

**University of Bedfordshire**

CIS016-1 - Principles of Programming

Student id: **Krijen Shahi(2432413)**

Contents

[Introduction 3](#_Toc216969251)

[Objective 3](#_Toc216969252)

[Task Description 4](#_Toc216969253)

[Requirement Analysis 4](#_Toc216969254)

[Function Requirement 5](#_Toc216969255)

[Non-Functional Requirement 5](#_Toc216969256)

[Usability Requirements 6](#_Toc216969257)

[Use Case Diagrams 7](#_Toc216969258)

[Activity Diagrams 9](#_Toc216969259)

[Class Diagram 10](#_Toc216969260)

[Database Design ERM/ERD 11](#_Toc216969261)

[Physical Database Design 12](#_Toc216969262)

[Skeleton Tables 12](#_Toc216969263)

[Data Dictionary 13](#_Toc216969264)

[Implementation (OOP in the context of the project) 15](#_Toc216969265)

[Computational Thinking 16](#_Toc216969266)

[UI Design 17](#_Toc216969267)

[Testing 20](#_Toc216969268)

[Discussion/Reflection/Critical Analysis 28](#_Toc216969269)

[Taxi Booking System 28](#_Toc216969270)

[What Went Well 28](#_Toc216969271)

[What Went Wrong 29](#_Toc216969272)

[Future Improvements 29](#_Toc216969273)

[Learning Experience and Challenges 29](#_Toc216969274)

[Conclusion 30](#_Toc216969275)

[References 30](#_Toc216969276)

[Appendix 30](#_Toc216969277)

[Figure 1 Sea Level 7](#_Toc216969303)

[Figure 2 Fish level 8](#_Toc216969304)

[Figure 3 Activity Diagram 9](#_Toc216969305)

[Figure 4 Class Diagram 10](#_Toc216969306)

[Figure 5ERM Diagram\ 11](#_Toc216969307)

[Figure 6 ERD 11](#_Toc216969308)

[Figure 7Successfull 23](#_Toc216969309)

[Figure 8 Email already Exist 24](#_Toc216969310)

[Figure 9Driver Creation 24](#_Toc216969311)

[Figure 10 Booking Created 25](#_Toc216969312)

[Figure 11 User Name Validation 25](#_Toc216969313)

[Figure 12 Assign driver to booking 26](#_Toc216969314)

[Figure 13 Prevent double booking 26](#_Toc216969315)

[Figure 14 Booking canceled successful 27](#_Toc216969316)

[Figure 15 Cancel validation 27](#_Toc216969317)

[Figure 16 Logout validation 28](#_Toc216969318)

[Table 1 Functional Requirement 5](#_Toc216969320)

[Table 2 Non-Functional Requirement 5](#_Toc216969321)

[Table 3 Usability Requirement 6](#_Toc216969322)

[Table 4 Data Dictionary of Customer Table 13](#_Toc216969323)

[Table 5 Data Dictionary of Drivers table 13](#_Toc216969324)

[Table 6 Data Dictionary of Bookings Table 14](#_Toc216969325)

[Table 7 Data Dictionary of Users Table 14](#_Toc216969326)

# Introduction

Comparing this with some years back ride-sharing apps have revolutionized the archaic process of booking taxi and negotiating over fares. This changing trend forced us to create a reliable and easy-to-use taxi booking software. Our project was to develop a solution that provides an easy way for customers to book rides, drivers to manage their trips and the admin panel interfaces everything.

I have complete functioning system in Python, with a proper OOP concept and computational thinking. I have broken down the projects into clear, reuseable components , which help me in organizing the code-based structure and making it neat and clean. Systems support a multi role option for different user type (e.g., customers, drivers, and administrators) have different roles and features in the system.

We solved the problem step-by-step, breaking it down, simplifying complex parts, identifying what could work the same way in different places, and carefully planning each function. The result is a system that’s can manage and grow over time.

This report describes how the system has been designed and built, how the test cases are done, and what I have learned along the way including the hurdles I faced and how I overcome through them.

# Objective

The main objective of this project was to design and build a well performing taxi booking system which supports three different types of users i.e customer, admin and driver. I have made sure that the system worked well for all three user types that are customers, drivers, and administrators, each with their own responsibilities.

One of my main objectives was to build a well functional, role-based system. Using Python and object-oriented programming, I have developed a software where each user type has a different types of roles and features which helped in keeping the system organized and made it easier to understand how different users interact with it.

I have also worked on applying problem-solving and computational thinking throughout the development process which involved breaking the system into smaller parts, manageable parts, like separating user management from ride-booking logic. I reused common patterns, where possible (e.g., having the same login structure for all users), I also tried to hide some of the complex processing that was needed, like password security, in order to keep main code clean and readable.  
Lastly, I focused on the system being user-friendly and easier to operate. I tested the general functionality of the system, which includes user registration, secure login, booking of rides, and driver assignment to ensure a good user experience. Overall, the aim was to deliver a seamless and reliable experience for all those using the system.

# Task Description

The project requirement was to create a taxi booking with different three user types with their respective roles

- Customer Functions: Can Register their details, book a taxi, view, and cancel bookings.  
- Driver Functions: Log in to view assigned trips.  
- Administrator Functions: Log in, view all bookings, assign drivers to customer bookings, and prevent overlapping bookings.

The system is to be implemented in Python using a text-based interface; hence, the users will interact with the system through the selection of options from a menu. The project also involves the use of data persistence using text files: data for customers, trips, and driver assignments will be stored. Optional functionality includes GUI for enhanced user interaction.

# Requirement Analysis

As per our requirement to develop taxi booking system. I have listed down functional and non-functional requirement below.

## Function Requirement

Table 1 Functional Requirement

|  |  |  |
| --- | --- | --- |
| **Req No** | **Requirement** | **Priority** |
| FR1 | Customer must be able to register by providing their basic details like name, address, phone number and email address | MUST |
| FR2 | Customer must be able to book a taxi by entering their pickup location, drop-off location with date and time. | MUST |
| FR3 | Customer must be able to view their booking and cancel them as well. | MUST |
| FR4 | Driver must be able to login and see the details of their assigned rides. | MUST |
| FR5 | Admin user must be able to login and view all the bookings, assign the free drives to a customer booking. | MUST |
| FR6 | Admin must ensure that bookings assignment is not overlapped to drivers. | MUST |
| FR7 | Admin should be able to view and manage customer and trip data. | SHOULD |
| FR8 | The system should support the creation and editing of customer basic info. | SHOULD |
| FR9 | The system should provide summary of bookings to the customer. | SHOULD |
| FR10 | Driver should be able to confirm or reject the assigned trips. | SHOULD |

## Non-Functional Requirement

Table 2 Non-Functional Requirement

|  |  |  |
| --- | --- | --- |
| **Req No** | **Requirement** | **Priority** |
| NFR1 | The system must load user data within 2 seconds of startup. | MUST |
| NFR2 | The system must securely store user data, using encryption for sensitive information such as passwords. | MUST |
| NFR3 | The system must be able to handle at least 1000 active users without performance degradation. | SHOULD |
| NFR4 | The system must provide clear and user-friendly feedback in case of invalid inputs. | MUST |
| NFR5 | The system should have a simple, text-based interface that is easy to navigate for all user types. | MUST |

# Usability Requirements

Table 3 Usability Requirement

|  |  |  |
| --- | --- | --- |
| **Req No.** | **Requirement** | **Priority\*** |
| 1 | The interface should follow a consistent design across all screens. | MUST |
| 2 | Booking forms should be logically organized and easy to complete. | MUST |
| 3 | The system should provide clear navigation between customer, driver, and admin functions. | MUST |
| 4 | Text, labels, and messages should be clear, concise, and free of spelling or grammatical errors | MUST |
| 5 | Color schemes should be appropriate for a taxi booking application | SHOULD |
| 6 | Important actions (book ride, cancel booking, assign driver) should be clearly distinguishable | MUST |
| 7 | The system should provide feedback for user actions (success, error, or warning messages) | SHOULD |
| 8 | Data entry fields should include validation where appropriate (login, booking details, registration) | MUST |
| 9 | The layout should remain consistent across different screen sizes and resolutions | SHOULD |

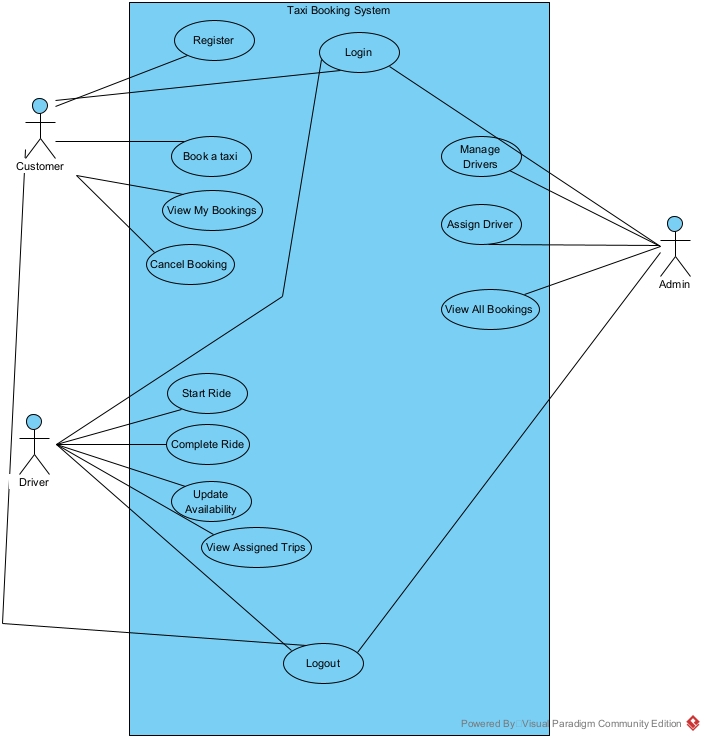
Use Case Diagrams

Figure 1 Sea Level

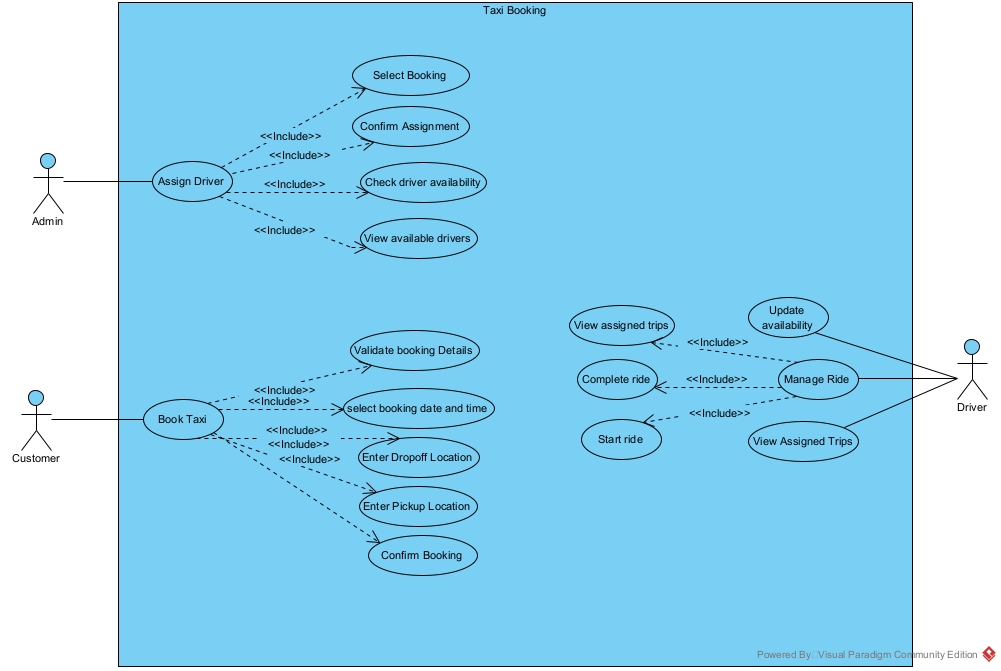


Figure 2 Fish level

Table 4 Calm level

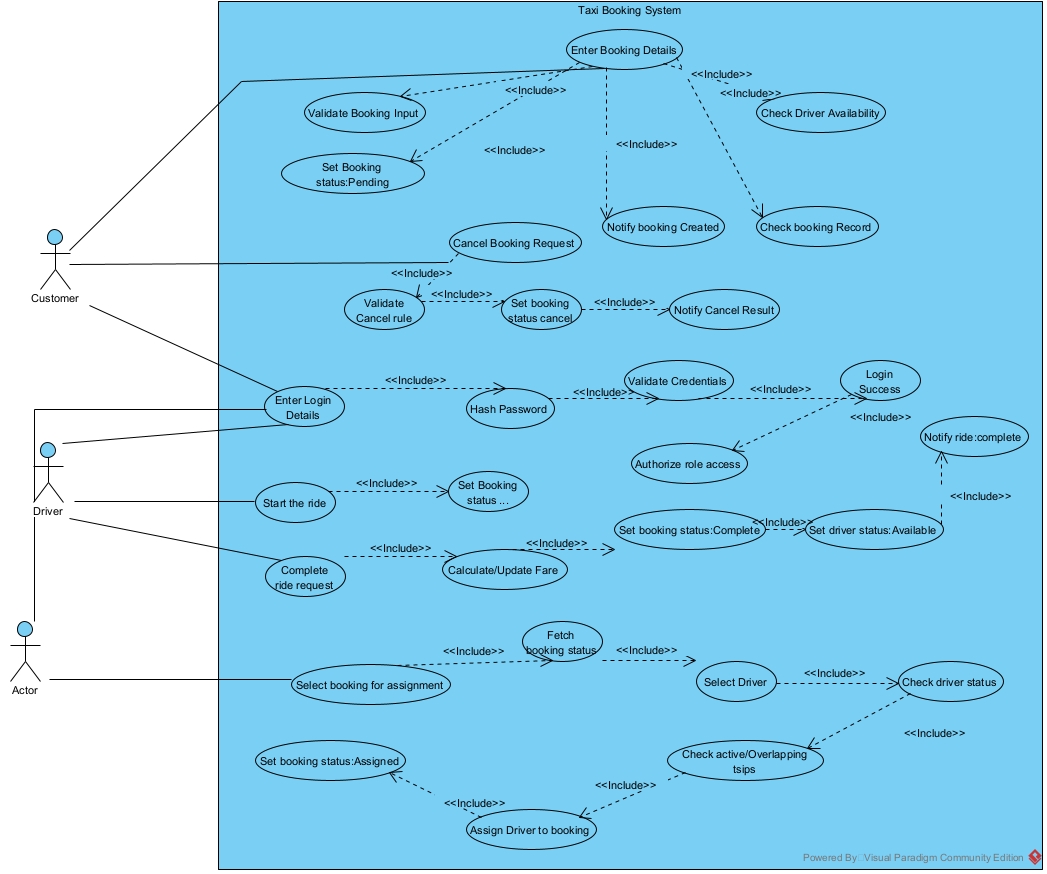


Figure 3 Calm level Use Case Diagram

# Activity Diagrams

Figure 4 Activity Diagram

# Class Diagram

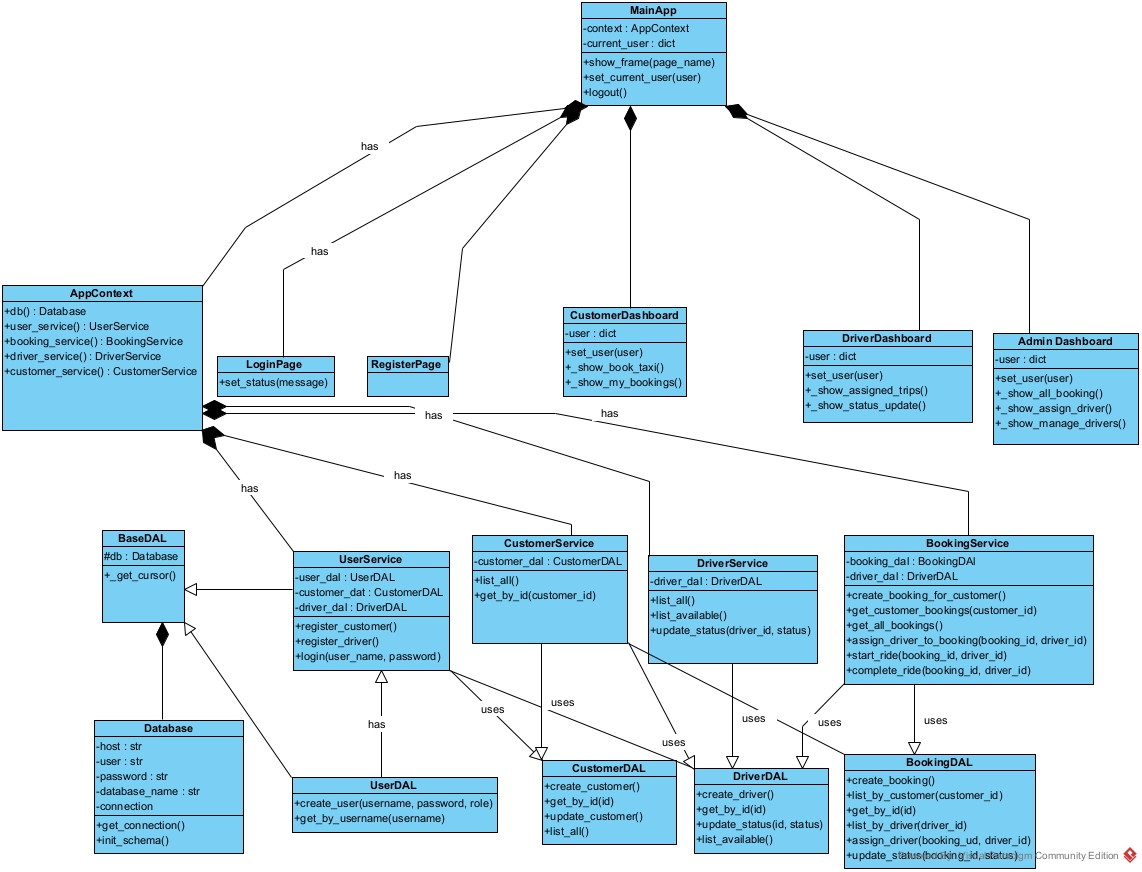


Figure 5 Class Diagram

# Database Design ERM/ERD

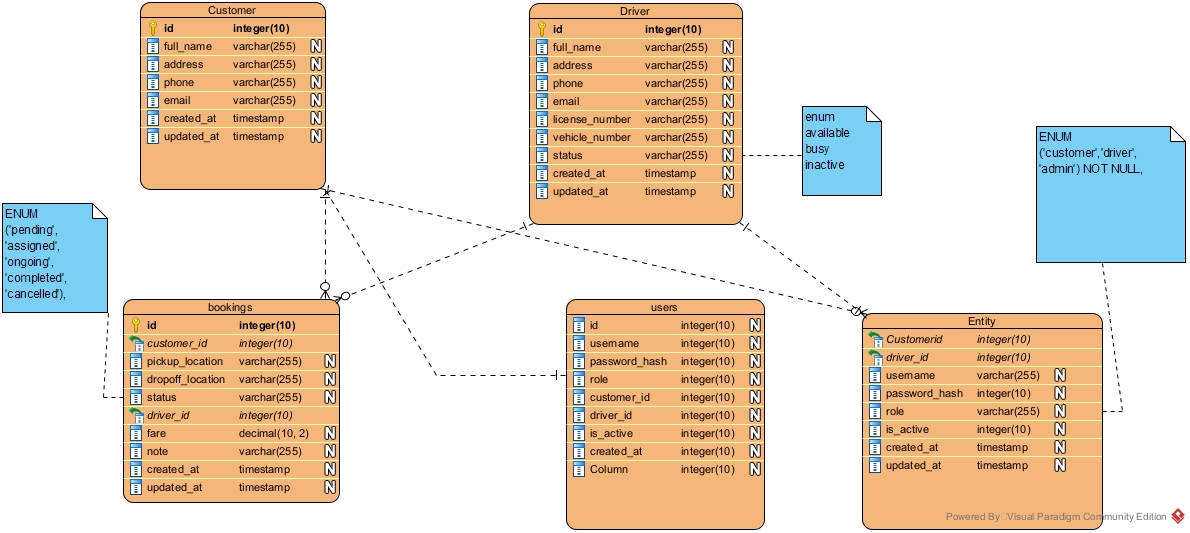


Figure 6ERM Diagram\

# 

Figure 7 ERD

# Physical Database Design

## Skeleton Tables

• **customers**

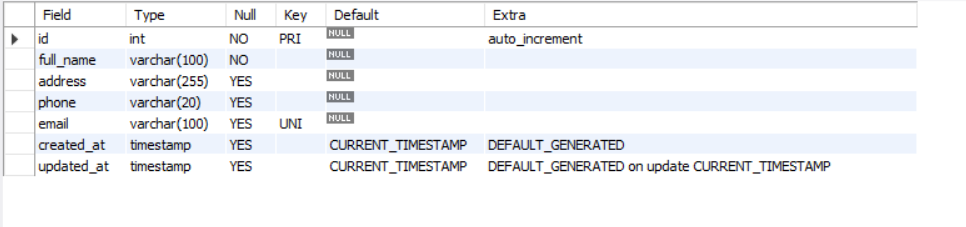


Figure 8 Customer Table

**Customer Table**: - (customer\_id, full\_name, address, phone, email, created\_at, updated\_at)

• **drivers**  
(driver\_id, full\_name, address, phone, email, license\_number, vehicle\_number, status, created\_at, updated\_at)

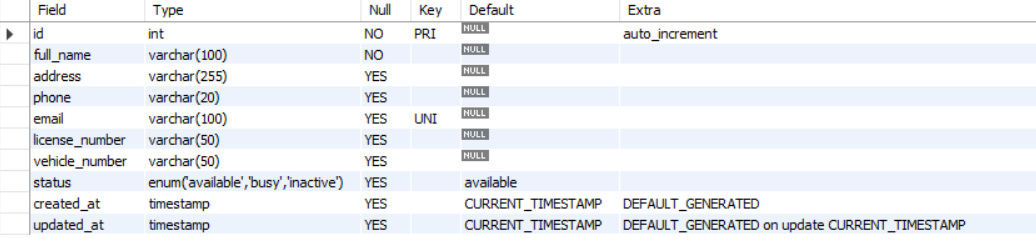


Figure 9 Driver Table

• **bookings**

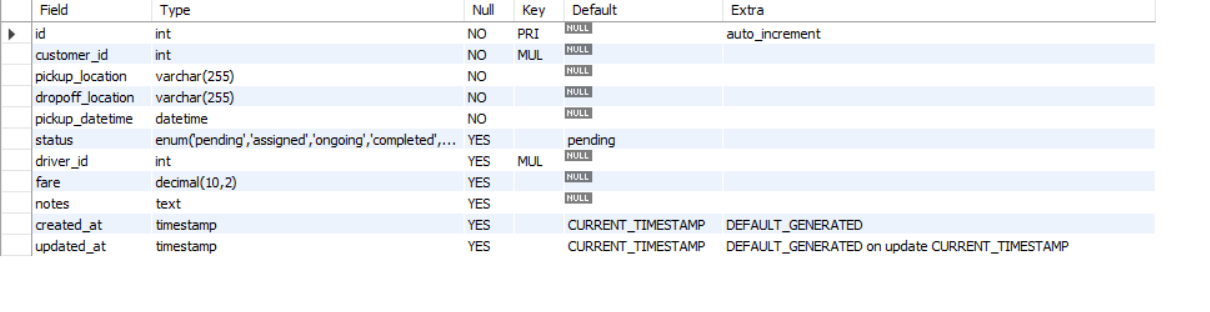


Figure 10 Bookings table

**Booking Table**:-(booking\_id, customer\_id\*, driver\_id\*, pickup\_location, dropoff\_location, pickup\_datetime, status, fare, notes, created\_at, updated\_at)

• **users**

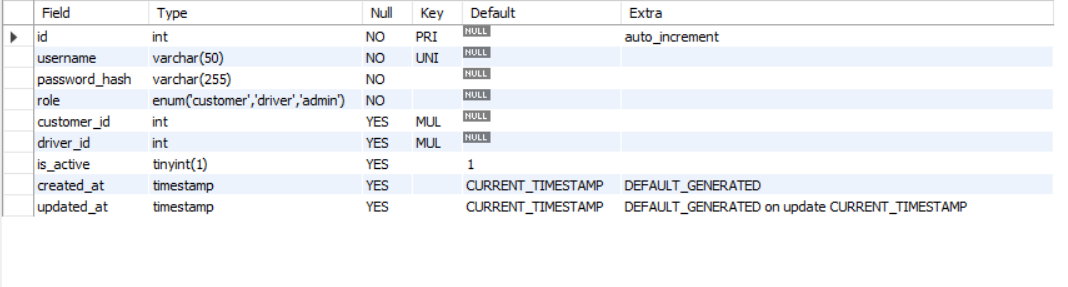


Figure 11 Users table

**User Table**:-(user\_id, username, password\_hash, role, customer\_id\*, driver\_id\*, is\_active, created\_at, updated\_at)

## Data Dictionary

Table 5 Data Dictionary of Customer Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Customers** | | | | | | | |
| **Description: This table stores customer information used for taxi bookings.** | | | | | | | |
| Field Name | Data Type | Length | Index | Null | Default value | Validation Rule | Description |
| customer\_id | Int |  | PK | No | Auto\_increment |  | Unique customer identifier |
| full\_name | Varchar | 100 |  | Yes |  |  | Customer full name |
| address | Varchar | 255 |  | Yes |  |  | Customer address |
| phone | Varchar | 20 |  | Yes |  |  | Customer contact |
| Email | Varchar | 100 | UQ | Yes |  |  | Customer email address |
| created\_at | timestamp |  |  | Yes | CURRENT\_ TIMESTAMP |  | Record creation date |
| updated\_at | timestamp |  |  |  | CURRENT\_ TIMESTAMP |  | Record creation date |

Table 6 Data Dictionary of Drivers table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Drivers** | | | | | | | |
| **Description: This table stores driver details and availability status.** | | | | | | | |
| Field Name | Data Type | Length | Index | Null | Default value | Validation Rule | Description |
| driver\_id | Int |  | PK | No | Auto\_increment |  | Unique driver identifier |
| full\_name | Varchar | 100 |  | Yes |  |  | Driver full name |
| address | Varchar | 255 |  | Yes |  |  | Driver address |
| phone | Varchar | 20 |  | Yes |  |  | Driver contact |
| email | Varchar | 100 | UQ | Yes |  |  | Driver email address |
| license\_number | Varchar | 50 |  | Yes |  |  | Driver License number |
| Vehicle\_number | Varchar | 50 |  | Yes |  |  | Vehicle registration number |
| Status | Enum |  |  | Yes | Available | Available, busy, inactive | Driver availability status |
| Created\_at | Timestamp |  |  | Yes | CURRENT\_TIMESTAMP |  | Record creation date |
| Updated\_at | Timestamp |  |  | Yes | CURRENT\_TIMESTAMP |  | Record update date |

Table 7 Data Dictionary of Bookings Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bookings** | | | | | | | |
| **Description: This table stores taxi booking details made by customers.** | | | | | | | |
| Field Name | Data Type | Length | Index | Null | Default value | Validation Rule | Description |
| Booking\_id | Int |  | PK | No | Auto\_increment |  | Unique booking identifier |
| Customer\_id | Int |  | FK | No |  | Must existing | Customer making the booking |
| address | Varchar | 255 |  | Yes |  |  | Driver address |
| phone | Varchar | 20 |  | Yes |  |  | Driver contact |
| email | Varchar | 100 | UQ | Yes |  |  | Driver email address |
| license\_number | Varchar | 50 |  | Yes |  |  | Driver License number |
| Vehicle\_number | Varchar | 50 |  | Yes |  |  | Vehicle registration number |
| Status | Enum |  |  | Yes | Available | Available, busy, inactive | Driver availability status |
| Created\_at | Timestamp |  |  | Yes | CURRENT\_TIMESTAMP |  | Record creation date |
| Updated\_at | Timestamp |  |  | Yes | CURRENT\_TIMESTAMP |  | Record update date |

Table 8 Data Dictionary of Users Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Users** | | | | | | | |
| **Description: This table stores system login credentials for customers, drivers, and administrators.** | | | | | | | |
| Field Name | Data Type | Length | Index | Null | Default value | Validation Rule | Description |
| user\_id | Int |  | PK | No | Auto\_ increment |  | Unique user identifier |
| username | Varchar | 50 | UQ | No |  | Unique | Login Username |
| password\_ hash | Varchar | 255 |  | No |  | Encrypted | User Password |
| role | Enum | 20 |  | No |  | Customer, driver, admin | User role |
| customer\_id | int |  | FK | Yes |  | References Customer | Linked customer id |
| driver\_id | int |  | FK | Yes |  | References Driver | Linked driver id |
| is\_active | Boolean |  |  | Yes | 1 |  | Account Status |
| created\_at | Enum |  |  | Yes | CURRENT\_ TIMESTAMP |  | Record creation date |
| updated\_at | Timestamp |  |  | Yes | CURRENT\_ TIMESTAMP |  | Record update date |

# Implementation (OOP in the context of the project)

The Taxi Booking System is implemented using an object-oriented, layered architecture that separates database access, business logic, and user interface concerns. This structure improves code maintainability, reusability, and clarity.

The system is divided into four layers: Data Access Layer (DAL), Business Logic (Service) Layer, User Interface (UI) Layer, and an Application Context for shared resources.

The Data Access Layer encapsulates all database interactions. A central Database class manages the MySQL connection and schema initialization. Entity-specific classes such as UserDAL, CustomerDAL, DriverDAL, and BookingDAL inherit from a common BaseDAL, allowing reuse of cursor-handling logic. This demonstrates encapsulation and inheritance, while abstracting raw SQL from higher layers.

The Service Layer implements business rules and workflows. Classes such as UserService and BookingService coordinate multiple DAL operations to perform tasks like user authentication, booking creation, driver assignment, and ride lifecycle management. These services provide high-level abstractions to the UI and encapsulate validation and rule enforcement logic, illustrating abstraction and composition.

The UI Layer is implemented using Tkinter, where each screen (e.g., login, customer dashboard, driver dashboard, admin dashboard) is represented as a class inheriting from tk. Frame. The MainApp class controls screen navigation and session state. This use of inheritance and role-based screen behavior demonstrates polymorphism and encapsulation in the presentation layer.

The AppContext class initializes and shares the database and service objects across the application, simplifying dependency management and ensuring consistent access to core resources.

Overall, the system effectively applies encapsulation, abstraction, inheritance, and polymorphism to achieve a clean separation of concerns. This object-oriented design makes the Taxi Booking System easy to understand, maintain, and extend with additional features.

# Computational Thinking

**Computational thinking** is the mental skill set used to solve problems in a way that can be carried out by a computer or, more broadly, to understand and structure complex processes logically. (Wing, 2006). Computational Thinking is used throughout the project of Taxi Booking System. This system indicates all four aspects that should be included in Computational Thinking.

Computational thinking is used in the Taxi Booking System to build and execute an organized and effective resolution. The system illustrates the main concepts of computational thinking of decomposition, abstraction, pattern recognition, and algorithmic thinking.

This is done through decomposition in conjunction with breaking the system into smaller parts and layers such as Data Access Layer, Service Layer, and User Interface Layer. The major functions, user management, booking management, and driver assignment, are managed by specific classes and methods, and it is easier to comprehend and maintain the system.

Each level has implementation details that are concealed through abstraction. The operations of the database are abstracted to DAL classes, the business rules are managed by service classes and the user interface interacts with only higher-level service methods. This eliminates complexity and maintains the separation of responsibilities.

Pattern recognition is also applied in the similar structure used in the system i.e. CRUD operations in the DAL classes, validation patterns in services, and role-based behavior in the customer, driver, and administrator user interface.

The use of algorithmic thinking is observed by the use of clear, step-by-step reasoning in user authentication, as well as the formation of a booking, assigning a driver, and completing a ride. Every process has a set of steps that must be properly and accurately followed to implement it.

# UI Design

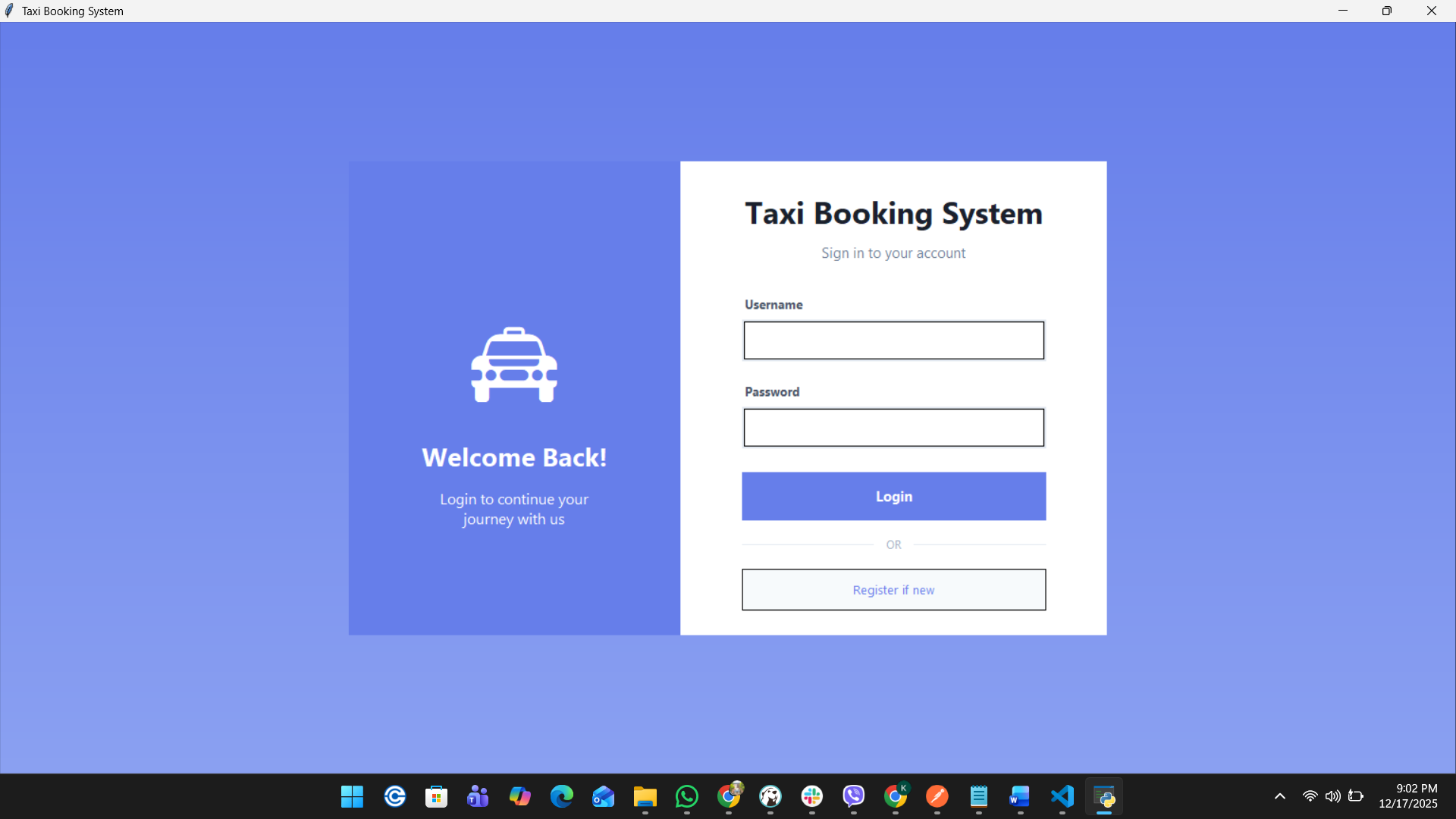


Figure 12 Login UI

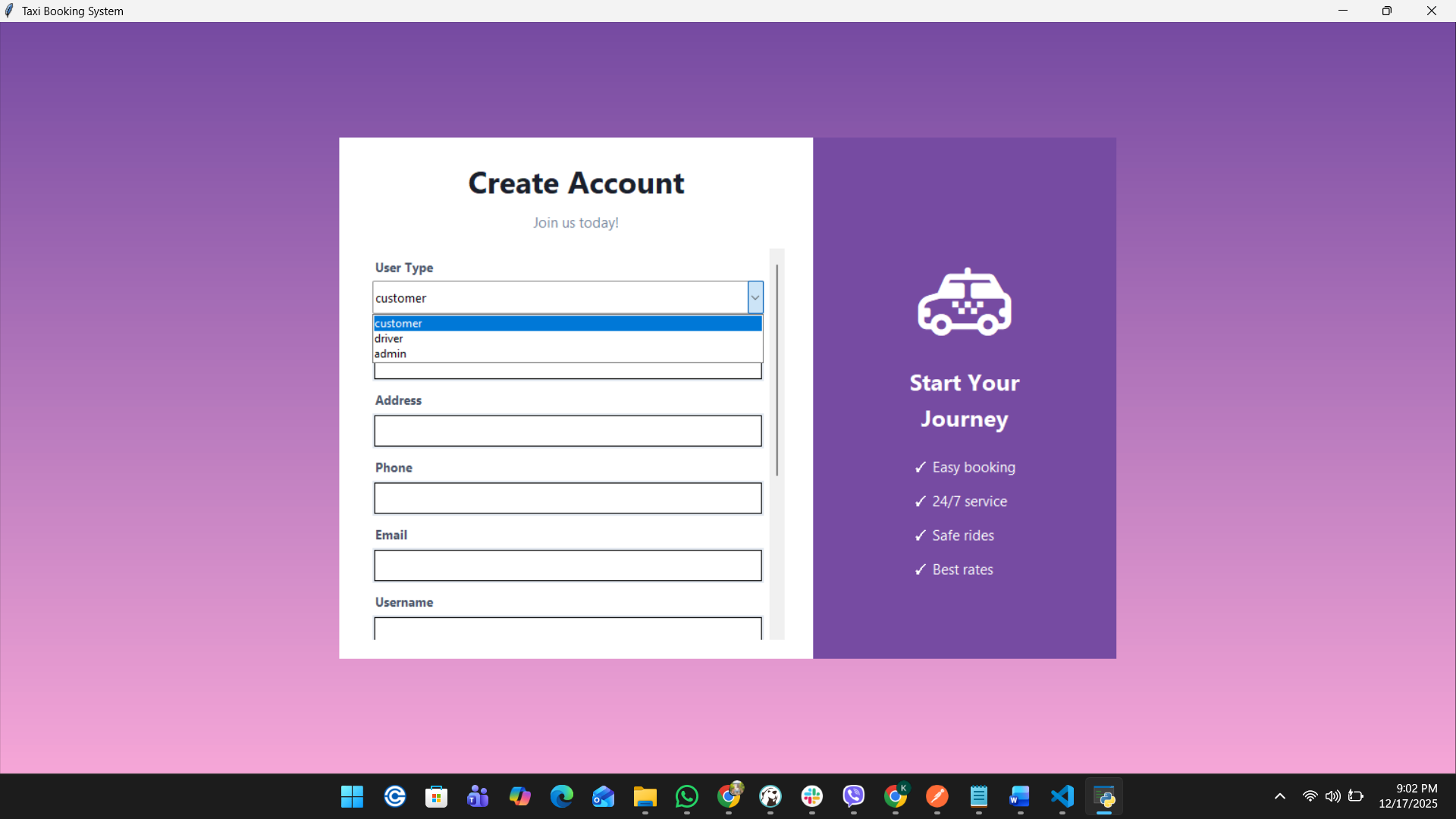


Figure 13 Registration Page UI

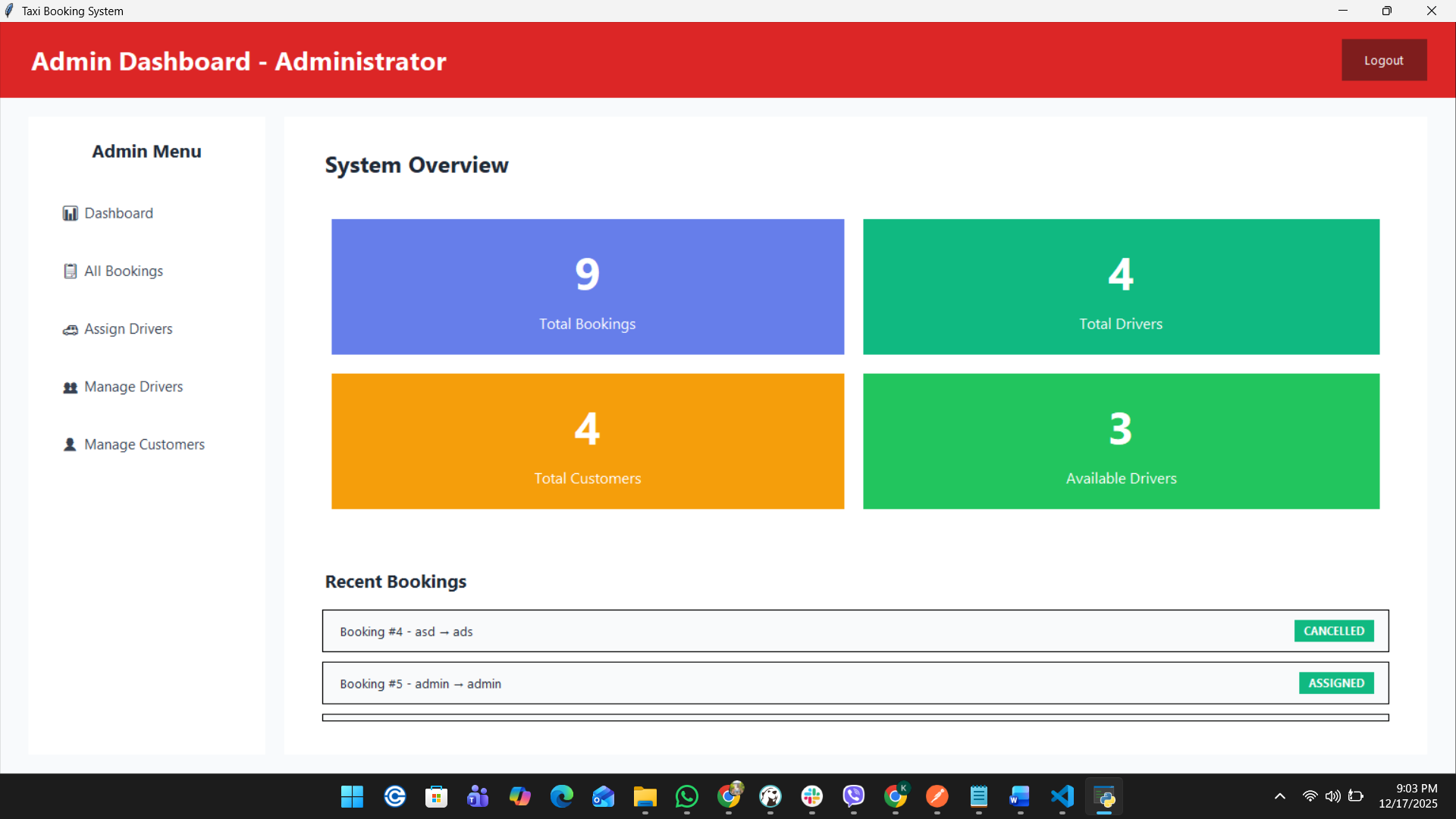


Figure 14 Admin Dashboard UI

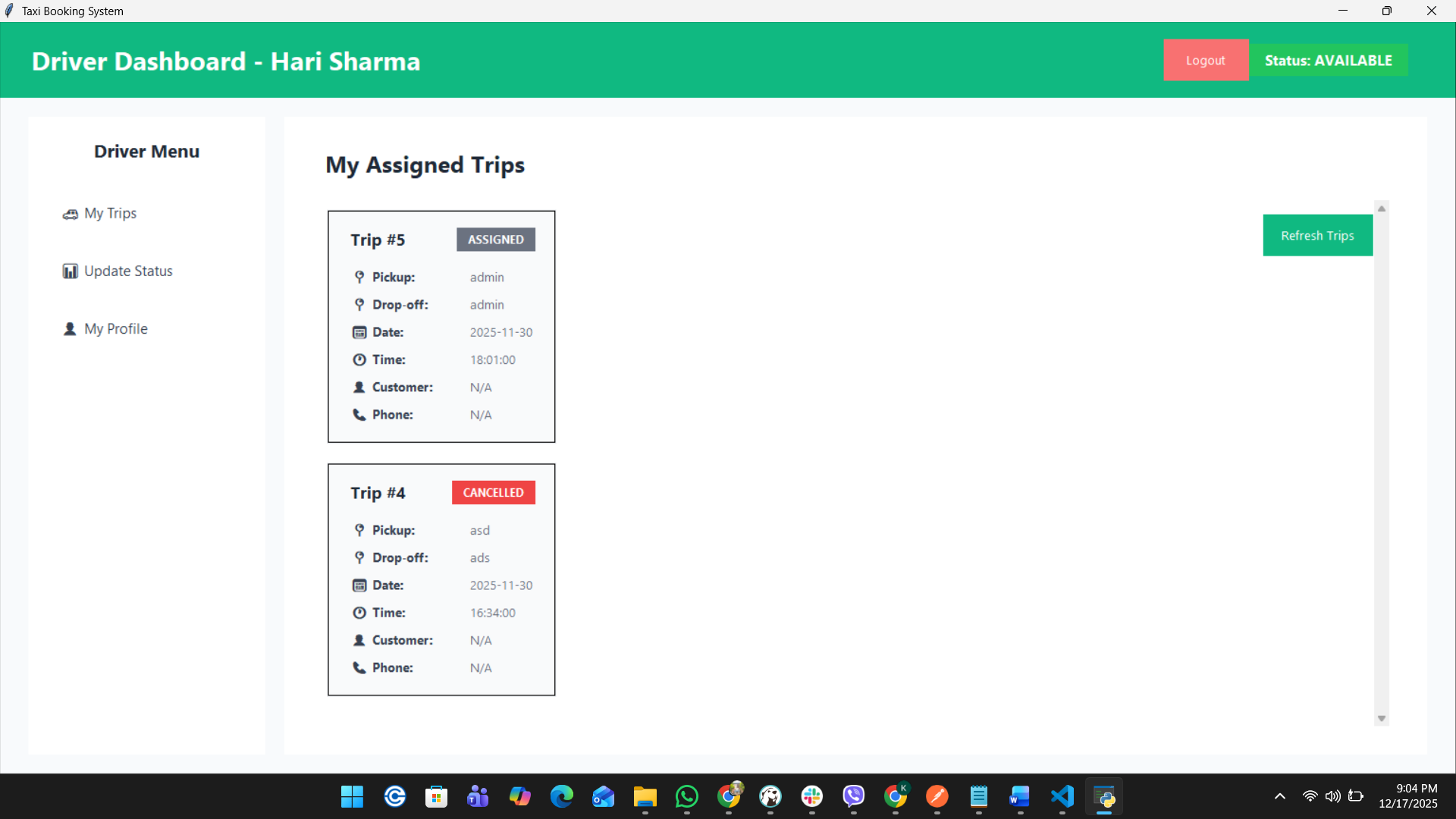


Figure 15 Driver Dashboard UI

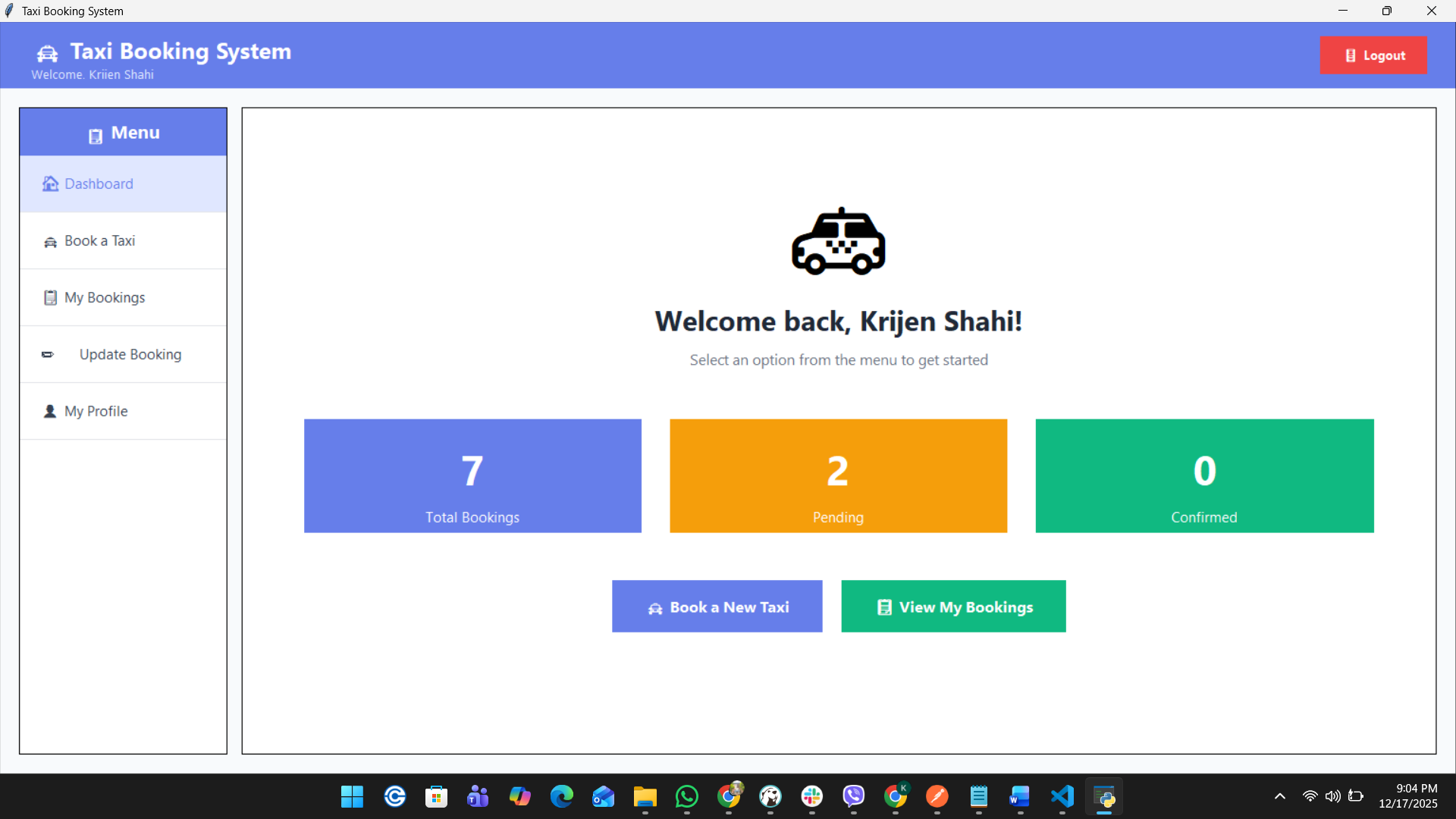


Figure 16 Customer Dashboard UI

# Testing

Table 9 Test Cases

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Date | Purpose of Testing | Input Data | Expected Output | Actual  Output | Pass/Fail | Screenshot |
| T-1 | Dec-14 | Verify admin login functionality | Username: admin  Password: admin | Redirect to admin portal | Dashboard loaded successfully | Pass |  |
| T-2 | Dec-14 | Test invalid login rejection | Username: admin Password: Test | Error message “Invalid username or password” | Error message displayed | Pass |  |
| T-3 | Dec-14 | Verify customer registration | Full Name: Suman Thapa  Address: Nepal Phone: 9860165286 Email: [suman1@gmail.com](mailto:suman1@gmail.com) Username: suman1 Password: Suman | Registration successful as a customer | Customer created successfully | Pass |  |
| T-4 | Dec-14 | Validate already email exists | Full Name: Suman Thapa  Address: Nepal Phone: 9860165286 Email: [suman@gmail.com](mailto:suman@gmail.com) Username: suman Password: Suman | A customer with this email already exists | Validated successfully | Pass |  |
| T-5 | Dec-14 | Validate user name exists | Full Name: Suman Thapa  Address: Nepal Phone: 9860165286 Email: [suman12@gmail.com](mailto:suman12@gmail.com) Username: suman Password: Suman | User name is already taken | Validated successfully | Pass |  |
| T-6 | Dec-14 | Verify driver registration | User Type:  Driver  Full Name: Jivan Gurung  Address: Satdobato  Phone: 9860827827 Email: [jivangurung@gmail.com](mailto:jivangurung@gmail.com) License Number: 12399812 Vehicle Number: 89132 Username:  jivan | Driver created successfully | Registration successful as driver. | Pass |  |
| T-7 | Dec-14 | Create taxi booking | Pickup Location:  Satdobato Drop-off Location: Thamel Pickup Date: 2025-12-17 | Booking created successfully | Booking successfully created | Pass |  |
| T-8 | Dec-14 |  | Validate booking without entry of mandatory field | Validated successfully | All fields are required. | Pass |  |
| T-9 | Dec-14 | Assign driver to booking (Admin) | Booking Id: 13 Driver: Jivan Gurung (Vehicle no 89132) | Driver assigned successfully | Assignment successful | Pass |  |
| T-10 | Dec-14 | Prevent driver double booking | Booking ID: 11  Driver: Jivan Gurung (Vehicle no 89132) | Driver has already active booking | Failed to assign driver: This driver already has an active booking. They must complete the current ride before a new one can be assigned. | Pass |  |
| T-11 | Dec-14 | Cancel booking by customer | Booking id 8 | Booking cancelled successfully | Booking cancelled | Pass |  |
| T-12 | Dec-14 | Cancel the ride with cancel status | Booking id 6 | Validated successfully | Failed to cancel booking: Cannot cancel a booking with status ‘Cancelled’ | Pass |  |
| T-13 | Dec-14 | Validate logout functionality | Click Logout | Redirect to Login Page | Login page displayed | Pass |  |

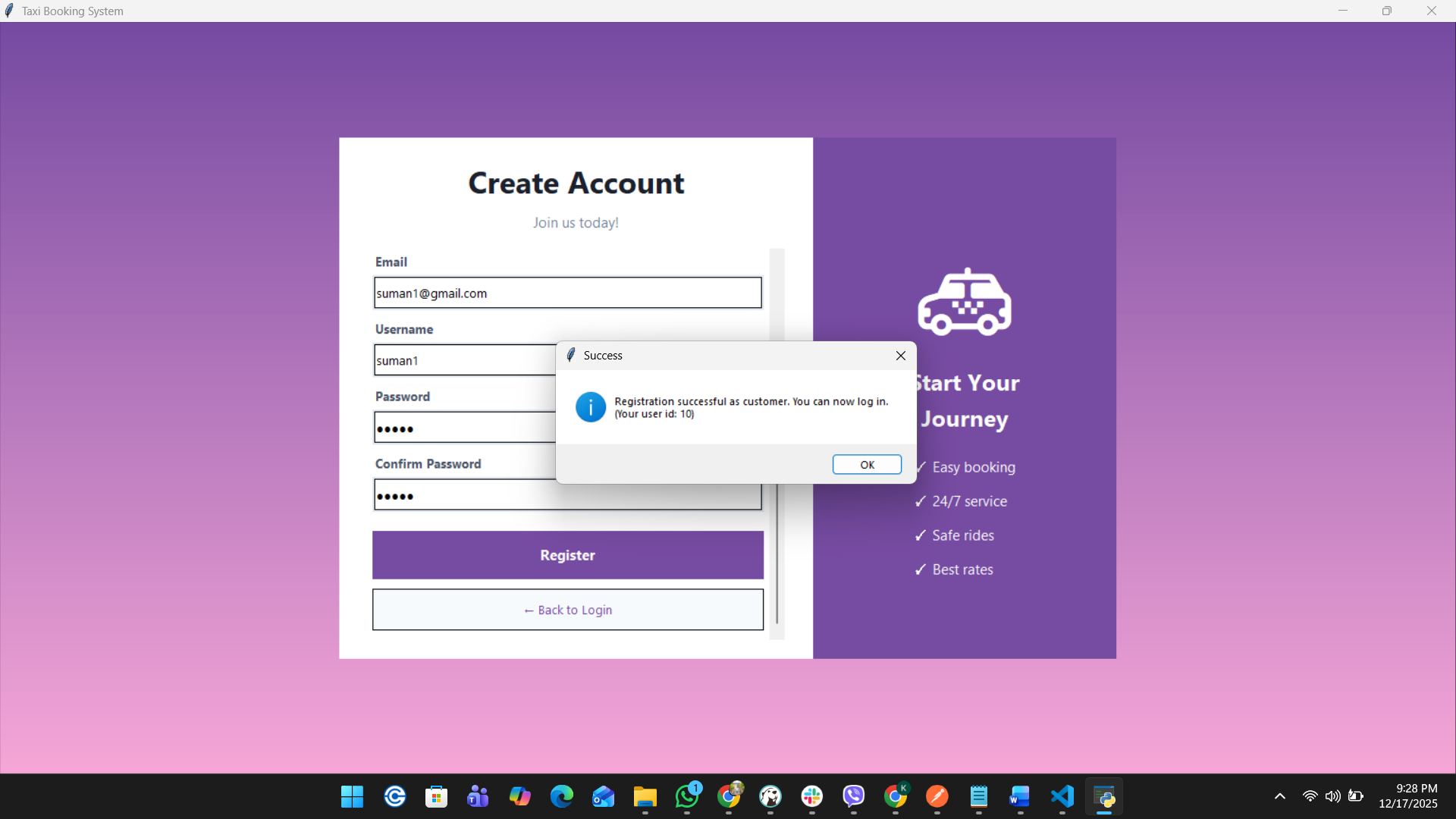
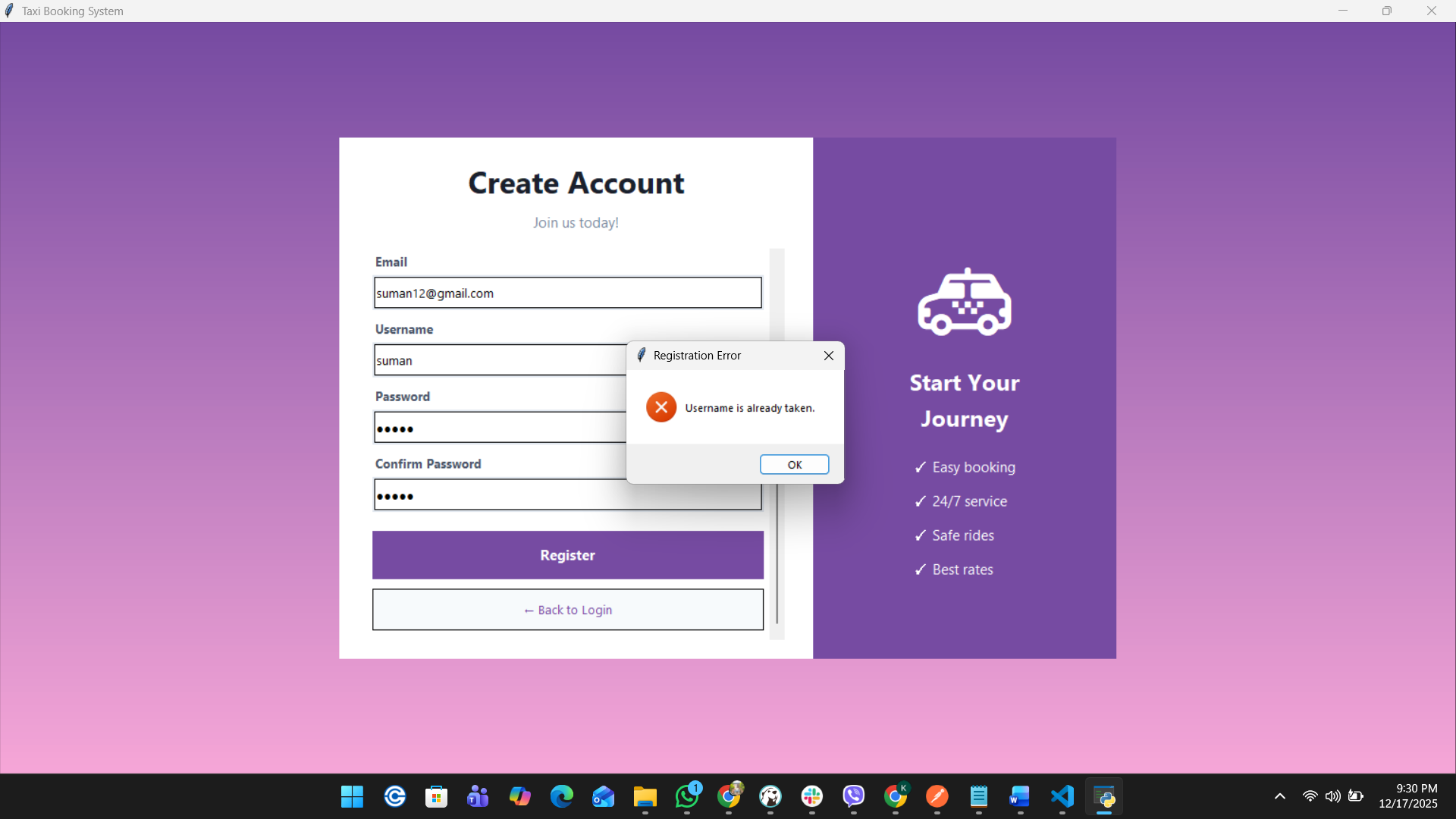


Figure 17Successfull



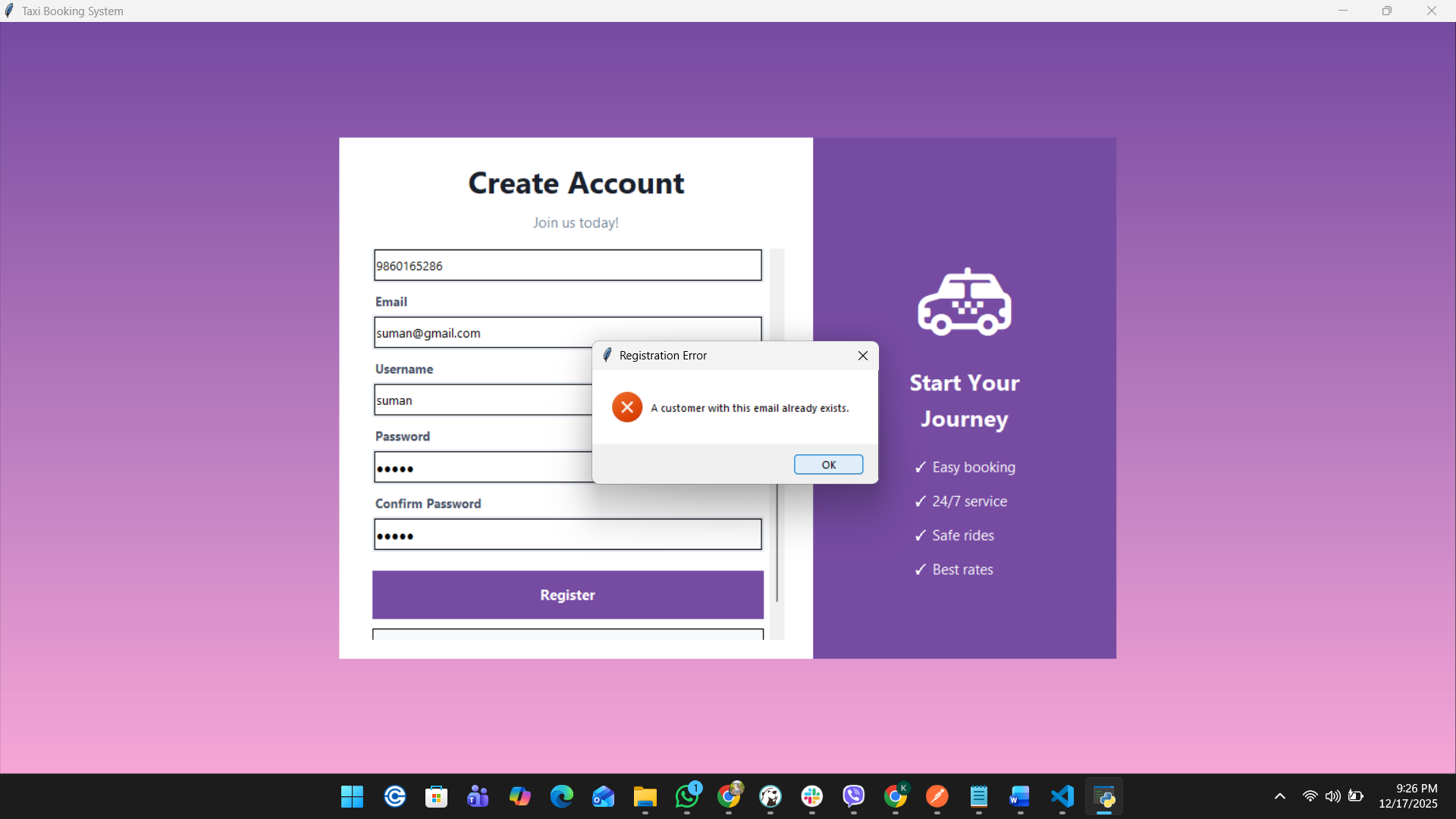


Figure 18 Email already Exist

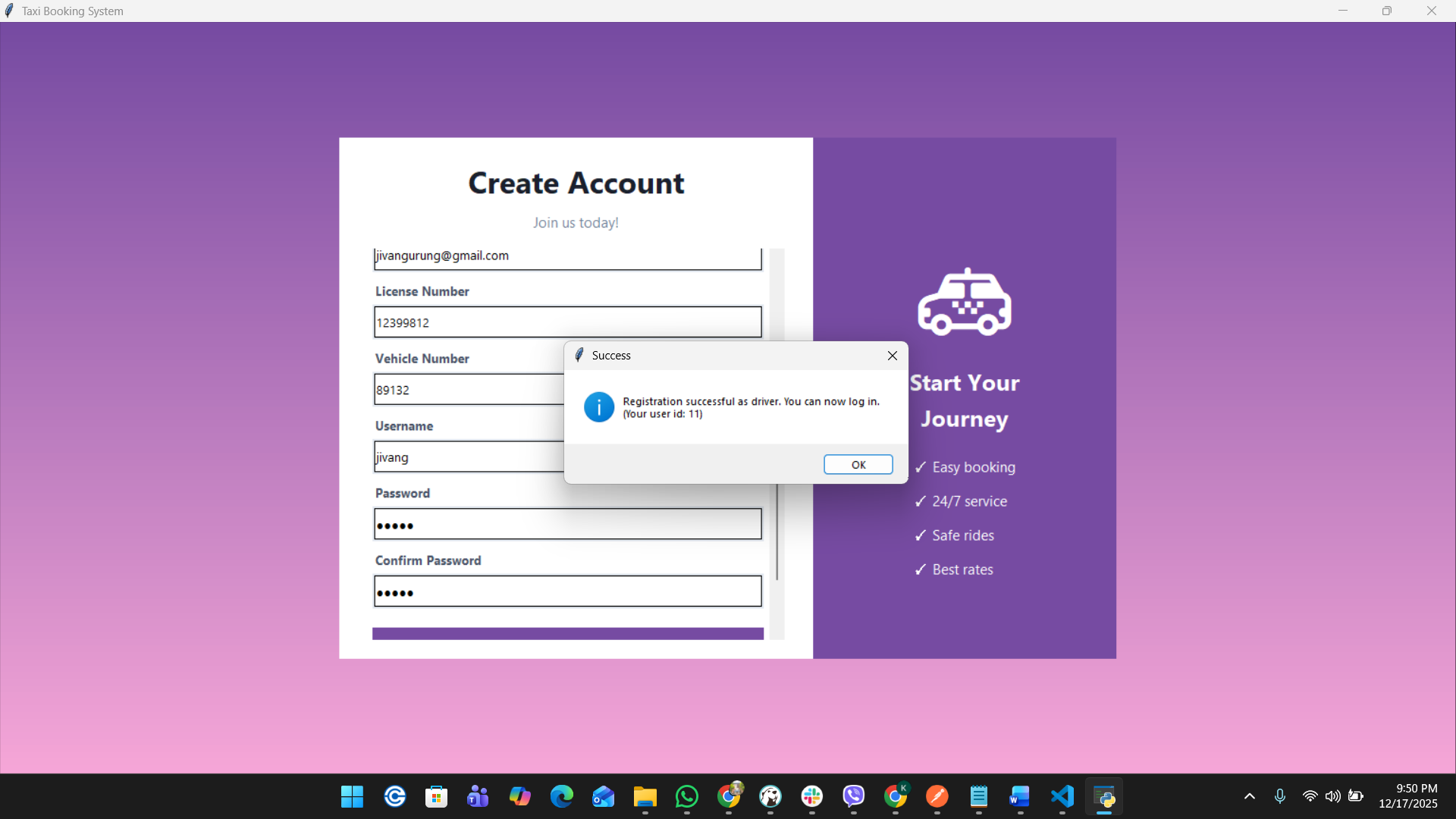


Figure 19Driver Creation

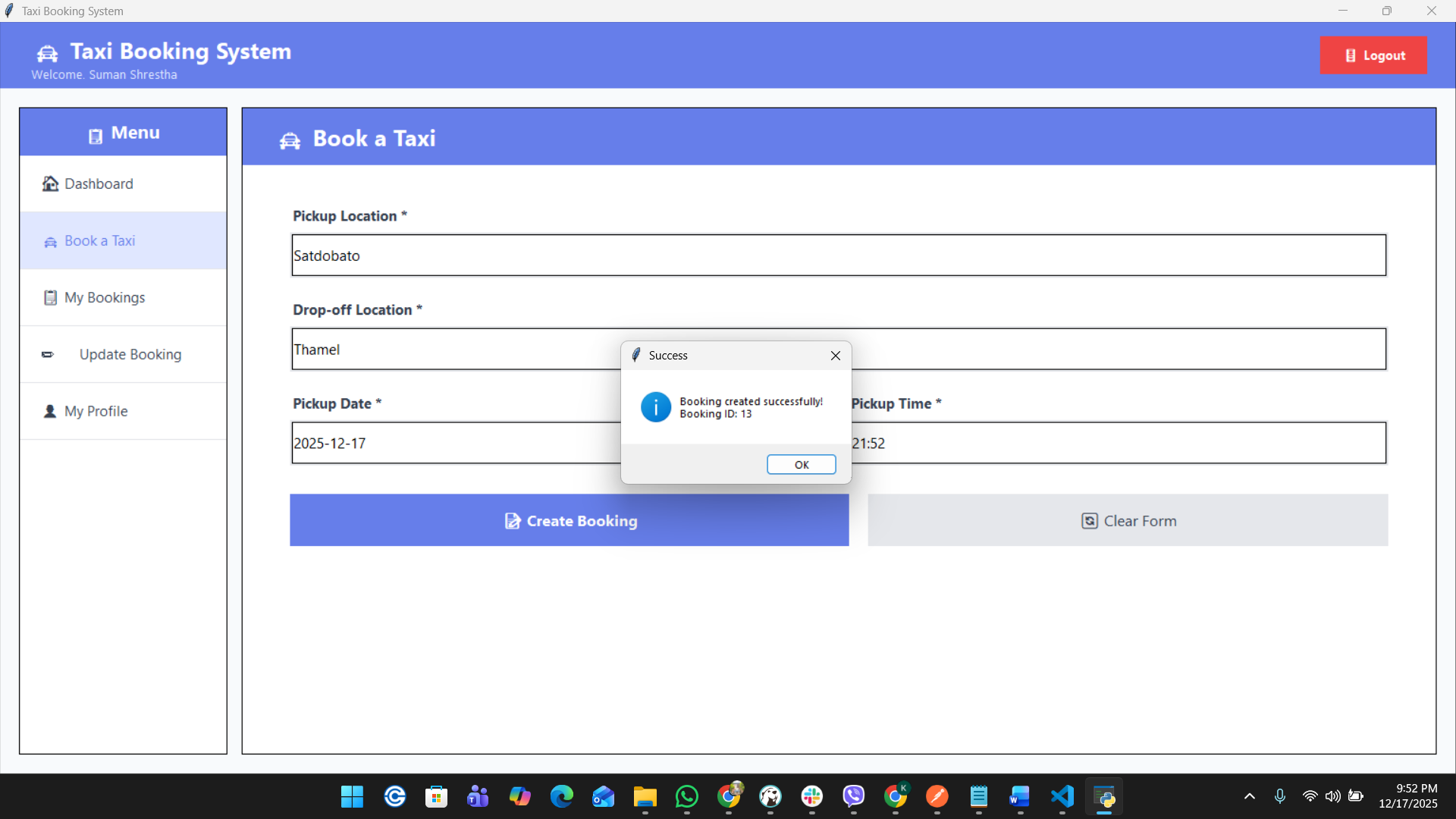


Figure 20 Booking Created

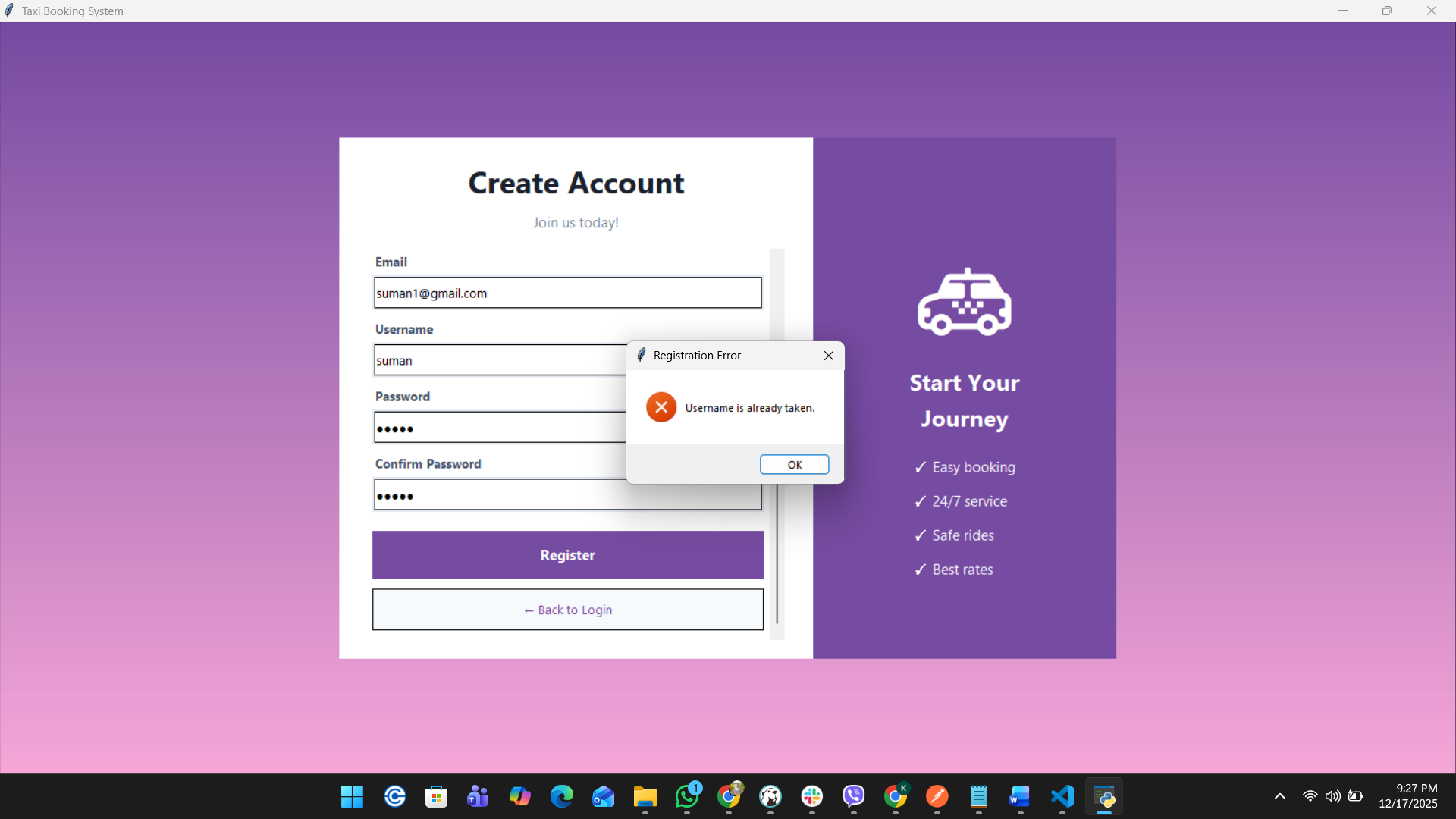


Figure 21 User Name Validation



Figure 22 Assign driver to booking

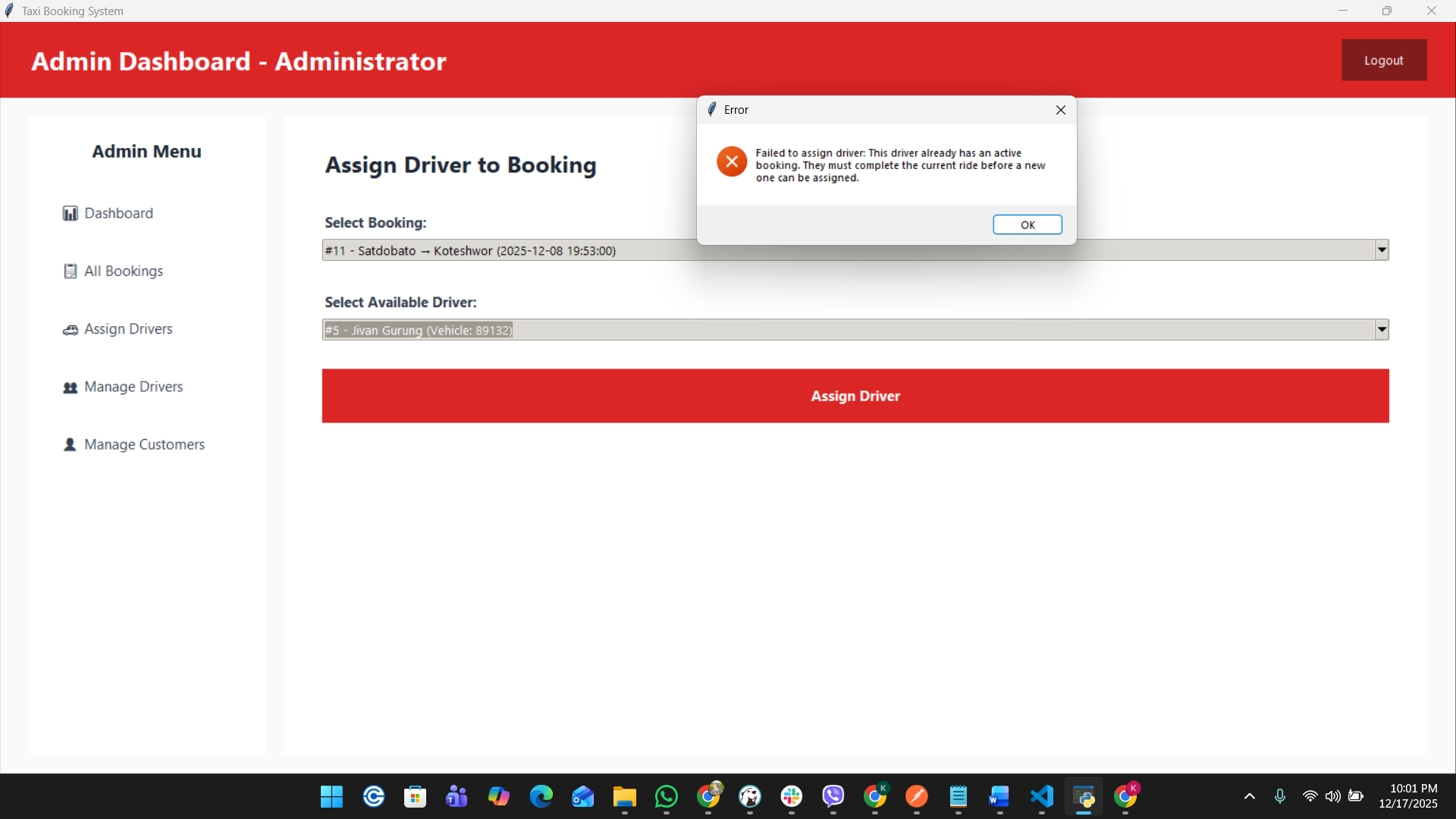


Figure 23 Prevent double booking

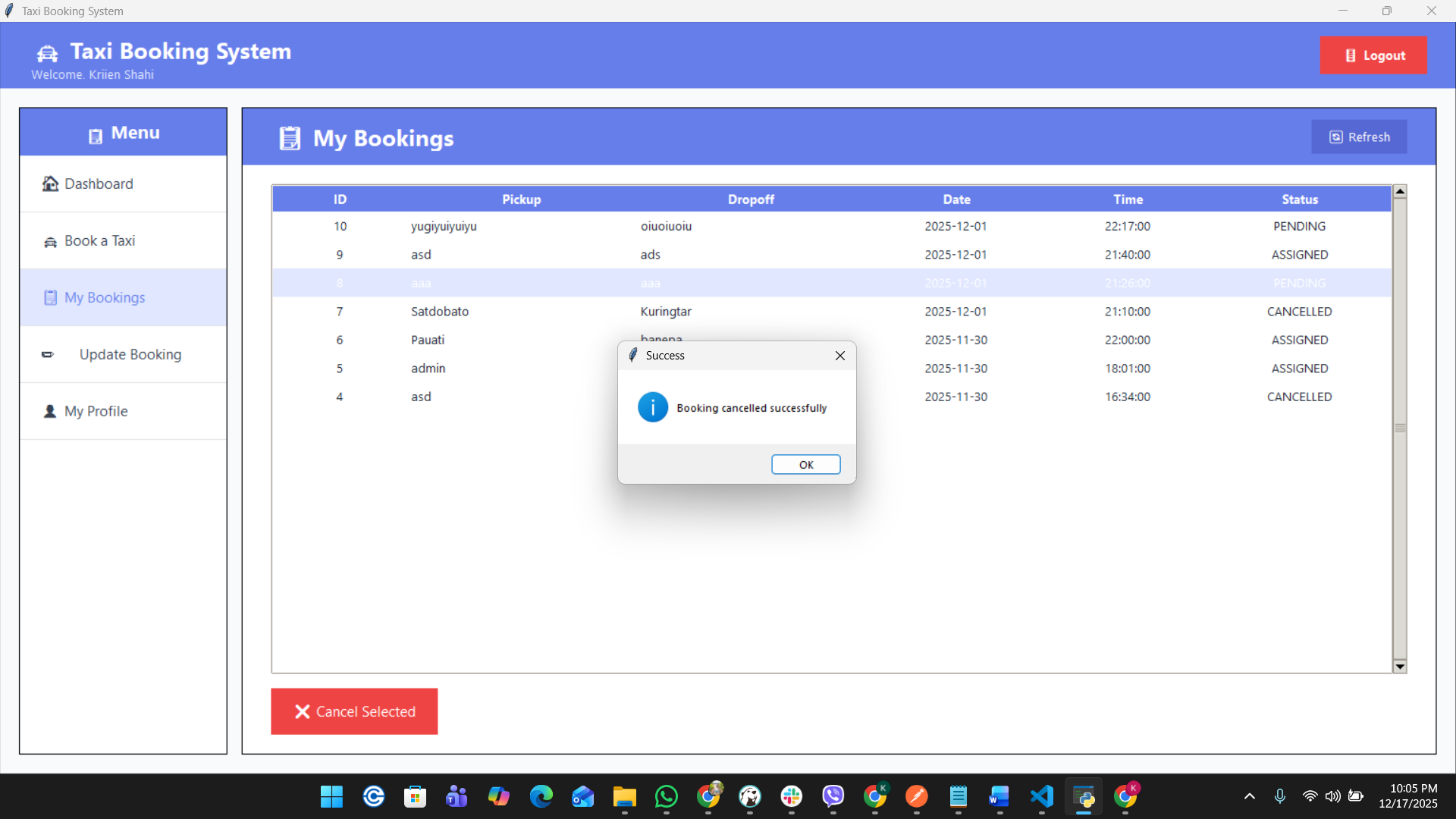


Figure 24 Booking canceled successful

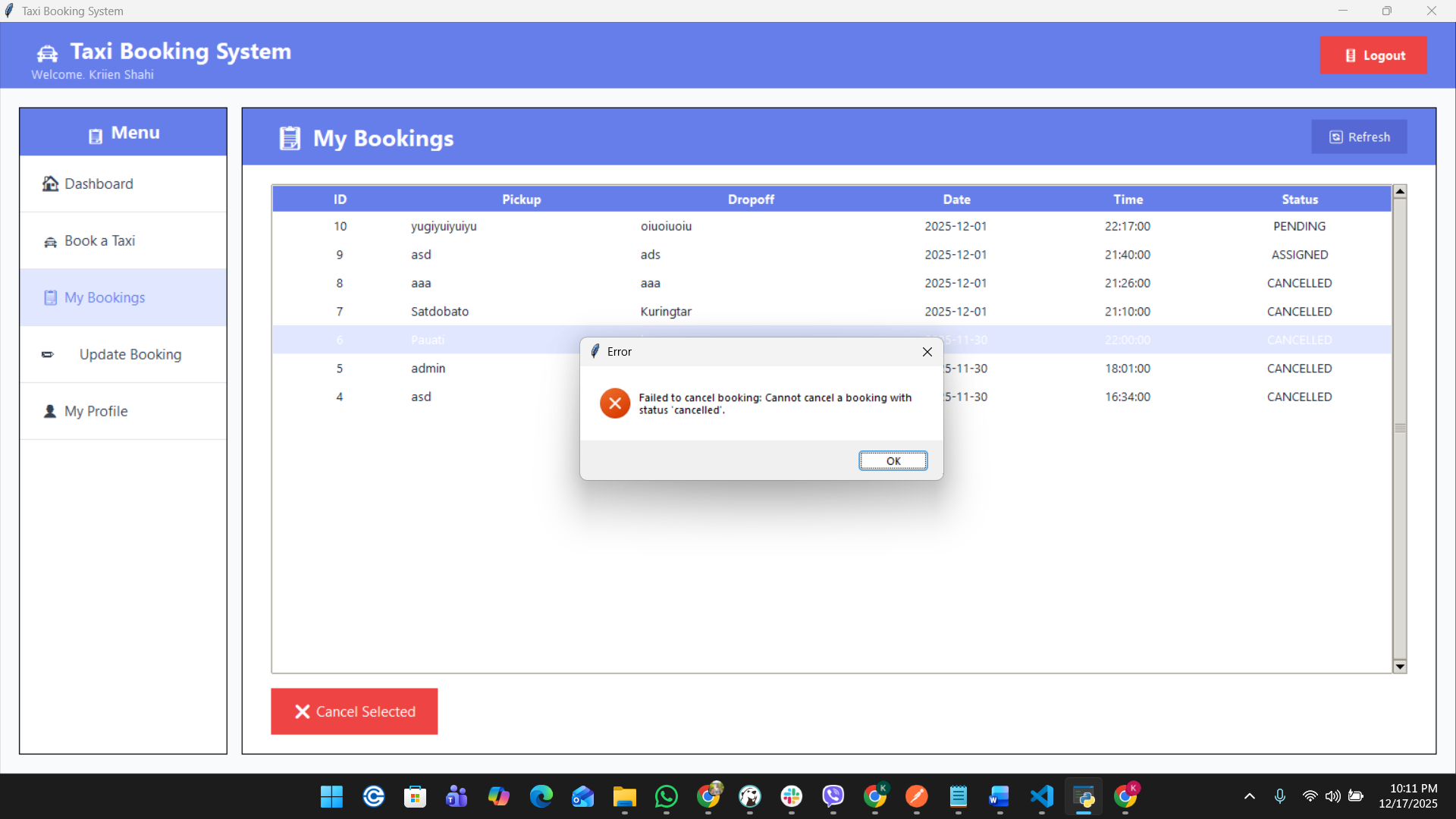


Figure 25 Cancel validation

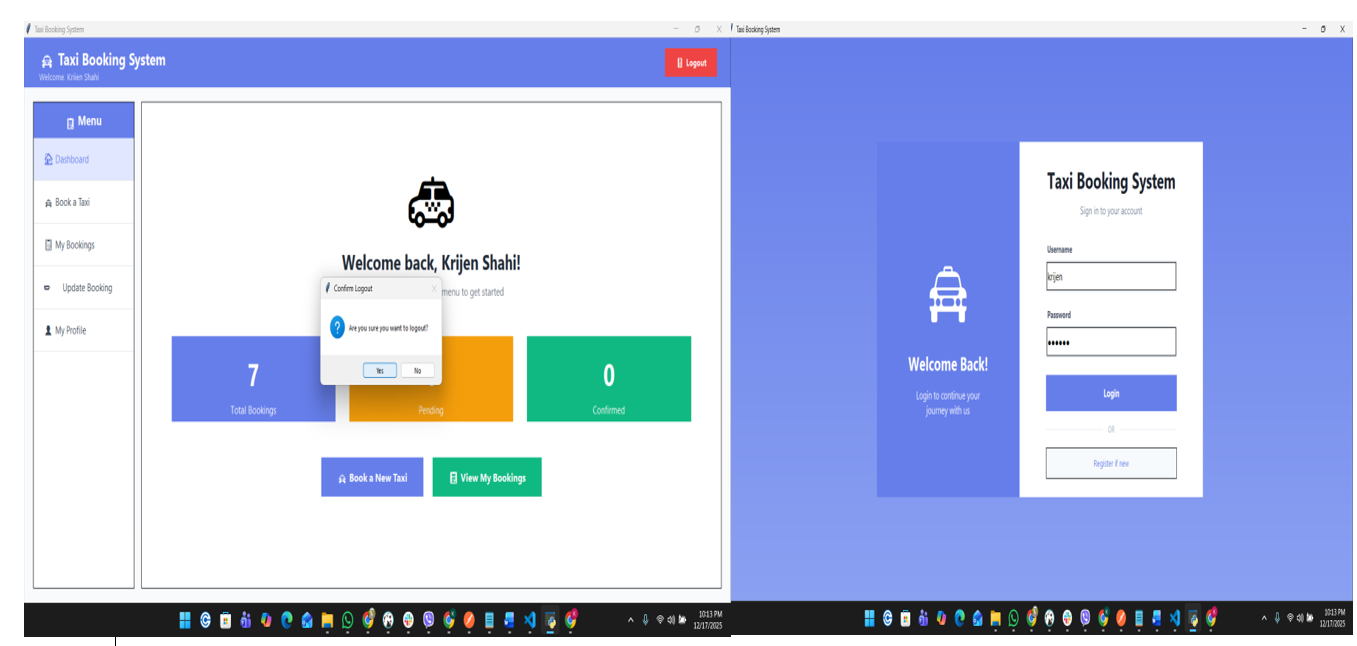


Figure 26 Logout validation

# Discussion/Reflection/Critical Analysis

## Taxi Booking System

As part of this assignment, I was assigned to develop **Taxi Booking System** to replicate a real-world transport service. The main function of the system was to manage daily taxi operations such as user login, taxi booking by customer, assign driver to the booking, and booking status management. The system was developed using an object-oriented approach with a layered structure, which helped us separate the user interface, business logic, and database operations.

## What Went Well

One of the most successful aspects of the project was the use of object-oriented programming concepts. By separating the system into different classes for users, drivers, bookings, and services, the code became easier to understand and maintain. The layered structure also helped us debug issues more efficiently, as problems could be traced to a specific layer.

Core features such as login validation, taxi booking, driver assignment, booking cancellation, and role-based dashboards were implemented successfully and worked as expected. Input validation and role restrictions ensured that customers, drivers, and administrators could only perform actions relevant to them. Teamwork was another positive aspect of the project. Tasks were divided among members based on individual strengths, and regular communication helped us complete the system within the given timeframe.

## What Went Wrong

The main difficulty we faced was time management. We underestimated the amount of time required to connect the user interface with backend logic and database operations. As deadlines approached, development became rushed, especially during testing and UI refinement.

Another challenge was implementing and managing business rules, such as preventing a driver from being assigned to multiple active bookings. These rules required coordination across multiple layers, which increased complexity. In some parts of the system, validation logic was repeated between the UI and backend, making the code harder to manage. With better early planning, these issues could have been reduced.

## Future Improvements

There are several improvements that could be made in the future. Enhancing the user interface responsiveness would improve usability across different screen sizes. Additional features such as booking reminders, notifications, payment integration, and ride history analytics would make the system more realistic.

From a technical perspective, the system architecture could be improved by further separating validation and business logic from the database layer. This would reduce duplication and make the system easier to extend and maintain.

## Learning Experience and Challenges

This project was a valuable learning experience that helped us understand how real-world software systems are designed and implemented. We improved our knowledge of object-oriented programming, database integration, and layered architecture. Managing this project alongside other academic and personal responsibilities was challenging, but it taught us the importance of planning, teamwork, and time management.

# Conclusion

Overall, the Taxi Booking System project was both challenging and rewarding. Despite time constraints and technical difficulties, the system met the assignment requirements and provided valuable insights into real-world software development. The skills and lessons gained from this project will be useful for future academic and professional work.

# References

# Appendix