left")

        background\_image = tk.PhotoImage(file=BG\_LOCATION)

        homebg\_lable.configure(image=background\_image)

        signup\_redirect\_button = tk.Button(home\_frame)

        signup\_redirect\_button.place(relx=0.783, rely=0.054, height=43, width=111)

        signup\_redirect\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Sign Up""",

            command=lambda: HomePage.redirect\_to\_signup\_page(root),

        )

        login\_redirect\_button = tk.Button(home\_frame)

        login\_redirect\_button.place(relx=0.893, rely=0.054, height=43, width=111)

        login\_redirect\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Log In""",

            command=lambda: HomePage.redirect\_to\_login\_page(root),

        )

        app\_exit\_button = tk.Button(home\_frame)

        app\_exit\_button.place(relx=0.100, rely=0.865, height=43, width=111)

        app\_exit\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""EXIT""",

            command=lambda: HomePage.app\_exit(root),

        )

        home\_frame.mainloop()

    @staticmethod

    def app\_exit(root):

        response = messagebox.askquestion(

            title="Exit !", message="Do you really want to exit ?", icon="warning"

        )

*if* response == "yes":

            root.destroy()

    @staticmethod

    def redirect\_to\_signup\_page(root):

        sign\_up\_page.SignUpPage(root)

    @staticmethod

    def redirect\_to\_admin\_page(root):

        LogInPage(root)

    @staticmethod

    def redirect\_to\_driver\_page(root):

        DriverLogInPage(root)

    @staticmethod

    def redirect\_to\_login\_page(root):

        log\_in\_page.LogInPage(root)

**Package : sign\_up\_page.py**

code :

import tkinter as tk

from tkinter import messagebox

from pydantic import ValidationError

from helper.constants import BG\_LOCATION

from .user\_model import UserModel

from .user\_controllers import UserController

from helper.exceptions import CustomException

class SignUpPage:

    def \_\_init\_\_(self, root):

        self.root = root

        self.create\_signup\_frame()

    def create\_signup\_frame(self):

        self.signup\_frame = tk.Frame(self.root)

        self.signup\_frame.configure(background="#FFFFFF")

        self.signup\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_signup\_frame(signup\_frame=self.signup\_frame)

    @staticmethod

    def build\_signup\_frame(signup\_frame):

        welcome = tk.Label(signup\_frame)

        welcome.place(relx=0.593, rely=0.090, height=60, width=323)

        welcome.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            text="""Welcome To Turbo TB""",

            font="-family {Noto Sans} -size 22 -weight bold",

        )

        quoest = tk.Label(signup\_frame)

        quoest.place(relx=0.644, rely=0.160, height=20, width=205)

        quoest.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14 -slant italic",

            text="""Travel For Your Heart""",

        )

        firstname\_label = tk.Label(signup\_frame)

        firstname\_label.place(relx=0.538, rely=0.231, height=20, width=135)

        firstname\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 16",

            compound="left",

            background="#FFFFFF",

            text="""First Name""",

        )

        firstname\_entry = tk.Entry(signup\_frame)

        firstname\_entry.place(relx=0.538, rely=0.27, height=33, relwidth=0.189)

        firstname\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        lastname\_label = tk.Label(signup\_frame)

        lastname\_label.place(relx=0.773, rely=0.231, height=20, width=135)

        lastname\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Last Name""",

        )

        lastname\_entry = tk.Entry(signup\_frame)

        lastname\_entry.place(relx=0.773, rely=0.27, height=33, relwidth=0.189)

        lastname\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        contact\_label = tk.Label(signup\_frame)

        contact\_label.place(relx=0.538, rely=0.338, height=21, width=135)

        contact\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 16",

            text="""Contact""",

        )

        contact\_entry = tk.Entry(signup\_frame)

        contact\_entry.place(relx=0.538, rely=0.379, height=33, relwidth=0.189)

        contact\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        address\_label = tk.Label(signup\_frame)

        address\_label.place(relx=0.773, rely=0.338, height=21, width=135)

        address\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Address""",

        )

        address\_entry = tk.Entry(signup\_frame)

        address\_entry.place(relx=0.773, rely=0.379, height=33, relwidth=0.189)

        address\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        email\_label = tk.Label(signup\_frame)

        email\_label.place(relx=0.538, rely=0.446, height=20, width=185)

        email\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Email Address""",

        )

        email\_entry = tk.Entry(signup\_frame)

        email\_entry.place(relx=0.538, rely=0.486, height=33, relwidth=0.419)

        email\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        username\_label = tk.Label(signup\_frame)

        username\_label.place(relx=0.538, rely=0.554, height=31, width=135)

        username\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""User Name""",

        )

        username\_entry = tk.Entry(signup\_frame)

        username\_entry.place(relx=0.538, rely=0.608, height=33, relwidth=0.189)

        username\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        password\_label = tk.Label(signup\_frame)

        password\_label.place(relx=0.773, rely=0.554, height=21, width=135)

        password\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Password""",

        )

        password\_entry = tk.Entry(signup\_frame)

        password\_entry.place(relx=0.773, rely=0.608, height=33, relwidth=0.189)

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        confrim\_password\_label = tk.Label(signup\_frame)

        confrim\_password\_label.place(relx=0.533, rely=0.678, height=20, width=205)

        confrim\_password\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Confirm Password""",

        )

        confrim\_password\_entry = tk.Entry(signup\_frame)

        confrim\_password\_entry.place(relx=0.538, rely=0.73, height=33, relwidth=0.189)

        confrim\_password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        show\_pass\_button = tk.Button(signup\_frame)

        show\_pass\_button.place(relx=0.74, rely=0.735, height=23, width=51)

        show\_pass\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""OO""",

        )

        show\_pass\_button.bind(

            "<Button-1>",

            lambda event: SignUpPage.show\_pass(

                event,

                password\_entry,

                confrim\_password\_entry,

            ),

        )

        show\_pass\_button.bind(

            "<ButtonRelease-1>",

            lambda event: SignUpPage.hide\_pass(

                event,

                password\_entry,

                confrim\_password\_entry,

            ),

        )

        signup\_button = tk.Button(signup\_frame)

        signup\_button.place(relx=0.666, rely=0.855, height=43, width=201)

        signup\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Sign Up""",

            command=lambda: SignUpPage.user\_register(

                firstname\_entry,

                lastname\_entry,

                contact\_entry,

                address\_entry,

                email\_entry,

                username\_entry,

                password\_entry,

                confrim\_password\_entry,

                signup\_frame,

            ),

        )

        background\_label = tk.Label(signup\_frame)

        background\_label.place(relx=-0.029, rely=0.0, height=770, width=700)

        background\_label.configure(anchor="w", compound="left")

        background\_image = tk.PhotoImage(file=BG\_LOCATION)

        background\_label.configure(image=background\_image)

        back\_button = tk.Button(signup\_frame)

        back\_button.place(relx=0.100, rely=0.865, height=43, width=111)

        back\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: SignUpPage.redirect\_to\_homepage(signup\_frame),

        )

        signup\_frame.mainloop()

    @staticmethod

    def show\_pass(

        event,

        password\_entry,

        confrim\_password\_entry,

    ):

        password = password\_entry.get()

        confrim\_password = confrim\_password\_entry.get()

        password\_entry.config(show="", text=password)

        confrim\_password\_entry.config(show="", text=confrim\_password)

    @staticmethod

    def hide\_pass(

        event,

        password\_entry,

        confrim\_password\_entry,

    ):

        password\_entry.config(show="\*")

        confrim\_password\_entry.config(show="\*")

    @staticmethod

    def redirect\_to\_homepage(signup\_frame):

        signup\_frame.destroy()

    @staticmethod

    def user\_register(

        firstname\_entry,

        lastname\_entry,

        contact\_entry,

        address\_entry,

        email\_entry,

        username\_entry,

        password\_entry,

        confrim\_password\_entry,

        signup\_frame,

    ):

*try*:

            user = UserModel(

                firstname=firstname\_entry.get(),

                lastname=lastname\_entry.get(),

                contact=contact\_entry.get(),

                address=address\_entry.get(),

                email=email\_entry.get(),

                username=username\_entry.get(),

                password=password\_entry.get(),

                confirm\_password=confrim\_password\_entry.get(),

            )

            user\_control = UserController()

            user\_control.registration\_control(user, signup\_frame)

*except* CustomException *as* e:

            messagebox.showerror("Invalid Data", e)

**Package : log\_in\_page.py**

code :

import imp

import tkinter as tk

from tkinter import messagebox

from helper.constants import BG\_LOCATION

from .user\_model import UserModel

from .user\_controllers import UserController

from helper.exceptions import CustomException

class LogInPage:

    def \_\_init\_\_(self, root):

        self.root = root

        self.create\_login\_frame()

    def create\_login\_frame(self):

        self.login\_frame = tk.Frame(self.root)

        self.login\_frame.configure(background="#FFFFFF")

        self.login\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_login\_frame(login\_frame=self.login\_frame, root=self.root)

    @staticmethod

    def build\_login\_frame(login\_frame, root):

        welcome = tk.Label(login\_frame)

        welcome.place(relx=0.696, rely=0.068, height=59, width=114)

        welcome.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 22 -weight bold",

            text="""Log In""",

        )

        quoest = tk.Label(login\_frame)

        quoest.place(relx=0.674, rely=0.149, height=20, width=185)

        quoest.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14 -slant italic",

            text="""Welcome Back User""",

        )

        username\_label = tk.Label(login\_frame)

        username\_label.place(relx=0.585, rely=0.299, height=21, width=206)

        username\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Username""",

        )

        username\_entry = tk.Entry(login\_frame)

        username\_entry.place(relx=0.585, rely=0.353, height=33, relwidth=0.301)

        username\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        password\_label = tk.Label(login\_frame)

        password\_label.place(relx=0.585, rely=0.434, height=20, width=185)

        password\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Password""",

        )

        password\_entry = tk.Entry(login\_frame)

        password\_entry.place(relx=0.585, rely=0.488, height=33, relwidth=0.301)

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        login\_button = tk.Button(login\_frame)

        login\_button.place(relx=0.674, rely=0.611, height=43, width=201)

        login\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Log In""",

            command=lambda: LogInPage.user\_login(

                username\_entry,

                password\_entry,

                login\_frame,

                root,

            ),

        )

        background\_label = tk.Label(login\_frame)

        background\_label.place(relx=-0.029, rely=0.0, height=739, width=700)

        background\_label.configure(anchor="w", compound="left")

        background\_image = tk.PhotoImage(file=BG\_LOCATION)

        background\_label.configure(image=background\_image)

        back\_button = tk.Button(login\_frame)

        back\_button.place(relx=0.100, rely=0.865, height=43, width=111)

        back\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: LogInPage.redirect\_to\_home(login\_frame),

        )

        show\_pass\_button = tk.Button(login\_frame)

        show\_pass\_button.place(relx=0.900, rely=0.500, height=23, width=51)

        show\_pass\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""OO""",

        )

        show\_pass\_button.bind(

            "<Button-1>",

            lambda event: LogInPage.show\_pass(

                event,

                password\_entry,

            ),

        )

        show\_pass\_button.bind(

            "<ButtonRelease-1>",

            lambda event: LogInPage.hide\_pass(

                event,

                password\_entry,

            ),

        )

        login\_frame.mainloop()

    @staticmethod

    def show\_pass(

        event,

        password\_entry,

    ):

        password = password\_entry.get()

        password\_entry.config(show="", text=password)

    @staticmethod

    def hide\_pass(

        event,

        password\_entry,

    ):

        password\_entry.config(show="\*")

    def redirect\_to\_home(login\_frame):

        login\_frame.destroy()

    @staticmethod

    def user\_login(

        username\_entry,

        password\_entry,

        login\_frame,

        root,

    ):

*try*:

            user = UserModel(

                username=username\_entry.get(),

                password=password\_entry.get(),

                confirm\_password=password\_entry.get(),

            )

            login\_frame.destroy()

            user\_control = UserController()

            user\_control.login\_control(user, root)

*except* CustomException *as* e:

            messagebox.showerror("Invalid Data", e)

**Package : dashboard\_page.py**

code :

import imp

import tkinter as tk

from tkinter import NO, W, Scrollbar, ttk

from helper.constants import LOGO\_LOCATION

from PIL import Image, ImageTk

from io import BytesIO

from .profile\_page import ProfilePage

from bookings.booking\_form import BookingPage

from .user\_model import UserModel

class DashboardPage:

    def \_\_init\_\_(self, root, record, user\_controller):

        self.root = root

        self.record = record

        self.user\_controller = user\_controller

        self.create\_dashboard\_frame()

    def create\_dashboard\_frame(self):

        self.dashboard\_frame = tk.Frame(self.root)

        self.dashboard\_frame.configure(background="#FFFFFF")

        self.dashboard\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_dashboard\_frame(

            dashboard\_frame=self.dashboard\_frame,

            root=self.root,

            record=self.record,

            user\_controller=self.user\_controller,

        )

    @staticmethod

    def build\_dashboard\_frame(

        dashboard\_frame,

        root,

        record,

        user\_controller,

    ):

        title = tk.Label(dashboard\_frame)

        title.place(relx=0.452, rely=0.285, height=40, width=212)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            foreground="#3056D3",

            text="""Your Trip History""",

        )

        profile\_label = tk.Label(dashboard\_frame)

        profile\_label.place(relx=0.91, rely=0.068, height=61, width=70)

        profile\_label.configure(

            anchor="w",

            compound="left",

            borderwidth=2,

            relief="solid",

        )

*for* data *in* record:

            fetched\_username = data[6].rstrip()

            fetched\_profile\_picture = data[8]

        profile\_img = Image.open(BytesIO(fetched\_profile\_picture))

        small\_img = profile\_img.resize((70, 61))

        view\_profile = ImageTk.PhotoImage(small\_img)

        profile\_label.configure(image=view\_profile)

        logo\_label = tk.Label(dashboard\_frame)

        logo\_label.place(relx=0.110, rely=0.068, height=71, width=80)

        logo\_label.configure(

            anchor="w",

            compound="left",

            text="""Logo""",

            background="#FFFFFF",

        )

        photo\_location = tk.PhotoImage(file=LOGO\_LOCATION)

        logo\_label.configure(image=photo\_location)

        username\_button = tk.Button(dashboard\_frame)

        username\_button.place(relx=0.799, rely=0.095, height=33, width=131)

        username\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            text=fetched\_username,

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            command=lambda: DashboardPage.redirect\_to\_profile\_page(

                root,

                dashboard\_frame,

                record,

                user\_controller,

            ),

        )

        tabel\_label = tk.Label(dashboard\_frame)

        tabel\_label.place(relx=0.110, rely=0.407, height=229, width=1160)

        tabel\_label.configure(anchor="w", compound="left", text="""Table""")

        book\_now\_button = tk.Button(dashboard\_frame)

        book\_now\_button.place(relx=0.466, rely=0.814, height=43, width=171)

        book\_now\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#FFFFFF",

            text="""Book Now""",

            command=lambda: DashboardPage.redirect\_to\_booking\_form(

                root,

                dashboard\_frame,

                record,

            ),

        )

        refresh\_button = tk.Button(dashboard\_frame)

        refresh\_button.place(relx=0.799, rely=0.814, height=33, width=131)

        refresh\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            text="Refresh Page",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            command=lambda: DashboardPage.refresh\_page(

                user\_controller,

                record,

                root,

                dashboard\_frame,

            ),

        )

        DashboardPage.booking\_history\_frame(dashboard\_frame, record, user\_controller)

        dashboard\_frame.mainloop()

    @staticmethod

    def redirect\_to\_profile\_page(

        root,

        dashboard\_frame,

        record,

        user\_controller,

    ):

        ProfilePage(root, dashboard\_frame, record, user\_controller)

    @staticmethod

    def redirect\_to\_booking\_form(

        root,

        dashboard\_frame,

        record,

    ):

        BookingPage(

            root,

            dashboard\_frame,

            record,

        )

    @staticmethod

    def refresh\_page(user\_controller, record, root, dashboard\_frame):

        dashboard\_frame.destroy()

*for* data *in* record:

            fetched\_username = data[6]

            fetched\_password = data[7]

        user = UserModel(

            username=fetched\_username,

            password=fetched\_password,

            confirm\_password=fetched\_password,

        )

        user\_control = user\_controller()

        user\_control.login\_control(user, root)

    @staticmethod

    def booking\_history\_frame(dashboard\_frame, record, user\_controller):

*for* data *in* record:

            userid = data[0]

        history\_tabel\_frame = tk.Frame(dashboard\_frame, bg="#FFFFFF")

        history\_tabel\_frame.place(

            relx=0.110,

            rely=0.407,

            height=229,

            width=1160,

        )

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

        table\_scroll\_bar = Scrollbar(history\_tabel\_frame)

        table\_scroll\_bar.place(relx=0.980, rely=0, height=229, width=15)

        history\_booking\_table = ttk.Treeview(

            history\_tabel\_frame,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        history\_booking\_table.place(

            relx=0,

            rely=0,

            height=229,

            width=1120,

        )

        table\_scroll\_bar.config(

            command=history\_booking\_table.yview,

        )

        history\_booking\_table["columns"] = (

            "Booking ID",

            "Full Name",

            "Date and Time",

            "Destination",

            "Total Cost",

            "Status",

        )

        history\_booking\_table.column("#0", width=0, stretch=NO)

        history\_booking\_table.column("Booking ID", width=20, anchor=W)

        history\_booking\_table.column("Full Name", width=70, anchor=W)

        history\_booking\_table.column("Date and Time", width=70, anchor=W)

        history\_booking\_table.column("Destination", width=240, anchor=W)

        history\_booking\_table.column("Total Cost", width=40, anchor=W)

        history\_booking\_table.column("Status", width=40, anchor=W)

        history\_booking\_table.heading("#0", text="", anchor=W)

        history\_booking\_table.heading("Booking ID", text="Booking ID", anchor=W)

        history\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        history\_booking\_table.heading("Date and Time", text="Date and Time", anchor=W)

        history\_booking\_table.heading("Destination", text="Destination", anchor=W)

        history\_booking\_table.heading("Total Cost", text="Total Cost", anchor=W)

        history\_booking\_table.heading("Status", text="Status", anchor=W)

        history\_trip\_control = user\_controller()

        trip\_fetched\_data = history\_trip\_control.history\_trip\_detail\_fetcher(userid)

*for* data *in* trip\_fetched\_data:

            history\_booking\_table.insert(

                "",

                index="end",

                values=(

                    data[0],

                    (data[2], data[3]),

                    (data[6], "---", data[7], ":", data[8]),

                    data[10],

                    data[11],

                    data[16],

                ),

            )

**Package : profile\_page.py**

code :

import tkinter as tk

from tkinter import NO, W, Scrollbar, messagebox, filedialog

from tkinter import ttk

from helper.constants import LOGO\_LOCATION

from PIL import Image, ImageTk

from io import BytesIO

from .user\_model import UserModel

from helper.exceptions import CustomException

from tkintermapview import TkinterMapView

class ProfilePage:

    def \_\_init\_\_(self, root, dashboard\_frame, record, user\_controller):

        self.root = root

        self.dashboard\_frame = dashboard\_frame

        self.record = record

        self.user\_controller = user\_controller

        self.create\_profile\_frame()

    def create\_profile\_frame(self):

        self.profile\_frame = tk.Frame(self.root)

        self.profile\_frame.configure(background="#FFFFFF")

        self.profile\_frame.place(

            relx=-0.010,

            rely=0,

            height=768,

            width=1366,

        )

        self.build\_profile\_frame(

            profile\_frame=self.profile\_frame,

            root=self.root,

            dashboard\_frame=self.dashboard\_frame,

            record=self.record,

            user\_controller=self.user\_controller,

        )

    @staticmethod

    def build\_profile\_frame(

        profile\_frame,

        root,

        dashboard\_frame,

        record,

        user\_controller,

    ):

        logo = tk.Label(profile\_frame)

        logo.place(x=91, y=60, height=81, width=82)

        logo.configure(

            anchor="w",

            compound="left",

            text="""logo""",

            background="#FFFFFF",

        )

        photo\_location = tk.PhotoImage(file=LOGO\_LOCATION)

        logo.configure(image=photo\_location)

        logout\_button = tk.Button(profile\_frame)

        logout\_button.place(x=1159, y=60, height=43, width=111)

        logout\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Log Out""",

            command=lambda: ProfilePage.log\_out(

                profile\_frame,

                dashboard\_frame,

            ),

        )

*for* data *in* record:

            featched\_username = data[6].rstrip()

            fetched\_email = data[5].rstrip()

            fetched\_address = data[4].rstrip()

            fetched\_contact = data[3].rstrip()

            fetched\_profile = data[8]

        profile\_img = Image.open(BytesIO(fetched\_profile))

        big\_img = profile\_img.resize((153, 133))

        view\_profile = ImageTk.PhotoImage(big\_img)

        profile\_photo = tk.Label(profile\_frame)

        profile\_photo.place(x=59, y=210, height=133, width=153)

        profile\_photo.configure(

            anchor="w",

            compound="left",

            text="""profile""",

            borderwidth=2,

            relief="solid",

        )

        profile\_photo.configure(image=view\_profile)

        username = tk.Label(profile\_frame)

        username.place(x=259, y=210, height=31, width=115)

        username.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16 -weight bold",

            text=featched\_username,

        )

        email = tk.Label(profile\_frame)

        email.place(x=259, y=260, height=21, width=67)

        email.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Email :""",

        )

        contact = tk.Label(profile\_frame)

        contact.place(x=259, y=290, height=20, width=95)

        contact.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Phone no :""",

        )

        address = tk.Label(profile\_frame)

        address.place(x=259, y=320, height=20, width=87)

        address.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Address :""",

        )

        email\_data = tk.Label(profile\_frame)

        email\_data.place(x=320, y=255, height=30, width=300)

        email\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text=fetched\_email,

        )

        contact\_data = tk.Label(profile\_frame)

        contact\_data.place(x=359, y=290, height=20, width=95)

        contact\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text=fetched\_contact,

        )

        address\_data = tk.Label(profile\_frame)

        address\_data.place(x=350, y=317, height=30, width=300)

        address\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text=fetched\_address,

        )

        edit\_profile\_picture = tk.Button(profile\_frame)

        edit\_profile\_picture.place(x=59, y=380, height=23, width=91)

        edit\_profile\_picture.configure(

            background="#FFFFFF",

            borderwidth="2",

            compound="left",

            text="""Edit Profile >""",

            command=lambda: ProfilePage.edit\_account\_profile(

                user\_controller,

                record,

                profile\_frame,

                dashboard\_frame,

            ),

        )

        edit\_bio\_button = tk.Button(profile\_frame)

        edit\_bio\_button.place(x=259, y=380, height=23, width=101)

        edit\_bio\_button.configure(

            background="#FFFFFF",

            borderwidth="2",

            compound="left",

            text="""Edit Bio >""",

            command=lambda: ProfilePage.create\_edit\_bio\_page(

                record,

                user\_controller,

                profile\_frame,

                dashboard\_frame,

            ),

        )

        upcoming\_trips = tk.Label(profile\_frame)

        upcoming\_trips.place(x=550, y=440, height=41, width=245)

        upcoming\_trips.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            foreground="#3056D3",

            text="""Your Upcoming Trips""",

        )

        back\_button = tk.Button(profile\_frame)

        back\_button.place(x=50, y=630, height=43, width=111)

        back\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: ProfilePage.redirect\_to\_dashboard(profile\_frame),

        )

        ProfilePage.upcoming\_trip\_table\_viwer(

            profile\_frame,

            user\_controller,

            record,

        )

        profile\_frame.mainloop()

    @staticmethod

    def redirect\_to\_dashboard(profile\_frame):

        profile\_frame.destroy()

    @staticmethod

    def log\_out(profile\_frame, dashboard\_frame):

        response = messagebox.askquestion(

            title="Log Out !",

            message="Do you really want to Log Out ?",

            icon="warning",

        )

*if* response == "yes":

            profile\_frame.destroy()

            dashboard\_frame.destroy()

    @staticmethod

    def create\_edit\_bio\_page(

        record,

        user\_controller,

        profile\_frame,

        dashboard\_frame,

    ):

        edit\_bio\_frame = tk.Toplevel(profile\_frame)

        edit\_bio\_frame.title("Edit Bio")

        edit\_bio\_frame.resizable(0, 0)

        edit\_bio\_frame.configure(background="#FFFFFF")

        edit\_bio\_frame.geometry("460x579+420+91")

        ProfilePage.build\_edit\_bio\_page(

            edit\_bio\_frame,

            record,

            user\_controller,

            profile\_frame,

            dashboard\_frame,

        )

    @staticmethod

    def build\_edit\_bio\_page(

        edit\_bio\_frame,

        record,

        user\_controller,

        profile\_frame,

        dashboard\_frame,

    ):

        welcome = tk.Label(edit\_bio\_frame)

        welcome.place(relx=0.413, rely=0.069, height=41, width=104)

        welcome.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            text="""Edit Bio""",

            font="-family {Noto Sans} -size 16 -weight bold",

        )

*for* data *in* record:

            fetched\_firstname = data[1].rstrip()

            fetched\_lastname = data[2].rstrip()

            fetched\_email = data[5].rstrip()

        firstname = tk.Label(edit\_bio\_frame)

        firstname.place(relx=0.065, rely=0.19, height=41, width=104)

        firstname.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 14",

            compound="left",

            background="#FFFFFF",

            text="""First Name""",

        )

        firstname\_entry = tk.Entry(edit\_bio\_frame)

        firstname\_entry.insert(0, fetched\_firstname)

        firstname\_entry.place(

            relx=0.065,

            rely=0.259,

            height=23,

            relwidth=0.404,

        )

        firstname\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            state="disabled",

        )

        lastname = tk.Label(edit\_bio\_frame)

        lastname.place(relx=0.543, rely=0.19, height=41, width=104)

        lastname.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Last Name""",

        )

        lastname\_entry = tk.Entry(edit\_bio\_frame)

        lastname\_entry.insert(0, fetched\_lastname)

        lastname\_entry.place(

            relx=0.543,

            rely=0.259,

            height=23,

            relwidth=0.404,

        )

        lastname\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            state="disabled",

        )

        contact = tk.Label(edit\_bio\_frame)

        contact.place(relx=0.065, rely=0.311, height=41, width=104)

        contact.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            text="""Contact""",

        )

        contact\_entry = tk.Entry(edit\_bio\_frame)

        contact\_entry.place(

            relx=0.065,

            rely=0.38,

            height=23,

            relwidth=0.404,

        )

        contact\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        address = tk.Label(edit\_bio\_frame)

        address.place(relx=0.543, rely=0.311, height=41, width=104)

        address.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Address""",

        )

        address\_entry = tk.Entry(edit\_bio\_frame)

        address\_entry.place(

            relx=0.543,

            rely=0.38,

            height=23,

            relwidth=0.404,

        )

        address\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        email = tk.Label(edit\_bio\_frame)

        email.place(relx=0.065, rely=0.432, height=41, width=300)

        email.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Email Address""",

        )

        email\_entry = tk.Entry(edit\_bio\_frame)

        email\_entry.insert(0, fetched\_email)

        email\_entry.place(

            relx=0.065,

            rely=0.501,

            height=23,

            relwidth=0.883,

        )

        email\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            state="disabled",

        )

        username = tk.Label(edit\_bio\_frame)

        username.place(relx=0.065, rely=0.553, height=41, width=104)

        username.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""User Name""",

        )

        username\_entry = tk.Entry(edit\_bio\_frame)

        username\_entry.place(

            relx=0.065,

            rely=0.622,

            height=23,

            relwidth=0.404,

        )

        username\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        password = tk.Label(edit\_bio\_frame)

        password.place(relx=0.543, rely=0.553, height=41, width=300)

        password.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Old Password""",

        )

        password\_entry = tk.Entry(edit\_bio\_frame)

        password\_entry.place(

            relx=0.543,

            rely=0.622,

            height=23,

            relwidth=0.404,

        )

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        confirm\_password = tk.Label(edit\_bio\_frame)

        confirm\_password.place(

            relx=0.065,

            rely=0.674,

            height=41,

            width=300,

        )

        confirm\_password.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""New Password""",

        )

        confrim\_password\_entry = tk.Entry(edit\_bio\_frame)

        confrim\_password\_entry.place(

            relx=0.065,

            rely=0.76,

            height=23,

            relwidth=0.404,

        )

        confrim\_password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        show\_password = tk.Button(edit\_bio\_frame)

        show\_password.place(

            relx=0.5,

            rely=0.757,

            height=23,

            width=51,

        )

        show\_password.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""OO""",

        )

        show\_password.bind(

            "<Button-1>",

            lambda event: ProfilePage.show\_pass(

                event,

                password\_entry,

                confrim\_password\_entry,

            ),

        )

        show\_password.bind(

            "<ButtonRelease-1>",

            lambda event: ProfilePage.hide\_pass(

                event,

                password\_entry,

                confrim\_password\_entry,

            ),

        )

        update\_button = tk.Button(edit\_bio\_frame)

        update\_button.place(relx=0.522, rely=0.864, height=43, width=111)

        update\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Update""",

            command=lambda: ProfilePage.edit\_user\_bio(

                record,

                edit\_bio\_frame,

                profile\_frame,

                dashboard\_frame,

                user\_controller,

                contact\_entry,

                address\_entry,

                username\_entry,

                password\_entry,

                confrim\_password\_entry,

            ),

        )

        delete\_button = tk.Button(edit\_bio\_frame)

        delete\_button.place(relx=0.690, rely=0.757, height=23, width=120)

        delete\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10",

            text="""Delete Account""",

            command=lambda: ProfilePage.delete\_account(

                user\_controller,

                profile\_frame,

                dashboard\_frame,

                edit\_bio\_frame,

                record,

            ),

        )

        cancel\_button = tk.Button(edit\_bio\_frame)

        cancel\_button.place(relx=0.250, rely=0.864, height=43, width=111)

        cancel\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Cancel""",

            command=lambda: ProfilePage.cancel(edit\_bio\_frame),

        )

        edit\_bio\_frame.grab\_set()

        edit\_bio\_frame.mainloop()

    @staticmethod

    def show\_pass(

        event,

        password\_entry,

        confrim\_password\_entry,

    ):

        password = password\_entry.get()

        confrim\_password = confrim\_password\_entry.get()

        password\_entry.config(show="", text=password)

        confrim\_password\_entry.config(show="", text=confrim\_password)

    @staticmethod

    def hide\_pass(

        event,

        password\_entry,

        confrim\_password\_entry,

    ):

        password\_entry.config(show="\*")

        confrim\_password\_entry.config(show="\*")

    @staticmethod

    def cancel(edit\_bio\_frame):

        edit\_bio\_frame.destroy()

    @staticmethod

    def delete\_account(

        user\_controller,

        profile\_frame,

        dashboard\_frame,

        edit\_bio\_frame,

        record,

    ):

        account\_control = user\_controller()

        response = messagebox.askquestion(

            "Delete Account",

            "Do you really want to delete your account? \n  \

            you wont be able to recover your account again",

        )

*if* response == "yes":

            account\_control.delete\_account(

                profile\_frame,

                dashboard\_frame,

                edit\_bio\_frame,

                record,

            )

    @staticmethod

    def edit\_account\_profile(

        user\_controller,

        record,

        profile\_frame,

        dashboaard\_frame,

    ):

*for* data *in* record:

            fetched\_password = data[7]

        file = filedialog.askopenfilename()

*if* file:

            user = UserModel(

                profile=file,

                password=fetched\_password,

                confirm\_password=fetched\_password,

            )

            profile\_picture\_control = user\_controller()

            profile\_picture\_control.change\_profile(

                user,

                record,

                profile\_frame,

                dashboaard\_frame,

            )

    @staticmethod

    def edit\_user\_bio(

        record,

        edit\_bio\_frame,

        profile\_frame,

        dashboard\_frame,

        user\_controller,

        contact\_entry,

        address\_entry,

        username\_entry,

        password\_entry,

        confrim\_password\_entry,

    ):

*for* data *in* record:

            fetched\_contact = data[3]

            fetched\_address = data[4]

            fetched\_username = data[6]

            fetched\_password = data[7]

*# checking if data is empty or not*

*if* contact\_entry.get() == "":

            new\_contact = fetched\_contact

*else*:

            new\_contact = contact\_entry.get()

*if* address\_entry.get() == "":

            new\_address = fetched\_address

*else*:

            new\_address = address\_entry.get()

*if* username\_entry.get() == "":

            new\_username = fetched\_username

*else*:

            new\_username = username\_entry.get()

*if* confrim\_password\_entry.get() == "":

            new\_password\_update = fetched\_password

*else*:

            new\_password\_update = confrim\_password\_entry.get()

*if* password\_entry.get() == fetched\_password:

            old\_password = password\_entry.get()

*else*:

            old\_password = password\_entry.get()

*try*:

            user = UserModel(

                contact=new\_contact,

                address=new\_address,

                username=new\_username,

                password=old\_password,

                confirm\_password=old\_password,

                new\_password=new\_password\_update,

            )

            user\_bio\_control = user\_controller()

            user\_bio\_control.update\_user\_bio(

                user,

                record,

                edit\_bio\_frame,

                profile\_frame,

                dashboard\_frame,

            )

*except* CustomException *as* e:

            messagebox.showerror("Invalid Data", e, parent=edit\_bio\_frame)

    @staticmethod

    def upcoming\_trip\_table\_viwer(profile\_frame, user\_controller, record):

*for* data *in* record:

            userid = data[0]

        upcoming\_trip\_table = tk.Frame(profile\_frame, bg="#FFFFFF")

        upcoming\_trip\_table.place(

            x=49,

            y=490,

            height=130,

            width=1212,

        )

*# Upcoming booking details table*

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

*# driver Table Scroll Bar*

        table\_scroll\_bar = Scrollbar(upcoming\_trip\_table)

        table\_scroll\_bar.place(relx=0.985, rely=0, height=130, width=15)

*# driver Table Build*

        upcoming\_booking\_table = ttk.Treeview(

            upcoming\_trip\_table,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        upcoming\_booking\_table.place(

            relx=0,

            rely=0,

            height=280,

            width=1180,

        )

*# upcoming a scroll bar*

        table\_scroll\_bar.config(

            command=upcoming\_booking\_table.yview,

        )

        upcoming\_booking\_table["columns"] = (

            "Booking ID",

            "Full Name",

            "Date and Time",

            "Destination",

            "Total Cost",

            "Status",

        )

*# Format Our Columns*

        upcoming\_booking\_table.column("#0", width=0, stretch=NO)

        upcoming\_booking\_table.column("Booking ID", width=20, anchor=W)

        upcoming\_booking\_table.column("Full Name", width=70, anchor=W)

        upcoming\_booking\_table.column("Date and Time", width=70, anchor=W)

        upcoming\_booking\_table.column("Destination", width=240, anchor=W)

        upcoming\_booking\_table.column("Total Cost", width=40, anchor=W)

        upcoming\_booking\_table.column("Status", width=40, anchor=W)

*# Create Heading*

        upcoming\_booking\_table.heading("#0", text="", anchor=W)

        upcoming\_booking\_table.heading("Booking ID", text="Booking ID", anchor=W)

        upcoming\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        upcoming\_booking\_table.heading("Date and Time", text="Date and Time", anchor=W)

        upcoming\_booking\_table.heading("Destination", text="Destination", anchor=W)

        upcoming\_booking\_table.heading("Total Cost", text="Total Cost", anchor=W)

        upcoming\_booking\_table.heading("Status", text="Status", anchor=W)

        upcoming\_trip\_control = user\_controller()

        trip\_fetched\_data = upcoming\_trip\_control.upcoming\_trip\_detail\_fetcher(userid)

*for* data *in* trip\_fetched\_data:

            upcoming\_booking\_table.insert(

                "",

                index="end",

                values=(

                    data[0],

                    (data[2], data[3]),

                    (data[6], "---", data[7], ":", data[8]),

                    data[10],

                    data[11],

                    data[16],

                ),

            )

            upcoming\_booking\_table.bind(

                "<Double-1>",

                lambda event: ProfilePage.user\_profile\_booking\_detail(

                    event,

                    upcoming\_booking\_table,

                    user\_controller,

                ),

            )

    @staticmethod

    def user\_profile\_booking\_detail(

        event,

        upcoming\_booking\_table,

        user\_controller,

    ):

        selected = upcoming\_booking\_table.focus()

        selected\_row = upcoming\_booking\_table.item(

            selected,

            "values",

        )

        selected\_booking\_id = selected\_row[0]

*# Booking Details Data Fetcher*

        booking\_data\_control = user\_controller()

        fetched\_booking\_details = (

            booking\_data\_control.user\_profile\_booking\_data\_fetcher(

                selected\_booking\_id,

            )

        )

        data = fetched\_booking\_details[0]

        upcoming\_booking\_detail\_frame = tk.Toplevel(upcoming\_booking\_table)

        upcoming\_booking\_detail\_frame.title("Assign Taxi")

        upcoming\_booking\_detail\_frame.geometry("682x641+193+31")

        upcoming\_booking\_detail\_frame.resizable(0, 0)

        upcoming\_booking\_detail\_frame.configure(background="#FFFFFF")

*# BUILDING BOOKING DETAIL PAGE*

        title = tk.Label(upcoming\_booking\_detail\_frame)

        title.place(relx=0.360, rely=0.031, height=43, width=250)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            text="""Booking Details""",

        )

        fullname\_label = tk.Label(upcoming\_booking\_detail\_frame)

        fullname\_label.place(relx=0.029, rely=0.156, height=34, width=122)

        fullname\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Full Name :""",

        )

        fullname\_data = tk.Label(upcoming\_booking\_detail\_frame)

        fullname\_data.place(relx=0.22, rely=0.156, height=34, width=280)

        fullname\_data.configure(

            anchor="w",

            background="#FFFFFF",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[2] + " " + data[3],

        )

        map\_view\_frame = tk.Frame(upcoming\_booking\_detail\_frame)

        map\_view\_frame.place(relx=0.640, rely=0.156, height=140, width=220)

        map\_view\_frame.configure(

            background="#FFFFFF",

            borderwidth="2",

            highlightthickness="2",

        )

        location\_detail\_map\_view = TkinterMapView(map\_view\_frame)

        location\_detail\_map\_view.set\_tile\_server(

            "https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",

            max\_zoom=60,

        )

        pickup\_cordinates = eval(data[19])

        destination\_coordinates = eval(data[20])

        location\_detail\_map\_view.set\_marker(

            pickup\_cordinates[0],

            pickup\_cordinates[1],

        )

        location\_detail\_map\_view.set\_marker(

            destination\_coordinates[0],

            destination\_coordinates[1],

        )

        location\_detail\_map\_view.set\_position(

            pickup\_cordinates[0],

            pickup\_cordinates[1],

        )

        location\_detail\_map\_view.set\_path(

            position\_list=[pickup\_cordinates, destination\_coordinates]

        )

        location\_detail\_map\_view.pack()

        passenger\_no\_label = tk.Label(upcoming\_booking\_detail\_frame)

        passenger\_no\_label.place(relx=0.029, rely=0.234, height=34, width=175)

        passenger\_no\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""No of Passenger :""",

        )

        passenger\_no\_data = tk.Label(upcoming\_booking\_detail\_frame)

        passenger\_no\_data.place(relx=0.293, rely=0.234, height=34, width=48)

        passenger\_no\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[4],

        )

        booking\_status = tk.Label(upcoming\_booking\_detail\_frame)

        booking\_status.place(relx=0.029, rely=0.312, height=35, width=180)

        booking\_status.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Booking Status :""",

        )

        booking\_status\_data = tk.Label(upcoming\_booking\_detail\_frame)

        booking\_status\_data.place(relx=0.250, rely=0.312, height=35, width=150)

        booking\_status\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[16],

        )

        pick\_up\_date\_lable = tk.Label(upcoming\_booking\_detail\_frame)

        pick\_up\_date\_lable.place(relx=0.029, rely=0.39, height=34, width=133)

        pick\_up\_date\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Pick Up Date :""",

        )

        pick\_up\_date\_data = tk.Label(upcoming\_booking\_detail\_frame)

        pick\_up\_date\_data.place(relx=0.249, rely=0.39, height=35, width=163)

        pick\_up\_date\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[6],

        )

        time\_label = tk.Label(upcoming\_booking\_detail\_frame)

        time\_label.place(relx=0.513, rely=0.39, height=34, width=69)

        time\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Time :""",

        )

        data7 = str(data[7])

        data8 = str(data[8])

        time\_data = tk.Label(upcoming\_booking\_detail\_frame)

        time\_data.place(relx=0.616, rely=0.39, height=34, width=100)

        time\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=(data7 + ":" + data8),

        )

        pick\_up\_address\_label = tk.Label(upcoming\_booking\_detail\_frame)

        pick\_up\_address\_label.place(relx=0.029, rely=0.468, height=35, width=175)

        pick\_up\_address\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Pick Up Address :""",

        )

        pickup\_address\_data = tk.Label(upcoming\_booking\_detail\_frame)

        pickup\_address\_data.place(relx=0.293, rely=0.468, height=35, width=323)

        pickup\_address\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[9],

        )

        destination\_label = tk.Label(upcoming\_booking\_detail\_frame)

        destination\_label.place(relx=0.029, rely=0.546, height=34, width=132)

        destination\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Destination :""",

        )

        destination\_address\_lable = tk.Label(upcoming\_booking\_detail\_frame)

        destination\_address\_lable.place(relx=0.249, rely=0.546, height=34, width=323)

        destination\_address\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[10],

        )

        total\_cost\_label = tk.Label(upcoming\_booking\_detail\_frame)

        total\_cost\_label.place(relx=0.029, rely=0.624, height=34, width=111)

        total\_cost\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Total Cost :""",

        )

        total\_cost\_data = tk.Label(upcoming\_booking\_detail\_frame)

        total\_cost\_data.place(relx=0.22, rely=0.624, height=34, width=206)

        total\_cost\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[11] + " .NPR",

        )

        payment\_method\_lable = tk.Label(upcoming\_booking\_detail\_frame)

        payment\_method\_lable.place(relx=0.029, rely=0.702, height=33, width=185)

        payment\_method\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Payment Method :""",

        )

        payment\_method\_data = tk.Label(upcoming\_booking\_detail\_frame)

        payment\_method\_data.place(relx=0.293, rely=0.702, height=34, width=206)

        payment\_method\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[12],

        )

        driver\_details\_lable = tk.Label(upcoming\_booking\_detail\_frame)

        driver\_details\_lable.place(relx=0.029, rely=0.780, height=33, width=185)

        driver\_details\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Driver Details :""",

        )

        driver\_details\_name = tk.Label(upcoming\_booking\_detail\_frame)

        driver\_details\_name.place(relx=0.250, rely=0.780, height=34, width=206)

        driver\_details\_name.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[23],

        )

        driver\_details\_number = tk.Label(upcoming\_booking\_detail\_frame)

        driver\_details\_number.place(relx=0.600, rely=0.780, height=34, width=206)

        driver\_details\_number.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[25],

        )

        taxi\_details\_lable = tk.Label(upcoming\_booking\_detail\_frame)

        taxi\_details\_lable.place(relx=0.025, rely=0.858, height=33, width=185)

        taxi\_details\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Taxi Details :""",

        )

        data29 = str(data[31])

        data30 = str(data[32])

*if* (data29) == "None":

            data29 = ""

*if* (data30) == "None":

            data30 = ""

        taxi\_details\_name = tk.Label(upcoming\_booking\_detail\_frame)

        taxi\_details\_name.place(relx=0.220, rely=0.858, height=34, width=465)

        taxi\_details\_name.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data29 + " " + data30,

        )

        taxi\_details\_number = tk.Label(upcoming\_booking\_detail\_frame)

        taxi\_details\_number.place(relx=0.220, rely=0.936, height=34, width=206)

        taxi\_details\_number.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[33],

        )

        back\_button = tk.Button(upcoming\_booking\_detail\_frame)

        back\_button.place(relx=0.865, rely=0.031, height=33, width=71)

        back\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#FFFFFF",

            text="""Back""",

        )

        cancel\_booking\_button = tk.Button(upcoming\_booking\_detail\_frame)

        cancel\_booking\_button.place(relx=0.580, rely=0.936, height=34, width=130)

        cancel\_booking\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10 -weight bold",

            text="""Cancel Booking""",

            command=lambda: ProfilePage.user\_cancel\_booking(

                user\_controller, upcoming\_booking\_detail\_frame, data

            ),

        )

*if* data[16] == "Accepted":

            cancel\_booking\_button.config(state="disable")

        upcoming\_booking\_detail\_frame.wait\_visibility()

        upcoming\_booking\_detail\_frame.grab\_set()

        upcoming\_booking\_detail\_frame.mainloop()

    @staticmethod

    def user\_cancel\_booking(

        user\_controller,

        upcoming\_booking\_detail\_frame,

        data,

    ):

        booking\_id = data[0]

        response = messagebox.askquestion(

            "Cancel Booking", "DO you really want to cancel this booking ?"

        )

*if* response == "yes":

            user\_booking\_cancel\_control = user\_controller()

            user\_booking\_cancel\_control.cancel\_booking(booking\_id)

            upcoming\_booking\_detail\_frame.destroy()

**Package : user\_model.py**

code :

import re

from pydantic import BaseModel, root\_validator, validator

from helper.exceptions import CustomException

name\_reg = re.compile("^[A-Z][a-z]+$")  *# firstname, lastname regex*

contact\_reg = re.compile("^[0-9]{10}$")  *# contact regex*

address\_reg = re.compile(".+")  *# address not null regex*

email\_reg = re.compile("[a-z0-9]+@[a-z]+\.[a-z]{2,3}")  *# email address regex*

username\_reg = re.compile("[a-z][a-z0-9]\*([.\_-][a-z0-9]+){0,3}$")  *# username regex*

class UserModel(BaseModel):

    firstname: str = ""

    lastname: str = ""

    contact: str = ""

    address: str = ""

    email: str = ""

    username: str = ""

    password: str = ""

    confirm\_password: str = ""

    user\_id: str = ""

    message: str = ""

    profile: str = ""

    new\_password: str = ""

    @validator("firstname", "lastname")

    def validate\_first\_name(cls, value):

*if* not value:

*raise* CustomException("name cannot be empty")

*if* not name\_reg.match(value):

*raise* CustomException("invalid format for name")

*return* value

    @validator("contact")

    def validate\_contact(cls, value):

*if* not value:

*raise* CustomException("contact cannot be empty")

*if* not contact\_reg.match(value):

*raise* CustomException("invalid format for contact")

*return* value

    @validator("address")

    def validate\_address(cls, value):

*if* not value:

*raise* CustomException("address cannot be empty")

*if* not address\_reg.match(value):

*raise* CustomException("invalid format for address")

*return* value

    @validator("email")

    def validate\_email(cls, value):

*if* not value:

*raise* CustomException("email cannot be empty")

*if* not email\_reg.match(value):

*raise* CustomException("invalid format for email")

*return* value

    @validator("username")

    def validate\_username(cls, value):

*if* not value:

*raise* CustomException("username cannot be empty")

*if* not username\_reg.match(value):

*raise* CustomException("invalid format for username")

*return* value

    @root\_validator

    def validate\_password(cls, value):

        password = value.get("password")

        confirm\_password = value.get("confirm\_password")

*if* not password:

*raise* CustomException("password cannot be empty")

*if* password != confirm\_password:

*raise* CustomException("The two passwords did not match.")

*return* value

**Package : user\_controllers.py**

code :

from curses import use\_default\_colors

from tkinter import messagebox

from helper.turbo\_db import Turbo\_db

import psycopg2

from .dashboard\_page import DashboardPage

class UserController:

    def \_\_init\_\_(self) -> None:

        self.\_connection = Turbo\_db.turbo\_connection()

    def registration\_control(self, user, signup\_frame):

*if* self.authenticate(user):

            self.registration\_sucess(signup\_frame)

    def authenticate(self, user):

*try*:

            cursor = self.\_connection.cursor()

            statement = """CREATE TABLE IF NOT EXISTS users(

                userID   SERIAL PRIMARY KEY,

                firstname   VARCHAR(50) NOT NULL,

                lastname   VARCHAR(50) NOT NULL,

                contact   VARCHAR(20) NOT NULL,

                address   VARCHAR(50) NOT NULL,

                email   VARCHAR(50) NOT NULL UNIQUE,

                username   VARCHAR(20) NOT NULL UNIQUE,

                password   VARCHAR(20) NOT NULL,

                profile BYTEA

            );"""

            defult\_image = open(

                "/home/vivu/Class Stuff V2/ISD\_Files/TurboTB\_System/img/profile.png",

                "rb",

            ).read()

            data\_insert = """INSERT INTO users(firstname, lastname, contact,

            address, email, username, password, profile) VALUES (%s, %s, %s, %s, %s, %s, %s, %s);

            """

            data\_values = (

                user.firstname,

                user.lastname,

                user.contact,

                user.address,

                user.email,

                user.username,

                user.password,

                psycopg2.Binary(defult\_image),

            )

            cursor.execute(statement)

            cursor.execute(data\_insert, data\_values)

            self.\_connection.commit()

*return* True

*except* Exception *as* error:

*if* "username" in str(error):

                messagebox.showerror(

                    "Invalid Data",

                    "username already exist!!",

                )

*elif* "email" in str(error):

                messagebox.showerror(

                    "Invalid Data",

                    "Already have account linked \

                    to this account.",

                )

*else*:

                print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def registration\_sucess(self, signup\_frame):

        messagebox.showinfo(

            "Account Created",

            "Congatulation, Your Accunt Has Been Created",

        )

        signup\_frame.destroy()

    def login\_control(self, user, root):

*if* self.login\_authenticate(user):

            self.login\_sucess(root)

*else*:

            messagebox.showerror("Invalid Data", "username or password not matched")

    def login\_authenticate(self, user):

        cursor = self.\_connection.cursor()

        statement = "SELECT \* FROM users WHERE username=%s AND password = %s;"

        data = (user.username, user.password)

        cursor.execute(statement, data)

        self.record = cursor.fetchall()

*if* self.record:

*for* data *in* self.record:

                self.fetch\_user\_id = data[0]

                self.fetched\_firstname = data[1].rstrip()

                self.fetched\_lastname = data[2].rstrip()

                self.fetch\_username = data[6]

                self.fetch\_password = data[7].rstrip()

                self.fetched\_profile = data[8]

*return* True

*return* False

    def login\_sucess(self, root):

        DashboardPage(

            root,

            self.record,

            UserController,

        ),

    def delete\_account(

        self,

        profile\_frame,

        dashboard\_frame,

        edit\_bio\_frame,

        record,

    ):

*for* data *in* record:

            fetched\_username = data[6]

*try*:

            cursor = self.\_connection.cursor()

            statement = """DELETE FROM users WHERE username = %s;"""

            data\_values = (fetched\_username,)

            cursor.execute(statement, data\_values)

            self.\_connection.commit()

            messagebox.showinfo(

                "Sucess!!",

                "Account Sucessfully deleted !",

            )

            edit\_bio\_frame.destroy()

            profile\_frame.destroy()

            dashboard\_frame.destroy()

*except* Exception *as* e:

            print(e)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def change\_profile(

        self,

        user,

        record,

        profile\_frame,

        dashboard\_frame,

    ):

*for* data *in* record:

            fetched\_firstname = data[6]

*try*:

            image\_path = user.profile

            new\_profile = open(image\_path, "rb").read()

            cursor = self.\_connection.cursor()

            statement = """UPDATE users

            SET profile = %s WHERE username = %s;"""

            dataValues = (

                psycopg2.Binary(new\_profile),

                fetched\_firstname,

            )

            cursor.execute(statement, dataValues)

            self.\_connection.commit()

            messagebox.showinfo(

                "Sucess!!",

                "Profile Picture Changed !",

            )

*# profile\_frame.destroy()*

*# dashboard\_frame.destroy()*

*except* Exception *as* e:

            print(e)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def update\_user\_bio(

        self,

        user,

        record,

        edit\_bio\_frame,

        profile\_frame,

        dashboard\_frame,

    ):

*for* data *in* record:

            fetched\_username = data[6]

*try*:

            cursor = self.\_connection.cursor()

            statement = """UPDATE users

            SET contact = COALESCE(NULLIF(%s,''),contact),

            address = COALESCE(NULLIF(%s,''),address),

            username = COALESCE(NULLIF(%s,''),username),

            password = COALESCE(NULLIF(%s,''),password)

            WHERE username = %s;"""

            dataValues = (

                user.contact,

                user.address,

                user.username,

                user.new\_password,

                fetched\_username,

            )

            cursor.execute(statement, dataValues)

            self.\_connection.commit()

            messagebox.showinfo(

                "Sucess!!",

                "Updated !",

            )

            edit\_bio\_frame.destroy()

*except* Exception *as* error:

*if* "username" in str(error):

                messagebox.showerror(

                    "Invalid Data",

                    "username already exist!!",

                    parent=edit\_bio\_frame,

                )

*else*:

                print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def upcoming\_trip\_detail\_fetcher(self, user\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM booking as b LEFT JOIN drivers as d on b.driver\_id = d.driverid LEFT JOIN taxi as t on d.taxi\_number = t.taxi\_number WHERE b.booking\_status = 'Pending' AND user\_id = %s OR b.booking\_status = 'Accepted' AND user\_id = %s ;"

            uid = str(user\_id)

            data = (uid, uid)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def user\_profile\_booking\_data\_fetcher(self, selected\_booking\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM booking as b LEFT JOIN drivers as d on b.driver\_id = d.driverid LEFT JOIN taxi as t on d.taxi\_number = t.taxi\_number WHERE b.booking\_id = %s;"

            bid = str(selected\_booking\_id)

            data = bid

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def cancel\_booking(self, booking\_id):

*try*:

            statement = "UPDATE booking SET booking\_status = 'Canceled' WHERE booking\_id = %s AND booking\_status = 'Pending';"

            b\_id = str(booking\_id)

            cursor = self.\_connection.cursor()

            data = b\_id

            cursor.execute(statement, data)

            self.\_connection.commit()

*except* Exception *as* error:

            print(error)

    def history\_trip\_detail\_fetcher(self, user\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM booking as b LEFT JOIN drivers as d on b.driver\_id = d.driverid LEFT JOIN taxi as t on d.taxi\_number = t.taxi\_number WHERE b.booking\_status = 'Completed' AND user\_id = %s OR b.booking\_status = 'Canceled' AND user\_id = %s ;"

            uid = str(user\_id)

            data = (uid, uid)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

* **Module : bookings**

**Package : booking\_form.py**

code :

from datetime import date, timedelta

from helper.exceptions import CustomException

from .booking\_model import BookingModel

from .booking\_controllers import BookingController

import math

from tkcalendar import DateEntry

from tkinter import StringVar, messagebox

from tkintermapview import TkinterMapView, convert\_coordinates\_to\_address

import haversine as hs

import tkinter as tk

class BookingPage:

    def \_\_init\_\_(self, root, dashboard\_frame, record) -> None:

        self.root = root

        self.dashboard\_frame = dashboard\_frame

        self.record = record

        self.create\_booking\_form\_frame()

    def create\_booking\_form\_frame(self):

        self.booking\_form\_frame = tk.Frame(self.root)

        self.booking\_form\_frame.configure(background="#FFFFFF")

        self.booking\_form\_frame.place(

            relx=-0.010,

            rely=0,

            height=768,

            width=1366,

        )

        self.build\_booking\_form\_frame(

            dashboard\_frame=self.dashboard\_frame,

            record=self.record,

            booking\_form\_frame=self.booking\_form\_frame,

        )

    pickup\_location\_coordinates = None

    destination\_coordinates = None

    button\_name = None

    @staticmethod

    def build\_booking\_form\_frame(

        dashboard\_frame,

        record,

        booking\_form\_frame,

    ):

        title\_label = tk.Label(booking\_form\_frame)

        title\_label.place(relx=0.052, rely=0.042, height=59, width=143)

        title\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            text="""Book Now""",

        )

        message\_label = tk.Message(booking\_form\_frame)

        message\_label.place(relx=0.052, rely=0.122, relheight=0.067, relwidth=0.671)

        message\_label.configure(

            anchor="w",

            background="#FFFFFF",

            foreground="#637381",

            padx="1",

            pady="1",

            text="""some information""",

            width=907,

        )

*for* data *in* record:

            fetched\_firstname = data[1].rstrip()

            fetched\_lastname = data[2].rstrip()

            fetched\_user\_id = data[0]

        fullname = tk.Label(booking\_form\_frame)

        fullname.place(relx=0.044, rely=0.312, height=31, width=93)

        fullname.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Full Name :""",

        )

        firstname\_entry = tk.Entry(booking\_form\_frame)

        firstname\_entry.place(relx=0.126, rely=0.312, height=33, relwidth=0.173)

        firstname\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        firstname\_entry.insert(0, fetched\_firstname)

        lastname\_entry = tk.Entry(booking\_form\_frame)

        lastname\_entry.place(relx=0.318, rely=0.312, height=33, relwidth=0.173)

        lastname\_entry.configure(

            background="#EFF0F2", selectbackground="#c4c4c4", font="TkFixedFont"

        )

        lastname\_entry.insert(0, fetched\_lastname)

        firstname = tk.Label(booking\_form\_frame)

        firstname.place(relx=0.126, rely=0.366, height=20, width=93)

        firstname.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""First Name""",

        )

        lastname = tk.Label(booking\_form\_frame)

        lastname.place(relx=0.318, rely=0.366, height=20, width=93)

        lastname.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""Last Name""",

        )

        no\_of\_passenger = tk.Label(booking\_form\_frame)

        no\_of\_passenger.place(relx=0.044, rely=0.421, height=31, width=205)

        no\_of\_passenger.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Number Of Passenger :""",

        )

        no\_of\_passenger\_entry = tk.Spinbox(

            booking\_form\_frame, from\_=1.0, to=4.0, state="readonly"

        )

        no\_of\_passenger\_entry.place(

            relx=0.192, rely=0.421, relheight=0.038, relwidth=0.079

        )

        no\_of\_passenger\_entry.configure(

            background="#EFF0F2",

            font="TkDefaultFont",

            highlightbackground="black",

            selectbackground="#c4c4c4",

        )

        no\_of\_taxi\_label = tk.Label(booking\_form\_frame)

        no\_of\_taxi\_label.place(relx=0.044, rely=0.488, height=31, width=175)

        no\_of\_taxi\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Number Of Taxi :""",

        )

        no\_of\_taxi\_entry = tk.Spinbox(

            booking\_form\_frame, from\_=1.0, to=2.0, state="disable"

        )

        no\_of\_taxi\_entry.place(relx=0.192, rely=0.488, relheight=0.038, relwidth=0.079)

        no\_of\_taxi\_entry.configure(

            activebackground="#f9f9f9",

            background="#EFF0F2",

            font="TkDefaultFont",

            highlightbackground="black",

            selectbackground="#c4c4c4",

        )

        taxi\_message = tk.Message(booking\_form\_frame)

        taxi\_message.place(relx=0.281, rely=0.501, relheight=0.028, relwidth=0.152)

        taxi\_message.configure(

            anchor="w",

            background="#FFFFFF",

            foreground="#637381",

            padx="1",

            pady="1",

            text="""a taxi can carry only upto 4 people""",

            width=205,

        )

        pickup\_date = tk.Label(booking\_form\_frame)

        pickup\_date.place(relx=0.044, rely=0.556, height=30, width=134)

        pickup\_date.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Pick Up Date :""",

        )

        today\_date = date.today()

        last\_date = date.today() + timedelta(days=5)

        date\_entry = DateEntry(

            booking\_form\_frame,

            selectmode="day",

            mindate=today\_date,

            maxdate=last\_date,

            state="readonly",

            date\_pattern="yyyy/mm/dd",

        )

        date\_entry.place(relx=0.185, rely=0.556, height=33, relwidth=0.092)

        date\_label = tk.Label(booking\_form\_frame)

        date\_label.place(relx=0.185, rely=0.612, height=20, width=93)

        date\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""MM/DD/YY""",

        )

        hour\_entry = tk.Spinbox(booking\_form\_frame, from\_=0.0, to=23.0)

        hour\_entry.place(relx=0.285, rely=0.556, relheight=0.038, relwidth=0.079)

        hour\_entry.configure(

            activebackground="#f9f9f9",

            background="#EFF0F2",

            font="TkDefaultFont",

            highlightbackground="black",

            selectbackground="#c4c4c4",

        )

        minute\_entry = tk.Spinbox(booking\_form\_frame, from\_=0.0, to=55.0, increment=5.0)

        minute\_entry.place(relx=0.370, rely=0.556, relheight=0.038, relwidth=0.079)

        minute\_entry.configure(

            activebackground="#f9f9f9",

            background="#EFF0F2",

            font="TkDefaultFont",

            highlightbackground="black",

            selectbackground="#c4c4c4",

        )

        time\_label = tk.Label(booking\_form\_frame)

        time\_label.place(relx=0.320, rely=0.612, height=20, width=93)

        time\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""HH(24hrs) : MM""",

        )

        pickup\_address = tk.Label(booking\_form\_frame)

        pickup\_address.place(relx=0.044, rely=0.651, height=31, width=134)

        pickup\_address.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Pick Up Address :""",

        )

        pickup\_address\_entry = tk.Entry(booking\_form\_frame, state="readonly")

        pickup\_address\_entry.place(relx=0.185, rely=0.651, height=33, relwidth=0.265)

        pickup\_address\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        pickup\_address\_button = tk.Button(booking\_form\_frame)

        pickup\_address\_button.place(

            relx=0.47,

            rely=0.651,

            height=33,

            relwidth=0.122,

        )

        pickup\_address\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Select Address""",

        )

        pickup\_address\_button.bind(

            "<ButtonRelease-1>",

            lambda event: BookingPage.create\_location\_picker(

                event,

                booking\_form\_frame,

                pickup\_address\_entry,

            ),

        )

        pickup\_address\_button.bind("<Button-1>", BookingPage.button\_name\_finder)

        destination = tk.Label(booking\_form\_frame)

        destination.place(relx=0.044, rely=0.746, height=31, width=134)

        destination.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Destination :""",

        )

        destination\_entry = tk.Entry(booking\_form\_frame, state="readonly")

        destination\_entry.place(relx=0.185, rely=0.746, height=33, relwidth=0.265)

        destination\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        destination\_button = tk.Button(booking\_form\_frame)

        destination\_button.place(relx=0.47, rely=0.746, height=33, relwidth=0.122)

        destination\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Select Address""",

        )

        destination\_button.bind(

            "<ButtonRelease-1>",

            lambda event: BookingPage.create\_location\_picker(

                event,

                booking\_form\_frame,

                destination\_entry,

            ),

        )

        destination\_button.bind("<Button-1>", BookingPage.button\_name\_finder\_2)

        total\_cost = tk.Label(booking\_form\_frame)

        total\_cost.place(relx=0.044, rely=0.855, height=40, width=144)

        total\_cost.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Total Charge :""",

        )

        cost = tk.Label(booking\_form\_frame)

        cost.place(relx=0.155, rely=0.855, height=40, width=143)

        cost.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""XXXXXX""",

        )

        view\_total\_cost\_button = tk.Button(booking\_form\_frame)

        view\_total\_cost\_button.place(

            relx=0.260,

            rely=0.855,

            height=33,

            relwidth=0.122,

        )

        view\_total\_cost\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""View Cost""",

        )

        view\_total\_cost\_button.bind(

            "<Button-1>",

            lambda event: BookingPage.show\_cost(

                event,

                no\_of\_taxi\_entry,

                cost,

            ),

        )

        view\_total\_cost\_button.bind(

            "<ButtonRelease-1>",

            lambda event: BookingPage.hide\_cost(

                event,

                cost,

            ),

        )

        book\_now\_button = tk.Button(booking\_form\_frame)

        book\_now\_button.place(relx=0.820, rely=0.840, height=53, width=141)

        book\_now\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 16",

            foreground="#FFFFFF",

            text="""Book Now""",

            command=lambda: BookingPage.user\_booking\_register(

                booking\_form\_frame,

                record,

                firstname\_entry,

                lastname\_entry,

                no\_of\_passenger\_entry,

                no\_of\_taxi\_entry,

                date\_entry,

                hour\_entry,

                minute\_entry,

                pickup\_address\_entry,

                destination\_entry,

                rb\_valv,

                card\_number\_entry,

                exp\_entry,

                cvv\_entry,

                cost,

            ),

        )

        payment\_method = tk.LabelFrame(booking\_form\_frame)

        payment\_method.place(relx=0.570, rely=0.285, relheight=0.21, relwidth=0.34)

        payment\_method.configure(

            relief="groove", background="#FFFFFF", text="""Payment Method"""

        )

        card\_number\_entry = tk.Entry(payment\_method)

        card\_number\_entry.place(

            relx=0.047, rely=0.583, height=33, relwidth=0.508, bordermode="ignore"

        )

        card\_number\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        exp\_entry = tk.Entry(payment\_method)

        exp\_entry.place(

            relx=0.611, rely=0.583, height=33, relwidth=0.142, bordermode="ignore"

        )

        exp\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        cvv\_entry = tk.Entry(payment\_method)

        cvv\_entry.place(

            relx=0.806, rely=0.583, height=33, relwidth=0.142, bordermode="ignore"

        )

        cvv\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        card\_number = tk.Label(payment\_method)

        card\_number.place(

            relx=0.043, rely=0.84, height=20, width=93, bordermode="ignore"

        )

        card\_number.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""Card Number""",

        )

        exp\_date = tk.Label(payment\_method)

        exp\_date.place(relx=0.611, rely=0.84, height=20, width=62, bordermode="ignore")

        exp\_date.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""Exp Date""",

        )

        cvv = tk.Label(payment\_method)

        cvv.place(relx=0.806, rely=0.84, height=20, width=63, bordermode="ignore")

        cvv.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            foreground="#637381",

            text="""CVV""",

        )

        rb\_valv = StringVar()

        credit\_card = tk.Radiobutton(

            payment\_method,

            value="Credit Card",

            variable=rb\_valv,

            command=lambda: BookingPage.enable\_card\_details(

                card\_number\_entry,

                exp\_entry,

                cvv\_entry,

            ),

        )

        credit\_card.place(relx=0.043, rely=0.256, relheight=0.212, relwidth=0.267)

        credit\_card.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            justify="left",

            selectcolor="#d9d9d9",

            text="""Credit Card""",

        )

        cashInHand = tk.Radiobutton(

            payment\_method,

            value="Cash In Hand",

            variable=rb\_valv,

        )

        cashInHand.place(relx=0.350, rely=0.256, relheight=0.212, relwidth=0.267)

        cashInHand.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            justify="left",

            selectcolor="#d9d9d9",

            text="""Cash In Hand""",

            command=lambda: BookingPage.disable\_card\_details(

                card\_number\_entry,

                exp\_entry,

                cvv\_entry,

            ),

        )

        rb\_valv.set("Credit Card")

        back = tk.Button(booking\_form\_frame)

        back.place(x=1165, y=60, height=35, width=90)

        back.config(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: BookingPage.redirect\_to\_dashboard(

                booking\_form\_frame,

            ),

        )

        booking\_form\_frame.mainloop()

    @staticmethod

    def redirect\_to\_dashboard(booking\_form\_frame):

        booking\_form\_frame.destroy()

    @staticmethod

    def disable\_card\_details(

        card\_number\_entry,

        exp\_entry,

        cvv\_entry,

    ):

        card\_number\_entry.delete(0, "end")

        card\_number\_entry.config(state="disabled")

        exp\_entry.delete(0, "end")

        exp\_entry.config(state="disabled")

        cvv\_entry.delete(0, "end")

        cvv\_entry.config(state="disabled")

        card\_number\_entry.update()

        exp\_entry.update()

        cvv\_entry.update()

    @staticmethod

    def enable\_card\_details(card\_number\_entry, exp\_entry, cvv\_entry):

        card\_number\_entry.config(state="normal")

        exp\_entry.config(state="normal")

        cvv\_entry.config(state="normal")

        card\_number\_entry.update()

        exp\_entry.update()

        cvv\_entry.update()

    @staticmethod

    def create\_location\_picker(

        event,

        booking\_form\_frame,

        location\_entry,

    ):

        location\_picker\_frame = tk.Toplevel(booking\_form\_frame)

        location\_picker\_frame.title("Location Picker")

        location\_picker\_frame.resizable(0, 0)

        location\_picker\_frame.configure(background="#FFFFFF")

        location\_picker\_frame.geometry("580x580")

        BookingPage.build\_location\_picker\_frame(

            location\_picker\_frame,

            location\_entry,

        )

    @staticmethod

    def build\_location\_picker\_frame(

        location\_picker\_frame,

        location\_entry,

    ):

        google\_map\_view = TkinterMapView(

            location\_picker\_frame,

            width=580,

            height=480,

        )

        google\_map\_view.set\_tile\_server(

            "https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",

            max\_zoom=22,

        )

        google\_map\_view.place(relx=0, rely=0.07)

        google\_map\_view.set\_address("kathmandu, nepal")

        google\_map\_view.set\_zoom(13)

        selected\_address = tk.Label(location\_picker\_frame)

        selected\_address.place(relx=0.0, rely=0.010, height=33, relwidth=0.999)

        selected\_address.config(background="#FFFFFF")

        google\_map\_view.add\_left\_click\_map\_command(

            lambda coordinates: BookingPage.location\_marker(

                coordinates,

                selected\_address,

            ),

        )

        search\_entry = tk.Entry(location\_picker\_frame)

        search\_entry.place(relx=0.060, rely=0.920, height=33, relwidth=0.300)

        search\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        search\_button = tk.Button(location\_picker\_frame)

        search\_button.place(relx=0.400, rely=0.920, height=33, relwidth=0.122)

        search\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Search""",

            command=lambda: BookingPage.search\_location(

                google\_map\_view,

                search\_entry,

            ),

        )

        select\_location\_button = tk.Button(location\_picker\_frame)

        select\_location\_button.place(

            relx=0.550,

            rely=0.920,

            height=33,

            relwidth=0.290,

        )

        select\_location\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 12",

            text="""Select Location""",

            command=lambda: BookingPage.address\_selected\_display(

                location\_entry,

                location\_picker\_frame,

            ),

        )

        location\_picker\_frame.grab\_set()

        location\_picker\_frame.mainloop()

    @staticmethod

    def search\_location(google\_map\_view, search\_entry):

        google\_map\_view.set\_address(search\_entry.get())

        google\_map\_view.set\_zoom(15)

    clicked\_address = None

    @staticmethod

    def location\_marker(

        coordinates,

        selected\_address,

    ):

*if* BookingPage.button\_name == "pickup\_button":

            BookingPage.pickup\_location\_coordinates = coordinates

*else*:

            BookingPage.destination\_coordinates = coordinates

        clicked\_coordinates\_1 = coordinates[0]

        clicked\_coordinates\_2 = coordinates[1]

        BookingPage.clicked\_address = convert\_coordinates\_to\_address(

            clicked\_coordinates\_1,

            clicked\_coordinates\_2,

        )

        selected\_address.config(

            text=(

                BookingPage.clicked\_address.street,

                BookingPage.clicked\_address.postal,

                BookingPage.clicked\_address.city,

                BookingPage.clicked\_address.country,

            ),

            font="-family {Noto Sans} -size 14",

        )

    @staticmethod

    def address\_selected\_display(

        location\_entry,

        location\_picker\_frame,

    ):

        location\_entry.config(state="normal")

        location\_entry.delete(0, "end")

        location\_entry.insert(

            0,

            (

                BookingPage.clicked\_address.street,

                BookingPage.clicked\_address.postal,

                BookingPage.clicked\_address.city,

                BookingPage.clicked\_address.country,

            ),

        )

        location\_entry.config(state="readonly")

        location\_picker\_frame.destroy()

    final\_cost = ""

    @staticmethod

    def calculate\_total\_cost(no\_of\_taxi\_entry, cost):

*try*:

            total\_cost = (

                math.ceil(

                    hs.haversine(

                        BookingPage.pickup\_location\_coordinates,

                        BookingPage.destination\_coordinates,

                    )

                )

                \* 59.25

            )

            no\_of\_taxi = int(no\_of\_taxi\_entry.get())

            showCost = total\_cost \* no\_of\_taxi

            BookingPage.final\_cost = str(round(showCost, 2))

            cost.config(text="NPR. " + BookingPage.final\_cost)

*except* Exception *as* error:

            cost.config(text="XXXXXX")

    @staticmethod

    def button\_name\_finder(event):

        BookingPage.button\_name = "pickup\_button"

    @staticmethod

    def button\_name\_finder\_2(event):

        BookingPage.button\_name = "destination\_button"

    @staticmethod

    def show\_cost(

        event,

        no\_of\_taxi\_entry,

        cost,

    ):

        BookingPage.calculate\_total\_cost(no\_of\_taxi\_entry, cost)

    @staticmethod

    def hide\_cost(

        event,

        cost,

    ):

        cost.config(text="XXXXXX")

    @staticmethod

    def user\_booking\_register(

        booking\_form\_frame,

        record,

        firstname\_entry,

        lastname\_entry,

        no\_of\_passenger\_entry,

        no\_of\_taxi\_entry,

        date\_entry,

        hour\_entry,

        minute\_entry,

        pickup\_address\_entry,

        destination\_entry,

        rb\_valv,

        card\_number\_entry,

        exp\_entry,

        cvv\_entry,

        cost,

    ):

*for* data *in* record:

            fetched\_user\_id = data[0]

        BookingPage.calculate\_total\_cost(no\_of\_taxi\_entry, cost)

*try*:

            user\_booking = BookingModel(

                user\_id=fetched\_user\_id,

                firstname=firstname\_entry.get(),

                lastname=lastname\_entry.get(),

                no\_of\_pass=no\_of\_passenger\_entry.get(),

                no\_of\_taxi=no\_of\_taxi\_entry.get(),

                pick\_up\_date=date\_entry.get(),

                pick\_up\_hrs=hour\_entry.get(),

                pick\_up\_min=minute\_entry.get(),

                pick\_up\_location=pickup\_address\_entry.get(),

                destination=destination\_entry.get(),

                payment\_method=rb\_valv.get(),

                card\_number=card\_number\_entry.get(),

                exp\_date=exp\_entry.get(),

                cvv=cvv\_entry.get(),

                total\_cost=BookingPage.final\_cost,

                pickup\_coordinates=str(BookingPage.pickup\_location\_coordinates),

                destination\_coordinates=str(BookingPage.destination\_coordinates),

            )

            booking\_control = BookingController()

            booking\_control.booking\_control(user\_booking, booking\_form\_frame)

*except* CustomException *as* e:

            messagebox.showerror("Invalid Data", e)

**Package : booking\_model.py**

code :

from ast import Return

from datetime import datetime

import re

from pydantic import BaseModel, ValidationError, root\_validator, validator

from helper.exceptions import CustomException

name\_reg = re.compile("^[A-Z][a-z]+$")  *# firstname, lastname regex*

class BookingModel(BaseModel):

    booking\_id: str = ""

    user\_id: str = ""

    firstname: str = ""

    lastname: str = ""

    no\_of\_pass: str = ""

    no\_of\_taxi: str = ""

    pick\_up\_date: str = ""

    pick\_up\_hrs: int = 0

    pick\_up\_min: int = 0

    pick\_up\_location: str = ""

    destination: str = ""

    total\_cost: str = ""

    payment\_method: str = ""

    card\_number: str = ""

    exp\_date: str = ""

    cvv: str = ""

    pickup\_coordinates: str = ""

    destination\_coordinates: str = ""

    @validator("firstname", "lastname")

    def validate\_name(cls, value):

*if* not value:

*raise* CustomException("name cannot be empty")

*if* not name\_reg.match(value):

*raise* CustomException("invalid format for name")

*return* value

    @root\_validator

    def validate\_time(cls, value):

        pick\_up\_hrs = value.get("pick\_up\_hrs")

        pick\_up\_min = value.get("pick\_up\_min")

        pick\_up\_date = value.get("pick\_up\_date")

*if* not pick\_up\_hrs:

*raise* CustomException("time(Hour) cannot be empty")

*if* not pick\_up\_min:

*raise* CustomException("time(Minutes) cannot be empty")

*if* pick\_up\_hrs == str:

*raise* CustomException("time(Hour) invalid")

*if* pick\_up\_min == str:

*raise* CustomException("time(Minutes) invalid")

*if* pick\_up\_hrs >= 24:

*raise* CustomException("time(Hour) invalid")

*if* pick\_up\_min >= 60:

*raise* CustomException("time(Minutes) invalid")

*else*:

            trip\_date = datetime.strptime(pick\_up\_date, "%Y/%m/%d").date()

            trip\_time = str(pick\_up\_hrs) + ":" + str(pick\_up\_min) + ":00"

            time\_final = datetime.strptime(trip\_time, "%H:%M:%S").time()

            trip\_date\_and\_time = datetime.combine(trip\_date, time\_final)

            today = datetime.now().isoformat(" ", "seconds")

            today\_date = datetime.strptime(today, "%Y-%m-%d %H:%M:%S")

*if* today\_date > trip\_date\_and\_time:

*raise* CustomException("invalid time given")

*else*:

*return* value

    @root\_validator

    def validate\_location(cls, value):

        pick\_up\_location = value.get("pick\_up\_location")

        destination = value.get("destination")

*if* not pick\_up\_location:

*raise* CustomException("pickup location cannot be empty")

*if* not destination:

*raise* CustomException("destination location cannot be empty")

*return* value

    @root\_validator

    def validate\_card\_details(cls, value):

        payment\_method = value.get("payment\_method")

        credit\_number = value.get("card\_number")

        exp\_date = value.get("exp\_date")

        cvv = value.get("cvv")

*if* payment\_method == "Credit Card":

*if* credit\_number == "":

*raise* CustomException("credit number cannot be empty!!")

*if* exp\_date == "":

*raise* CustomException("exp date cannot be empty!!")

*if* cvv == "":

*raise* CustomException("cvv cannot be empty!!")

*return* value

*else*:

*return* value

from ast import Return

from datetime import datetime

import re

from pydantic import BaseModel, ValidationError, root\_validator, validator

from helper.exceptions import CustomException

name\_reg = re.compile("^[A-Z][a-z]+$")  *# firstname, lastname regex*

class BookingModel(BaseModel):

    booking\_id: str = ""

    user\_id: str = ""

    firstname: str = ""

    lastname: str = ""

    no\_of\_pass: str = ""

    no\_of\_taxi: str = ""

    pick\_up\_date: str = ""

    pick\_up\_hrs: int = 0

    pick\_up\_min: int = 0

    pick\_up\_location: str = ""

    destination: str = ""

    total\_cost: str = ""

    payment\_method: str = ""

    card\_number: str = ""

    exp\_date: str = ""

    cvv: str = ""

    pickup\_coordinates: str = ""

    destination\_coordinates: str = ""

    @validator("firstname", "lastname")

    def validate\_name(cls, value):

*if* not value:

*raise* CustomException("name cannot be empty")

*if* not name\_reg.match(value):

*raise* CustomException("invalid format for name")

*return* value

    @root\_validator

    def validate\_time(cls, value):

        pick\_up\_hrs = value.get("pick\_up\_hrs")

        pick\_up\_min = value.get("pick\_up\_min")

        pick\_up\_date = value.get("pick\_up\_date")

*if* not pick\_up\_hrs:

*raise* CustomException("time(Hour) cannot be empty")

*if* not pick\_up\_min:

*raise* CustomException("time(Minutes) cannot be empty")

*if* pick\_up\_hrs == str:

*raise* CustomException("time(Hour) invalid")

*if* pick\_up\_min == str:

*raise* CustomException("time(Minutes) invalid")

*if* pick\_up\_hrs >= 24:

*raise* CustomException("time(Hour) invalid")

*if* pick\_up\_min >= 60:

*raise* CustomException("time(Minutes) invalid")

*else*:

            trip\_date = datetime.strptime(pick\_up\_date, "%Y/%m/%d").date()

            trip\_time = str(pick\_up\_hrs) + ":" + str(pick\_up\_min) + ":00"

            time\_final = datetime.strptime(trip\_time, "%H:%M:%S").time()

            trip\_date\_and\_time = datetime.combine(trip\_date, time\_final)

            today = datetime.now().isoformat(" ", "seconds")

            today\_date = datetime.strptime(today, "%Y-%m-%d %H:%M:%S")

*if* today\_date > trip\_date\_and\_time:

*raise* CustomException("invalid time given")

*else*:

*return* value

    @root\_validator

    def validate\_location(cls, value):

        pick\_up\_location = value.get("pick\_up\_location")

        destination = value.get("destination")

*if* not pick\_up\_location:

*raise* CustomException("pickup location cannot be empty")

*if* not destination:

*raise* CustomException("destination location cannot be empty")

*return* value

    @root\_validator

    def validate\_card\_details(cls, value):

        payment\_method = value.get("payment\_method")

        credit\_number = value.get("card\_number")

        exp\_date = value.get("exp\_date")

        cvv = value.get("cvv")

*if* payment\_method == "Credit Card":

*if* credit\_number == "":

*raise* CustomException("credit number cannot be empty!!")

*if* exp\_date == "":

*raise* CustomException("exp date cannot be empty!!")

*if* cvv == "":

*raise* CustomException("cvv cannot be empty!!")

*return* value

*else*:

*return* value

**Package : booking\_control.py**

code :

from tkinter import messagebox

from helper.turbo\_db import Turbo\_db

from datetime import date, datetime

class BookingController:

    def \_\_init\_\_(self) -> None:

        self.\_connection = Turbo\_db.turbo\_connection()

    def booking\_control(self, user\_booking, booking\_form\_frame):

*if* self.authenticate\_booking(user\_booking):

            self.booking\_sucess(booking\_form\_frame)

    def authenticate\_booking(self, user\_booking):

        booking\_created\_at\_date = date.today()

        now = datetime.now()

        booking\_created\_at\_time = now.strftime("%H:%M:%S")

*try*:

            cursor = self.\_connection.cursor()

            statement = """CREATE TABLE IF NOT EXISTS booking(

                booking\_id          SERIAL PRIMARY KEY,

                user\_id             INT NOT NULL,

                firstname           VARCHAR(50) NOT NULL,

                lastname            VARCHAR(50) NOT NULL,

                no\_of\_passenger     INT NOT NULL,

                no\_of\_taxi          INT NOT NULL,

                pickup\_date         DATE NOT NULL,

                pickup\_time\_hrs     INT NOT NULL,

                pickup\_time\_min     INT NOT NULL,

                pickup\_location     VARCHAR(100) NOT NULL,

                destination         VARCHAR(100) NOT NULL,

                total\_cost          VARCHAR(20) NOT NULL,

                payment\_method      VARCHAR(20) NOT NULL,

                card\_number         VARCHAR(50),

                card\_exp            VARCHAR(20),

                card\_cvv            VARCHAR(10),

                booking\_status      VARCHAR(20),

                created\_at\_date     DATE NOT NULL,

                created\_at\_time     VARCHAR(30),

                pickup\_coordinate   VARCHAR(100),

                destination\_coordinate  VARCHAR(100),

                driver\_id           INT

            );"""

            dataInsert = """INSERT INTO booking(user\_id, firstname, lastname, \

                no\_of\_passenger, no\_of\_taxi, pickup\_date, pickup\_time\_hrs, \

                pickup\_time\_min, pickup\_location, destination, total\_cost, \

                payment\_method, card\_number, card\_exp, card\_cvv, booking\_status, created\_at\_date, created\_at\_time, \

                pickup\_coordinate, destination\_coordinate) \

                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);"""

            dataValues = (

                user\_booking.user\_id,

                user\_booking.firstname,

                user\_booking.lastname,

                user\_booking.no\_of\_pass,

                user\_booking.no\_of\_taxi,

                user\_booking.pick\_up\_date,

                user\_booking.pick\_up\_hrs,

                user\_booking.pick\_up\_min,

                user\_booking.pick\_up\_location,

                user\_booking.destination,

                user\_booking.total\_cost,

                user\_booking.payment\_method,

                user\_booking.card\_number,

                user\_booking.exp\_date,

                user\_booking.cvv,

                "Pending",

                booking\_created\_at\_date,

                booking\_created\_at\_time,

                user\_booking.pickup\_coordinates,

                user\_booking.destination\_coordinates,

            )

            cursor.execute(statement)

            cursor.execute(dataInsert, dataValues)

            self.\_connection.commit()

*except* Exception *as* error:

            print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

*return* True

    def booking\_sucess(self, booking\_form\_frame):

        messagebox.showinfo(

            "Booking Completed",

            "Congatulation, Your Booking Request Has Been Sent",

        )

        booking\_form\_frame.destroy()

* **Module : admin**

**Package : admin\_model.py**

code :

from pydantic import BaseModel

class AdminModel(BaseModel):

    username: str = ""

    password: str = ""

**Package : admin\_login.py**

code :

import tkinter as tk

from admin.admin\_controllers import AdminController

from helper.constants import BG\_LOCATION

from .admin\_model import AdminModel

class LogInPage:

    def \_\_init\_\_(self, root):

        self.root = root

        self.create\_login\_frame()

    def create\_login\_frame(self):

        self.login\_frame = tk.Frame(self.root)

        self.login\_frame.configure(background="#FFFFFF")

        self.login\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_login\_frame(login\_frame=self.login\_frame, root=self.root)

    @staticmethod

    def build\_login\_frame(login\_frame, root):

        welcome = tk.Label(login\_frame)

        welcome.place(relx=0.650, rely=0.149, height=59, width=200)

        welcome.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 22 -weight bold",

            text="""Admin Log In""",

        )

        username\_label = tk.Label(login\_frame)

        username\_label.place(relx=0.585, rely=0.299, height=21, width=206)

        username\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Username""",

        )

        username\_entry = tk.Entry(login\_frame)

        username\_entry.place(relx=0.585, rely=0.353, height=33, relwidth=0.301)

        username\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        password\_label = tk.Label(login\_frame)

        password\_label.place(relx=0.585, rely=0.434, height=20, width=185)

        password\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Password""",

        )

        password\_entry = tk.Entry(login\_frame)

        password\_entry.place(relx=0.585, rely=0.488, height=33, relwidth=0.301)

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        login\_button = tk.Button(login\_frame)

        login\_button.place(relx=0.674, rely=0.611, height=43, width=201)

        login\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Log In""",

            command=lambda: LogInPage.admin\_login(

                username\_entry,

                password\_entry,

                login\_frame,

                root,

            ),

        )

        background\_label = tk.Label(login\_frame)

        background\_label.place(relx=-0.029, rely=0.0, height=739, width=700)

        background\_label.configure(anchor="w", compound="left")

        background\_image = tk.PhotoImage(file=BG\_LOCATION)

        background\_label.configure(image=background\_image)

        exit\_button = tk.Button(login\_frame)

        exit\_button.place(relx=0.100, rely=0.865, height=43, width=111)

        exit\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: LogInPage.exit\_application(login\_frame),

        )

        show\_pass\_button = tk.Button(login\_frame)

        show\_pass\_button.place(relx=0.900, rely=0.500, height=23, width=51)

        show\_pass\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""OO""",

        )

        show\_pass\_button.bind(

            "<Button-1>",

            lambda event: LogInPage.show\_pass(

                event,

                password\_entry,

            ),

        )

        show\_pass\_button.bind(

            "<ButtonRelease-1>",

            lambda event: LogInPage.hide\_pass(

                event,

                password\_entry,

            ),

        )

        login\_frame.mainloop()

    @staticmethod

    def show\_pass(

        event,

        password\_entry,

    ):

        password = password\_entry.get()

        password\_entry.config(show="", text=password)

    @staticmethod

    def hide\_pass(

        event,

        password\_entry,

    ):

        password\_entry.config(show="\*")

    def exit\_application(login\_frame):

        login\_frame.destroy()

    @staticmethod

    def admin\_login(

        username\_entry,

        password\_entry,

        login\_frame,

        root,

    ):

        admin = AdminModel(

            username=username\_entry.get(),

            password=password\_entry.get(),

        )

        user\_control = AdminController()

        user\_control.admin\_login\_control(admin, login\_frame, root)

**Package : admin\_controller.py**

code :

from tkinter import messagebox

from helper.turbo\_db import Turbo\_db

from .control\_panel import ControlPanelPage

class AdminController:

    def \_\_init\_\_(self) -> None:

        self.\_connection = Turbo\_db.turbo\_connection()

    def admin\_login\_control(self, admin, login\_frame, root):

*if* self.login\_authenticate(admin):

            self.login\_sucess(root, login\_frame)

*else*:

            messagebox.showerror(

                "Invalid Data",

                "username or password not matched",

            )

    def login\_authenticate(self, admin):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM admin WHERE username=%s AND password = %s;"

            data = (admin.username, admin.password)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*if* self.record:

*return* True

*return* False

*except* Exception *as* error:

            print(error)

    def login\_sucess(self, root, login\_page):

        messagebox.showinfo(title="Congratulation", message="Welcome ")

        login\_page.destroy()

        ControlPanelPage(root, login\_page, AdminController)

    def customer\_data\_fetcher(self):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT u.userid, b.booking\_id, b.created\_at\_date, CONCAT(u.firstname,' ',u.lastname) as fullname, u.contact, u.address, u.email, b.booking\_status from users as u JOIN booking as b on u.userid = b.user\_id where b.booking\_status = 'Pending' order by b.created\_at\_date, b.created\_at\_time;"

            cursor.execute(statement)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def customer\_search\_fetcher(self, search\_entry, selected\_sort\_by):

*if* selected\_sort\_by.get() == "Pending":

            statement = "SELECT u.userid, b.booking\_id, b.created\_at\_date, CONCAT(u.firstname,' ',u.lastname) as fullname, u.contact, u.address, u.email, b.booking\_status from users as u JOIN booking as b on u.userid = b.user\_id where concat(u.firstname , ' ' , u.lastname) like concat('%%',%s,'%%') and b.booking\_status = 'Pending' order by b.created\_at\_date, b.created\_at\_time;"

*else*:

*if* selected\_sort\_by.get() == "Accepted":

                statement = "SELECT u.userid, b.booking\_id, b.created\_at\_date, CONCAT(u.firstname,' ',u.lastname) as fullname, u.contact, u.address, u.email, b.booking\_status from users as u JOIN booking as b on u.userid = b.user\_id where concat(u.firstname , ' ' , u.lastname) like concat('%%',%s,'%%') and b.booking\_status = 'Accepted' order by b.created\_at\_date, b.created\_at\_time;"

*else*:

*if* selected\_sort\_by.get() == "Canceled":

                    statement = "SELECT u.userid, b.booking\_id, b.created\_at\_date, CONCAT(u.firstname,' ',u.lastname) as fullname, u.contact, u.address, u.email, b.booking\_status from users as u JOIN booking as b on u.userid = b.user\_id where concat(u.firstname , ' ' , u.lastname) like concat('%%',%s,'%%') and b.booking\_status = 'Canceled' order by b.created\_at\_date, b.created\_at\_time;"

*else*:

*if* selected\_sort\_by.get() == "All":

                        statement = "SELECT u.userid, b.booking\_id, b.created\_at\_date, CONCAT(u.firstname,' ',u.lastname) as fullname, u.contact, u.address, u.email, b.booking\_status from users as u JOIN booking as b on u.userid = b.user\_id where concat(u.firstname , ' ' , u.lastname) like concat('%%',%s,'%%') order by b.created\_at\_date, b.created\_at\_time;"

*try*:

            cursor = self.\_connection.cursor()

            data = (search\_entry.get(),)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def driver\_data\_fetcher(self):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM drivers;"

            cursor.execute(statement)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def driver\_search\_fetcher(self, search\_entry, selected\_sort\_by):

*if* selected\_sort\_by.get() == "Driver ID":

            statement = "SELECT \* FROM drivers WHERE fullname like concat('%%',%s,'%%') ORDER BY driverid;"

*else*:

*if* selected\_sort\_by.get() == "Full Name":

                statement = "SELECT \* FROM drivers WHERE fullname like concat('%%',%s,'%%') ORDER BY fullname;"

*else*:

*if* selected\_sort\_by.get() == "Available":

                    statement = "SELECT \* FROM drivers WHERE fullname like concat('%%',%s,'%%') and driver\_status = 'Available' ORDER BY driverid;"

*else*:

*if* selected\_sort\_by.get() == "Booked":

                        statement = "SELECT \* FROM drivers WHERE fullname like concat('%%',%s,'%%') and driver\_status = 'Booked' ORDER BY driverid;"

*else*:

*return*

*try*:

            cursor = self.\_connection.cursor()

            data = (search\_entry.get(),)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def booking\_details\_data\_fetcher(self, selected\_booking\_id):

*try*:

            statement = "SELECT \* FROM booking WHERE booking\_id = %s"

            cursor = self.\_connection.cursor()

            data = selected\_booking\_id

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def cancel\_booking(self, booking\_id):

*try*:

            statement = "UPDATE booking SET booking\_status = 'Canceled' WHERE booking\_id = %s AND booking\_status = 'Pending';"

            b\_id = str(booking\_id)

            cursor = self.\_connection.cursor()

            data = b\_id

            cursor.execute(statement, data)

            self.\_connection.commit()

*except* Exception *as* error:

            print(error)

    def available\_driver\_fetcher(self):

*try*:

            statement = "SELECT \* FROM drivers WHERE driver\_status = 'Available'"

            cursor = self.\_connection.cursor()

            cursor.execute(statement)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def booked\_driver\_fetcher(self):

*try*:

            statement = "SELECT \* FROM drivers WHERE driver\_status = 'Booked'"

            cursor = self.\_connection.cursor()

            cursor.execute(statement)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def assign\_controller(self, driver\_id, booking\_id):

*try*:

            booking\_statement = "Update booking set booking\_status = 'Accepted', driver\_id = %s WHERE booking\_id = %s ;"

            did = str(driver\_id)

            bid = str(booking\_id)

            booking\_data = (did, bid)

            driver\_statement = (

                "Update drivers set driver\_status = 'Booked' where driverid = %s;"

            )

            driver\_data = did

            cursor = self.\_connection.cursor()

            cursor.execute(booking\_statement, booking\_data)

            cursor.execute(driver\_statement, driver\_data)

            self.\_connection.commit()

*except* Exception *as* error:

            print(error)

**Package : control\_panel.py**

code :

from select import select

from sre\_parse import State

import tkinter as tk

from tkinter import W, Scrollbar, ttk

from tkinter import messagebox

from tkinter.messagebox import NO

from click import command

from taxi.taxi\_register import TaxiRegisterPage

from driver.driver\_register import DriverRegisterPage

from tkintermapview import TkinterMapView

class ControlPanelPage:

    def \_\_init\_\_(self, root, login\_frame, admin\_controller):

        self.root = root

        self.admin\_controller = admin\_controller

        self.login\_frame = login\_frame

        self.create\_control\_panel\_frame()

    def create\_control\_panel\_frame(self):

        self.control\_panel\_frame = tk.Frame(self.root)

        self.control\_panel\_frame.configure(background="#FFFFFF")

        self.control\_panel\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_control\_panel\_frame(

            self.root, self.control\_panel\_frame, self.admin\_controller

        )

    @staticmethod

    def build\_control\_panel\_frame(root, control\_panel\_frame, admin\_controller):

        title = tk.Label(control\_panel\_frame)

        title.place(relx=0.450, rely=0.025, height=71, width=224)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 22 -weight bold",

            text="""Control Panel""",

        )

        line\_seperator = ttk.Separator(control\_panel\_frame)

        line\_seperator.place(relx=0.130, rely=0.123, relwidth=0.800)

        taxi\_register\_button = tk.Button(control\_panel\_frame)

        taxi\_register\_button.place(relx=0.850, rely=0.130, height=33, width=101)

        taxi\_register\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Taxi Register""",

            command=lambda: TaxiRegisterPage(

                root,

                control\_panel\_frame,

            ),

        )

        driver\_register\_button = tk.Button(control\_panel\_frame)

        driver\_register\_button.place(relx=0.750, rely=0.130, height=33, width=110)

        driver\_register\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Driver Register""",

            command=lambda: DriverRegisterPage(

                root,

                control\_panel\_frame,

            ),

        )

        customer\_table\_button = tk.Button(control\_panel\_frame)

        customer\_table\_button.place(relx=0.130, rely=0.850, height=33, width=101)

        customer\_table\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Customer""",

            command=lambda: ControlPanelPage.customer\_tabel\_data\_fetcher(

                admin\_controller, control\_panel\_frame

            ),

        )

        driver\_table\_button = tk.Button(control\_panel\_frame)

        driver\_table\_button.place(relx=0.220, rely=0.850, height=33, width=101)

        driver\_table\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Driver""",

            command=lambda: ControlPanelPage.driver\_tabel\_data\_fetcher(

                admin\_controller,

                control\_panel\_frame,

            ),

        )

        ControlPanelPage.customer\_tabel\_data\_fetcher(

            admin\_controller, control\_panel\_frame

        )

        control\_panel\_frame.mainloop()

    @staticmethod

    def customer\_table\_frame\_build(

        record,

        control\_panel\_frame,

        admin\_controller,

    ):

        customer\_table\_frame = tk.Frame(control\_panel\_frame)

        customer\_table\_frame.place(

            relx=0.130,

            rely=0.200,

            height=470,

            width=1095,

        )

        customer\_table\_frame.config(

            background="#FFFFFF",

            borderwidth=2,

            relief="solid",

            highlightbackground="#000000",

        )

        table\_title = tk.Label(customer\_table\_frame)

        table\_title.place(relx=0.020, rely=0.010, height=60, width=200)

        table\_title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Customer Table""",

        )

        search\_entry = tk.Entry(customer\_table\_frame)

        search\_entry.place(relx=0.200, rely=0.050, height=25, relwidth=0.150)

        search\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        search\_button = tk.Button(customer\_table\_frame)

        search\_button.place(relx=0.360, rely=0.050, height=25, relwidth=0.069)

        search\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 10",

            text="""Search""",

            command=lambda: ControlPanelPage.customer\_tabel\_search\_fetcher(

                admin\_controller,

                search\_entry,

                selected\_sort\_by,

                customer\_table\_frame,

            ),

        )

        selected\_sort\_by = tk.StringVar()

        sort\_by\_combo = ttk.Combobox(

            customer\_table\_frame,

            textvariable=selected\_sort\_by,

            state="readonly",

            values=(

                "Pending",

                "Accepted",

                "Canceled",

                "All",

            ),

        )

        sort\_by\_combo.place(relx=0.460, rely=0.050, height=25, relwidth=0.150)

        sort\_by\_combo.current(0)

        sort\_by\_combo.bind(

            "<<ComboboxSelected>>",

            lambda event: ControlPanelPage.customer\_tabel\_search\_fetcher(

                admin\_controller,

                search\_entry,

                selected\_sort\_by,

                customer\_table\_frame,

            ),

        ),

        sort\_by\_label = tk.Label(

            customer\_table\_frame,

            text=": View",

            anchor=W,

            background="#FFFFFF",

            foreground="#4A4A4A",

        )

        sort\_by\_label.place(

            relx=0.620,

            rely=0.050,

            height=25,

            relwidth=0.060,

        )

        ControlPanelPage.customer\_table\_build(

            customer\_table\_frame,

            record,

            admin\_controller,

        )

    @staticmethod

    def driver\_table\_frame\_build(record, control\_panel\_frame, admin\_controller):

        driver\_table\_frame = tk.Frame(control\_panel\_frame)

        driver\_table\_frame.place(

            relx=0.130,

            rely=0.200,

            height=470,

            width=1095,

        )

        driver\_table\_frame.config(

            background="#FFFFFF",

            borderwidth=2,

            relief="solid",

            highlightbackground="#000000",

        )

        table\_title = tk.Label(driver\_table\_frame)

        table\_title.place(relx=0.020, rely=0.010, height=60, width=200)

        table\_title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Driver Table""",

        )

        search\_entry = tk.Entry(driver\_table\_frame)

        search\_entry.place(relx=0.200, rely=0.050, height=25, relwidth=0.150)

        search\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        search\_button = tk.Button(driver\_table\_frame)

        search\_button.place(relx=0.360, rely=0.050, height=25, relwidth=0.069)

        search\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 10",

            text="""Search""",

            command=lambda: ControlPanelPage.driver\_tabel\_search\_fetcher(

                admin\_controller,

                search\_entry,

                selected\_sort\_by,

                driver\_table\_frame,

            ),

        )

        selected\_sort\_by = tk.StringVar()

        sort\_by\_combo = ttk.Combobox(

            driver\_table\_frame,

            textvariable=selected\_sort\_by,

            state="readonly",

            values=("Driver ID", "Full Name", "Available", "Booked"),

        )

        sort\_by\_combo.place(relx=0.460, rely=0.050, height=25, relwidth=0.150)

        sort\_by\_combo.current(0)

        sort\_by\_combo.bind(

            "<<ComboboxSelected>>",

            lambda event: ControlPanelPage.driver\_tabel\_search\_fetcher(

                admin\_controller,

                search\_entry,

                selected\_sort\_by,

                driver\_table\_frame,

            ),

        ),

        sort\_by\_label = tk.Label(

            driver\_table\_frame,

            text=": sort by",

            anchor=W,

            background="#FFFFFF",

            foreground="#4A4A4A",

        )

        sort\_by\_label.place(

            relx=0.620,

            rely=0.050,

            height=25,

            relwidth=0.060,

        )

        ControlPanelPage.driver\_table\_build(driver\_table\_frame, record)

    @staticmethod

    def driver\_table\_build(dirver\_table\_frame, record):

*# Driver TABLE LOOK*

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

*# driver Table Scroll Bar*

        table\_scroll\_bar = Scrollbar(dirver\_table\_frame)

        table\_scroll\_bar.place(relx=0.980, rely=0.020, height=450, width=15)

*# driver Table Build*

        driver\_booking\_table = ttk.Treeview(

            dirver\_table\_frame,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        driver\_booking\_table.place(

            relx=0.020,

            rely=0.150,

            height=375,

            width=1025,

        )

*# driver a scroll bar*

        table\_scroll\_bar.config(

            command=driver\_booking\_table.yview,

        )

*# Define Column*

        driver\_booking\_table["columns"] = (

            "Driver ID",

            "Full Name",

            "License Number",

            "Contact",

            "Taxi Number",

            "Status",

        )

*# Format Our Columns*

        driver\_booking\_table.column("#0", width=0, stretch=NO)

        driver\_booking\_table.column("Driver ID", width=70, anchor=W)

        driver\_booking\_table.column("Full Name", width=140, anchor=W)

        driver\_booking\_table.column("License Number", width=120, anchor=W)

        driver\_booking\_table.column("Contact", width=120, anchor=W)

        driver\_booking\_table.column("Taxi Number", width=120, anchor=W)

        driver\_booking\_table.column("Status", width=120, anchor=W)

*# Create Heading*

        driver\_booking\_table.heading("#0", text="", anchor=W)

        driver\_booking\_table.heading("Driver ID", text="Driver ID", anchor=W)

        driver\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        driver\_booking\_table.heading("License Number", text="License Number", anchor=W)

        driver\_booking\_table.heading("Contact", text="Contact", anchor=W)

        driver\_booking\_table.heading("Taxi Number", text="Taxi Number", anchor=W)

        driver\_booking\_table.heading("Status", text="Status", anchor=W)

*for* data *in* record:

            driver\_booking\_table.insert(

                "",

                index="end",

                values=(

                    data[0],

                    data[1],

                    data[2],

                    data[3],

                    data[4],

                    data[5],

                ),

            )

        dirver\_table\_frame.mainloop()

    @staticmethod

    def customer\_table\_build(customer\_table\_frame, record, admin\_controller):

*# CUSTOMISING TABLE LOOK*

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

*# Customer Table Scroll Bar*

        table\_scroll\_bar = Scrollbar(customer\_table\_frame)

        table\_scroll\_bar.place(relx=0.980, rely=0.020, height=450, width=15)

*# Customer Table Build*

        customer\_booking\_table = ttk.Treeview(

            customer\_table\_frame,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        customer\_booking\_table.place(

            relx=0.020,

            rely=0.150,

            height=375,

            width=1025,

        )

        customer\_booking\_table.bind(

            "<Double-1>",

            lambda event: ControlPanelPage.assign\_taxi\_to\_user(

                event,

                customer\_booking\_table,

                admin\_controller,

            ),

        )

*# Configure a scroll bar*

        table\_scroll\_bar.config(

            command=customer\_booking\_table.yview,

        )

*# Define Column*

        customer\_booking\_table["columns"] = (

            "User ID",

            "Booking ID",

            "Created At",

            "Full Name",

            "Contact",

            "Address",

            "Email",

            "Booking Status",

        )

*# Format Our Columns*

        customer\_booking\_table.column("#0", width=0, stretch=NO)

        customer\_booking\_table.column("User ID", width=70, anchor=W)

        customer\_booking\_table.column("Booking ID", width=70, anchor=W)

        customer\_booking\_table.column("Created At", width=120, anchor=W)

        customer\_booking\_table.column("Full Name", width=140, anchor=W)

        customer\_booking\_table.column("Contact", width=70, anchor=W)

        customer\_booking\_table.column("Address", width=170, anchor=W)

        customer\_booking\_table.column("Email", width=180, anchor=W)

        customer\_booking\_table.column("Booking Status", width=100, anchor=W)

*# Create Heading*

        customer\_booking\_table.heading("#0", text="", anchor=W)

        customer\_booking\_table.heading("User ID", text="User ID", anchor=W)

        customer\_booking\_table.heading("Booking ID", text="Booking ID", anchor=W)

        customer\_booking\_table.heading("Created At", text="Created At", anchor=W)

        customer\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        customer\_booking\_table.heading("Contact", text="Contact", anchor=W)

        customer\_booking\_table.heading("Address", text="Address", anchor=W)

        customer\_booking\_table.heading("Email", text="Email", anchor=W)

        customer\_booking\_table.heading(

            "Booking Status", text="Booking Status", anchor=W

        )

*for* data *in* record:

            customer\_booking\_table.insert(

                parent="",

                index="end",

                text="",

                values=(

                    data[0],

                    data[1],

                    data[2],

                    data[3],

                    data[4],

                    data[5],

                    data[6],

                    data[7],

                ),

            )

        customer\_table\_frame.mainloop()

    @staticmethod

    def customer\_tabel\_data\_fetcher(admin\_controller, control\_panel\_frame):

        user\_booking = admin\_controller()

        required\_customer\_data = user\_booking.customer\_data\_fetcher()

        ControlPanelPage.customer\_table\_frame\_build(

            required\_customer\_data,

            control\_panel\_frame,

            admin\_controller,

        )

    @staticmethod

    def driver\_tabel\_data\_fetcher(admin\_controller, control\_panel\_frame):

        user\_booking = admin\_controller()

        required\_customer\_data = user\_booking.driver\_data\_fetcher()

        ControlPanelPage.driver\_table\_frame\_build(

            required\_customer\_data,

            control\_panel\_frame,

            admin\_controller,

        )

    @staticmethod

    def customer\_tabel\_search\_fetcher(

        admin\_controller,

        search\_entry,

        selected\_sort\_by,

        customer\_table\_frame,

    ):

        user\_booking = admin\_controller()

        required\_customer\_search = user\_booking.customer\_search\_fetcher(

            search\_entry,

            selected\_sort\_by,

        )

        ControlPanelPage.customer\_table\_build(

            customer\_table\_frame,

            required\_customer\_search,

            admin\_controller,

        )

    @staticmethod

    def driver\_tabel\_search\_fetcher(

        admin\_controller,

        search\_entry,

        selected\_sort\_by,

        driver\_table\_frame,

    ):

        user\_booking = admin\_controller()

        required\_customer\_search = user\_booking.driver\_search\_fetcher(

            search\_entry,

            selected\_sort\_by,

        )

        ControlPanelPage.driver\_table\_build(

            driver\_table\_frame,

            required\_customer\_search,

        )

    @staticmethod

    def assign\_taxi\_to\_user(event, customer\_booking\_table, admin\_controller):

        selected = customer\_booking\_table.focus()

        selected\_row = customer\_booking\_table.item(

            selected,

            "values",

        )

        selected\_booking\_id = selected\_row[1]

*# Booking Details Data Fetcher*

        booking\_data\_control = admin\_controller()

        fetched\_booking\_details = booking\_data\_control.booking\_details\_data\_fetcher(

            selected\_booking\_id,

        )

        data = fetched\_booking\_details[0]

        assign\_taxi\_frame = tk.Toplevel(customer\_booking\_table)

        assign\_taxi\_frame.title("Assign Taxi")

        assign\_taxi\_frame.geometry("682x641+193+31")

        assign\_taxi\_frame.resizable(0, 0)

        assign\_taxi\_frame.configure(background="#FFFFFF")

*# BUILDING BOOKING DETAIL PAGE*

        title = tk.Label(assign\_taxi\_frame)

        title.place(relx=0.360, rely=0.031, height=43, width=250)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            text="""Booking Details""",

        )

        fullname\_label = tk.Label(assign\_taxi\_frame)

        fullname\_label.place(relx=0.029, rely=0.156, height=34, width=122)

        fullname\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Full Name :""",

        )

        fullname\_data = tk.Label(assign\_taxi\_frame)

        fullname\_data.place(relx=0.22, rely=0.156, height=34, width=280)

        fullname\_data.configure(

            anchor="w",

            background="#FFFFFF",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[2] + " " + data[3],

        )

        map\_view\_frame = tk.Frame(assign\_taxi\_frame)

        map\_view\_frame.place(relx=0.640, rely=0.156, height=140, width=220)

        map\_view\_frame.configure(

            background="#FFFFFF",

            borderwidth="2",

            highlightthickness="2",

        )

        location\_detail\_map\_view = TkinterMapView(map\_view\_frame)

        location\_detail\_map\_view.set\_tile\_server(

            "https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",

            max\_zoom=60,

        )

        pickup\_cordinates = eval(data[19])

        destination\_coordinates = eval(data[20])

        location\_detail\_map\_view.set\_marker(

            pickup\_cordinates[0],

            pickup\_cordinates[1],

        )

        location\_detail\_map\_view.set\_marker(

            destination\_coordinates[0],

            destination\_coordinates[1],

        )

        location\_detail\_map\_view.set\_position(

            pickup\_cordinates[0],

            pickup\_cordinates[1],

        )

        location\_detail\_map\_view.set\_path(

            position\_list=[pickup\_cordinates, destination\_coordinates]

        )

        location\_detail\_map\_view.pack()

        passenger\_no\_label = tk.Label(assign\_taxi\_frame)

        passenger\_no\_label.place(relx=0.029, rely=0.234, height=34, width=175)

        passenger\_no\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""No of Passenger :""",

        )

        passenger\_no\_data = tk.Label(assign\_taxi\_frame)

        passenger\_no\_data.place(relx=0.293, rely=0.234, height=34, width=48)

        passenger\_no\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[4],

        )

        taxi\_number = tk.Label(assign\_taxi\_frame)

        taxi\_number.place(relx=0.029, rely=0.312, height=35, width=111)

        taxi\_number.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""No of Taxi :""",

        )

        taxi\_number\_data = tk.Label(assign\_taxi\_frame)

        taxi\_number\_data.place(relx=0.22, rely=0.312, height=35, width=48)

        taxi\_number\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[5],

        )

        pick\_up\_date\_lable = tk.Label(assign\_taxi\_frame)

        pick\_up\_date\_lable.place(relx=0.029, rely=0.39, height=34, width=133)

        pick\_up\_date\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Pick Up Date :""",

        )

        pick\_up\_date\_data = tk.Label(assign\_taxi\_frame)

        pick\_up\_date\_data.place(relx=0.249, rely=0.39, height=35, width=163)

        pick\_up\_date\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[6],

        )

        pick\_up\_address\_label = tk.Label(assign\_taxi\_frame)

        pick\_up\_address\_label.place(relx=0.029, rely=0.468, height=35, width=175)

        pick\_up\_address\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Pick Up Address :""",

        )

        time\_label = tk.Label(assign\_taxi\_frame)

        time\_label.place(relx=0.513, rely=0.39, height=34, width=69)

        time\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Time :""",

        )

        data7 = str(data[7])

        data8 = str(data[8])

        time\_data = tk.Label(assign\_taxi\_frame)

        time\_data.place(relx=0.616, rely=0.39, height=34, width=100)

        time\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=(data7 + ":" + data8),

        )

        pickup\_address\_data = tk.Label(assign\_taxi\_frame)

        pickup\_address\_data.place(relx=0.293, rely=0.468, height=35, width=323)

        pickup\_address\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[9],

        )

        destination\_label = tk.Label(assign\_taxi\_frame)

        destination\_label.place(relx=0.029, rely=0.546, height=34, width=132)

        destination\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Destination :""",

        )

        destination\_address\_lable = tk.Label(assign\_taxi\_frame)

        destination\_address\_lable.place(relx=0.249, rely=0.546, height=34, width=323)

        destination\_address\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[10],

        )

        total\_cost\_label = tk.Label(assign\_taxi\_frame)

        total\_cost\_label.place(relx=0.029, rely=0.624, height=34, width=111)

        total\_cost\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Total Cost :""",

        )

        total\_cost\_data = tk.Label(assign\_taxi\_frame)

        total\_cost\_data.place(relx=0.22, rely=0.624, height=34, width=206)

        total\_cost\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[11] + " .NPR",

        )

        payment\_method\_lable = tk.Label(assign\_taxi\_frame)

        payment\_method\_lable.place(relx=0.029, rely=0.702, height=33, width=185)

        payment\_method\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Payment Method :""",

        )

        payment\_method\_data = tk.Label(assign\_taxi\_frame)

        payment\_method\_data.place(relx=0.293, rely=0.702, height=34, width=206)

        payment\_method\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[12],

        )

        credit\_number\_lable = tk.Label(assign\_taxi\_frame)

        credit\_number\_lable.place(relx=0.029, rely=0.78, height=34, width=236)

        credit\_number\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            text=data[13],

            highlightthickness="2",

            foreground="#4A4A4A",

            font="-family {Noto Sans} -size 14",

            compound="left",

        )

        exp\_date\_lable = tk.Label(assign\_taxi\_frame)

        exp\_date\_lable.place(relx=0.411, rely=0.78, height=34, width=96)

        exp\_date\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[14],

        )

        cvv\_lable = tk.Label(assign\_taxi\_frame)

        cvv\_lable.place(relx=0.587, rely=0.78, height=34, width=96)

        cvv\_lable.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            highlightthickness="2",

            text=data[15],

        )

        show\_credit\_detail\_lable = tk.Checkbutton(assign\_taxi\_frame)

        show\_credit\_detail\_lable.place(

            relx=0.748, rely=0.78, relheight=0.051, relwidth=0.164

        )

        show\_credit\_detail\_lable.configure(

            activebackground="beige",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            highlightthickness="0",

            justify="left",

            selectcolor="#d9d9d9",

            text="""Show Details""",

        )

        assign\_taxi\_button = tk.Button(assign\_taxi\_frame)

        assign\_taxi\_button.place(relx=0.264, rely=0.889, height=33, width=141)

        assign\_taxi\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10 -weight bold",

            foreground="#FFFFFF",

            text="""Assign Taxi""",

            command=lambda: ControlPanelPage.taxi\_assign\_final\_frame(

                assign\_taxi\_frame, admin\_controller, data

            ),

        )

        cancel\_booking\_button = tk.Button(assign\_taxi\_frame)

        cancel\_booking\_button.place(relx=0.499, rely=0.889, height=33, width=141)

        cancel\_booking\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10 -weight bold",

            text="""Cancel Booking""",

            foreground="#FFFFFF",

            command=lambda: ControlPanelPage.cancel\_booking\_request(

                admin\_controller, data, assign\_taxi\_frame

            ),

        )

*if* data[16] == "Canceled":

            assign\_taxi\_button.config(state="disable")

            cancel\_booking\_button.config(state="disable")

*if* data[16] == "Accepted":

            assign\_taxi\_button.config(state="disable")

        back\_button = tk.Button(assign\_taxi\_frame)

        back\_button.place(relx=0.865, rely=0.031, height=33, width=71)

        back\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 10",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: ControlPanelPage.exit\_assign\_taxi\_page(

                assign\_taxi\_frame,

            ),

        )

        assign\_taxi\_frame.wait\_visibility()

        assign\_taxi\_frame.grab\_set()

        assign\_taxi\_frame.mainloop()

    @staticmethod

    def exit\_assign\_taxi\_page(assign\_taxi\_frame):

        assign\_taxi\_frame.destroy()

    @staticmethod

    def cancel\_booking\_request(admin\_controller, data, assign\_taxi\_frame):

        booking\_id = data[0]

        response = messagebox.askquestion(

            "Warning",

            "Do you really want to cancele this booking?",

            parent=assign\_taxi\_frame,

        )

*if* response == "yes":

            cancel\_booking\_control = admin\_controller()

            cancel\_booking\_control.cancel\_booking(booking\_id)

            assign\_taxi\_frame.destroy()

    @staticmethod

    def taxi\_assign\_final\_frame(assign\_taxi\_frame, admin\_controller, booking\_id):

        assign\_table\_frame = tk.Toplevel(assign\_taxi\_frame)

        assign\_table\_frame.geometry("600x400+193+31")

        assign\_table\_frame.resizable(0, 0)

        assign\_table\_frame.configure(background="#FFFFFF")

        search\_entry = tk.Entry(assign\_table\_frame)

        search\_entry.place(relx=0.030, rely=0.070, height=25, relwidth=0.200)

        search\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        search\_button = tk.Button(assign\_table\_frame)

        search\_button.place(relx=0.250, rely=0.070, height=25, relwidth=0.100)

        search\_button.configure(

            activebackground="beige",

            borderwidth="1",

            compound="left",

            font="-family {Noto Sans} -size 10",

            text="""Search""",

        )

        selected\_driver\_type = tk.StringVar()

        sort\_by\_combo = ttk.Combobox(

            assign\_table\_frame,

            textvariable=selected\_driver\_type,

            state="readonly",

            values=(

                "Avalable",

                "Booked",

            ),

        )

        sort\_by\_combo.place(relx=0.370, rely=0.070, height=25, relwidth=0.200)

        sort\_by\_combo.current(0)

        sort\_by\_combo.bind(

            "<<ComboboxSelected>>",

            lambda event: ControlPanelPage.available\_driver\_table\_view(

                assign\_table\_frame,

                admin\_controller,

                selected\_driver\_type,

                booking\_id,

                assign\_taxi\_frame,

            ),

        )

*# Define Column*

*if* selected\_driver\_type.get() == "Avalable":

            ControlPanelPage.available\_driver\_table\_view(

                assign\_table\_frame,

                admin\_controller,

                selected\_driver\_type,

                booking\_id,

                assign\_taxi\_frame,

            )

        assign\_table\_frame.wait\_visibility()

        assign\_table\_frame.grab\_set()

        assign\_table\_frame.mainloop()

    @staticmethod

    def available\_driver\_table\_view(

        assign\_table\_frame,

        admin\_controller,

        selected\_driver\_type,

        booking\_id,

        assign\_taxi\_frame,

    ):

*if* selected\_driver\_type.get() == "Avalable":

            table\_style = ttk.Style()

            table\_style.theme\_use("default")

            table\_style.configure(

                "Treeview",

                background="#FFFFFF",

                foreground="#4A4A4A",

                rowheight="35",

                fieldbackground="#FFFFFF",

            )

            table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

*# driver Table Scroll Bar*

            table\_scroll\_bar = Scrollbar(assign\_table\_frame)

            table\_scroll\_bar.place(relx=0.970, rely=0.150, height=280, width=15)

*# driver Table Build*

            driver\_booking\_table = ttk.Treeview(

                assign\_table\_frame,

                yscrollcommand=table\_scroll\_bar.set,

                selectmode="extended",

            )

            driver\_booking\_table.place(

                relx=0.030,

                rely=0.150,

                height=280,

                width=550,

            )

*# driver a scroll bar*

            table\_scroll\_bar.config(

                command=driver\_booking\_table.yview,

            )

            driver\_booking\_table["columns"] = (

                "Driver ID",

                "Full Name",

                "Contact",

                "Taxi Number",

            )

*# Format Our Columns*

            driver\_booking\_table.column("#0", width=0, stretch=NO)

            driver\_booking\_table.column("Driver ID", width=40, anchor=W)

            driver\_booking\_table.column("Full Name", width=70, anchor=W)

            driver\_booking\_table.column("Contact", width=70, anchor=W)

            driver\_booking\_table.column("Taxi Number", width=80, anchor=W)

*# Create Heading*

            driver\_booking\_table.heading("#0", text="", anchor=W)

            driver\_booking\_table.heading("Driver ID", text="Driver ID", anchor=W)

            driver\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

            driver\_booking\_table.heading("Contact", text="Contact", anchor=W)

            driver\_booking\_table.heading("Taxi Number", text="Taxi Number", anchor=W)

            availavble\_driver\_control = admin\_controller()

            record = availavble\_driver\_control.available\_driver\_fetcher()

*for* data *in* record:

                driver\_booking\_table.insert(

                    "",

                    index="end",

                    values=(

                        data[0],

                        data[1],

                        data[3],

                        data[4],

                    ),

                )

            assign\_button = tk.Button(assign\_table\_frame)

            assign\_button.place(relx=0.400, rely=0.880, height=35, width=120)

            assign\_button.configure(

                background="#007074",

                borderwidth="2",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#FFFFFF",

                text="""Assign""",

                command=lambda: ControlPanelPage.selcted\_driver\_row\_assign(

                    driver\_booking\_table,

                    admin\_controller,

                    booking\_id,

                    assign\_taxi\_frame,

                ),

            )

*else*:

            table\_style = ttk.Style()

            table\_style.theme\_use("default")

            table\_style.configure(

                "Treeview",

                background="#FFFFFF",

                foreground="#4A4A4A",

                rowheight="35",

                fieldbackground="#FFFFFF",

            )

            table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

*# driver Table Scroll Bar*

            table\_scroll\_bar = Scrollbar(assign\_table\_frame)

            table\_scroll\_bar.place(relx=0.970, rely=0.150, height=280, width=15)

*# driver Table Build*

            driver\_booking\_table = ttk.Treeview(

                assign\_table\_frame,

                yscrollcommand=table\_scroll\_bar.set,

                selectmode="extended",

            )

            driver\_booking\_table.place(

                relx=0.030,

                rely=0.150,

                height=280,

                width=550,

            )

*# driver a scroll bar*

            table\_scroll\_bar.config(

                command=driver\_booking\_table.yview,

            )

            driver\_booking\_table["columns"] = (

                "Driver ID",

                "Full Name",

                "Contact",

                "Taxi Number",

            )

*# Format Our Columns*

            driver\_booking\_table.column("#0", width=0, stretch=NO)

            driver\_booking\_table.column("Driver ID", width=40, anchor=W)

            driver\_booking\_table.column("Full Name", width=70, anchor=W)

            driver\_booking\_table.column("Contact", width=70, anchor=W)

            driver\_booking\_table.column("Taxi Number", width=80, anchor=W)

*# Create Heading*

            driver\_booking\_table.heading("#0", text="", anchor=W)

            driver\_booking\_table.heading("Driver ID", text="Driver ID", anchor=W)

            driver\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

            driver\_booking\_table.heading("Contact", text="Contact", anchor=W)

            driver\_booking\_table.heading("Taxi Number", text="Taxi Number", anchor=W)

            availavble\_driver\_control = admin\_controller()

            record = availavble\_driver\_control.booked\_driver\_fetcher()

*for* data *in* record:

                driver\_booking\_table.insert(

                    "",

                    index="end",

                    values=(

                        data[0],

                        data[1],

                        data[3],

                        data[4],

                    ),

                )

            assign\_button = tk.Button(assign\_table\_frame)

            assign\_button.place(relx=0.400, rely=0.880, height=35, width=120)

            assign\_button.configure(

                background="#007074",

                borderwidth="2",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#FFFFFF",

                text="""Not Now""",

                state="disabled",

            )

    @staticmethod

    def selcted\_driver\_row\_assign(

        driver\_booking\_table,

        admin\_controller,

        booking\_details,

        assign\_taxi\_frame,

    ):

        selected\_row = driver\_booking\_table.focus()

        driver\_id = driver\_booking\_table.item(selected\_row, "values")

        seleted\_driver\_id = driver\_id[0]

        booking\_id = booking\_details[0]

        assign\_taxi\_controll = admin\_controller()

        assign\_taxi\_controll.assign\_controller(seleted\_driver\_id, booking\_id)

        messagebox.showinfo(

            "Sucess",

            "Driver ID "

            + str(seleted\_driver\_id)

            + " Sucessfully  assigned to booking id "

            + str(booking\_id),

        )

        assign\_taxi\_frame.destroy()

* **Module : driver**

**Package : driver\_login.py**

code :

import tkinter as tk

from helper.constants import BG\_LOCATION

from .driver\_model import DriverModel

from .driver\_controller import DriverController

class DriverLogInPage:

    def \_\_init\_\_(self, root):

        self.root = root

        self.create\_driver\_login\_frame()

    def create\_driver\_login\_frame(self):

        self.login\_frame = tk.Frame(self.root)

        self.login\_frame.configure(background="#FFFFFF")

        self.login\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_driver\_login\_frame(login\_frame=self.login\_frame, root=self.root)

    @staticmethod

    def build\_driver\_login\_frame(login\_frame, root):

        welcome = tk.Label(login\_frame)

        welcome.place(relx=0.650, rely=0.149, height=59, width=200)

        welcome.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 22 -weight bold",

            text="""Driver Log In""",

        )

        username\_label = tk.Label(login\_frame)

        username\_label.place(relx=0.585, rely=0.299, height=21, width=206)

        username\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Username""",

        )

        username\_entry = tk.Entry(login\_frame)

        username\_entry.place(relx=0.585, rely=0.353, height=33, relwidth=0.301)

        username\_entry.configure(

            background="#EFF0F2", font="TkFixedFont", selectbackground="#c4c4c4"

        )

        password\_label = tk.Label(login\_frame)

        password\_label.place(relx=0.585, rely=0.434, height=20, width=185)

        password\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 16",

            text="""Password""",

        )

        password\_entry = tk.Entry(login\_frame)

        password\_entry.place(relx=0.585, rely=0.488, height=33, relwidth=0.301)

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        login\_button = tk.Button(login\_frame)

        login\_button.place(relx=0.674, rely=0.611, height=43, width=201)

        login\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Log In""",

            command=lambda: DriverLogInPage.admin\_login(

                username\_entry,

                password\_entry,

                login\_frame,

                root,

            ),

        )

        background\_label = tk.Label(login\_frame)

        background\_label.place(relx=-0.029, rely=0.0, height=739, width=700)

        background\_label.configure(anchor="w", compound="left")

        background\_image = tk.PhotoImage(file=BG\_LOCATION)

        background\_label.configure(image=background\_image)

        exit\_button = tk.Button(login\_frame)

        exit\_button.place(relx=0.100, rely=0.865, height=43, width=111)

        exit\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Back""",

            command=lambda: DriverLogInPage.exit\_application(login\_frame),

        )

        show\_pass\_button = tk.Button(login\_frame)

        show\_pass\_button.place(relx=0.900, rely=0.500, height=23, width=51)

        show\_pass\_button.configure(

            activebackground="beige",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""OO""",

        )

        show\_pass\_button.bind(

            "<Button-1>",

            lambda event: DriverLogInPage.show\_pass(

                event,

                password\_entry,

            ),

        )

        show\_pass\_button.bind(

            "<ButtonRelease-1>",

            lambda event: DriverLogInPage.hide\_pass(

                event,

                password\_entry,

            ),

        )

        login\_frame.mainloop()

    @staticmethod

    def show\_pass(

        event,

        password\_entry,

    ):

        password = password\_entry.get()

        password\_entry.config(show="", text=password)

    @staticmethod

    def hide\_pass(

        event,

        password\_entry,

    ):

        password\_entry.config(show="\*")

    def exit\_application(login\_frame):

        login\_frame.destroy()

    @staticmethod

    def admin\_login(

        username\_entry,

        password\_entry,

        login\_frame,

        root,

    ):

        driver = DriverModel(

            username=username\_entry.get(),

            password=password\_entry.get(),

        )

        user\_control = DriverController()

        user\_control.driver\_login\_control(driver, login\_frame, root)

**Package : driver\_dashboard.py**

code :

from datetime import datetime, date

import tkinter as tk

from tkinter import NO, W, Scrollbar, ttk

from tkinter import messagebox

from helper.constants import LOGO\_LOCATION

from tkintermapview import TkinterMapView

class DriverDashboard:

    def \_\_init\_\_(self, root, driver\_controller, record):

        self.root = root

        self.driver\_controller = driver\_controller

        self.record = record

        self.create\_dashboard\_frame()

    def create\_dashboard\_frame(self):

        self.dashboard\_frame = tk.Frame(self.root)

        self.dashboard\_frame.configure(background="#FFFFFF")

        self.dashboard\_frame.place(relx=-0.07, rely=0, height=768, width=1366)

        self.build\_dashboard\_frame(

            dashboard\_frame=self.dashboard\_frame,

            root=self.root,

            driver\_controller=self.driver\_controller,

            record=self.record,

        )

    @staticmethod

    def build\_dashboard\_frame(

        dashboard\_frame,

        root,

        driver\_controller,

        record,

    ):

        logo\_label = tk.Label(dashboard\_frame)

        logo\_label.place(relx=0.110, rely=0.068, height=71, width=80)

        logo\_label.configure(

            anchor="w",

            compound="left",

            background="#FFFFFF",

        )

        photo\_location = tk.PhotoImage(file=LOGO\_LOCATION)

        logo\_label.configure(image=photo\_location)

        DriverDashboard.upcoming\_trip\_frame(

            dashboard\_frame,

            driver\_controller,

            record,

        )

        customer\_table\_button = tk.Button(dashboard\_frame)

        customer\_table\_button.place(relx=0.110, rely=0.850, height=33, width=101)

        customer\_table\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Upcoming""",

            command=lambda: DriverDashboard.upcoming\_trip\_frame(

                dashboard\_frame,

                driver\_controller,

                record,

            ),

        )

        driver\_table\_button = tk.Button(dashboard\_frame)

        driver\_table\_button.place(relx=0.200, rely=0.850, height=33, width=101)

        driver\_table\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 9 -weight bold",

            foreground="#FFFFFF",

            text="""Completed""",

            command=lambda: DriverDashboard.completed\_trip\_frame(

                dashboard\_frame,

                driver\_controller,

                record,

            ),

        )

        profile\_refresh\_button = tk.Button(dashboard\_frame)

        profile\_refresh\_button.place(relx=0.720, rely=0.450, height=35, width=100)

        profile\_refresh\_button.configure(

            background="#FFFFFF",

            compound="left",

            text="Update",

            command=lambda: DriverDashboard.driver\_profile\_frame\_create(

                dashboard\_frame,

                driver\_controller,

                record,

            ),

        )

        DriverDashboard.driver\_profile\_frame\_create(

            dashboard\_frame,

            driver\_controller,

            record,

        )

        dashboard\_frame.mainloop()

    @staticmethod

    def upcoming\_trip\_frame(dashboard\_frame, driver\_controller, record):

        table\_frame = tk.Frame(dashboard\_frame, bg="#FFFFFF")

        table\_frame.place(relx=0.110, rely=0.590, height=160, width=1160)

        DriverDashboard.upcoming\_trip\_table(

            table\_frame,

            driver\_controller,

            record,

        )

    @staticmethod

    def upcoming\_trip\_table(table\_frame, driver\_controller, record):

        title = tk.Label(table\_frame)

        title.place(relx=0, rely=0, height=40, width=212)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            foreground="#3056D3",

            text="""Upcoming Trip""",

        )

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

        table\_scroll\_bar = Scrollbar(table\_frame)

        table\_scroll\_bar.place(relx=0.985, rely=0.250, height=118, width=15)

        upcoming\_booking\_table = ttk.Treeview(

            table\_frame,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        upcoming\_booking\_table.bind(

            "<Double-1>",

            lambda event: DriverDashboard.driver\_trip\_detail(

                event, table\_frame, trip\_fetched\_data, driver\_controller

            ),

        )

        upcoming\_booking\_table.place(

            relx=0,

            rely=0.250,

            height=118,

            width=1125,

        )

        table\_scroll\_bar.config(

            command=upcoming\_booking\_table.yview,

        )

        upcoming\_booking\_table["columns"] = (

            "Booking ID",

            "Full Name",

            "Date and Time",

            "Destination",

            "Total Cost",

            "Status",

        )

*# Format Our Columns*

        upcoming\_booking\_table.column("#0", width=0, stretch=NO)

        upcoming\_booking\_table.column("Booking ID", width=20, anchor=W)

        upcoming\_booking\_table.column("Full Name", width=70, anchor=W)

        upcoming\_booking\_table.column("Date and Time", width=70, anchor=W)

        upcoming\_booking\_table.column("Destination", width=240, anchor=W)

        upcoming\_booking\_table.column("Total Cost", width=40, anchor=W)

        upcoming\_booking\_table.column("Status", width=40, anchor=W)

*# Create Heading*

        upcoming\_booking\_table.heading("#0", text="", anchor=W)

        upcoming\_booking\_table.heading("Booking ID", text="Booking ID", anchor=W)

        upcoming\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        upcoming\_booking\_table.heading("Date and Time", text="Date and Time", anchor=W)

        upcoming\_booking\_table.heading("Destination", text="Destination", anchor=W)

        upcoming\_booking\_table.heading("Total Cost", text="Total Cost", anchor=W)

        upcoming\_booking\_table.heading("Status", text="Status", anchor=W)

*for* data *in* record:

            driver\_id = data[0]

        upcoming\_trip\_control = driver\_controller()

        trip\_fetched\_data = upcoming\_trip\_control.upcoming\_trip\_detail\_fetcher(

            driver\_id

        )

*for* data *in* trip\_fetched\_data:

            upcoming\_booking\_table.insert(

                "",

                index="end",

                values=(

                    data[0],

                    (data[2], data[3]),

                    (data[6], "---", data[7], ":", data[8]),

                    data[10],

                    data[11],

                    data[16],

                ),

            )

    @staticmethod

    def completed\_trip\_frame(dashboard\_frame, driver\_controller, record):

        table\_frame = tk.Frame(dashboard\_frame, bg="#FFFFFF")

        table\_frame.place(relx=0.110, rely=0.590, height=160, width=1160)

        DriverDashboard.completed\_trip\_table(

            table\_frame,

            driver\_controller,

            record,

        )

    @staticmethod

    def driver\_trip\_detail(

        event,

        table\_frame,

        trip\_fetched\_data,

        driver\_controller,

    ):

*for* data *in* trip\_fetched\_data:

            driver\_trip\_frame = tk.Toplevel(table\_frame)

            driver\_trip\_frame.title("Assign Taxi")

            driver\_trip\_frame.geometry("682x641+193+31")

            driver\_trip\_frame.resizable(0, 0)

            driver\_trip\_frame.configure(background="#FFFFFF")

            title = tk.Label(driver\_trip\_frame)

            title.place(relx=0.360, rely=0.031, height=43, width=250)

            title.configure(

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 18 -weight bold",

                text="""Booking Details""",

            )

            fullname\_label = tk.Label(driver\_trip\_frame)

            fullname\_label.place(relx=0.029, rely=0.156, height=34, width=122)

            fullname\_label.configure(

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Full Name :""",

            )

            fullname\_data = tk.Label(driver\_trip\_frame)

            fullname\_data.place(relx=0.22, rely=0.156, height=34, width=280)

            fullname\_data.configure(

                anchor="w",

                background="#FFFFFF",

                borderwidth="2",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[2] + " " + data[3],

            )

            map\_view\_frame = tk.Frame(driver\_trip\_frame)

            map\_view\_frame.place(relx=0.640, rely=0.156, height=140, width=220)

            map\_view\_frame.configure(

                background="#FFFFFF",

                borderwidth="2",

                highlightthickness="2",

            )

            location\_detail\_map\_view = TkinterMapView(map\_view\_frame)

            location\_detail\_map\_view.set\_tile\_server(

                "https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&z={z}&s=Ga",

                max\_zoom=60,

            )

            pickup\_cordinates = eval(data[19])

            destination\_coordinates = eval(data[20])

            location\_detail\_map\_view.set\_marker(

                pickup\_cordinates[0],

                pickup\_cordinates[1],

            )

            location\_detail\_map\_view.set\_marker(

                destination\_coordinates[0],

                destination\_coordinates[1],

            )

            location\_detail\_map\_view.set\_position(

                pickup\_cordinates[0],

                pickup\_cordinates[1],

            )

            location\_detail\_map\_view.set\_path(

                position\_list=[pickup\_cordinates, destination\_coordinates]

            )

            location\_detail\_map\_view.pack()

            passenger\_no\_label = tk.Label(driver\_trip\_frame)

            passenger\_no\_label.place(relx=0.029, rely=0.234, height=34, width=175)

            passenger\_no\_label.configure(

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""No of Passenger :""",

            )

            passenger\_no\_data = tk.Label(driver\_trip\_frame)

            passenger\_no\_data.place(relx=0.293, rely=0.234, height=34, width=48)

            passenger\_no\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[4],

            )

            taxi\_number = tk.Label(driver\_trip\_frame)

            taxi\_number.place(relx=0.029, rely=0.312, height=35, width=111)

            taxi\_number.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""No of Taxi :""",

            )

            taxi\_number\_data = tk.Label(driver\_trip\_frame)

            taxi\_number\_data.place(relx=0.22, rely=0.312, height=35, width=48)

            taxi\_number\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[5],

            )

            pick\_up\_date\_lable = tk.Label(driver\_trip\_frame)

            pick\_up\_date\_lable.place(relx=0.029, rely=0.39, height=34, width=133)

            pick\_up\_date\_lable.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Pick Up Date :""",

            )

            pick\_up\_date\_data = tk.Label(driver\_trip\_frame)

            pick\_up\_date\_data.place(relx=0.249, rely=0.39, height=35, width=163)

            pick\_up\_date\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[6],

            )

            pick\_up\_address\_label = tk.Label(driver\_trip\_frame)

            pick\_up\_address\_label.place(relx=0.029, rely=0.468, height=35, width=175)

            pick\_up\_address\_label.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Pick Up Address :""",

            )

            time\_label = tk.Label(driver\_trip\_frame)

            time\_label.place(relx=0.513, rely=0.39, height=34, width=69)

            time\_label.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Time :""",

            )

            data7 = str(data[7])

            data8 = str(data[8])

            time\_data = tk.Label(driver\_trip\_frame)

            time\_data.place(relx=0.616, rely=0.39, height=34, width=100)

            time\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=(data7 + ":" + data8),

            )

            pickup\_address\_data = tk.Label(driver\_trip\_frame)

            pickup\_address\_data.place(relx=0.293, rely=0.468, height=35, width=323)

            pickup\_address\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 10",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[9],

            )

            destination\_label = tk.Label(driver\_trip\_frame)

            destination\_label.place(relx=0.029, rely=0.546, height=34, width=132)

            destination\_label.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Destination :""",

            )

            destination\_address\_lable = tk.Label(driver\_trip\_frame)

            destination\_address\_lable.place(

                relx=0.249, rely=0.546, height=34, width=323

            )

            destination\_address\_lable.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 10",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[10],

            )

            total\_cost\_label = tk.Label(driver\_trip\_frame)

            total\_cost\_label.place(relx=0.029, rely=0.624, height=34, width=111)

            total\_cost\_label.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Total Cost :""",

            )

            total\_cost\_data = tk.Label(driver\_trip\_frame)

            total\_cost\_data.place(relx=0.22, rely=0.624, height=34, width=206)

            total\_cost\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[11] + " .NPR",

            )

            payment\_method\_lable = tk.Label(driver\_trip\_frame)

            payment\_method\_lable.place(relx=0.029, rely=0.702, height=33, width=185)

            payment\_method\_lable.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                text="""Payment Method :""",

            )

            payment\_method\_data = tk.Label(driver\_trip\_frame)

            payment\_method\_data.place(relx=0.293, rely=0.702, height=34, width=206)

            payment\_method\_data.configure(

                activebackground="#f9f9f9",

                anchor="w",

                background="#FFFFFF",

                compound="left",

                font="-family {Noto Sans} -size 14",

                foreground="#4A4A4A",

                highlightthickness="2",

                text=data[12],

            )

            trip\_completed\_button = tk.Button(driver\_trip\_frame)

            trip\_completed\_button.place(relx=0.365, rely=0.880, height=33, width=160)

            trip\_completed\_button.configure(

                activebackground="beige",

                background="#007074",

                borderwidth="2",

                compound="left",

                font="-family {Noto Sans} -size 10 -weight bold",

                foreground="#FFFFFF",

                text="""Trip Completed""",

                command=lambda: DriverDashboard.trip\_completed(

                    trip\_fetched\_data,

                    driver\_controller,

                    driver\_trip\_frame,

                ),

            )

*# Comparing today date and trip date*

            trip\_date = data[6]

            trip\_time = str(data[7]) + ":" + str(data[8]) + ":00"

            time\_final = datetime.strptime(trip\_time, "%H:%M:%S").time()

            trip\_date\_and\_time = datetime.combine(trip\_date, time\_final)

            today = datetime.now().isoformat(" ", "seconds")

            today\_date = datetime.strptime(today, "%Y-%m-%d %H:%M:%S")

*if* today\_date < trip\_date\_and\_time:

                trip\_completed\_button.config(state="disable")

                trip\_completed\_button.update()

*else*:

                trip\_completed\_button.config(state="normal")

                trip\_completed\_button.update()

*if* data[16] == "Completed":

                trip\_completed\_button.config(state="disable")

                trip\_completed\_button.update()

            back\_button = tk.Button(driver\_trip\_frame)

            back\_button.place(relx=0.865, rely=0.031, height=33, width=71)

            back\_button.configure(

                activebackground="beige",

                borderwidth="2",

                compound="left",

                font="-family {Noto Sans} -size 10",

                foreground="#FFFFFF",

                text="""Back""",

            )

            driver\_trip\_frame.wait\_visibility()

            driver\_trip\_frame.grab\_set()

            driver\_trip\_frame.mainloop()

    @staticmethod

    def trip\_completed(trip\_fetched\_data, driver\_controller, driver\_trip\_frame):

*for* data *in* trip\_fetched\_data:

            booking\_id = data[0]

            driver\_id = data[22]

        trip\_complete\_control = driver\_controller()

        trip\_complete\_control.completed\_trip\_controller(booking\_id, driver\_id)

        messagebox.showinfo("Sucess", "Marked as completed ")

        driver\_trip\_frame.destroy()

    @staticmethod

    def completed\_trip\_table(table\_frame, driver\_controller, record):

        title = tk.Label(table\_frame)

        title.place(relx=0, rely=0, height=40, width=212)

        title.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 18 -weight bold",

            foreground="#3056D3",

            text="""completed Trip""",

        )

        table\_style = ttk.Style()

        table\_style.theme\_use("default")

        table\_style.configure(

            "Treeview",

            background="#FFFFFF",

            foreground="#4A4A4A",

            rowheight="35",

            fieldbackground="#FFFFFF",

        )

        table\_style.map("Treeview", background=[("selected", "#C9C9C9")])

        table\_scroll\_bar = Scrollbar(table\_frame)

        table\_scroll\_bar.place(relx=0.985, rely=0.250, height=118, width=15)

        completed\_booking\_table = ttk.Treeview(

            table\_frame,

            yscrollcommand=table\_scroll\_bar.set,

            selectmode="extended",

        )

        completed\_booking\_table.bind(

            "<Double-1>",

            lambda event: DriverDashboard.driver\_trip\_detail(

                event, table\_frame, trip\_fetched\_data, driver\_controller

            ),

        )

        completed\_booking\_table.place(

            relx=0,

            rely=0.250,

            height=118,

            width=1125,

        )

        table\_scroll\_bar.config(

            command=completed\_booking\_table.yview,

        )

        completed\_booking\_table["columns"] = (

            "Booking ID",

            "Full Name",

            "Date and Time",

            "Destination",

            "Total Cost",

            "Status",

        )

*# Format Our Columns*

        completed\_booking\_table.column("#0", width=0, stretch=NO)

        completed\_booking\_table.column("Booking ID", width=20, anchor=W)

        completed\_booking\_table.column("Full Name", width=70, anchor=W)

        completed\_booking\_table.column("Date and Time", width=70, anchor=W)

        completed\_booking\_table.column("Destination", width=240, anchor=W)

        completed\_booking\_table.column("Total Cost", width=40, anchor=W)

        completed\_booking\_table.column("Status", width=40, anchor=W)

*# Create Heading*

        completed\_booking\_table.heading("#0", text="", anchor=W)

        completed\_booking\_table.heading("Booking ID", text="Booking ID", anchor=W)

        completed\_booking\_table.heading("Full Name", text="Full Name", anchor=W)

        completed\_booking\_table.heading("Date and Time", text="Date and Time", anchor=W)

        completed\_booking\_table.heading("Destination", text="Destination", anchor=W)

        completed\_booking\_table.heading("Total Cost", text="Total Cost", anchor=W)

        completed\_booking\_table.heading("Status", text="Status", anchor=W)

*for* data *in* record:

            driver\_id = data[0]

        completed\_trip\_control = driver\_controller()

        trip\_fetched\_data = completed\_trip\_control.completed\_trip\_detail\_fetcher(

            driver\_id

        )

*for* data *in* trip\_fetched\_data:

            completed\_booking\_table.insert(

                "",

                index="end",

                values=(

                    data[0],

                    (data[2], data[3]),

                    (data[6], "---", data[7], ":", data[8]),

                    data[10],

                    data[11],

                    data[16],

                ),

            )

    @staticmethod

    def driver\_profile\_frame\_create(

        dashboard\_frame,

        driver\_controller,

        record,

    ):

        profile\_details\_label = tk.Frame(dashboard\_frame)

        profile\_details\_label.place(relx=0.720, rely=0.068, height=280, width=325)

        profile\_details\_label.configure(

            background="#FFFFFF",

            highlightthickness="2",

            relief="flat",

        )

*for* data *in* record:

            driver\_id = data[0]

        driver\_profile\_data\_control = driver\_controller()

        profile\_details = driver\_profile\_data\_control.driver\_profile\_data\_fetcher(

            driver\_id

        )

        driver\_profile\_data\_control = driver\_controller()

        booking\_details = driver\_profile\_data\_control.driver\_booking\_data\_fetcher(

            driver\_id

        )

        profile\_data = profile\_details[0]

        booking\_data = booking\_details[0]

        username\_data = tk.Label(profile\_details\_label)

        username\_data.place(relx=0.215, rely=0.036, height=41, width=184)

        username\_data.configure(

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14 -weight bold",

            foreground="#4A4A4A",

            text=profile\_data[6],

        )

        name\_label = tk.Label(profile\_details\_label)

        name\_label.place(relx=0.062, rely=0.25, height=31, width=64)

        name\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Name :""",

        )

        name\_data = tk.Label(profile\_details\_label)

        name\_data.place(relx=0.277, rely=0.25, height=31, width=184)

        name\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text=profile\_data[1],

        )

        contact\_label = tk.Label(profile\_details\_label)

        contact\_label.place(relx=0.062, rely=0.357, height=31, width=74)

        contact\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Contact :""",

        )

        contact\_data = tk.Label(profile\_details\_label)

        contact\_data.place(relx=0.308, rely=0.357, height=31, width=104)

        contact\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text=profile\_data[3],

        )

        taxi\_label = tk.Label(profile\_details\_label)

        taxi\_label.place(relx=0.062, rely=0.464, height=31, width=54)

        taxi\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Taxi :""",

        )

        taxi\_data = tk.Label(profile\_details\_label)

        taxi\_data.place(relx=0.246, rely=0.464, height=31, width=184)

        taxi\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text=(profile\_data[9] + " " + profile\_data[10]),

        )

        taxi\_number\_label = tk.Label(profile\_details\_label)

        taxi\_number\_label.place(relx=0.062, rely=0.571, height=31, width=114)

        taxi\_number\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Taxi number :""",

        )

        taxi\_number\_data = tk.Label(profile\_details\_label)

        taxi\_number\_data.place(relx=0.431, rely=0.571, height=31, width=134)

        taxi\_number\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text=profile\_data[11],

        )

        total\_revenue\_label = tk.Label(profile\_details\_label)

        total\_revenue\_label.place(relx=0.062, rely=0.714, height=41, width=154)

        total\_revenue\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="""Total  Revenue :""",

        )

        total\_revenue\_data = tk.Label(profile\_details\_label)

        total\_revenue\_data.place(relx=0.031, rely=0.857, height=31, width=304)

        total\_revenue\_data.configure(

            anchor="center",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            foreground="#4A4A4A",

            text="XXXXX",

        )

        data\_view\_button = tk.Button(profile\_details\_label)

        data\_view\_button.place(relx=0.554, rely=0.75, height=23, width=41)

        data\_view\_button.configure(

            background="#FFFFFF",

            compound="left",

            text="""00""",

        )

        data\_view\_button.bind(

            "<Button-1>",

            lambda event: DriverDashboard.show\_revenue(

                event,

                total\_revenue\_data,

                booking\_data,

            ),

        )

        data\_view\_button.bind(

            "<ButtonRelease-1>",

            lambda event: DriverDashboard.hide\_revenue(

                event,

                total\_revenue\_data,

            ),

        )

        DriverDashboard.driver\_payment\_frame\_create(

            dashboard\_frame,

            record,

            driver\_controller,

        )

    @staticmethod

    def show\_revenue(event, total\_revenue\_data, booking\_data):

        total\_revenue\_data.config(text=booking\_data)

        total\_revenue\_data.update()

    @staticmethod

    def hide\_revenue(

        event,

        total\_revenue\_data,

    ):

        total\_revenue\_data.config(text="XXXXX")

    @staticmethod

    def driver\_payment\_frame\_create(

        dashboard\_frame,

        record,

        driver\_controller,

    ):

        today = date.today().strftime("%Y-%m")

*for* data *in* record:

            driver\_id = data[0]

        driver\_profile\_data\_control = driver\_controller()

        monthly\_revenue\_details = (

            driver\_profile\_data\_control.driver\_monthly\_data\_fetcher(

                driver\_id,

                today,

            )

        )

        monthly\_income = monthly\_revenue\_details[0]

        profile\_details\_label = tk.Frame(dashboard\_frame)

        profile\_details\_label.place(relx=0.470, rely=0.250, height=140, width=325)

        profile\_details\_label.configure(

            background="#FFFFFF",

            highlightthickness="2",

            relief="flat",

        )

        payment\_label = tk.Label(profile\_details\_label)

        payment\_label.place(relx=0.215, rely=0.036, height=41, width=184)

        payment\_label.configure(

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14 -weight bold",

            foreground="#4A4A4A",

            text="Monthly Revenue",

        )

        gross\_income\_label = tk.Label(profile\_details\_label)

        gross\_income\_label.place(relx=0.062, rely=0.320, height=31, width=120)

        gross\_income\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Gross Income :""",

        )

        gross\_income\_data = tk.Label(profile\_details\_label)

        gross\_income\_data.place(relx=0.445, rely=0.320, height=31, width=160)

        gross\_income\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text="XXXXX",

        )

        service\_cost\_label = tk.Label(profile\_details\_label)

        service\_cost\_label.place(relx=0.062, rely=0.520, height=31, width=110)

        service\_cost\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Service Cost :""",

        )

        service\_cost\_data = tk.Label(profile\_details\_label)

        service\_cost\_data.place(relx=0.410, rely=0.520, height=31, width=160)

        service\_cost\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text="XXXXX",

        )

        net\_income\_label = tk.Label(profile\_details\_label)

        net\_income\_label.place(relx=0.062, rely=0.720, height=31, width=100)

        net\_income\_label.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#4A4A4A",

            text="""Net Income :""",

        )

        net\_income\_data = tk.Label(profile\_details\_label)

        net\_income\_data.place(relx=0.380, rely=0.720, height=31, width=160)

        net\_income\_data.configure(

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 11",

            foreground="#4A4A4A",

            text="XXXXX",

        )

        data\_view\_button = tk.Button(profile\_details\_label)

        data\_view\_button.place(relx=0.850, rely=0.055, height=23, width=41)

        data\_view\_button.configure(

            background="#FFFFFF",

            compound="left",

            text="""00""",

        )

        data\_view\_button.bind(

            "<Button-1>",

            lambda event: DriverDashboard.show\_monthly\_revenue(

                event,

                gross\_income\_data,

                service\_cost\_data,

                net\_income\_data,

                monthly\_income,

            ),

        )

        data\_view\_button.bind(

            "<ButtonRelease-1>",

            lambda event: DriverDashboard.hide\_monthly\_revenue(

                event,

                gross\_income\_data,

                service\_cost\_data,

                net\_income\_data,

            ),

        )

    @staticmethod

    def show\_monthly\_revenue(

        event,

        gross\_income\_data,

        service\_cost\_data,

        net\_income\_data,

        monthly\_income,

    ):

        gross\_income = str(round(monthly\_income[0], 2))

        service\_cost = str(round(monthly\_income[1], 2))

        net\_income = str(round(monthly\_income[2], 2))

        gross\_income\_data.config(text=gross\_income)

        gross\_income\_data.update()

        service\_cost\_data.config(text=service\_cost)

        service\_cost\_data.update()

        net\_income\_data.config(text=net\_income)

        net\_income\_data.update()

    @staticmethod

    def hide\_monthly\_revenue(

        event,

        gross\_income\_data,

        service\_cost\_data,

        net\_income\_data,

    ):

        gross\_income\_data.config(text="XXXXX")

        gross\_income\_data.update()

        service\_cost\_data.config(text="XXXXX")

        service\_cost\_data.update()

        net\_income\_data.config(text="XXXXX")

        net\_income\_data.update()

**Package : driver\_register.py**

code :

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

from helper.exceptions import CustomException

from taxi.taxi\_register\_controller import TaxiController

from .driver\_model import DriverModel

from .driver\_controller import DriverController

class DriverRegisterPage:

    def \_\_init\_\_(self, root, control\_panel\_frame):

        self.root = root

        self.control\_panel\_frame = control\_panel\_frame

        self.create\_driver\_register\_frame()

    def create\_driver\_register\_frame(self):

        self.driver\_register\_frame = tk.Toplevel(self.root)

        self.driver\_register\_frame.title("Driver Register")

        self.driver\_register\_frame.configure(background="#FFFFFF")

        self.driver\_register\_frame.resizable(0, 0)

        self.driver\_register\_frame.geometry("460x579+420+91")

        self.build\_driver\_register\_frame(

            driver\_register\_frame=self.driver\_register\_frame,

        )

    @staticmethod

    def build\_driver\_register\_frame(driver\_register\_frame):

        welcome = tk.Label(driver\_register\_frame)

        welcome.place(relx=0.313, rely=0.060, height=41, width=250)

        welcome.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            text="""Driver Register""",

            font="-family {Noto Sans} -size 16 -weight bold",

        )

        fullname = tk.Label(driver\_register\_frame)

        fullname.place(relx=0.065, rely=0.19, height=41, width=104)

        fullname.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 14",

            compound="left",

            background="#FFFFFF",

            text="""Full Name""",

        )

        fullname\_entry = tk.Entry(driver\_register\_frame)

        fullname\_entry.place(

            relx=0.065,

            rely=0.259,

            height=23,

            relwidth=0.404,

        )

        fullname\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        license\_label = tk.Label(driver\_register\_frame)

        license\_label.place(relx=0.543, rely=0.19, height=41, width=200)

        license\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""License Number""",

        )

        license\_entry = tk.Entry(driver\_register\_frame)

        license\_entry.place(

            relx=0.543,

            rely=0.259,

            height=23,

            relwidth=0.404,

        )

        license\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        contact = tk.Label(driver\_register\_frame)

        contact.place(relx=0.065, rely=0.311, height=41, width=104)

        contact.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            text="""Contact""",

        )

        contact\_entry = tk.Entry(driver\_register\_frame)

        contact\_entry.place(

            relx=0.065,

            rely=0.38,

            height=23,

            relwidth=0.404,

        )

        contact\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        taxi\_number\_label = tk.Label(driver\_register\_frame)

        taxi\_number\_label.place(

            relx=0.543,

            rely=0.311,

            height=41,

            width=200,

        )

        taxi\_number\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Taxi Number""",

        )

        selected\_taxi\_number = tk.StringVar()

        taxi\_number\_entry = ttk.Combobox(

            driver\_register\_frame,

            textvariable=selected\_taxi\_number,

            state="readonly",

            values=("taxi\_number"),

        )

        taxi\_number\_entry.place(

            relx=0.543,

            rely=0.38,

            height=23,

            relwidth=0.404,

        )

        taxi\_number\_entry.bind(

            "<<ComboboxSelected>>",

            lambda event: DriverRegisterPage.taxi\_number\_fetcher(

                event,

                driver\_register\_frame,

                taxi\_number\_entry,

                selected\_taxi\_number,

            ),

        )

        username = tk.Label(driver\_register\_frame)

        username.place(relx=0.065, rely=0.430, height=41, width=104)

        username.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            text="""username""",

        )

        username\_entry = tk.Entry(driver\_register\_frame)

        username\_entry.place(

            relx=0.065,

            rely=0.500,

            height=23,

            relwidth=0.404,

        )

        username\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        password = tk.Label(driver\_register\_frame)

        password.place(relx=0.543, rely=0.430, height=41, width=104)

        password.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            text="""password""",

        )

        password\_entry = tk.Entry(driver\_register\_frame)

        password\_entry.place(

            relx=0.543,

            rely=0.500,

            height=23,

            relwidth=0.404,

        )

        password\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

            show="\*",

        )

        update\_button = tk.Button(driver\_register\_frame)

        update\_button.place(relx=0.522, rely=0.864, height=43, width=111)

        update\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12 -weight bold",

            foreground="#FFFFFF",

            text="""Register""",

            command=lambda: DriverRegisterPage.driver\_registration(

                driver\_register\_frame,

                fullname\_entry,

                license\_entry,

                contact\_entry,

                selected\_taxi\_number,

                username\_entry,

                password\_entry,

            ),

        )

        cancel\_button = tk.Button(driver\_register\_frame)

        cancel\_button.place(relx=0.250, rely=0.864, height=43, width=111)

        cancel\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Cancel""",

            command=lambda: DriverRegisterPage.cancel\_button(

                driver\_register\_frame,

            ),

        )

        driver\_register\_frame.grab\_set()

        driver\_register\_frame.mainloop()

    @staticmethod

    def taxi\_number\_fetcher(

        event,

        driver\_register\_frame,

        taxi\_number\_entry,

        selected\_taxi\_number,

    ):

        TaxiController.taxi\_number\_fetcher(taxi\_number\_entry)

        DriverRegisterPage.taxi\_detail\_viewer(

            driver\_register\_frame,

            selected\_taxi\_number,

        )

    @staticmethod

    def taxi\_detail\_viewer(

        driver\_register\_frame,

        selected\_taxi\_number,

    ):

        driver\_register\_frame = tk.Frame(driver\_register\_frame)

        driver\_register\_frame.place(

            relx=0.065,

            rely=0.570,

            height=150,

            relwidth=0.883,

        )

        driver\_register\_frame.config(

            background="#FFFFFF",

            borderwidth=2,

            relief="solid",

            highlightbackground="#000000",

        )

        welcome = tk.Label(driver\_register\_frame)

        welcome.place(relx=0.065, rely=0.10, height=41, width=365)

        welcome.configure(

            background="#FFFFFF",

            compound="left",

            text="""Driver Register""",

            font="-family {Noto Sans} -size 14 -weight bold",

            anchor="center",

        )

        taxi\_number = tk.Label(driver\_register\_frame)

        taxi\_number.place(relx=0.065, rely=0.35, height=41, width=110)

        taxi\_number.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

            text="""Taxi Number :""",

        )

        taxi\_number\_data = tk.Label(driver\_register\_frame)

        taxi\_number\_data.place(relx=0.360, rely=0.35, height=41, width=110)

        taxi\_number\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

        )

        taxi\_age = tk.Label(driver\_register\_frame)

        taxi\_age.place(relx=0.680, rely=0.35, height=41, width=75)

        taxi\_age.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

            text="""Taxi Age :""",

        )

        taxi\_age\_data = tk.Label(driver\_register\_frame)

        taxi\_age\_data.place(relx=0.880, rely=0.35, height=41, width=40)

        taxi\_age\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

        )

        taxi\_discription = tk.Label(driver\_register\_frame)

        taxi\_discription.place(relx=0.065, rely=0.65, height=41, width=100)

        taxi\_discription.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

            text="""Discription :""",

        )

        taxi\_discription\_data = tk.Label(driver\_register\_frame)

        taxi\_discription\_data.place(relx=0.360, rely=0.70, height=41, width=250)

        taxi\_discription\_data.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 12",

            compound="left",

            background="#FFFFFF",

        )

        DriverRegisterPage.taxi\_data\_fetcher(

            welcome,

            taxi\_number\_data,

            taxi\_age\_data,

            taxi\_discription\_data,

            selected\_taxi\_number,

        )

    @staticmethod

    def taxi\_data\_fetcher(

        welcome,

        taxi\_number\_data,

        taxi\_age\_data,

        taxi\_discription\_data,

        selected\_taxi\_number,

    ):

        TaxiController.taxi\_data\_fetcher(

            welcome,

            taxi\_number\_data,

            taxi\_age\_data,

            taxi\_discription\_data,

            selected\_taxi\_number,

        )

    @staticmethod

    def cancel\_button(driver\_register\_frame):

        driver\_register\_frame.destroy()

    @staticmethod

    def driver\_registration(

        driver\_register\_frame,

        fullname\_entry,

        license\_entry,

        contact\_entry,

        selected\_taxi\_number,

        username\_entry,

        password\_entry,

    ):

*try*:

            driver = DriverModel(

                fullname=fullname\_entry.get(),

                license\_number=license\_entry.get(),

                contact=contact\_entry.get(),

                taxi\_number=selected\_taxi\_number.get(),

                username=username\_entry.get(),

                password=password\_entry.get(),

            )

            driver\_control = DriverController()

            driver\_control.driver\_registration\_control(

                driver,

                driver\_register\_frame,

            )

*except* CustomException *as* error:

            messagebox.showerror("invalid", error, parent=driver\_register\_frame)

**Package : driver\_model.py**

code :

from pydantic import BaseModel, validator

from helper.exceptions import CustomException

import re

name\_reg = re.compile(

    "^[A-Z][a-zA-Z]{3,}(?: [A-Z][a-zA-Z]\*){0,2}$",

)  *# fullname regex*

contact\_reg = re.compile("^[0-9]{10}$")  *# contact regex*

username\_reg = re.compile("[a-z][a-z0-9]\*([.\_-][a-z0-9]+){0,3}$")  *# username regex*

class DriverModel(BaseModel):

    fullname: str = ""

    license\_number: str = ""

    contact: str = ""

    taxi\_number: str = ""

    username: str = ""

    password: str = ""

    @validator("fullname")

    def fullname\_validator(cls, value):

*if* not value:

*raise* CustomException("please enter your fullname !!")

*if* not name\_reg.match(value):

*raise* CustomException("invalid firname format !!")

*return* value

    @validator("license\_number")

    def validate\_license\_number(cls, value):

*if* not value:

*raise* CustomException("license number cannot be empty")

*return* value

    @validator("contact")

    def validate\_contact(cls, value):

*if* not value:

*raise* CustomException("contact cannot be empty")

*if* not contact\_reg.match(value):

*raise* CustomException("invalid format for contact")

*return* value

    @validator("taxi\_number")

    def validate\_taxi\_number(cls, value):

*if* not value:

*raise* CustomException("taxi number cannot be empty !!")

*if* value == "taxi\_number":

*raise* CustomException("please select taxi number!!")

*return* value

    @validator("username")

    def validate\_username(cls, value):

*if* not value:

*raise* CustomException("username cannot be empty")

*if* not username\_reg.match(value):

*raise* CustomException("invalid format for username")

*return* value

**Package : driver\_controller.py**

code :

from tkinter import messagebox

from helper.turbo\_db import Turbo\_db

from .driver\_dashboard import DriverDashboard

class DriverController:

    def \_\_init\_\_(self) -> None:

        self.\_connection = Turbo\_db.turbo\_connection()

    def driver\_registration\_control(

        self,

        driver,

        driver\_register\_frame,

    ):

*if* self.authenticate(driver):

            self.registration\_sucess(driver\_register\_frame)

    def authenticate(self, driver):

*try*:

            cursor = self.\_connection.cursor()

            statement = """CREATE TABLE IF NOT EXISTS drivers(

                driverID  SERIAL PRIMARY KEY,

                fullname   VARCHAR(50) NOT NULL,

                license\_number    VARCHAR(50) NOT NULL UNIQUE,

                contact VARCHAR(20) NOT NULL,

                taxi\_number VARCHAR(50) NOT NULL,

                driver\_status   VARCHAR(50) NOT NULL,

                username        VARCHAR(50) NOT NULL UNIQUE,

                password        VARCHAR(50) NOT NULL

            );"""

            data\_insert = """INSERT INTO drivers(fullname, license\_number,

            contact, taxi\_number, driver\_status, username, password) VALUES (%s, %s, %s, %s, %s, %s, %s);

            """

            data\_values = (

                driver.fullname,

                driver.license\_number,

                driver.contact,

                driver.taxi\_number,

                "Available",

                driver.username,

                driver.password,

            )

            cursor.execute(statement)

            cursor.execute(data\_insert, data\_values)

            taxi\_statement = (

                """UPDATE taxi SET status = 'Assigned' WHERE taxi\_number = %s;"""

            )

            data\_taxi = (driver.taxi\_number,)

            cursor.execute(taxi\_statement, data\_taxi)

            self.\_connection.commit()

*return* True

*except* Exception *as* e:

            print(e)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def registration\_sucess(self, driver\_register\_frame):

        driver\_register\_frame.destroy()

    def driver\_login\_control(self, driver, login\_frame, root):

*if* self.login\_authenticate(driver):

            self.login\_sucess(root, login\_frame)

*else*:

            messagebox.showerror(

                "Invalid Data",

                "username or password not matched",

            )

    def login\_authenticate(self, admin):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM drivers WHERE username=%s AND password = %s;"

            data = (admin.username, admin.password)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*if* self.record:

*return* True

*return* False

*except* Exception *as* error:

            print(error)

    def login\_sucess(self, root, login\_page):

        messagebox.showinfo(title="Congratulation", message="Welcome ")

        login\_page.destroy()

        DriverDashboard(root, DriverController, self.record)

    def upcoming\_trip\_detail\_fetcher(self, driver\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM booking as b LEFT JOIN drivers as d on b.driver\_id = d.driverid LEFT JOIN taxi as t on d.taxi\_number = t.taxi\_number WHERE d.driverid = %s AND b.booking\_status = 'Accepted';"

            did = str(driver\_id)

            data = did

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def completed\_trip\_detail\_fetcher(self, driver\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT \* FROM booking as b LEFT JOIN drivers as d on b.driver\_id = d.driverid LEFT JOIN taxi as t on d.taxi\_number = t.taxi\_number WHERE d.driverid = %s AND b.booking\_status = 'Completed';"

            did = str(driver\_id)

            data = did

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def completed\_trip\_controller(self, booking\_id, driver\_id):

*try*:

            cursor = self.\_connection.cursor()

            booking\_statement = (

                "Update booking set booking\_status = 'Completed' Where booking\_id = %s;"

            )

            bid = str(booking\_id)

            did = str(driver\_id)

            booking\_data = bid

            cursor.execute(booking\_statement, booking\_data)

            driver\_statement = (

                "Update drivers set driver\_status = 'Available' Where driverid = %s;"

            )

            driver\_data = did

            cursor.execute(driver\_statement, driver\_data)

            self.\_connection.commit()

*except* Exception *as* error:

            print(error)

    def driver\_booking\_data\_fetcher(self, driver\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "select sum(CAST(total\_cost AS decimal)) from booking as b join drivers as d on b.driver\_id = d.driverid where b.booking\_status = 'Completed' AND d.driverid = %s; "

            did = str(driver\_id)

            data = did

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def driver\_profile\_data\_fetcher(self, driver\_id):

*try*:

            cursor = self.\_connection.cursor()

            statement = "select \*  from drivers as d join taxi as t on d.taxi\_number = t.taxi\_number where d.driverid = %s;"

            did = str(driver\_id)

            data = did

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

    def driver\_monthly\_data\_fetcher(self, driver\_id, today):

*try*:

            cursor = self.\_connection.cursor()

            statement = "SELECT sum(CAST(total\_cost AS decimal)) as gross\_incrome, sum(CAST(total\_cost AS decimal)/3) as service\_cost, sum(CAST(total\_cost AS decimal)-CAST(total\_cost AS decimal)/3) as net\_income from booking as b join drivers as d on b.driver\_id = d.driverid where b.booking\_status = 'Completed' AND d.driverid = %s AND CAST(b.pickup\_date AS VARCHAR(50)) like concat('%%',%s,'%%');"

            did = str(driver\_id)

            td = str(today)

            data = (did, td)

            cursor.execute(statement, data)

            self.record = cursor.fetchall()

*return* self.record

*except* Exception *as* error:

            print(error)

* **Module : taxi**

**Package : taxi\_model.py**

code :

from pydantic import BaseModel, validator

from helper.exceptions import CustomException

class TaxiModel(BaseModel):

    brand: str = ""

    model: str = ""

    taxi\_number: str = ""

    taxi\_age: str = ""

    discription: str = ""

    @validator("brand")

    def validate\_brand(cls, value):

*if* not value:

*raise* CustomException("brand cannot be empty")

*return* value

    @validator("model")

    def validate\_model(cls, value):

*if* not value:

*raise* CustomException("model cannot be empty")

*return* value

    @validator("taxi\_number")

    def validate\_taxi\_number(cls, value):

*if* not value:

*raise* CustomException("taxi number cannot be empty")

*return* value

**Package : taxi\_register.py**

code :

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

from helper.exceptions import CustomException

from .taxi\_model import TaxiModel

from .taxi\_register\_controller import TaxiController

class TaxiRegisterPage:

    def \_\_init\_\_(self, root, control\_panel\_frame):

        self.root = root

        self.control\_panel\_frame = control\_panel\_frame

        self.create\_taxi\_register\_frame()

    def create\_taxi\_register\_frame(self):

        self.taxi\_register\_frame = tk.Toplevel(self.root)

        self.taxi\_register\_frame.title("Taxi Register")

        self.taxi\_register\_frame.configure(background="#FFFFFF")

        self.taxi\_register\_frame.resizable(0, 0)

        self.taxi\_register\_frame.geometry("460x440+420+91")

        self.build\_taxi\_register\_frame(

            taxi\_register\_frame=self.taxi\_register\_frame,

        )

    @staticmethod

    def build\_taxi\_register\_frame(taxi\_register\_frame):

        welcome = tk.Label(taxi\_register\_frame)

        welcome.place(relx=0.340, rely=0.069, height=41, width=200)

        welcome.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            text="""Taxi Register""",

            font="-family {Noto Sans} -size 16 -weight bold",

        )

        brand\_label = tk.Label(taxi\_register\_frame)

        brand\_label.place(relx=0.065, rely=0.19, height=41, width=104)

        brand\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            font="-family {Noto Sans} -size 14",

            compound="left",

            background="#FFFFFF",

            text="""Brand""",

        )

        list\_of\_brand = [

            "Hyundai",

            "Maruti",

            "Honda",

        ]

        selected\_brand = tk.StringVar()

        selected\_model = tk.StringVar()

        selected\_age = tk.StringVar()

        brand\_entry = ttk.Combobox(

            taxi\_register\_frame,

            textvariable=selected\_brand,

            values=list\_of\_brand,

        )

        brand\_entry.place(

            relx=0.065,

            rely=0.29,

            height=25,

            relwidth=0.404,

        )

        brand\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            state="readonly",

        )

        brand\_entry.bind(

            "<<ComboboxSelected>>",

            lambda event: TaxiRegisterPage.model\_viewer(

                event,

                model\_entry,

                selected\_brand,

            ),

        )

        model\_user = tk.Label(taxi\_register\_frame)

        model\_user.place(relx=0.543, rely=0.19, height=41, width=104)

        model\_user.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Model""",

        )

        model\_entry = ttk.Combobox(

            taxi\_register\_frame,

            textvariable=selected\_model,

            state="readonly",

        )

        model\_entry.place(

            relx=0.543,

            rely=0.29,

            height=25,

            relwidth=0.404,

        )

        model\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

        )

        taxi\_number\_label = tk.Label(taxi\_register\_frame)

        taxi\_number\_label.place(relx=0.065, rely=0.370, height=41, width=170)

        taxi\_number\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            compound="left",

            background="#FFFFFF",

            font="-family {Noto Sans} -size 14",

            text="""Taxi Number""",

        )

        taxi\_number\_entry = tk.Entry(taxi\_register\_frame)

        taxi\_number\_entry.place(

            relx=0.065,

            rely=0.470,

            height=25,

            relwidth=0.404,

        )

        taxi\_number\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        taxi\_age\_label = tk.Label(taxi\_register\_frame)

        taxi\_age\_label.place(relx=0.543, rely=0.370, height=41, width=104)

        taxi\_age\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="""Age""",

        )

        taxi\_age\_entry = tk.Spinbox(

            taxi\_register\_frame,

            from\_=1.0,

            to=9.0,

            state="readonly",

            textvariable=selected\_age,

        )

        taxi\_age\_entry.place(

            relx=0.543,

            rely=0.470,

            height=25,

            relwidth=0.404,

        )

        taxi\_age\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        discription\_label = tk.Label(taxi\_register\_frame)

        discription\_label.place(relx=0.065, rely=0.550, height=41, width=300)

        discription\_label.configure(

            activebackground="#f9f9f9",

            anchor="w",

            background="#FFFFFF",

            compound="left",

            font="-family {Noto Sans} -size 14",

            text="Discription",

        )

        discription\_entry = tk.Text(taxi\_register\_frame)

        discription\_entry.place(

            relx=0.065,

            rely=0.650,

            height=60,

            relwidth=0.883,

        )

        discription\_entry.configure(

            background="#EFF0F2",

            font="TkFixedFont",

            selectbackground="#c4c4c4",

        )

        register\_taxi\_button = tk.Button(taxi\_register\_frame)

        register\_taxi\_button.place(relx=0.522, rely=0.850, height=35, width=95)

        register\_taxi\_button.configure(

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Register""",

            command=lambda: TaxiRegisterPage.taxi\_registration(

                taxi\_register\_frame,

                selected\_brand,

                selected\_model,

                taxi\_number\_entry,

                selected\_age,

                discription\_entry,

            ),

        )

        cancel\_button = tk.Button(taxi\_register\_frame)

        cancel\_button.place(relx=0.250, rely=0.850, height=35, width=95)

        cancel\_button.configure(

            activebackground="beige",

            background="#007074",

            borderwidth="2",

            compound="left",

            font="-family {Noto Sans} -size 12",

            foreground="#FFFFFF",

            text="""Cancel""",

            command=lambda: TaxiRegisterPage.exit\_taxi\_registration(

                taxi\_register\_frame,

            ),

        )

        taxi\_register\_frame.grab\_set()

        taxi\_register\_frame.mainloop()

    @staticmethod

    def model\_viewer(event, model\_entry, selected\_brand):

        hyundai\_model = ["Grand i10 NIOS", "Aura", "Venue"]

        maruti\_model = ["Swift dzire", "Eeco", "Ritz"]

        Honda\_model = ["Amaze V", "city E", "mobilio s"]

*if* selected\_brand.get() == "Hyundai":

            model\_entry["values"] = hyundai\_model

            model\_entry.current(0)

            model\_entry.update()

*if* selected\_brand.get() == "Maruti":

            model\_entry["values"] = maruti\_model

            model\_entry.current(0)

            model\_entry.update()

*if* selected\_brand.get() == "Honda":

            model\_entry["values"] = Honda\_model

            model\_entry.current(0)

            model\_entry.update()

    @staticmethod

    def exit\_taxi\_registration(taxi\_register\_frame):

        taxi\_register\_frame.destroy()

    @staticmethod

    def taxi\_registration(

        taxi\_register\_frame,

        selected\_brand,

        selected\_model,

        taxi\_number\_entry,

        selected\_age,

        discription\_entry,

    ):

*try*:

            taxi = TaxiModel(

                brand=selected\_brand.get(),

                model=selected\_model.get(),

                taxi\_number=taxi\_number\_entry.get(),

                taxi\_age=(selected\_age.get()),

                discription=discription\_entry.get("1.0", "end"),

            )

            taxi\_number\_entry.delete("0", "end")

            discription\_entry.delete("1.0", "end")

            taxi\_controller = TaxiController()

            taxi\_controller.taxi\_registration\_control(

                taxi,

                taxi\_register\_frame,

            )

*except* CustomException *as* error:

            messagebox.showerror(

                "Invalid Data",

                error,

                parent=taxi\_register\_frame,

            )

**Package : taxi\_register\_controller.py**

code :

from tkinter import messagebox

from helper.turbo\_db import Turbo\_db

class TaxiController:

    def \_\_init\_\_(self) -> None:

        self.\_connection = Turbo\_db.turbo\_connection()

    def taxi\_registration\_control(self, taxi, taxi\_register\_frame):

*if* self.authenticate(taxi, taxi\_register\_frame):

            self.registration\_sucess(taxi\_register\_frame)

    def authenticate(self, taxi, taxi\_register\_frame):

*try*:

            cursor = self.\_connection.cursor()

            statement = """CREATE TABLE IF NOT EXISTS taxi(

                taxiID  SERIAL PRIMARY KEY,

                brand   VARCHAR(50) NOT NULL,

                model    VARCHAR(50) NOT NULL,

                taxi\_number VARCHAR(20) NOT NULL UNIQUE,

                taxi\_age INT NOT NULL,

                discription   VARCHAR(50) NOT NULL ,

                status  VARCHAR(40)

            );"""

            data\_insert = """INSERT INTO taxi(brand, model, taxi\_number,

            taxi\_age, discription,status) VALUES (%s, %s, %s, %s, %s, %s);

            """

            data\_values = (

                taxi.brand,

                taxi.model,

                taxi.taxi\_number,

                taxi.taxi\_age,

                taxi.discription,

                "Available",

            )

            cursor.execute(statement)

            cursor.execute(data\_insert, data\_values)

            self.\_connection.commit()

*return* True

*except* Exception *as* error:

            messagebox.showerror(

                "Invalid",

                "taxi number already exits",

                parent=taxi\_register\_frame,

            )

            print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* self.\_connection is not None:

                self.\_connection.close()

    def registration\_sucess(self, taxi\_register\_frame):

        messagebox.showinfo(

            "sucess",

            "taxi register",

            parent=taxi\_register\_frame,

        )

    @staticmethod

    def taxi\_number\_fetcher(taxi\_number\_entry):

        \_connection = Turbo\_db.turbo\_connection()

*try*:

            cursor = \_connection.cursor()

            statement = "SELECT taxi\_number FROM taxi WHERE status='Available';"

            cursor.execute(statement)

            taxi\_number = cursor.fetchall()

            taxi\_number\_entry["values"] = taxi\_number

            taxi\_number\_entry.update()

*except* Exception *as* error:

            print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* \_connection is not None:

                \_connection.close()

    @staticmethod

    def taxi\_data\_fetcher(

        welcome,

        taxi\_number\_data,

        taxi\_age\_data,

        taxi\_discription\_data,

        selected\_taxi\_number,

    ):

        \_connection = Turbo\_db.turbo\_connection()

*try*:

            cursor = \_connection.cursor()

            statement = "SELECT \* FROM taxi WHERE taxi\_number=%s;"

            data = (selected\_taxi\_number.get(),)

            cursor.execute(statement, data)

            taxi\_data = cursor.fetchall()

*if* taxi\_data:

*for* data *in* taxi\_data:

                    fetched\_car\_brand = data[1]

                    fetched\_car\_model = data[2]

                    fetched\_car\_number = data[3]

                    fetched\_car\_age = data[4]

                    fetched\_car\_discription = data[5]

*else*:

*return* False

            welcome.config(text=fetched\_car\_brand + " " + fetched\_car\_model)

            taxi\_number\_data.config(text=fetched\_car\_number)

            taxi\_age\_data.config(text=fetched\_car\_age)

            taxi\_discription\_data.config(text=fetched\_car\_discription)

            welcome.update()

            taxi\_number\_data.update()

            taxi\_number\_data.update()

            taxi\_discription\_data.update()

*except* Exception *as* error:

            print(error)

*finally*:

*if* cursor is not None:

                cursor.close()

*if* \_connection is not None:

                \_connection.close()