Taxi Booking System

Assignment II

Mathematics and Concepts for Computational Thinking CISO93-1

Final Report

UID: 2431910

Name: Pranish Samsohang

Submitted to University of Bedfordshire

Table of Content

List of Figures

List of tables

Abstract

A taxi booking system is a must need application for daily travelers. It is useful for daily local travelers as well as international tourists. As it would be lot easier for them to just pick a taxi rather than bargaining with the local taxi drivers in a language they are not used to. Moreover, the taxi booking system comes with an efficient, user-friendly UI. Where the customer and the driver can connect to each other which will result the driver and customer enjoying a ride in a mutual respect. The main objective of this system is to eliminate the problem of taxi drivers and customer as they both are unable to find each other at crucial times. This is where the system comes in helping both of them find each other, as the admin will assign the customer to the driver as per their location. The result will be less human error and better customer satisfaction. As The long-term development of a mobile booking taxi application service depends on the continued use of its passengers. (Weng, 2017)

Introduction

In this modern age, sound transportation is a must. A good means of transportation can lead to many important aspects like time saving which will eventually lead to better results, However, better transportation has become a major issue in today's world. As there is a major problem going out. Which is inability to find taxi. People have been finding it difficult to pick a taxi for themselves. In the same way taxi drivers are also not able to find customers. The taxi booking system has the perfect solution for such scenario. As the core objective of this system is to connect the drivers and the customers. Which will eventually lead to a better transportation experience. Personalized transport service to the individuals but also save taxi driver's energy, time, and expenses consumed in search of passengers. (Gang, 2021)

Objective

- To ensure safe modes of transportation for the users
- To help the drivers find more customers in a less period of time
- To reduce any human errors
- To help customer find a better deal than manual riding

Requirement Analysis

Functional requirements

User Management

- This system will allow users to register him/herself as a customer.
- This system will allow users to login using the right credential
- This system will allow users to be able to book a ride after giving location details like pickup and drop off locations.

Admin

- The admin should be able to login using the right credentials
- The admin should be able to view the overall workflow of the system
- The admin should be able to view the ride requests sent by the customer
- The admin should be able to view the drivers and their status whether they are busy or free.
- The admin should be able to assign task to the drivers who are free.

Driver

- The driver should be able to login using the right credentials

- The driver should be able to view the assigned task
- The driver should be able to view the details like locations of the customer.
- The driver should be able to notify the admin after task completion

Non–Functional requirements

Performance

- The system shall be responsive in every user action like login, register etc.
- The system shall exchange the location of driver and customer without any delays

Security

- This system shall store the user credentials
- This system will use role-based login to prevent unauthorized access.

Usability

- This system should be user-friendly.
- The first-time users should also be able to use it without any problems.

Reliability- The system will be able to run without crashing.

System Requirements

The Taxi Booking System is built on different technological stack. Which are given below-

- Programming language- Python
- GUI Framework- Tkinter
- Database-My SQL
- IDE- VS code
- Operating system- Windows

Task Description

Table 1 Task Description

S. N	Task	Description
1	User registration	The user must be able to register.
2	User Login	The user must be able to login using their credential
3	Request for a ride	The customer must be able to request for a ride
4	Accept a ride	The driver should be able to accept the ride
5	Assign customer	The admin can assign the customer to the driver.
7	Admin Login	The admin should be able to login him/herself.
8	Admin Panel	The admin can check which customer in assigned to which driver and also check if the driver is free or not
9	Driver Login	The driver should be able to login using their credentials
10	View Task	The driver should be able to view customer assigned to him
11	Task Completed	The driver should be able to inform the admin that the task has been completed so that the admin will be informed that the driver is free.

Use-Case Diagram

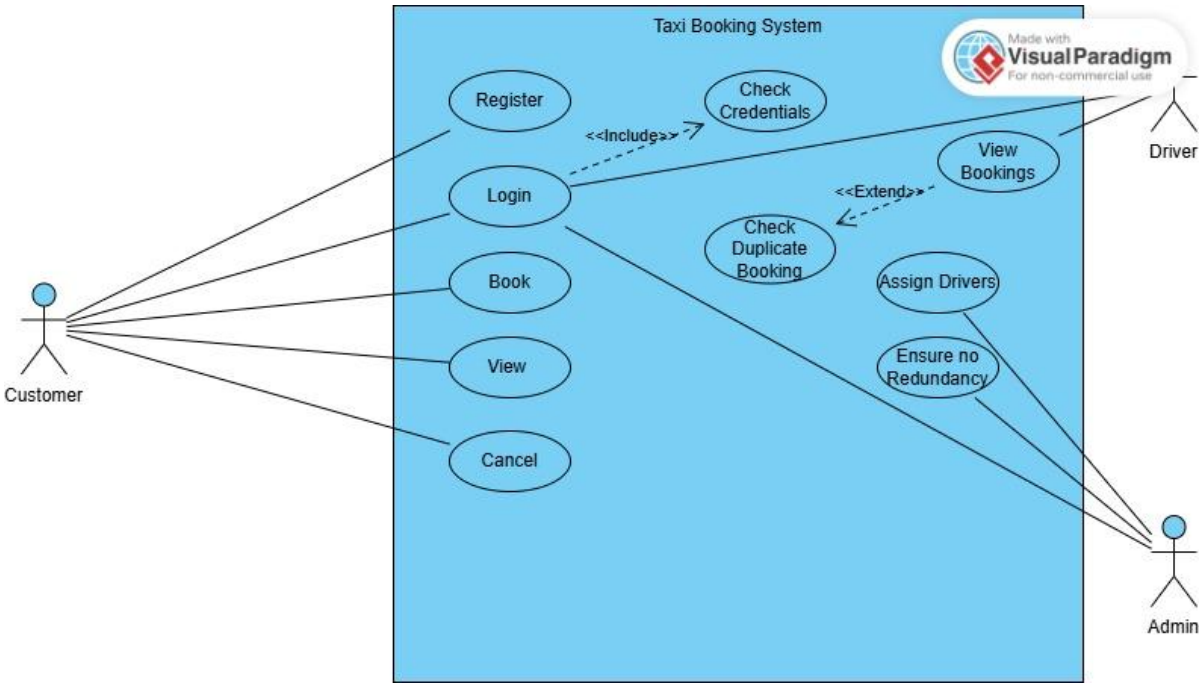


Figure 1Use Case Diagram

Register

Anyone can register him/herself as a user, admin or a driver. The system requires credential which will be saved in user's table. However, the driver credential will be stored in driver's table

Login

The user, admin and driver will be able to login using his/her credential and choosing his/her role. Furthermore, users won't be able to login himself despite having right credentials if the role isn't selected right.

Book

After logging in, the user will be able to book a taxi by giving details like time, pick-up location, drop-off location and name.

View assigned task

Driver will be able to view customer assigned to him. Following the details given by the customer like location details.

Assign driver

The admin will be able to assign driver with a customer. However, the admin will only be able to assign customer to the driver who have completed their task.

Task Completed

The driver will be able to inform the admin that he/she has completed the ride. In this manner the admin will be able to see that the driver is free and assign him a new customer.

Activity Diagram

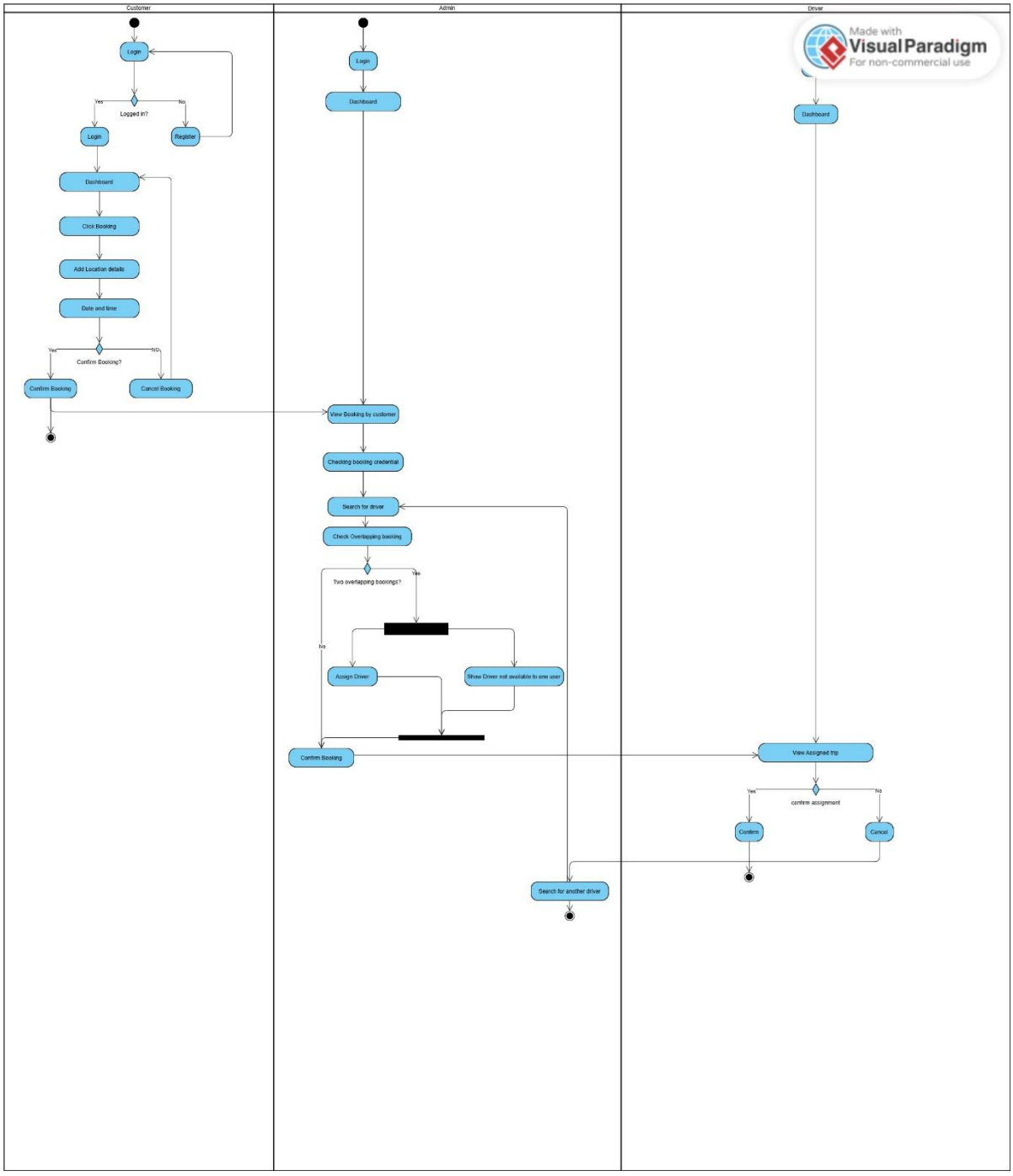


Figure 2 Activity Diagram

User

The user will be able to login using their credentials and selecting the user in the drop down menu. If the user is not registered the user can register him/herself in the register page. After

logging in as a user, the user is required to give the details like location, destination and phone number. After the details the user will be able to request for a ride.

Admin

The admin will be able to login using their credentials and selecting the admin in the drop down menu. If the admin is not registered the admin can register him/herself in the register page. After logging in as and admin, the admin can observe table with ride request. In another tab the admin can also see the status of the driver. Moving on the admin should be able to assign user to the driver in the assign driver section.

Driver

The driver shall login using the right credentials. If the driver is yet to be registered he/she can register himself in the driver section. After logging as a driver, the driver must be able to see the users assigned to him. The driver can also see the details about the user like locations and destinations. Furthermore, the driver can also inform the admin about the ride completion by clicking the ride completed button.

Class Diagram

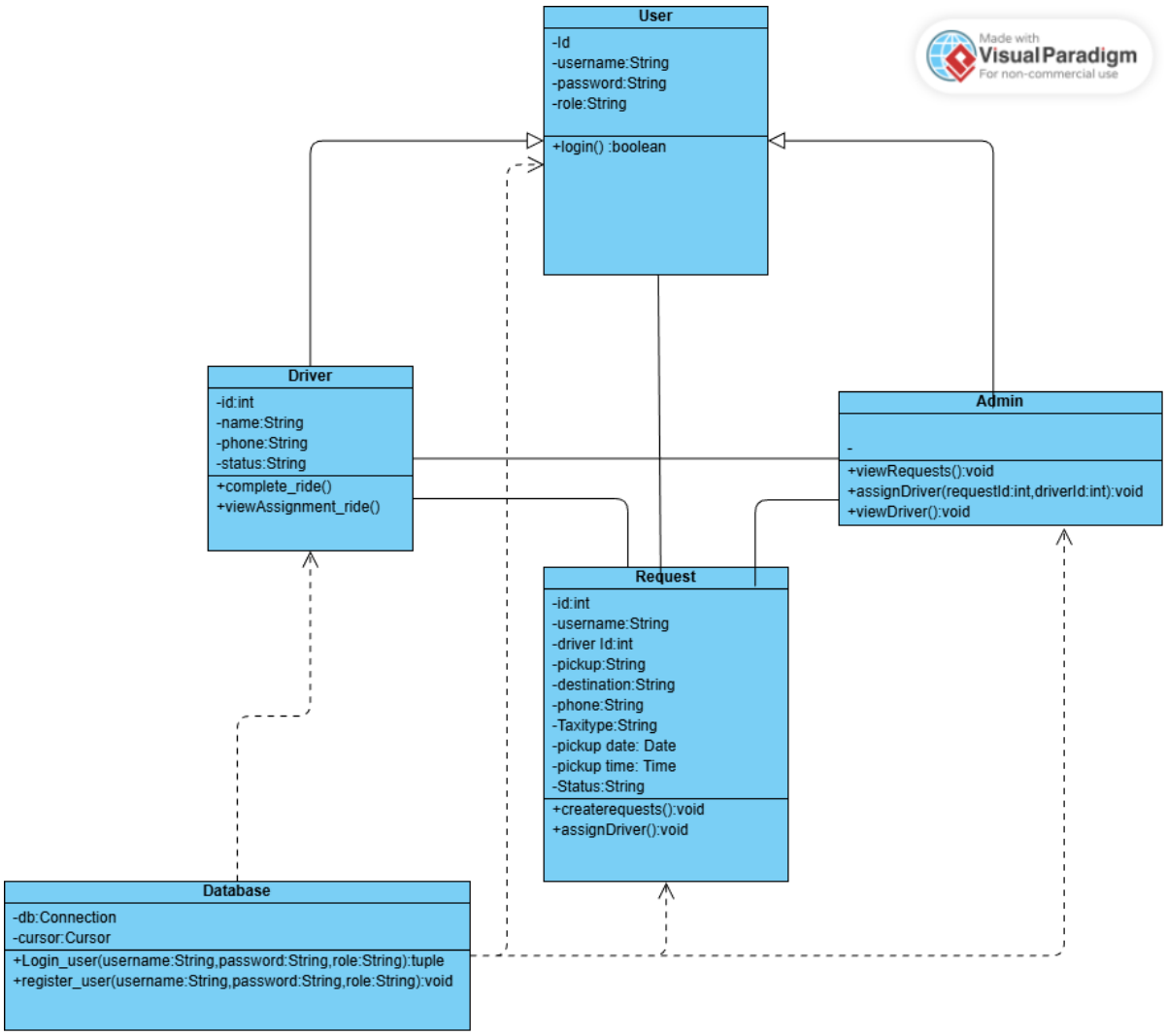


Figure 3 Class Diagram 1

User

The user class represents a general system user who can log in to the system. It serves as the base class for the admin and driver and stores common authentication information.z

Admin

The admin class represents a privileged user responsible for managing the overall system. It can view taxi requests, assign drivers to requests, and monitor driver information.

Requests

The Request class represents a taxi booking system made by a user. It stores pickup details, destination, time, taxi type, assigned driver, and ride status.

Driver

The driver class is for the taxi driver who will observe the assigned rides and complete trips.

Database

The database class manages database connectivity and execute all SQL operations such as user authentication, user registration, and data retrieval. It acts as the data access layer of the system

Database Design

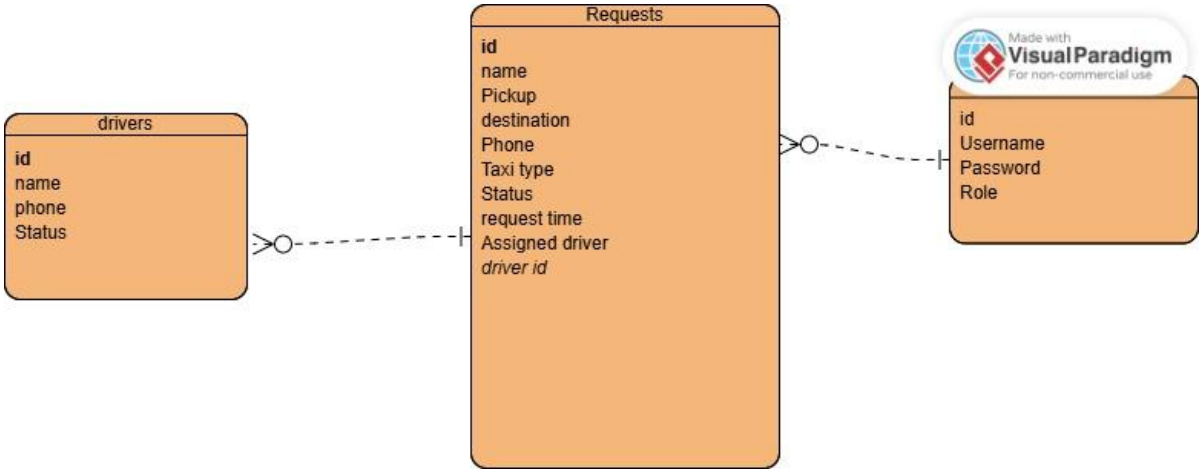


Figure 4 ERD Diagram

Skeleton Tables

Users

	id	username	password	role
▶	1	User	123	User
	2	Admin	123	Admin
	3	Driver	123	Driver
*	NULL	NULL	NULL	NULL

Figure 5 User

Requests

id	username	pickup	destination	phone	taxi_type	pickup_date	pickup_time	status	dr
1	User	l	m	9	Mini	2025-10-10	12:25:30	Completed	1
2	User	Kathmandu	Bhaktapur	9808421918	Mini	2025-12-16	12:15:56	Completed	1
3	User	Kupandole	New Road	9808421918	Mini	2025-12-14	12:04:12	Completed	1
4	User	K	u	9845351214	Mini	2024-12-06	12:15:16	Completed	1
5	User	k	u	9	Mini	2025-12-09	12:23:21	Pending	NULL
6	User	Kathmandu	Nagarkot	98746546515	Mini	2022-10-10	12:12:12	Completed	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6 Requests

Drivers

id	name	phone	status
1	Driver	9841538677	Available
NULL	NULL	NULL	NULL

Figure 7 Drivers

Data dictionary

User

Table 2 User

Field Name	Data Type	length	Index	Null	Default value	Validation rule	Description
Id	int		PK	No			Identify Id
Username	varchar	50		No			Identify Username
Password	varchar	50		No			Identify Password
role	varchar	50		No			Identify role

Requests

Table 3 Requests

Field Name	Data Type	Length	Index	Null	Default Value	Validation rule	Description
Id	int		PK	No			Identify Id
Username	Varchar	50		No			Identify Username
Pick-up destination	Varchar	50		No			Identify Pick-up destination
Phone	Int			No			Identify Phone
Taxi-type	Varchar	50		No			Identify Taxi-type
Pickup-date	Date			No			Identify Pickup-date
Pick-up time	Time			No			Identify Pickup-time
status	Varchar	50		No			Identify status
driver	Varchar	50		No			Identify driver

Driver

Table 4 Driver

Field Name	Data Type	Length	Index	Null	Default Value	Validation rule	Description
Id	Int		PK	No			Identify Id
Name	Varchar	50		No			Identify Name
Phone	Int			No			Identify Phone
status	varchar	50		No			Identify Status

Implementation

The implementation of taxi booking system includes converting the system design into a working software. The system was built by using python for backend, Python (Tkinter) was used for the graphical user interface of this system. Moreover, the system also uses MySQL as its backend database.

The relational database used by the system consists of tables like users, requests and drivers. Tables consist of PRIMARY KEY, FOREIGN KEY and CONSTRAINTS.

The system includes login and authentication module which plays and important role. The login is where the users, drivers and admin can login as per their role. Registration module is also available if the user or the driver is not registered yet.

The request module is vital for the user. The user will be able to request taxi by giving different details. Driver and user management module is for the admin. The admin will be able to assign customer to the drivers.

Computational thinking

The taxi booking system strictly follow the core principles of computational thinking. The principles of computational thinking focuses on breaking down complex problems into smaller simple and manageable parts, identifying patterns, hiding unnecessary details and also designing a step by step solution.

Decomposition

The whole taxi booking system is divided into many smaller manageable modules. Some of the examples are login, request taxi, assign customer. It makes the system more organized. As the complex problem is divided into different manageable parts, it will be far easier to modify the code or fix the bug in any cases. The main component includes

Pattern Recognition

The taxi booking system recognizes duplicate booking as it will avoid the admin to assign new user to the driver who's already been booked or in a ride.

Abstraction

Abstraction focuses on important details rather than minor details. The users can only see relevant information like driver information. Similarly the driver can only see the passenger details. The system will abstract minor details like how the driver is assigned to them.

Algorithm Design

Algorithm defines the step by step logic of the entire system. The system works on the following algorithm

- User sends a taxi request
- Admin assigns the driver to the user
- The user gets driver details
- The driver notifies the admin after the task completion

UI design

Login Page

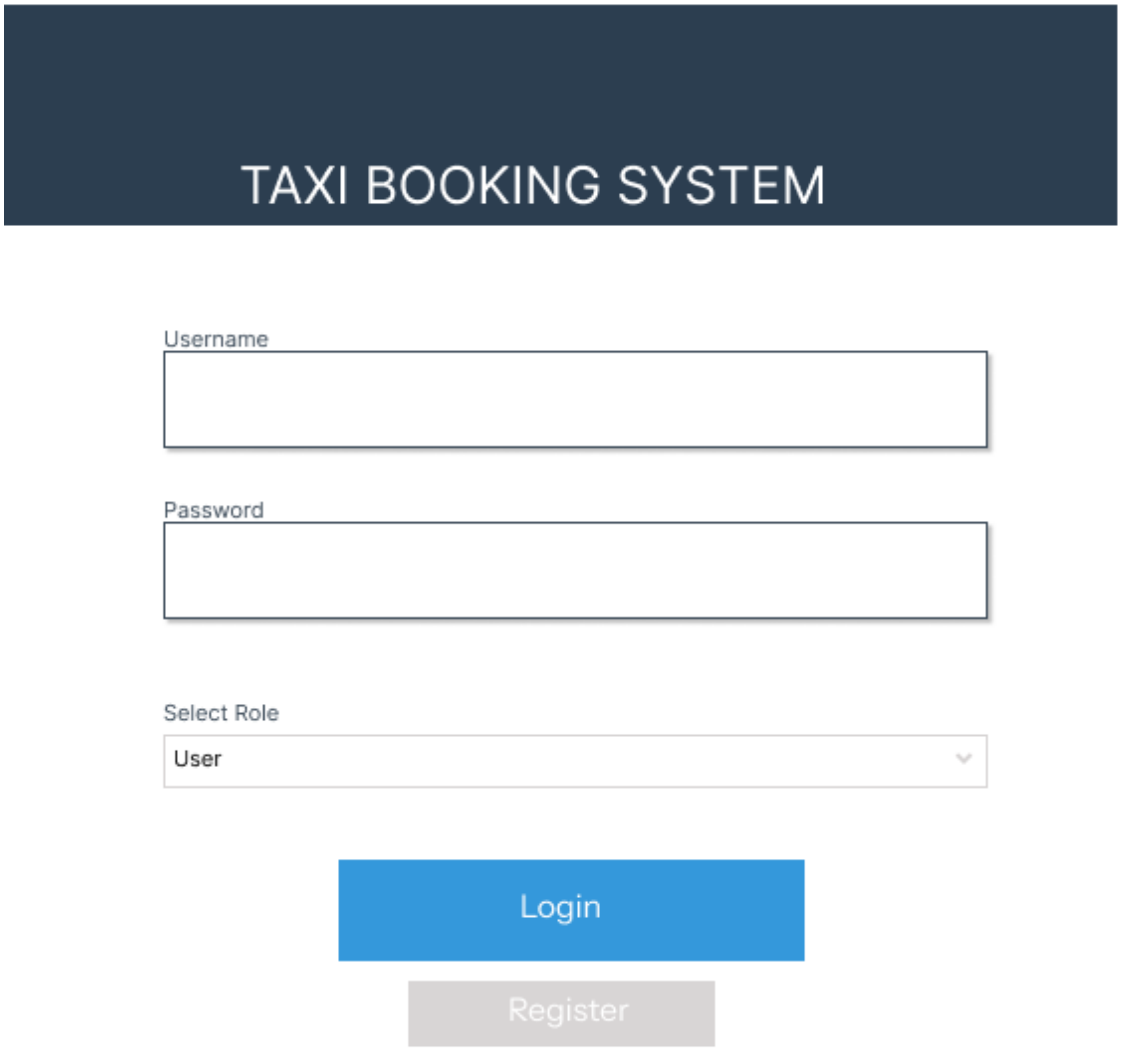


Figure 8 Login Page

User Dashboard

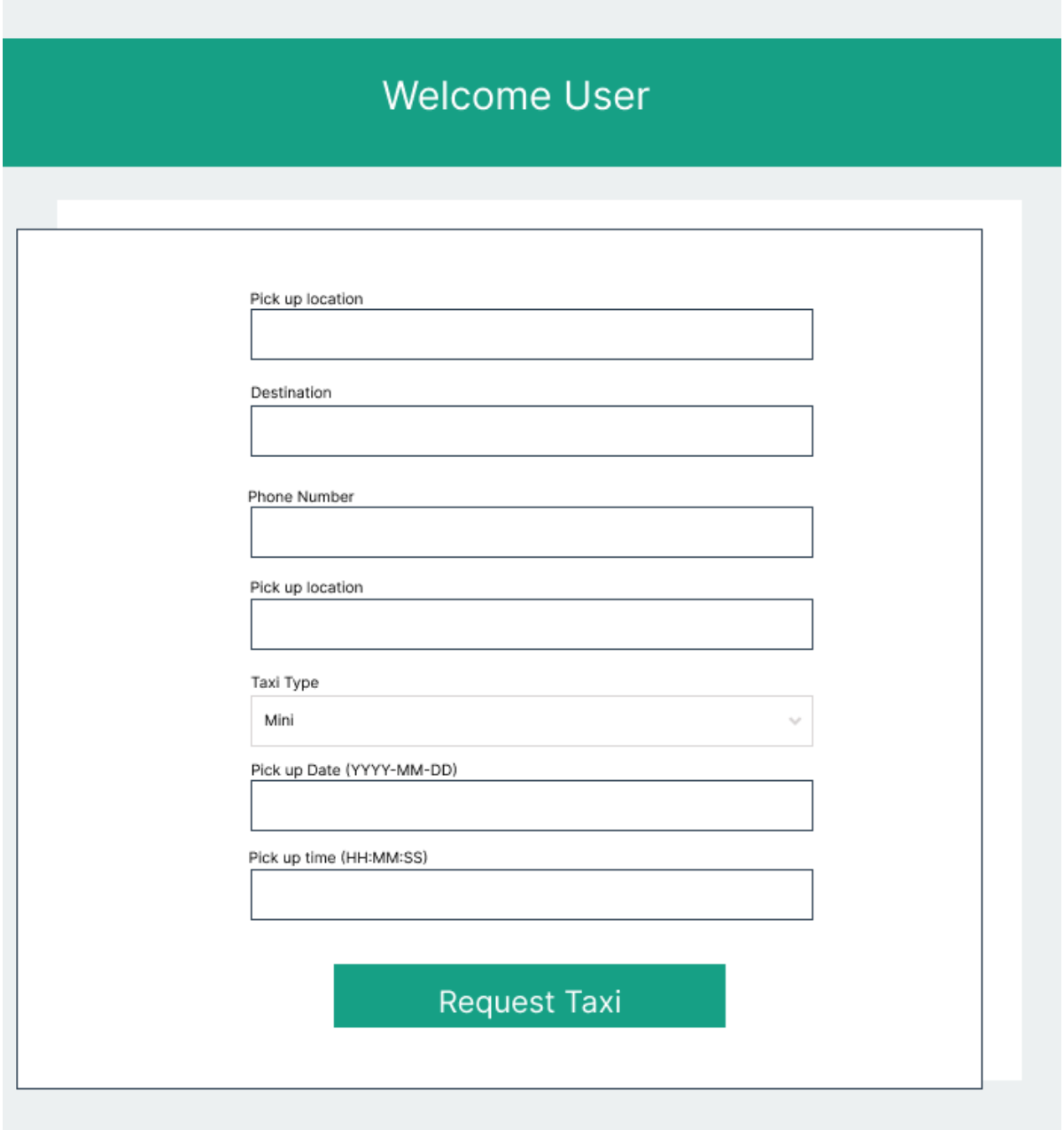


Figure 9 User Dashboard

Admin Dashboard

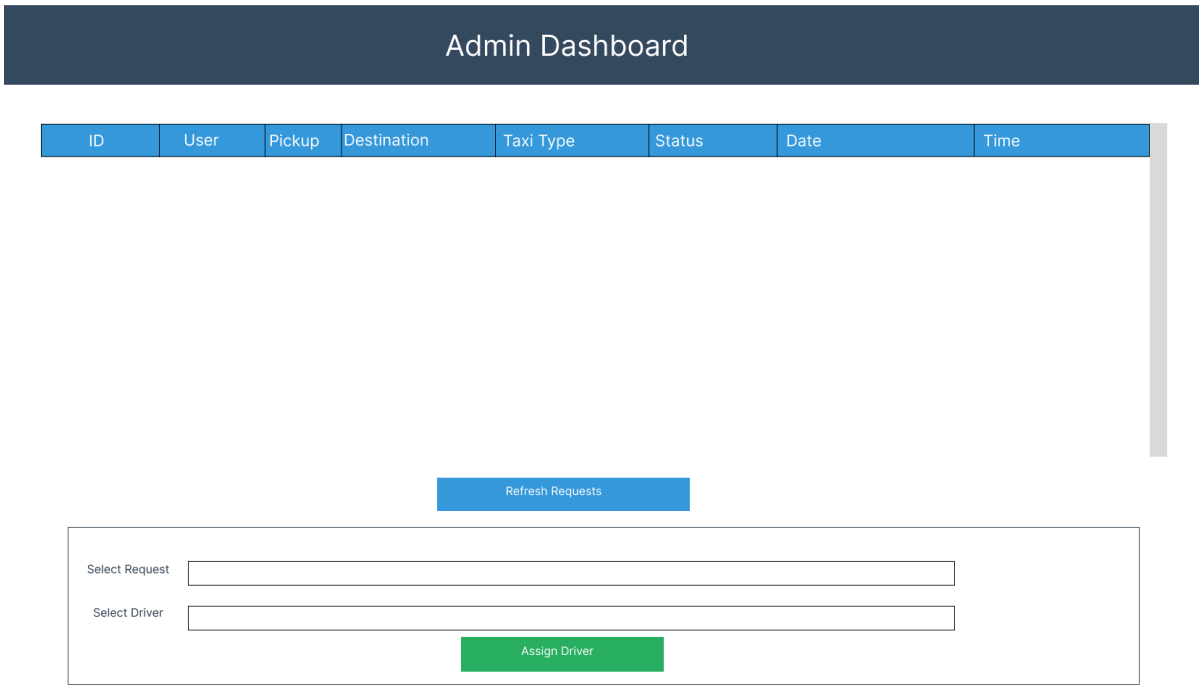


Figure 10 Admin Dashboard

Driver Dashboard

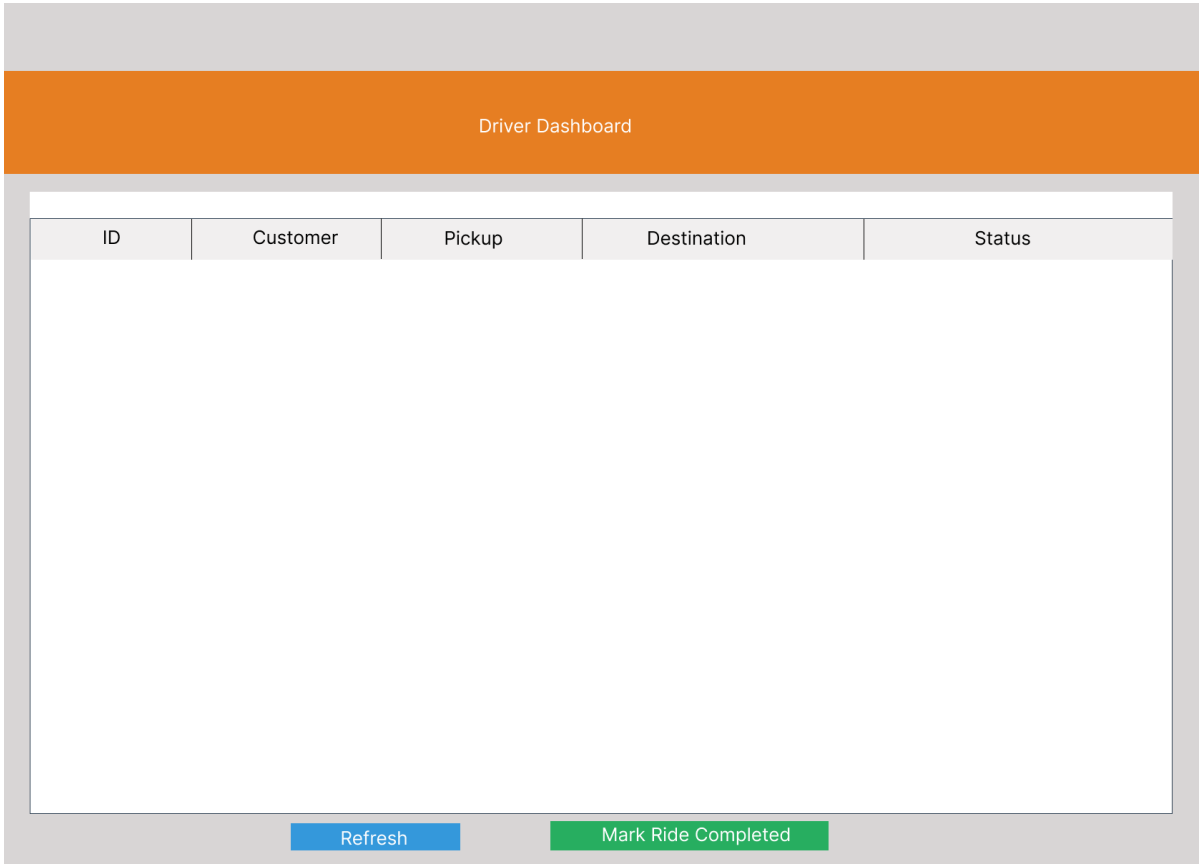


Figure 11 Driver Dashboard

Testing

Table 5 Testing

SN	Input	Expected Output	Actual Output	Result
1	Enter the name and password in the login page	The user should be able to login	The user was able to login	Pass
2	Login according to the role using a drop box in the login page	All three actors must be able to login according to their role	All three actors were able to login	Pass
3	Enter the ride details and click the “request taxi button”	The user should be able to request a TAXI	The user was able to request a TAXI	Pass
4	Login as an admin	The admin must be able to see the user requests	The admin was able to see the user requests	Pass
5	Login as an admin	The admin should also be able to see the drivers available	The admin was able to see all the drivers available	Pass
6	After logging as an admin, select customer and driver in the assign driver section.	The admin must be able to assign a customer to a driver	The admin was able to assign a customer to the driver	Pass
7	After logging as an admin, select	The admin must not be able to	The admin cannot assign a	Pass

	customer and driver in the assign driver section.	see the drivers which are busy	driver who is already assigned	
8	Login as a driver	The driver must be able to see the details of the customer assigned to him	The driver was able to see the customer assigned to him	Pass
9	Login as a driver and click on ride completed	On task completion, the driver must be able to inform the admin about the ride completion	The driver was able to inform the admin about the ride completion	Pass

Discussion

This project was finally able to meet its primary objectives, like design and implementing the functional requirements. Talking about the system. The system has implemented user-friendly interface. The system interacts with “hw” database, with a sound flow. The system has been tested number of times and is yet to fail. However, the system has some limitations, like it doesn’t calculate the fare and the distance between the driver and the customer. Moving on some other limitations, the user won’t be able to give a review to the driver.

If we compare this system to other existing manual taxi booking system which still exists, our system is able to reduce human error and also improves the efficiency by automating data flow and record management. Throughout the entire project I have gained a lot of experiences in system analysis, how to design a database.

In the future, this system can be enhanced by adding extra feature like calculating fair, adding a review section and also adding stronger security measures.

Conclusion

The main objective of the taxi booking system was to eliminate the manual taxi booking practice in Nepal. As manual taxi booking system consists of human error, miscommunication and other problems like demand, supply imbalance. The system implements feature like secure login mechanism where the actors like users, drivers and admin must authenticate before logging in, the customer will be able to request a taxi and the driver will be able to see the customer details.

The taxi booking system eliminates manual effort, minimizes errors, and provides a structured module compared to the existing traditional method. The system meets its primary objectives however, future enhancements like cloud based security, and live location tracking can further improve its performance.

References

Gang, C. (2021). From Consumer Satisfaction to Recommendation of Mobile App–Based Services: An Overview of Mobile Taxi Booking Apps. *Sage Journals*.

Weng, G. S. (2017). Mobile taxi booking application service's continuance usage intention by users. *Transportation Research Part D: Transport and Environment*.

Artificial intelligence tools (ChatGPT by OpenAI) were used to assist in brainstorming, improving grammar, and clarifying explanations.

Figma Design Link - https://www.figma.com/design/SAU94U5l56ogLV6JwaTrHW/Untitled?node-id=0-1&t=3rAPHM29N52vYVB1-1

Appendix

main_app

```
class MainApp(BaseWindow):
    def __init__(self):
            self.db = Database()
        except Exception as e:
            messagebox.showerror("Database Connection Error",
                                 f"Could not connect to the database. Please check your MySQL server and credentials in db_manager.py. Error: {e}")
            self.window.destroy()
            return

        self.set_db_manager(self.db)
        self.window.resizable(False, False)

        self._setup_ui()

    def _setup_ui(self):
        """Sets up the UI elements for the login window."""
        # Header Frame
        header_frame = tk.Frame(self.window, bg="#2c3e50", height=120)
        header_frame.pack(fill="x")
        header_frame.pack_propagate(False)

        tk.Label(header_frame, text="🚕", font=("Segoe UI", 40),
                 bg="#2c3e50").pack(pady=(15, 0))
        tk.Label(header_frame, text="TAXI BOOKING SYSTEM",
                 font=("Segoe UI", 20, "bold"), fg="white", bg="#2c3e50").pack()

        # Login Form Frame
        form_frame = tk.Frame(self.window, bg="#f8f9fa")
        form_frame.pack(pady=40, padx=50, fill="both", expand=True)

        # Username
        self.entry_username = self.create_input_field(form_frame, "Username")

        # Password
        self.entry_password = self.create_input_field(form_frame, "Password", is_password=True)

        # Role Selector
        tk.Label(form_frame, text="Select Role", font=("Segoe UI", 11),
                 fg="#2c3e50", bg="#f8f9fa", anchor="w").pack(fill="x", pady=(15, 5))
        self.combo_role = ttk.Combobox(form_frame, values=["User", "Admin", "Driver"],
                                       state="readonly", font=("Segoe UI", 11))
        self.combo_role.pack(fill="x", ipady=5)
        self.combo_role.set("User")

        # Buttons
        tk.Button(form_frame, text="Login", width=25, command=self.login,
                  font=("Segoe UI", 12, "bold"), bg="#3498db", fg="white",
                  relief="flat", cursor="hand2", pady=10).pack(pady=(25, 10))

        tk.Button(form_frame, text="Register", width=25, command=self.open_register_page,
                  font=("Segoe UI", 11), bg="#95a5a6", fg="white",
                  relief="flat", cursor="hand2", pady=8).pack()

    def login(self):
        """Handles the login process."""
        username = self.entry_username.get()
        password = self.entry_password.get()
        role = self.combo_role.get()

        if username == "" or password == "":
            messagebox.showerror("Error", "All fields are required")
            return

        try:
            # Use the instance created in __init__
            result = self.db.login_user(username, password, role)
        except Exception as e:
            messagebox.showerror("Database Error", f"An error occurred during login: {e}")
            return

        if result:
            messagebox.showinfo("Success", f"Login Successful as {role}")

            # Pass the database instance to the next window
            if role == "Admin":
                AdminDashboard(self.window, self.db)
            elif role == "User":
                UserDashboard(username, self.window, self.db)
            elif role == "Driver":
                DriverDashboard(username, self.window, self.db)
        else:
            messagebox.showerror("Error", "Invalid username, password or role")

    def open_register_page(self):
        """Opens the registration window."""
        # Pass the database instance to the next window
        RegisterWindow(self.window, self.db)

if __name__ == "__main__":
    app = MainApp()
    app.run()
```

Gui_base

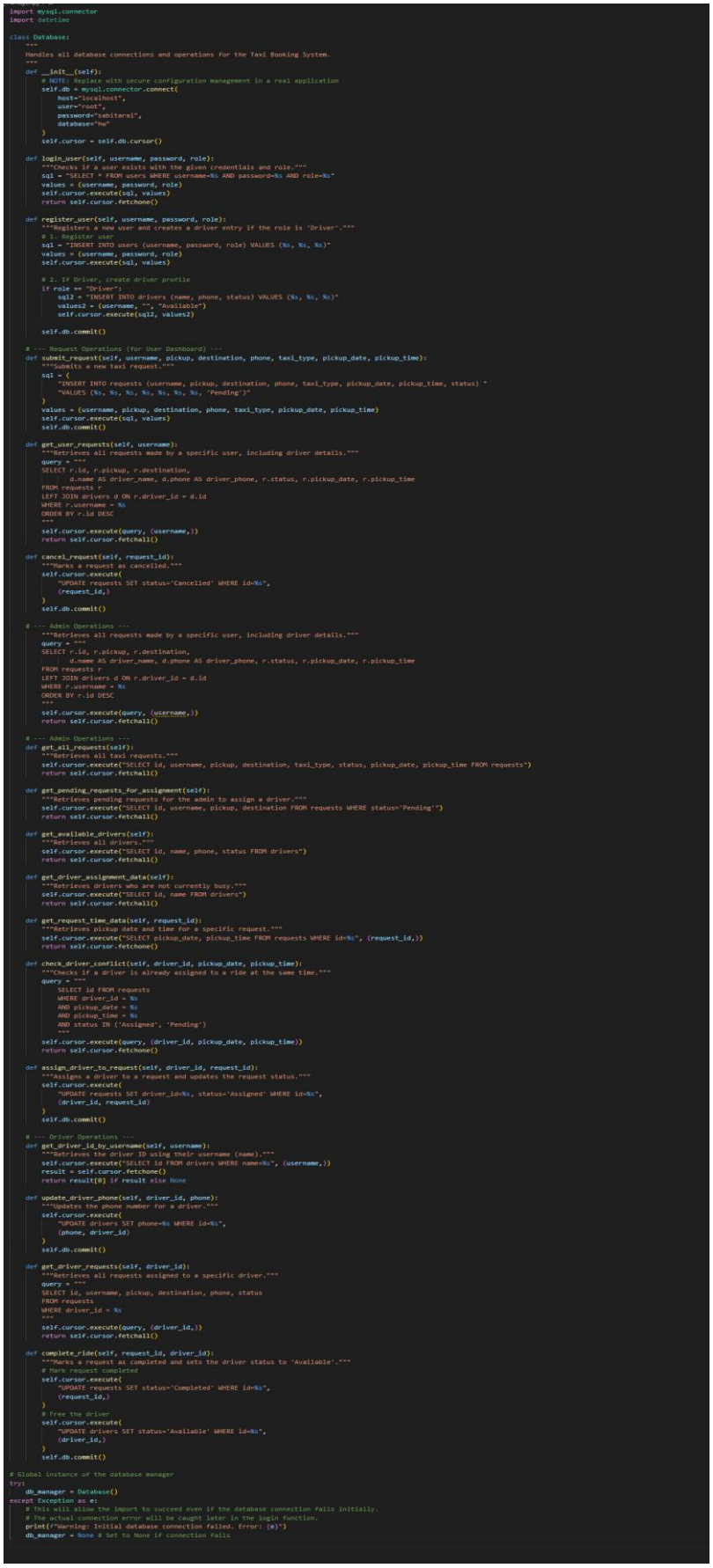
Driver_dashboard

```python
import tkinter as tk
from tkinter import ttk, messagebox
from gui_base import BaseWindow, TableView

class DriverDashboard(BaseWindow):
    """
    Handles the Driver Dashboard window, allowing drivers to view assigned rides
    and mark them as completed.
    """
    def __init__(self, username, parent, db_manager):
        # Removed parent_window.destroy() - now handled by BaseWindow.withdraw()
        super().__init__(parent=parent, title="Driver Dashboard - Taxi Booking System", geometry="850x550", bg="#ecf0f1")
        self.set_db_manager(db_manager)
        self.username = username
        self.window.resizable(True, True)

        self.setup_header(f"🚗 Driver Dashboard - {username}", "#e67e22", font_size=20)

        self.driver_id = self.db.get_driver_id_by_username(username)
        if not self.driver_id:
            messagebox.showerror("Error", "Driver not found in drivers table!")
            self.destroy()
            return

        # Scrollable Content Area
        self.scrollable_frame = self.setup_scrollable_frame(bg="#ecf0f1")

        self._setup_phone_update_section()
        self._setup_requests_table()

        self.load_requests()

    def _setup_phone_update_section(self):
        """Sets up the phone number update section."""
        phone_frame = tk.LabelFrame(self.scrollable_frame, text="Update Phone Number",
                                    font=("Segoe UI", 12, "bold"),
                                    bg="#ecf0f1", fg="#2c3e50", relief="solid", bd=1)
        phone_frame.pack(pady=10, padx=20, fill="x")

        content_frame = tk.Frame(phone_frame, bg="#ecf0f1")
        content_frame.pack(padx=10, pady=10, fill="x")

        tk.Label(content_frame, text="Phone:", font=("Segoe UI", 10),
                 bg="#ecf0f1", fg="#2c3e50").pack(side="left", padx=5)

        self.phone_entry = tk.Entry(content_frame, width=30, font=("Segoe UI", 10))
        self.phone_entry.pack(side="left", padx=5, ipady=5, fill="x", expand=True)

        tk.Button(content_frame, text="Save", command=self.save_phone,
                  bg="#3498db", fg="white", font=("Segoe UI", 10, "bold"),
                  relief="flat", cursor="hand2").pack(side="left", padx=10)

    def save_phone(self):
        """Handles the logic for saving the driver's phone number."""
        phone = self.phone_entry.get().strip()

        if phone == "":
            messagebox.showerror("Error", "Phone cannot be empty")
            return

        try:
            self.db.update_driver_phone(self.driver_id, phone)
            messagebox.showinfo("Success", "Phone number updated successfully")
        except Exception as e:
            messagebox.showerror("Database Error", f"Could not update phone: {e}")

    def _setup_requests_table(self):
        """Sets up the assigned requests table."""
        table_frame = tk.LabelFrame(self.scrollable_frame, text="Assigned Rides",
                                    font=("Segoe UI", 14, "bold"),
                                    bg="#ecf0f1", fg="#2c3e50", relief="solid", bd=1)
        table_frame.pack(pady=10, padx=20, fill="both", expand=True)

        table_container = tk.Frame(table_frame, bg="white")
        table_container.pack(padx=20, pady=20, fill="both", expand=True)

        columns = ("id", "customer", "pickup", "destination", "customer_phone", "status")
        self.drv_table_view = TableView(table_container, columns, height=12)
        self.drv_table_view.configure_column_widths({
            "id": 50, "customer": 100, "pickup": 150, "destination": 150,
            "customer_phone": 100, "status": 100
        })

        # Button Frame
        button_frame = tk.Frame(self.scrollable_frame, bg="#ecf0f1")
        button_frame.pack(pady=15)

        refresh_btn = tk.Button(button_frame, text="🔄 Refresh", font=("Segoe UI", 11, "bold"),
                                command=self.load_requests, bg="#3498db", fg="white",
                                relief="flat", cursor="hand2")
        refresh_btn.pack(side="left", padx=10, ipady=5, ipadx=10)

        complete_btn = tk.Button(button_frame, text="✓ Mark Ride Completed",
                                 font=("Segoe UI", 11, "bold"), bg="#27ae60", fg="white",
                                 relief="flat", cursor="hand2", command=self.complete_ride)
        complete_btn.pack(side="left", padx=10, ipady=5, ipadx=10)

    def load_requests(self):
        """Loads and displays all requests assigned to this driver."""
        try:
            rows = self.db.get_driver_requests(self.driver_id)
            self.drv_table_view.clear_table()
            self.drv_table_view.insert_rows(rows)
        except Exception as e:
            messagebox.showerror("Error", f"Could not load requests: {e}")

    def complete_ride(self):
        """Marks the selected ride as completed and frees the driver."""
        selected_data = self.drv_table_view.get_selected_item_data()
        if not selected_data:
            messagebox.showerror("Error", "Select a ride to complete")
            return

        req_id = selected_data[0]

        try:
            self.db.complete_ride(req_id, self.driver_id)
            messagebox.showinfo("Success", "Ride marked as Completed")
            self.load_requests()
        except Exception as e:
            messagebox.showerror("Database Error", f"Could not complete ride: {e}")
```

Admin dashboard

```
import tkinter as tk
from tkinter import ttk, messagebox
import datetime
from gui_base import BaseWindow, TableView

class AdminDashboard(BaseWindow):
    """
    Handles the Admin Dashboard window, including managing requests and drivers.
    """
    def __init__(self, parent, db_manager):
        # Removed parent_window.destroy() - now handled by BaseWindow.withdraw()
        super().__init__(parent=parent, title="Admin Dashboard - Taxi Booking System", geometry="1000x700", bg="#ecf0f1")
        self.set_db_manager(db_manager)
        self.window.resizable(True, True)

        self.setup_header("👑 Admin Dashboard", bg="#34495e", font_size=22)

        # Notebook with custom style
        style = ttk.Style()
        style.theme_use('default')
        style.configure("TNotebook", background="#ecf0f1", borderwidth=0)
        style.configure("TNotebook.Tab", font=("Segoe UI", 11, "bold"),
                        padding=[20, 10], background="#bdc3c7")
        style.map("TNotebook.Tab", background=[("selected", "#3498db")],
                  foreground=[("selected", "white")])

        self.notebook = ttk.Notebook(self.window)
        self.notebook.pack(expand=True, fill="both", padx=10, pady=10)

        self._setup_requests_tab()
        self._setup_drivers_tab()

        # Initial data load
        self.load_requests()
        self.load_drivers()
        self.load_assign_data()

    def _setup_requests_tab(self):
        """Sets up the Taxi Requests tab."""
        req_frame = tk.Frame(self.notebook, bg="white")
        self.notebook.add(req_frame, text="📋 Taxi Requests")

        # Scrollable content area for the requests tab
        self.req_scrollable_frame = self.setup_scrollable_frame(req_frame, bg="white")

        tk.Label(self.req_scrollable_frame, text="All Taxi Requests", font=("Segoe UI", 16, "bold"),
                 bg="white", fg="#2c3e50").pack(pady=15)

        # Table Frame
        table_frame = tk.Frame(self.req_scrollable_frame, bg="white")
        table_frame.pack(fill="x", padx=20, pady=10)

        columns = ("ID", "User", "Pickup", "Destination", "Taxi Type", "Status", "Date", "Time")
        self.req_table_view = TableView(table_frame, columns, height=10)
        self.req_table_view.configure_column_widths({
            "ID": 50, "User": 100, "Pickup": 130, "Destination": 150,
            "Taxi Type": 80, "Status": 80, "Date": 100, "Time": 80
        })

        refresh_req_btn = tk.Button(self.req_scrollable_frame, text="🔄 Refresh Requests", command=self.load_requests,
                                    bg="#3498db", fg="white", font=("Segoe UI", 11, "bold"),
                                    relief="flat", cursor="hand2")
        refresh_req_btn.pack(pady=10, ipady=4, ipadx=10)

        self._setup_assign_driver_section(self.req_scrollable_frame)

    def _setup_assign_driver_section(self, parent):
        """Sets up the driver assignment section."""
        assign_frame = tk.LabelFrame(parent, text="🔗 Assign Driver to Request",
                                     font=("Segoe UI", 13, "bold"), bg="#f0f0fa",
                                     fg="#2c3e50", relief="solid", bd=2, padx=20, pady=15)
        assign_frame.pack(fill="x", padx=20, pady=10)

        assign_frame.grid_columnconfigure(1, weight=1)

        tk.Label(assign_frame, text="Select Request:", font=("Segoe UI", 10, "bold"),
                 bg="#f0f0fa", fg="#2c3e50").grid(row=0, column=0, padx=10, pady=10, sticky="w")
        self.combo_request = ttk.Combobox(assign_frame, state="readonly", width=55,
                                          font=("Segoe UI", 10))
        self.combo_request.grid(row=0, column=1, padx=10, pady=10, sticky="ew")

        tk.Label(assign_frame, text="Select Driver:", font=("Segoe UI", 10, "bold"),
                 bg="#f0f0fa", fg="#2c3e50").grid(row=1, column=0, padx=10, pady=10, sticky="w")
        self.combo_driver = ttk.Combobox(assign_frame, state="readonly", width=55,
                                         font=("Segoe UI", 10))
        self.combo_driver.grid(row=1, column=1, padx=10, pady=10, sticky="ew")

        assign_btn = tk.Button(assign_frame, text="✔ Assign Driver",
                               bg="#27ae60", fg="white", font=("Segoe UI", 12, "bold"),
                               relief="flat", cursor="hand2", command=self.assign_driver)
        assign_btn.grid(row=2, column=0, columnspan=2, pady=15, ipady=5, ipadx=20)

    def _setup_drivers_tab(self):
        """Sets up the Drivers tab."""
        drv_frame = tk.Frame(self.notebook, bg="white")
        self.notebook.add(drv_frame, text="🚗 Drivers")

        tk.Label(drv_frame, text="All Registered Drivers", font=("Segoe UI", 16, "bold"),
                 bg="white", fg="#2c3e50").pack(pady=15)

        drv_table_frame = tk.Frame(drv_frame, bg="white")
        drv_table_frame.pack(fill="both", expand=True, padx=20, pady=10)

        drv_columns = ("ID", "Name", "Phone", "Status")
        self.drv_table_view = TableView(drv_table_frame, drv_columns, height=15)
        self.drv_table_view.configure_column_widths({
            "ID": 50, "Name": 200, "Phone": 200, "Status": 100
        })

        refresh_drv_btn = tk.Button(drv_frame, text="🔄 Refresh Drivers",
                                    command=self.load_drivers, bg="#3498db", fg="white",
                                    font=("Segoe UI", 11, "bold"), relief="flat",
                                    cursor="hand2")
        refresh_drv_btn.pack(pady=15, ipady=4, ipadx=10)

    def load_requests(self):
        """Loads and displays all taxi requests in the table."""
        try:
            rows = self.db.get_all_requests()
            self.req_table_view.clear_table()

            processed_rows = []
            for row in rows:
                row_list = list(row)
                # Format date and time objects for display
                if isinstance(row_list[6], datetime.date):
                    row_list[6] = row_list[6].strftime('%Y-%m-%d')
                if isinstance(row_list[7], datetime.time):
                    row_list[7] = str(row_list[7])
                processed_rows.append(row_list)

            self.req_table_view.insert_rows(processed_rows)
        except Exception as e:
            messagebox.showerror("Error", f"Could not load requests: {e}")

    def load_drivers(self):
        """Loads and displays all registered drivers in the table."""
        try:
            rows = self.db.get_available_drivers()
            self.drv_table_view.clear_table()
            self.drv_table_view.insert_rows(rows)
        except Exception as e:
            messagebox.showerror("Error", f"Could not load drivers: {e}")

    def load_assign_data(self):
        """Loads data for the request and driver assignment comboboxes."""
        try:
            # Load pending requests
            req = self.db.get_pending_requests_for_assignment()
            self.combo_request["values"] = [f"{r[0]} - {r[1]} ({r[2]} \u2192 {r[3]})" for r in req]

            # Load all drivers for assignment
            drv = self.db.get_driver_assignment_data()
            self.combo_driver["values"] = [f"{d[0]} - {d[1]}" for d in drv]

        except Exception as e:
            messagebox.showerror("Error", f"Could not load assignment data: {e}")

    def assign_driver(self):
        """Handles the logic for assigning a driver to a selected request."""
        if not self.combo_request.get() or not self.combo_driver.get():
            messagebox.showerror("Error", "Please select request and driver")
            return

        try:
            req_id = self.combo_request.get().split(" - ")[0]
            drv_id = self.combo_driver.get().split(" - ")[0]

            # 1. Check for time conflict
            request_time_data = self.db.get_request_time_data(req_id)
            if not request_time_data:
                messagebox.showerror("Error", "Request data not found.")
                return

            pickup_date, pickup_time = request_time_data

            conflict = self.db.check_driver_conflict(drv_id, pickup_date, pickup_time)

            if conflict:
                messagebox.showerror(
                    "Conflict Error",
                    f"Driver ID {drv_id} is already assigned to Request ID {conflict[0]} at {pickup_date} {pickup_time}."
                )
                return

            # 2. Assign driver
            self.db.assign_driver_to_request(drv_id, req_id)

            messagebox.showinfo("Success", "Driver Assigned Successfully")

            # 3. Refresh data
            self.load_assign_data()
            self.load_requests()
            self.load_drivers()

        except Exception as e:
            messagebox.showerror("Database Error", f"Assignment failed: {e}")
```

Db manager



```
import mysql.connector
import datetime

class Database:
    """
    Handles all database connections and operations for the Taxi Booking System.
    """
    def __init__(self):
        # NOTE: Replace with secure configuration management in a real application
        self.db = mysql.connector.connect(
            host="localhost",
            user="root",
            password="sabitaxi",
            database="hw"
        )
        self.cursor = self.db.cursor()

    def login_user(self, username, password, role):
        """Checks if a user exists with the given credentials and role."""
        sql = "SELECT * FROM users WHERE username=%s AND password=%s AND role=%s"
        values = (username, password, role)
        self.cursor.execute(sql, values)
        return self.cursor.fetchone()

    def register_user(self, username, password, role):
        """Registers a new user and creates a driver entry if the role is 'Driver'."""
        # 1. Register user
        sql = "INSERT INTO users (username, password, role) VALUES (%s, %s, %s)"
        values = (username, password, role)
        self.cursor.execute(sql, values)

        # 2. If Driver, create driver profile
        if role == "Driver":
            sql2 = "INSERT INTO drivers (name, phone, status) VALUES (%s, %s, %s)"
            values2 = (username, "", "Available")
            self.cursor.execute(sql2, values2)

        self.db.commit()

    # --- Request Operations (for User Dashboard) ---
    def submit_request(self, username, pickup, destination, phone, taxi_type, pickup_date, pickup_time):
        """Submits a new taxi request."""
        sql = (
            "INSERT INTO requests (username, pickup, destination, phone, taxi_type, pickup_date, pickup_time, status) "
            "VALUES (%s, %s, %s, %s, %s, %s, %s, 'Pending')"
        )
        values = (username, pickup, destination, phone, taxi_type, pickup_date, pickup_time)
        self.cursor.execute(sql, values)
        self.db.commit()

    def get_user_requests(self, username):
        """Retrieves all requests made by a specific user, including driver details."""
        query = """
        SELECT r.id, r.pickup, r.destination,
               d.name AS driver_name, d.phone AS driver_phone, r.status, r.pickup_date, r.pickup_time
        FROM requests r
        LEFT JOIN drivers d ON r.driver_id = d.id
        WHERE r.username = %s
        ORDER BY r.id DESC
        """
        self.cursor.execute(query, (username,))
        return self.cursor.fetchall()

    def cancel_request(self, request_id):
        """Marks a request as cancelled."""
        self.cursor.execute(
            "UPDATE requests SET status='Cancelled' WHERE id=%s",
            (request_id,)
        )
        self.db.commit()

    # --- Admin Operations ---
        """Retrieves all requests made by a specific user, including driver details."""
        query = """
        SELECT r.id, r.pickup, r.destination,
               d.name AS driver_name, d.phone AS driver_phone, r.status, r.pickup_date, r.pickup_time
        FROM requests r
        LEFT JOIN drivers d ON r.driver_id = d.id
        WHERE r.username = %s
        ORDER BY r.id DESC
        """
        self.cursor.execute(query, (username,))
        return self.cursor.fetchall()

    # --- Admin Operations ---
    def get_all_requests(self):
        """Retrieves all taxi requests."""
        self.cursor.execute("SELECT id, username, pickup, destination, taxi_type, status, pickup_date, pickup_time FROM requests")
        return self.cursor.fetchall()

    def get_pending_requests_for_assignment(self):
        """Retrieves pending requests for the admin to assign a driver."""
        self.cursor.execute("SELECT id, username, pickup, destination FROM requests WHERE status='Pending'")
        return self.cursor.fetchall()

    def get_available_drivers(self):
        """Retrieves all drivers."""
        self.cursor.execute("SELECT id, name, phone, status FROM drivers")
        return self.cursor.fetchall()

    def get_driver_assignment_data(self):
        """Retrieves drivers who are not currently busy."""
        self.cursor.execute("SELECT id, name FROM drivers")
        return self.cursor.fetchall()

    def get_request_time_data(self, request_id):
        """Retrieves pickup date and time for a specific request."""
        self.cursor.execute("SELECT pickup_date, pickup_time FROM requests WHERE id=%s", (request_id,))
        return self.cursor.fetchone()

    def check_driver_conflict(self, driver_id, pickup_date, pickup_time):
        """Checks if a driver is already assigned to a ride at the same time."""
        query = """
            SELECT id FROM requests
            WHERE driver_id = %s
            AND pickup_date = %s
            AND pickup_time = %s
            AND status IN ('Assigned', 'Pending')
            """
        self.cursor.execute(query, (driver_id, pickup_date, pickup_time))
        return self.cursor.fetchone()

    def assign_driver_to_request(self, driver_id, request_id):
        """Assigns a driver to a request and updates the request status."""
        self.cursor.execute(
            "UPDATE requests SET driver_id=%s, status='Assigned' WHERE id=%s",
            (driver_id, request_id)
        )
        self.db.commit()

    # --- Driver Operations ---
    def get_driver_id_by_username(self, username):
        """Retrieves the driver ID using their username (name)."""
        self.cursor.execute("SELECT id FROM drivers WHERE name=%s", (username,))
        result = self.cursor.fetchone()
        return result[0] if result else None

    def update_driver_phone(self, driver_id, phone):
        """Updates the phone number for a driver."""
        self.cursor.execute(
            "UPDATE drivers SET phone=%s WHERE id=%s",
            (phone, driver_id)
        )
        self.db.commit()

    def get_driver_requests(self, driver_id):
        """Retrieves all requests assigned to a specific driver."""
        query = """
        SELECT id, username, pickup, destination, phone, status
        FROM requests
        WHERE driver_id = %s
        """
        self.cursor.execute(query, (driver_id,))
        return self.cursor.fetchall()

    def complete_ride(self, request_id, driver_id):
        """Marks a request as completed and sets the driver status to 'Available'."""
        # Mark request completed
        self.cursor.execute(
            "UPDATE requests SET status='Completed' WHERE id=%s",
            (request_id,)
        )
        # Free the driver
        self.cursor.execute(
            "UPDATE drivers SET status='Available' WHERE id=%s",
            (driver_id,)
        )
        self.db.commit()

# Global instance of the database manager
try:
    db_manager = Database()
except Exception as e:
    # This will allow the import to succeed even if the database connection fails initially.
    # The actual connection error will be caught later in the login function.
    print(f"Warning: Initial database connection failed. Error: {e}")
    db_manager = None # Set to None if connection fails
```

Register window

```python
import tkinter as tk
from tkinter import ttk, messagebox
from gui_base import BaseWindow

class RegisterWindow(BaseWindow):
    """
    Handles the user registration window.
    """
    def __init__(self, parent, db_manager):
        super().__init__(parent=parent, title="Register - Taxi Booking System", geometry="600x750", bg="#f8f9fa")
        self.set_db_manager(db_manager)
        self.window.resizable(False, False)

        self.setup_header("Create Account", "#2c3e50", font_size=24)

        # Main Content Frame
        content_frame = tk.Frame(self.window, bg="#f8f9fa", padx=40, pady=20)
        content_frame.pack(fill="both", expand=True)

        self.reg_username = self.create_input_field(content_frame, "Username")
        self.reg_password = self.create_input_field(content_frame, "Password", is_password=True)

        self.role_box = self.create_combobox_field(
            content_frame,
            "Select Role",
            values=["User", "Admin", "Driver"],
            default_value="User"
        )

        # Override default button creation to use a specific pack configuration
        register_btn = tk.Button(content_frame, text="Register", width=25, command=self.register_now,
                                 font=("Segoe UI", 12, "bold"), bg="#27ae60", fg="white",
                                 relief="flat", cursor="hand2")
        register_btn.pack(pady=30, ipady=5)

    def register_now(self):
        """Handles the registration logic."""
        u = self.reg_username.get()
        p = self.reg_password.get()
        r = self.role_box.get()

        if u == "" or p == "":
            messagebox.showerror("Error", "All fields required")
            return

        try:
            self.db.register_user(u, p, r)
            messagebox.showinfo("Success", "Registration Successful")
            self.destroy()
        except Exception as e:
            messagebox.showerror("Database Error", f"Could not register user: {e}")
```

User dashboard

```
import tkinter as tk
from tkinter import ttk, messagebox
import datetime
from gui_base import BaseWindow, TableView

class UserDashboard(BaseWindow):
    """
    Handles the User Dashboard window, allowing users to book taxis and view history.
    """
    def __init__(self, username, parent, db_manager):
        # Removed parent.window.destroy() - now handled by BaseWindow.withdraw()
        super().__init__(parent=parent, title="User Dashboard - Taxi Booking System", geometry="700x750", bg="#ecf0f1")
        self.set_db_manager(db_manager)
        self.username = username
        self.window.resizable(True, True)

        self.setup_header(f"🚕 Welcome, {username}!", "#16a085", font_size=20)

        # Scrollable Content Area
        self.scrollable_frame = self.setup_scrollable_frame(bg="#ecf0f1")

        self._setup_request_form()
        self._setup_history_table()

        self.load_user_requests()

    def _setup_request_form(self):
        """Sets up the taxi booking request form."""
        form_frame = tk.LabelFrame(self.scrollable_frame, text="Book a Taxi", font=("Segoe UI", 14, "bold"),
                                   bg="white", fg="#2c3e50", relief="solid", bd=1)
        form_frame.pack(pady=10, padx=20, fill="x")

        content = tk.Frame(form_frame, bg="white")
        content.pack(padx=30, pady=20)

        # Helper function to create fields in the content frame
        def create_field(label_text, is_password=False):
            tk.Label(content, text=label_text, font=("Segoe UI", 10),
                     fg="#2c3e50", bg="white", anchor="w").pack(fill="x", pady=(10, 5))
            show_char = "*" if is_password else ""
            entry = tk.Entry(content, width=45, show=show_char, font=("Segoe UI", 11),
                             relief="solid", bd=1, highlightthickness=2,
                             highlightcolor="#16a085", highlightbackground="#dcdcdc")
            entry.pack(ipady=6)
            return entry

        self.entry_pickup = create_field("Pickup Location")
        self.entry_destination = create_field("Destination")
        self.entry_phone = create_field("Phone Number")

        tk.Label(content, text="Taxi Type", font=("Segoe UI", 10),
                 fg="#2c3e50", bg="white", anchor="w").pack(fill="x", pady=(10, 5))
        self.taxi_combo = ttk.Combobox(content, width=43, state="readonly",
                                       values=["Mini", "Sedan", "SUV", "Luxury"],
                                       font=("Segoe UI", 11))
        self.taxi_combo.pack(ipady=5)
        self.taxi_combo.set("Mini")

        self.entry_date = create_field("Pickup Date (YYYY-MM-DD)")
        self.entry_time = create_field("Pickup Time (HH:MM:SS)")

        submit_btn = tk.Button(content, text="🚕 Request Taxi", width=25, command=self.submit_request,
                               font=("Segoe UI", 12, "bold"), bg="#16a085", fg="white",
                               relief="flat", cursor="hand2")
        submit_btn.pack(pady=20, ipady=5)

    def _setup_history_table(self):
        """Sets up the request history table."""
        history_frame = tk.LabelFrame(self.scrollable_frame, text="My Assigned Rides",
                                      font=("Segoe UI", 14, "bold"),
                                      bg="white", fg="#2c3e50", relief="solid", bd=1)
        history_frame.pack(pady=20, padx=20, fill="both", expand=True)

        table_container = tk.Frame(history_frame, bg="white")
        table_container.pack(padx=15, pady=15, fill="both", expand=True)

        columns = ("id", "pickup", "destination", "driver", "phone", "status", "date", "time")
        self.req_table_view = TableView(table_container, columns, height=8)
        self.req_table_view.configure_column_widths({
            "id": 50, "pickup": 100, "destination": 100, "driver": 80,
            "phone": 80, "status": 80, "date": 80, "time": 80
        })

        refresh_btn = tk.Button(self.scrollable_frame, text="🔄 Refresh", width=20, command=self.load_user_requests,
                                font=("Segoe UI", 11, "bold"), bg="#3498db", fg="white",
                                relief="flat", cursor="hand2")
        refresh_btn.pack(side="left", padx=10, ipady=4, ipadx=10)

        cancel_btn = tk.Button(self.scrollable_frame, text="❌ Cancel Selected Ride", width=25, command=self.cancel_ride,
                               font=("Segoe UI", 11, "bold"), bg="#e74c3c", fg="white",
                               relief="flat", cursor="hand2")
        cancel_btn.pack(side="left", padx=10, ipady=4, ipadx=10)

    def submit_request(self):
        """Handles the logic for submitting a new taxi request."""
        pickup = self.entry_pickup.get()
        destination = self.entry_destination.get()
        phone = self.entry_phone.get()
        taxi_type = self.taxi_combo.get()
        pickup_date = self.entry_date.get()
        pickup_time = self.entry_time.get()

        if not all([pickup, destination, phone, pickup_date, pickup_time]):
            messagebox.showerror("Error", "All fields are required")
            return

        try:
            datetime.datetime.strptime(pickup_date, '%Y-%m-%d')
            datetime.datetime.strptime(pickup_time, '%H:%M:%S')
        except ValueError:
            messagebox.showerror("Error", "Invalid Date (YYYY-MM-DD) or Time (HH:MM:SS) format.")
            return

        try:
            self.db.submit_request(self.username, pickup, destination, phone, taxi_type, pickup_date, pickup_time)
            messagebox.showinfo("Success", "Taxi request submitted!")
            self.load_user_requests()
        except Exception as e:
            messagebox.showerror("Database Error", f"Could not submit request: {e}")

    def cancel_ride(self):
        """Handles the logic for cancelling a selected ride."""
        selected_data = self.req_table_view.get_selected_item_data()
        if not selected_data:
            messagebox.showerror("Error", "Please select a ride to cancel.")
            return

        req_id = selected_data[0]
        status = selected_data[5]

        if status in ["Completed", "Cancelled"]:
            messagebox.showwarning("Warning", f"Ride {req_id} is already {status} and cannot be cancelled.")
            return

        if not messagebox.askyesno("Confirm Cancellation", f"Are you sure you want to cancel ride {req_id}?"):
            return

        try:
            self.db.cancel_request(req_id)
            messagebox.showinfo("Success", f"Ride {req_id} has been successfully cancelled.")
            self.load_user_requests()
        except Exception as e:
            messagebox.showerror("Database Error", f"Could not cancel ride: {e}")

    def load_user_requests(self):
        """Loads and displays the user's request history."""
        try:
            rows = self.db.get_user_requests(self.username)
            self.req_table_view.clear_table()

            processed_rows = []
            for row in rows:
                row_list = list(row)
                # Format date and time objects for display
                if isinstance(row_list[6], datetime.date):
                    row_list[6] = row_list[6].strftime('%Y-%m-%d')
                if isinstance(row_list[7], datetime.time):
                    row_list[7] = str(row_list[7])
                processed_rows.append(row_list)

            self.req_table_view.insert_rows(processed_rows)
        except Exception as e:
            messagebox.showerror("Error", f"Could not load requests: {e}")
```