

Cab Booking System

MINOR PROJECT REPORT

By

B.S.N.ROHITH (RA2211031010106)
G.BADRI (RA2211031010103)
P.JAYA KRISHNA (RA22110310101098)

Under the guidance of

Dr. Manickam.M

In partial fulfilment for the Course

of

21CSC203P –ADVANCED PROGRAMMING PRACTICE

in Networking and Communications



FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minorproject report for the course **21CSC203PADVANCED PROGRAMMING PRACTICE** entitled in "**CAB BOOKING SYSTEM**" is the bonafide work of **B.S.N Rohith(RA2211031010106)**, **G.badri(RA2211031010103)** and **P.jaya krishna(RA221103101098)** who carried out the work under my supervision.

SIGNATURE

Dr. Manickam.M
Assistant Professor
Department of Networking and Communication
SRM Institute of Science andTechnology
Kattankulathur

SIGNATURE

DR.K.Annapurani
Head of the Department
Department of Networking & communication
SRM Institute of science & technology
Kattankulathur

ABSTRACT

The provided Python code implements a graphical user interface (GUI) application for a Cab Booking System main classes: user and travel. The user class handles user authentication, allowing users to log in or create a new account. It utilizes SQLite for user data storage, creating a table to store usernames and passwords. Upon successful login, the application switches to the travel class, which represents the main functionality of the Cab Booking System. The GUI includes features for capturing customer details, specifying travel details such as pickup and drop locations, selecting cab options, and calculating the total cost of the trip based on various factors like distance, cab type, and additional services. The code employs various Tkinter widgets such as labels, entry fields, buttons, and combo boxes to create an interactive and user-friendly interface. Additionally, the program incorporates error handling mechanisms using Tkinter's messagebox to inform users about invalid inputs or errors during the booking process. The modular structure of the code makes it easy to manage different aspects of the Cab Booking System, such as user authentication and trip details. Overall, the application provides a comprehensive and visually appealing solution for users to interact with a virtual cab booking service.

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman**, for imparting confidence to complete my course project

We are highly thankful to our my Course project Faculty **Dr. Manickam.M**, Assistant Professor, Department of Networking and Communications, School of Computing, for his assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. K. Annapurni Paniyappan**, Professor, Department of Department of Networking and Communications, SRM Institute of Science and Technology, and my Departmental colleagues for their support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my course project.

TABLE OF CONTENTS

CHAPTER NO	CONTENTS	PAGE NO
1	INTRODUCTION	
	1.1 Motivation	
	1.2 Objective	
	1.3 Problem Statement	
	1.4 Challenges	
2	LITERATURE SURVEY	
3	REQUIREMENT	
	ANALYSIS	
4	ARCHITECTURE &	
	DESIGN	
5	IMPLEMENTATION	
6	CODE	
7	EXPERIMENT RESULTS	
	& ANALYSIS	
8	CONCLUSION	
9	REFERENCES	

1. INTRODUCTION

The provided Python code is an implementation of a Cab Booking System using the Tkinter library for GUI development and SQLite for database management. The system consists of two main classes: "user" for handling user authentication and login, and "travel" for managing the cab booking process. Users can either log in to an existing account or create a new one. Once logged in, they can input their details and book a cab by selecting various options like pickup and drop locations, cab type, and additional services. The code includes functionalities for calculating the fare based on distance, selected cab type, and added services. The GUI is well-structured with frames and labels for clear user interaction. Overall, it provides a simple yet comprehensive platform for users to manage their cab bookings effectively.

The GUI includes entry fields for capturing customer details, radio buttons for selecting cab types, checkboxes for additional services like travel insurance and extra luggage, and dropdown menus for choosing pickup and drop-off locations. The code incorporates dynamic updates, enabling real-time calculation of fare and presenting a detailed receipt to the user.

Furthermore, the system demonstrates good software engineering practices by using classes to encapsulate related functionalities and employing modular design. The clean and organized structure facilitates code readability and maintenance. Overall, this Cab Booking System code serves as a robust foundation for a practical and user-friendly application in the transportation domain.

The 'user' class handles user authentication, ensuring secure access to the booking system. On the other hand, the 'travel' class manages the core functionalities of the cab booking process.

1.1 MOTIVATION

Developing a Cab Booking System is a commendable venture that holds great potential for addressing real-world transportation needs. In today's fast-paced world, efficiency and convenience are paramount, and this system embodies the spirit of streamlining travel arrangements. By creating a user-friendly graphical interface with Tkinter, the code opens doors for a seamless booking experience, catering to a diverse audience with varying levels of technological familiarity.

The integration of an SQLite database not only enhances the reliability of the system but also underscores the commitment to data persistence and security. The assurance that user accounts and booking details are securely stored reflects a dedication to safeguarding sensitive information, a crucial aspect in the era of digital transactions.

The use of classes, such as 'user' and 'travel,' signifies a commitment to object-oriented programming principles, contributing to a well-structured and maintainable codebase. This approach not only aids in the organization of functionalities but also lays the groundwork for future enhancements and modifications. It reflects a forward-thinking mindset, essential for software that may evolve with changing user requirements and technological advancements.

The thoughtful design of the graphical interface, with input fields, radio buttons, checkboxes, and dropdown menus, ensures a comprehensive and intuitive user experience. The inclusion of dynamic updates for fare calculations and detailed receipts adds a layer of sophistication to the system, demonstrating a dedication to user satisfaction and a desire to exceed expectations.

In conclusion, the motivation behind the Cab Booking System code lies in its potential to make a tangible impact on the way people approach and experience transportation. It embodies efficiency, security, and user-centric design, aligning with the evolving expectations of a tech-savvy and fast-moving society.

1.2 OBJECTIVE

The primary objective of developing the Cab Booking System is to create a robust and user-friendly platform that facilitates efficient and convenient transportation services. This project aims to address the contemporary challenges associated with booking cabs, providing users with a seamless and technology-driven solution. By leveraging Tkinter for the graphical interface, the objective is to design an intuitive and accessible user experience, catering to a diverse audience and enhancing the overall ease of booking.

Furthermore, the integration of an SQLite database serves the purpose of ensuring data reliability and security. The objective is to establish a system that not only manages user accounts effectively but also securely stores booking details. This contributes to building trust among users, assuring them that their personal and transactional information is handled with the utmost care and confidentiality.

The use of object-oriented programming principles, specifically through the implementation of classes like 'user' and 'travel,' underscores the objective of creating a well-organized and maintainable codebase. This approach aims to enhance the system's scalability and adaptability, allowing for future modifications and feature enhancements as needed. The objective is to foster a code structure that aligns with best practices, promoting efficiency and sustainability in the development process.

The design of the graphical interface, featuring input fields, radio buttons, checkboxes, and dropdown menus, serves the objective of delivering a comprehensive and user-centric experience. The dynamic updates for fare calculations and detailed receipts contribute to the objective of providing users with real-time and accurate information, enhancing their overall satisfaction with the booking process.

1.3 PROBLEM STATEMENT

The Cab Booking System project addresses the prevalent challenges associated with traditional taxi booking methods, introducing a technology-driven solution to streamline the transportation booking process. The existing manual booking systems often lack efficiency, leading to delayed or unreliable services. This project aims to solve the problem of inefficiency by creating a robust software application that automates and enhances the entire cab booking experience.

One of the key issues this system seeks to resolve is the lack of a centralized platform for managing cab bookings. Traditional methods involve phone calls or physical stands, which can be inconvenient for both users and service providers. The Cab Booking System aims to centralize and digitize this process, providing a user-friendly interface for booking cabs efficiently and securely.

Another problem the project addresses is the potential for data inaccuracies and security concerns in traditional systems. Storing user information and travel details manually can lead to errors and compromise user data. By implementing an SQLite database and adhering to secure coding practices, the Cab Booking System aims to mitigate these issues, ensuring accurate data storage and safeguarding user privacy.

The project also tackles the challenge of user experience in cab booking. Traditional methods may lack a user-friendly interface and real-time updates on fares and booking details. The Cab Booking System aims to provide an intuitive graphical interface with dynamic features, enhancing the overall user experience and making the booking process more transparent and user-centric.

1.4 CHALLENGES

1. Consider organizing your code into functions or classes to improve readability and maintainability. Add comments to explain complex logic or sections of code for better understanding.
2. Implement more robust error handling. For instance, handle database connection errors or unexpected inputs gracefully. Ensure that user inputs are validated properly to avoid potential issues.
3. Avoid storing passwords directly in the code. Consider using a secure method like hashing. Implement proper authentication mechanisms to enhance security..
4. Enhance the database structure by adding more tables if needed. For instance, you might want to store booking history. Implement a context manager for the database connections to ensure proper resource management.
5. Improve the user interface by adding labels, tooltips, or other elements to guide the user. Consider using a consistent color scheme and layout for a more polished look.
6. Add features like booking history, user profiles, or the ability to cancel bookings. Implement a feature to calculate and display estimated arrival times based on current traffic conditions..
7. Optimize the code for better performance, especially in areas where data processing or calculations are intensive. Consider using more efficient data structures or algorithms where applicable..

These challenges aim to enhance various aspects of your code, making it more robust, secure, and user-friendly..

2. LITERATURE SURVEY

A crucial aspect of cab booking applications is the user experience. Existing studies explore the design and functionality of user interfaces, aiming to enhance user satisfaction and ease of use. User-friendly interfaces often include features like real-time cab tracking, multiple payment options, and efficient booking processes. Investigate how these design principles can be applied to improve the user experience in your system.

Implementing a Cab Booking System involves considering existing literature and solutions in the field. While the provided code serves as a foundational example, a literature survey can provide insights into various aspects, such as user experience, security, and system scalability.

Efficient database management is vital for handling user information, cab details, and booking records. Research existing database solutions and best practices for storing and retrieving data in transportation-related applications. Consider how data integrity, security, and scalability are addressed in similar systems to ensure the robustness of your Cab Booking System.

Security is a paramount concern in any online system dealing with user data and transactions. Explore literature on security measures implemented in ride-sharing or booking platforms. This includes user authentication, data encryption, and protection against common cyber threats. Understanding these security measures will help you implement a secure Cab Booking System.

As your Cab Booking System may experience varying levels of demand, scalability is a significant consideration. Investigate literature on challenges and solutions related to scalability in transportation systems. This can include server load balancing, database optimization, and efficient resource allocation to handle a growing user base.

3. REQUIREMENTS

Functional Requirements:

Users should be able to log in to the system using a username and password. A new user should be able to create an account by providing a unique username and password.

The system should utilize SQLite to store user information, including usernames and passwords.

The application should create a database named 'Users.db' and a table named 'user' to store user credentials.

The system should provide a graphical user interface (GUI) for users to interact with. The GUI should include a login panel with fields for entering a username and password. Users should have the option to create a new account, requiring a unique username and password.

The application should allow users to book a cab by selecting pickup and drop-off locations.

Users should be able to choose the number of passengers for pooling. The system should calculate and display the base charge, distance, traveling insurance, and extra luggage charges.

Users should be able to view the total cost, including taxes, subtotals, and paid amounts. The system should generate a receipt with details such as receipt reference, date, customer information, journey details, selected services, and the total amount.

Non-Functional Requirements:

The system should respond promptly to user interactions, ensuring a smooth user experience. The application's startup time should be minimal, and any delays in data processing, especially during database interactions, should be optimized for efficiency. The graphical user interface (GUI) should be responsive and provide seamless navigation between different frames.

The application should handle errors and exceptions gracefully, providing meaningful error messages to users. It should be robust against unexpected inputs and edge cases, ensuring that it

maintains stability during various usage scenarios. The database operations, especially those related to user authentication and data storage, should be reliable and secure.

The user interface should be intuitive and user-friendly, allowing users to easily understand and interact with the application. Clear and concise labels, appropriate widget placements, and informative messages should be provided to enhance usability. The system should also support accessibility features to accommodate users with diverse needs.

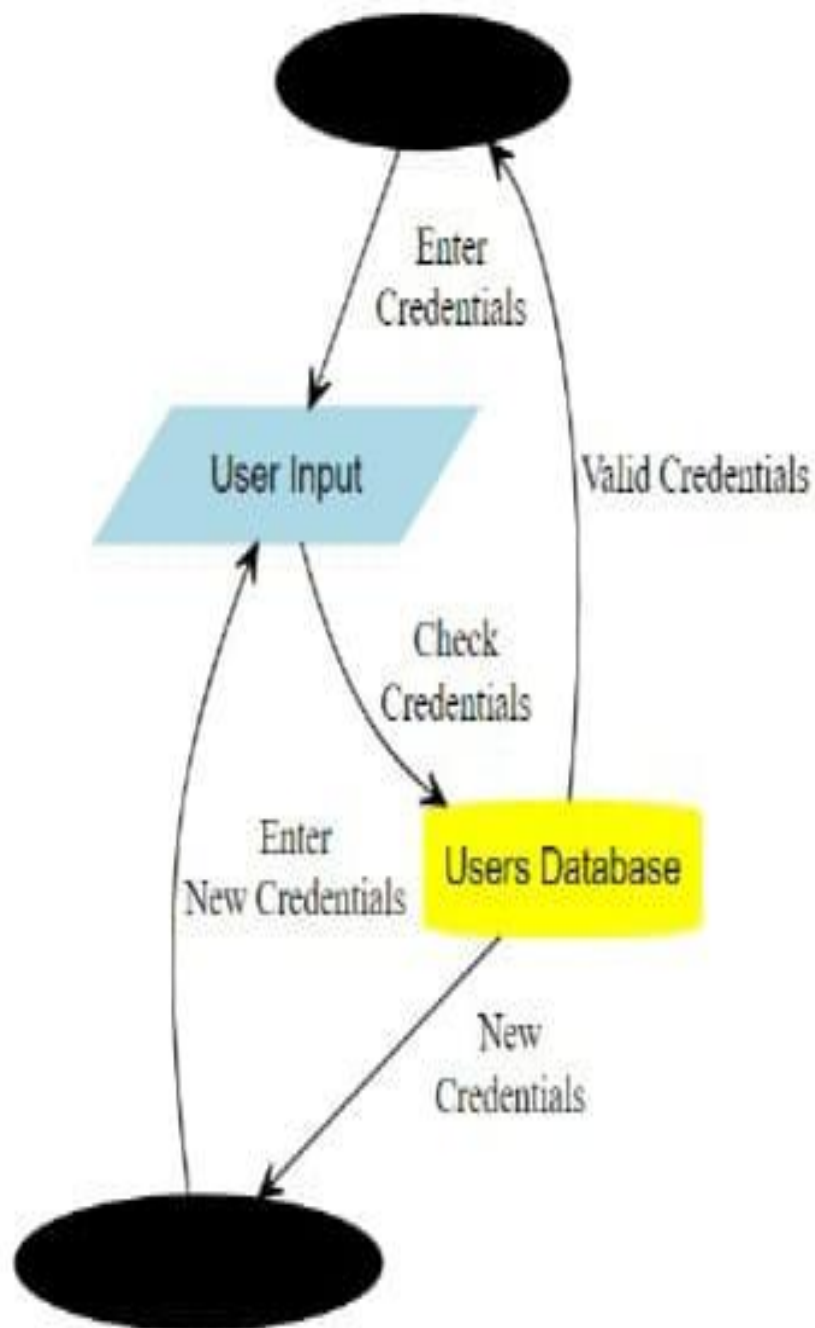
Security is a crucial aspect of the application. User passwords should be securely stored using hashing algorithms, and sensitive information should be protected during transmission. The application should guard against common security threats such as SQL injection and unauthorized access. Additionally, user sessions should be appropriately managed to prevent unauthorized access.

The codebase should follow best practices for readability, modularity, and documentation. Proper comments and documentation should be in place to facilitate future maintenance and updates. The application should be modular, allowing for easy extension of functionality without affecting existing components. Version control and collaborative development practices should be encouraged for ongoing maintenance.

The application should be designed to accommodate potential future growth in terms of users and features. This includes the ability to scale the database and handle increased user loads without compromising performance. The code should be structured to support additional functionality seamlessly.

The system should ensure the integrity of user data by implementing proper validation mechanisms. Input data, especially from user forms, should be validated to prevent data corruption or incorrect entries in the database.

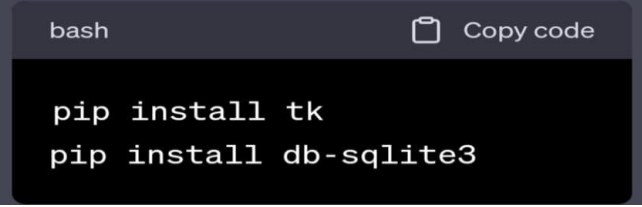
4. ARCHITECTURE AND DESIGN



5. IMPLEMENTATION

Step 1: Install Required Libraries

Ensure you have Python installed. You can install Tkinter (usually included with Python) and SQLite3 if not already installed

A terminal window with a dark background. The title bar shows 'bash' and a 'Copy code' button. The terminal contains two lines of text: 'pip install tk' and 'pip install db-sqlite3'.

```
bash Copy code
pip install tk
pip install db-sqlite3
```

Once your environment is set up, proceed to write the Python code. Create a new script, for instance, `cab_booking_system.py`. This script will be the backbone of your application. Refer to the provided code as a foundation and implement the classes and functions outlined. Customize them to suit your specific needs.

Next, establish an SQLite database to store essential information about users, bookings, and available cabs. Naming it something like `cab_booking.db` is a good practice. This database will serve as the backend storage for your system.

With the code in place and the database set up, execute the script (`cab_booking_system.py`). A Tkinter GUI should appear, allowing users to interact with the Cab Booking System. This interface provides the means for users to book cabs and manage their bookings seamlessly.

the code by identifying and addressing the root causes. Pay attention to error handling to ensure a robust and user-friendly system.

6.CODE

```
from tkinter import *
from tkinter import ttk
import random
import time
import datetime
from tkinter import messagebox as ms
import sqlite3

Item4 = 0

# make database and users (if not exists already) table at programme start up
with sqlite3.connect('Users.db') as db:
    c = db.cursor()

c.execute('CREATE TABLE IF NOT EXISTS user (username TEXT NOT NULL ,password TEXT NOT NULL)')
db.commit()
db.close()

#main Class
class user:
    def __init__(self, master):
        # Window
        self.master = master
        # Some Usefull variables
        self.username = StringVar()
        self.password = StringVar()
        self.n_username = StringVar()
        self.n_password = StringVar()
        #Create Widgets
        self.widgets()

    #Login Function
    def login(self):
        #Establish Connection
        with sqlite3.connect('Users.db') as db:
            c = db.cursor()

        #Find user If there is any take proper action
        find_user = ('SELECT * FROM user WHERE username = ? and password = ?')
        c.execute(find_user, [(self.username.get()), (self.password.get())])
        result = c.fetchall()
        if result:
            self.logf.pack_forget()
            self.head['text'] = "Welcome " + self.username.get()
            self.head.configure(fg="black")
```



```

        self.head.pack(fill=X)
        application = travel(root)

    else:
        ms.showerror('Oops!', 'Username Not Found.')

def new_user(self):
    #Establish Connection
    with sqlite3.connect('Users.db') as db:
        c = db.cursor()

    #Find Existing username if any take proper action
    find_user = ('SELECT * FROM user WHERE username = ?')
    c.execute(find_user,[(self.username.get())])
    if c.fetchall():
        ms.showerror('Error!', 'Username Already Taken!')
    else:
        ms.showinfo('Success!', 'Account Created!')
        self.log()

    #Create New Account
    insert = 'INSERT INTO user(username,password) VALUES(?,?)'
    c.execute(insert,[(self.n_username.get()),(self.n_password.get())])
    db.commit()

    #Frame Packing Methords
def log(self):
    self.username.set('')
    self.password.set('')
    self.crf.pack_forget()
    self.head['text'] = 'Login'
    self.logf.pack()
def cr(self):
    self.n_username.set('')
    self.n_password.set('')
    self.logf.pack_forget()
    self.head['text'] = 'Create Account'
    self.crf.pack()

#Draw Widgets
def widgets(self):
    self.head = Label(self.master,text = 'Login Panel',font = ('',30),pady =
10)

    self.head.pack()
    self.logf = Frame(self.master,padx =10,pady = 10)
    Label(self.logf,text = 'Username: ',font =
('',20),pady=5,padx=5).grid(sticky = W)
    Entry(self.logf,textvariable = self.username,bd = 5,font =
('',15)).grid(row=0,column=1)
    Label(self.logf,text = 'Password: ',font =
('',20),pady=5,padx=5).grid(sticky = W)

```

```

        Entry(self.logf,textvariable = self.password,bd = 5,font = ('',15),show =
        '*').grid(row=1,column=1)
        Button(self.logf,text = ' Login ',bd = 3 ,font =
        ('',15),padx=5,pady=5,command=self.login).grid()
        Button(self.logf,text = ' Create Account ',bd = 3 ,font =
        ('',15),padx=5,pady=5,command=self.cr).grid(row=2,column=1)
        self.logf.pack()

        self.crf = Frame(self.master,padx =10,pady = 10)
        Label(self.crf,text = 'Username: ',font =
        ('',20),pady=5,padx=5).grid(sticky = W)
        Entry(self.crf,textvariable = self.n_username,bd = 5,font =
        ('',15)).grid(row=0,column=1)
        Label(self.crf,text = 'Password: ',font =
        ('',20),pady=5,padx=5).grid(sticky = W)
        Entry(self.crf,textvariable = self.n_password,bd = 5,font = ('',15),show
        = '*').grid(row=1,column=1)
        Button(self.crf,text = 'Create Account',bd = 3 ,font =
        ('',15),padx=5,pady=5,command=self.new_user).grid()
        Button(self.crf,text = 'Go to Login',bd = 3 ,font =
        ('',15),padx=5,pady=5,command=self.log).grid(row=2,column=1)

class travel:

    def __init__(self,root):
        self.root = root
        self.root.title("Cab Booking System")
        self.root.geometry(geometry)
        self.root.configure(background='black')

        DateofOrder=StringVar()
        DateofOrder.set(time.strftime(" %d / %m / %Y "))
        Receipt_Ref=StringVar()
        PaidTax=StringVar()
        SubTotal=StringVar()
        TotalCost=StringVar()

        var1=IntVar()
        var2=IntVar()
        var3=IntVar()
        var4=IntVar()
        journeyType=IntVar()
        carType=IntVar()

        var11=StringVar()
        var12=StringVar()
        var13=StringVar()
        reset_counter=0

```

```

Firstname=StringVar()
Surname=StringVar()
Address=StringVar()
Postcode=StringVar()
Mobile=StringVar()
Telephone=StringVar()
Email=StringVar()

CabTax=StringVar()
Km=StringVar()
Travel_Ins=StringVar()
Luggage=StringVar()
Receipt=StringVar()

Standard=StringVar()
FordGalaxy=StringVar()
FordMondeo=StringVar()

CabTax.set("0")
Km.set("0")
Travel_Ins.set("0")
Luggage.set("0")

Standard.set("0")
FordGalaxy.set("0")
FordMondeo.set("0")

#=====Define
Function=====

def iExit():
    iExit= ms.askyesno("Prompt!", "Do you want to exit?")
    if iExit > 0:
        root.destroy()
        return

def Reset():
    CabTax.set("0")
    Km.set("0")
    Travel_Ins.set("0")
    Luggage.set("0")

    Standard.set("0")
    FordGalaxy.set("0")

```

```

FordMondeo.set("0")

Firstname.set("")
Surname.set("")
Address.set("")
Postcode.set("")
Mobile.set("")
Telephone.set("")
Email.set("")

PaidTax.set("")
SubTotal.set("")
TotalCost.set("")
self.txtReceipt1.delete("1.0",END)
self.txtReceipt2.delete("1.0",END)

var1.set(0)
var2.set(0)
var3.set(0)
var4.set(0)
journeyType.set(0)
carType.set(0)
var11.set("0")
var12.set("0")
var13.set("0")

self.cboPickup.current(0)
self.cboDrop.current(0)
self.cboPooling.current(0)

self.txtCabTax.configure(state=DISABLED)
self.txtKm.configure(state=DISABLED)
self.txtTravel_Ins.configure(state=DISABLED)
self.txtLuggage.configure(state=DISABLED)

self.txtStandard.configure(state=DISABLED)
self.txtFordGalaxy.configure(state=DISABLED)
self.txtFordMondeo.configure(state=DISABLED)
self.reset_counter=1

def Receiptt():
    if reset_counter == 0 and Firstname.get()!="" and Surname.get()!=""
and Address.get()!="" and Postcode.get()!="" and Mobile.get()!="" and
Telephone.get()!="" and Email.get()!="":
        self.txtReceipt1.delete("1.0",END)
        self.txtReceipt2.delete("1.0",END)
        x=random.randint(10853,500831)
        randomRef = str(x)
        Receipt_Ref.set(randomRef)

```

```

        self.txtReceipt1.insert(END, "Receipt Ref:\n")
        self.txtReceipt2.insert(END, Receipt_Ref.get() + "\n")
        self.txtReceipt1.insert(END, 'Date:\n')
        self.txtReceipt2.insert(END, DateofOrder.get() + "\n")
        self.txtReceipt1.insert(END, 'Cab No:\n')
        self.txtReceipt2.insert(END, 'TR ' + Receipt_Ref.get() + " BW\n")
        self.txtReceipt1.insert(END, 'Firstname:\n')
        self.txtReceipt2.insert(END, Firstname.get() + "\n")
        self.txtReceipt1.insert(END, 'Surname:\n')
        self.txtReceipt2.insert(END, Surname.get() + "\n")
        self.txtReceipt1.insert(END, 'Address:\n')
        self.txtReceipt2.insert(END, Address.get() + "\n")
        self.txtReceipt1.insert(END, 'Postal Code:\n')
        self.txtReceipt2.insert(END, Postcode.get() + "\n")
        self.txtReceipt1.insert(END, 'Telephone:\n')
        self.txtReceipt2.insert(END, Telephone.get() + "\n")
        self.txtReceipt1.insert(END, 'Mobile:\n')
        self.txtReceipt2.insert(END, Mobile.get() + "\n")
        self.txtReceipt1.insert(END, 'Email:\n')
        self.txtReceipt2.insert(END, Email.get() + "\n")
        self.txtReceipt1.insert(END, 'From:\n')
        self.txtReceipt2.insert(END, var11.get() + "\n")
        self.txtReceipt1.insert(END, 'To:\n')
        self.txtReceipt2.insert(END, var12.get() + "\n")
        self.txtReceipt1.insert(END, 'Pooling:\n')
        self.txtReceipt2.insert(END, var13.get() + "\n")
        self.txtReceipt1.insert(END, 'Standard:\n')
        self.txtReceipt2.insert(END, Standard.get() + "\n")
        self.txtReceipt1.insert(END, 'Prime Sedan:\n')
        self.txtReceipt2.insert(END, FordGalaxy.get() + "\n")
        self.txtReceipt1.insert(END, 'Premium Sedan:\n')
        self.txtReceipt2.insert(END, FordMondeo.get() + "\n")
        self.txtReceipt1.insert(END, 'Paid:\n')
        self.txtReceipt2.insert(END, PaidTax.get() + "\n")
        self.txtReceipt1.insert(END, 'SubTotal:\n')
        self.txtReceipt2.insert(END, str(SubTotal.get()) + "\n")
        self.txtReceipt1.insert(END, 'Total Cost:\n')
        self.txtReceipt2.insert(END, str(TotalCost.get()))

    else:
        self.txtReceipt1.delete("1.0", END)
        self.txtReceipt2.delete("1.0", END)
        self.txtReceipt1.insert(END, "\nNo Input")

def Cab_Tax():
    global Item1
    if var1.get() == 1:
        self.txtCabTax.configure(state = NORMAL)
        Item1=float(50)

```

```

        CabTax.set("Rs " + str(Item1))
    elif var1.get() == 0:
        self.txtCabTax.configure(state=DISABLED)
        CabTax.set("0")
        Item1=0

def Kilo():
    if var2.get() == 0:
        self.txtKm.configure(state=DISABLED)
        Km.set("0")
    elif var2.get() == 1 and var11.get() != "" and var12.get() != "":
        self.txtKm.configure(state=NORMAL)
        if var11.get() == "SRM":
            switch={"RAMAPURAM": 36,"AIRPORT":
22,"CHENGALPATTU":18,"SRM": 0}
            Km.set(switch[var12.get()])
        elif var11.get() == "RAMAPURAM":
            switch={"RAMAPURAM": 0,"AIRPORT":
12,"CHENGALPATTU":52,"SRM": 34}
            Km.set(switch[var12.get()])
        elif var11.get() == "AIRPORT":
            switch={"RAMAPURAM": 12,"AIRPORT":
0,"CHENGALPATTU":40,"SRM": 22}
            Km.set(switch[var12.get()])
        elif var11.get() == "CHENGALPATTU":
            switch={"RAMAPURAM": 52,"AIRPORT":
40,"CHENGALPATTU":0,"SRM": 18}
            Km.set(switch[var12.get()])

def Travelling():
    global Item3
    if var3.get() == 1:
        self.txtTravel_Ins.configure(state = NORMAL)
        Item3=float(10)
        Travel_Ins.set("Rs " + str(Item3))
    elif var3.get() == 0:
        self.txtTravel_Ins.configure(state = DISABLED)
        Travel_Ins.set("0")
        Item3=0

def Lug():
    global Item4
    if (var4.get()==1):
        self.txtLuggage.configure(state = NORMAL)
        Item4=float(30)
        Luggage.set("Rs "+ str(Item4))
    elif var4.get()== 0:

```

```

        self.txtLuggage.configure(state = DISABLED)
        Luggage.set("0")
        Item4=0

def selectCar():
    global Item5
    if carType.get() == 1:
        self.txtFordGalaxy.configure(state = DISABLED)
        FordGalaxy.set("0")
        self.txtFordMondeo.configure(state = DISABLED)
        FordMondeo.set("0")
        self.txtStandard.configure(state = NORMAL)
        Item5 = float(8)
        Standard.set("Rs " + str(Item5))
    elif carType.get() == 2:
        self.txtStandard.configure(state =DISABLED)
        Standard.set("0")
        self.txtFordMondeo.configure(state = DISABLED)
        FordMondeo.set("0")
        self.txtFordGalaxy.configure(state = NORMAL)
        Item5 = float(15)
        FordGalaxy.set("Rs " + str(Item5))
    else:
        self.txtStandard.configure(state =DISABLED)
        Standard.set("0")
        self.txtFordGalaxy.configure(state = DISABLED)
        FordGalaxy.set("0")
        self.txtFordMondeo.configure(state = NORMAL)
        Item5 = float(22)
        FordMondeo.set("Rs " + str(Item5))

def Total_Paid():
    if ((var1.get() == 1 and var2.get() == 1 and var3.get() == 1 or
var4.get() == 1) and carType.get() != 0 and journeyType.get() != 0 and
(var11.get() != "" and var12.get() !='')):
        if journeyType.get()==1:
            Item2=Km.get()
            Cost_of_fare = (Item1+(float(Item2)*Item5)+Item3+Item4)

            Tax = "Rs " + str('%.2f'%((Cost_of_fare) *0.09))
            ST = "Rs " + str('%.2f'%((Cost_of_fare)))
            TT = "Rs " + str('%.2f'%(Cost_of_fare+((Cost_of_fare)*0.9)))
        elif journeyType.get()==2:
            Item2=Km.get()
            Cost_of_fare = (Item1+(float(Item2)*Item5)*1.5+Item3+Item4)

            Tax = "Rs " + str('%.2f'%((Cost_of_fare) *0.09))
            ST = "Rs " + str('%.2f'%((Cost_of_fare)))

```

```

        TT = "Rs " + str('%.2f'%((Cost_of_fare+((Cost_of_fare)*0.9)))
    else:
        Item2=Km.get()
        Cost_of_fare = (Item1+(float(Item2)*Item5)*2+Item3+Item4)

        Tax = "Rs " + str('%.2f'%((Cost_of_fare) *0.09))
        ST = "Rs " + str('%.2f'%((Cost_of_fare)))
        TT = "Rs " + str('%.2f'%((Cost_of_fare+((Cost_of_fare)*0.9)))

        PaidTax.set(Tax)
        SubTotal.set(ST)
        TotalCost.set(TT)
    else:
        w = ms.showwarning("Error !","Invalid Input\nPlease try again
!!!")

#=====mainframe=====
=====

MainFrame=Frame(self.root)
MainFrame.pack(fill=BOTH,expand=True)

Tops = Frame(MainFrame, bd=10, width=1350,relief=RIDGE)
Tops.pack(side=TOP,fill=BOTH)

self.lblTitle=Label(Tops,font=('arial',50,'bold'),text="\t Cab Booking
System ")
self.lblTitle.grid()

#=====customerframedetail=====
=====

CustomerDetailsFrame=LabelFrame(MainFrame, width=1350,height=500,bd=20,
pady=5, relief=RIDGE)
CustomerDetailsFrame.pack(side=BOTTOM,fill=BOTH,expand=True)

FrameDetails=Frame(CustomerDetailsFrame, width=880,height=400,bd=10,
relief=RIDGE)
FrameDetails.pack(side=LEFT,fill=BOTH,expand=True)

CustomerName=LabelFrame(FrameDetails, width=150,height=250,bd=10,
font=('arial',12,'bold'),text="Customer Info", relief=RIDGE)
CustomerName.grid(row=0,column=0)

TravelFrame = LabelFrame(FrameDetails,bd=10, width=300,height=250,
font=('arial',12,'bold'),text="Booking Detail", relief=RIDGE)
TravelFrame.grid(row=0,column=1)

Book_Frame=LabelFrame(FrameDetails,width=300,height=150,relief=FLAT)

```



```

Book_Frame.grid(row=1,column=0)

CostFrame =
LabelFrame(FrameDetails,width=150,height=150,bd=5,relief=FLAT)
CostFrame.grid(row=1,column=1)

#=====receipt=====
=====

Receipt_BottonFrame=LabelFrame(CustomerDetailsFrame,bd=10,
width=450,height=400, relief=RIDGE)
Receipt_BottonFrame.pack(side=RIGHT,fill=BOTH,expand=True)

ReceiptFrame=LabelFrame(Receipt_BottonFrame, width=350,height=300,
font=('arial',12,'bold'),text="Receipt", relief=RIDGE)
ReceiptFrame.grid(row=0,column=0)

ButtonFrame=LabelFrame(Receipt_BottonFrame, width=350,height=100,
relief=RIDGE)
ButtonFrame.grid(row=1,column=0)

#=====CustomerName=====
=====

self.lblFirstname=Label(CustomerName,font=('arial',14,'bold'),text="First
name",bd=7)
self.lblFirstname.grid(row=0,column=0,sticky=W)
self.txtFirstname=Entry(CustomerName,font=('arial',14,'bold'),textvariabl
e=Firstname,bd=7,insertwidth=2,justify=RIGHT)
self.txtFirstname.grid(row=0,column=1)

self.lblSurname=Label(CustomerName,font=('arial',14,'bold'),text="Surname
",bd=7)
self.lblSurname.grid(row=1,column=0,sticky=W)
self.txtSurname=Entry(CustomerName,font=('arial',14,'bold'),textvariable=
Surname,bd=7,insertwidth=2,justify=RIGHT)
self.txtSurname.grid(row=1,column=1,sticky=W)

self.lblAddress=Label(CustomerName,font=('arial',14,'bold'),text="Address
",bd=7)
self.lblAddress.grid(row=2,column=0,sticky=W)
self.txtAddress=Entry(CustomerName,font=('arial',14,'bold'),textvariable=
Address,bd=7,insertwidth=2,justify=RIGHT)
self.txtAddress.grid(row=2,column=1)

self.lblPostcode=Label(CustomerName,font=('arial',14,'bold'),text="Postco
de",bd=7)
self.lblPostcode.grid(row=3,column=0,sticky=W)

```

```

        self.txtPostcode=Entry(CustomerName,font=('arial',14,'bold'),textvariable
=Postcode,bd=7,insertwidth=2,justify=RIGHT)
        self.txtPostcode.grid(row=3,column=1)

        self.lblTelephone=Label(CustomerName,font=('arial',14,'bold'),text="Telep
hone",bd=7)
        self.lblTelephone.grid(row=4,column=0,sticky=W)
        self.txtTelephone=Entry(CustomerName,font=('arial',14,'bold'),textvariabl
e=Telephone,bd=7,insertwidth=2,justify=RIGHT)
        self.txtTelephone.grid(row=4,column=1)

        self.lblMobile=Label(CustomerName,font=('arial',14,'bold'),text="Mobile",
bd=7)
        self.lblMobile.grid(row=5,column=0,sticky=W)
        self.txtMobile=Entry(CustomerName,font=('arial',14,'bold'),textvariable=M
obile,bd=7,insertwidth=2,justify=RIGHT)
        self.txtMobile.grid(row=5,column=1)

        self.lblEmail=Label(CustomerName,font=('arial',14,'bold'),text="Email",bd
=7)
        self.lblEmail.grid(row=6,column=0,sticky=W)
        self.txtEmail=Entry(CustomerName,font=('arial',14,'bold'),textvariable=Em
ail,bd=7,insertwidth=2,justify=RIGHT)
        self.txtEmail.grid(row=6,column=1)

        #=====Cab
Information=====
        self.lblPickup=Label(TravelFrame,font=('arial',14,'bold'),text="Pickup",b
d=7)
        self.lblPickup.grid(row=0,column=0,sticky=W)

        self.cboPickup =ttk.Combobox(TravelFrame, textvariable = var11 ,
state='readonly', font=('arial',20,'bold'), width=14)
        self.cboPickup['value']=('','SRM','CHENGALPATTU','AIRPORT','RAMAPURAM')
        self.cboPickup.current(0)
        self.cboPickup.grid(row=0,column=1)

        self.lblDrop=Label(TravelFrame,font=('arial',14,'bold'),text="Drop",bd=7)
        self.lblDrop.grid(row=1,column=0,sticky=W)

        self.cboDrop =ttk.Combobox(TravelFrame, textvariable = var12 ,
state='readonly', font=('arial',20,'bold'), width=14)
        self.cboDrop['value']=('','RAMAPURAM','AIRPORT','SRM','CHENGALPATTU')
        self.cboDrop.current(0)
        self.cboDrop.grid(row=1,column=1)

```

```

        self.lblPooling=Label(TravelFrame,font=('arial',14,'bold'),text="Pooling"
, bd=7)
        self.lblPooling.grid(row=2,column=0,sticky=W)

        self.cboPooling =ttk.Combobox(TravelFrame, textvariable = var13 ,
state='readonly', font=('arial',20,'bold'), width=14)
        self.cboPooling['value']=('','1','2','3','4')
        self.cboPooling.current(1)
        self.cboPooling.grid(row=2,column=1)

        #=====Cab
Information=====

        self.chkCabTax=Checkbutton(TravelFrame,text="Base Charge *",variable =
var1, onvalue=1, offvalue=0,font=('arial',16,'bold'),command=Cab_Tax).grid(row=3,
column=0, sticky=W)
        self.txtCabTax=Label(TravelFrame,font=('arial',14,'bold'),textvariable=Ca
bTax,bd=6,width=18,bg="white",state= DISABLED,justify=RIGHT,relief=SUNKEN)
        self.txtCabTax.grid(row=3,column=1)

        self.chkKm=Checkbutton(TravelFrame,text="Distance(KMs) *",variable =
var2, onvalue=1, offvalue=0,font=('arial',16,'bold'),command=Kilo).grid(row=4,
column=0, sticky=W)
        self.txtKm=Label(TravelFrame,font=('arial',14,'bold'),textvariable=Km,bd=
6,width=18,bg="white",state=
DISABLED,justify=RIGHT,relief=SUNKEN,highlightthickness=0)
        self.txtKm.grid(row=4,column=1)

        self.chkTravel_Ins=Checkbutton(TravelFrame,text="Travelling Insurance
*",variable = var3, onvalue=1,
offvalue=0,font=('arial',16,'bold'),command=Travelling).grid(row=5, column=0,
sticky=W)
        self.txtTravel_Ins=Label(TravelFrame,font=('arial',14,'bold'),textvariabl
e=Travel_Ins,bd=6,width=18,bg="white",state=
DISABLED,justify=RIGHT,relief=SUNKEN)
        self.txtTravel_Ins.grid(row=5,column=1)

        self.chkLuggage=Checkbutton(TravelFrame,text="Extra Luggage",variable =
var4, onvalue=1, offvalue=0,font=('arial',16,'bold'),command=Lug).grid(row=6,
column=0, sticky=W)
        self.txtLuggage=Label(TravelFrame,font=('arial',14,'bold'),textvariable=L
uggage,bd=6,width=18,bg="white",state= DISABLED,justify=RIGHT,relief=SUNKEN)
        self.txtLuggage.grid(row=6,column=1)

        #=====payment information
=====

```

```

        self.lblPaidTax=Label(CostFrame,font=('arial',14,'bold'),text="Paid
Tax\t\t",bd=7)
        self.lblPaidTax.grid(row=0,column=2,sticky=W)
        self.txtPaidTax =
Label(CostFrame,font=('arial',14,'bold'),textvariable=PaidTax,bd=7, width=10,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtPaidTax.grid(row=0,column=3)


        self.lblSubTotal=Label(CostFrame,font=('arial',14,'bold'),text="Sub
Total",bd=7)
        self.lblSubTotal.grid(row=1,column=2,sticky=W)
        self.txtSubTotal =
Label(CostFrame,font=('arial',14,'bold'),textvariable=SubTotal,bd=7, width=10,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtSubTotal.grid(row=1,column=3)


        self.lblTotalCost=Label(CostFrame,font=('arial',14,'bold'),text="Total
Cost",bd=7)
        self.lblTotalCost.grid(row=2,column=2,sticky=W)
        self.txtTotalCost =
Label(CostFrame,font=('arial',14,'bold'),textvariable=TotalCost,bd=7, width=10,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtTotalCost.grid(row=2,column=3)


        #=====Cabselect=====
=====


        self.chkStandard=Radiobutton(Book_Frame,text="Standard
Cab",value=1,variable =
carType,font=('arial',14,'bold'),command=selectCar).grid(row=0, column=0,
sticky=W)
        self.txtStandard = Label(Book_Frame,font=('arial',14,'bold'),width
=7,textvariable=Standard,bd=5, state= DISABLED,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtStandard.grid(row=0,column=1)


        self.chkFordGalaxyd=Radiobutton(Book_Frame,text="Ford Galaxy
Cab",value=2,variable =
carType,font=('arial',14,'bold'),command=selectCar).grid(row=1, column=0,
sticky=W)
        self.txtFordGalaxy= Label(Book_Frame,font=('arial',14,'bold'),width
=7,textvariable=FordGalaxy,bd=5, state= DISABLED,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtFordGalaxy.grid(row=1,column=1)

```

```

        self.chkFordMondeo = Radiobutton(Book_Frame,text="Ford Mondeo
Cab",value=3,variable =
carType,font=('arial',14,'bold'),command=selectCar).grid(row=2, column=0)
        self.txtFordMondeo = Label(Book_Frame,font=('arial',14,'bold'),width
=7,textvariable=FordMondeo,bd=5, state= DISABLED,
justify=RIGHT,bg="white",relief=SUNKEN)
        self.txtFordMondeo.grid(row=2,column=1)

        self.chkSingle =Radiobutton(Book_Frame,text="Single",value=1,variable =
journeyType,font=('arial',14,'bold')).grid(row=0, column=2, sticky=W)
        self.chkReturn =Radiobutton(Book_Frame,text="Return",value=2,variable =
journeyType,font=('arial',14,'bold')).grid(row=1, column=2, sticky=W)
        self.chkSpecialsNeeds
=Radiobutton(Book_Frame,text="SpecialNeeds",value=3,variable =
journeyType,font=('arial',14,'bold')).grid(row=2, column=2, sticky=W)

#=====Receipt=====
=====

        self.txtReceipt1 = Text(ReceiptFrame,width = 22, height =
21,font=('arial',10,'bold'),borderwidth=0)
        self.txtReceipt1.grid(row=0,column=0,columnspan=2)
        self.txtReceipt2 = Text(ReceiptFrame,width = 22, height =
21,font=('arial',10,'bold'),borderwidth=0)
        self.txtReceipt2.grid(row=0,column=2,columnspan=2)

#=====Button=====
=====

        self.btnTotal =
Button(ButtonFrame,padx=18,bd=7,font=('arial',11,'bold'),width =
2,text='Total',command=Total_Paid).grid(row=0,column=0)
        self.btnReceipt =
Button(ButtonFrame,padx=18,bd=7,font=('arial',11,'bold'),width =
2,text='Receipt',command=Receipttt).grid(row=0,column=1)
        self.btnReset =
Button(ButtonFrame,padx=18,bd=7,font=('arial',11,'bold'),width =
2,text='Reset',command=Reset).grid(row=0,column=2)
        self.btnExit =
Button(ButtonFrame,padx=18,bd=7,font=('arial',11,'bold'),width = 2,text='Exit',
command=iExit).grid(row=0,column=3)

#=====
=====

if __name__ == '__main__':

```

```
root = Tk()

#===== Getting Screen Width
=====
w = root.winfo_screenwidth()
h = root.winfo_screenheight()
geometry="%dx%d+%d+%d"%(w,h,0,0)

root.geometry("500x300+320+200")
root.title('Login Form')
application = user(root)
root.mainloop()
```

6. RESULTS

Login Form

Login Panel

Username: xyz@gmail.com

Password: *****

Login Create Account

Welcome tarun

Cab Booking System

Customer Info Firstname <input type="text"/> Surname <input type="text"/> Address <input type="text"/> Postcode <input type="text"/> Telephone <input type="text"/> Mobile <input type="text"/> Email <input type="text"/>	Booking Detail.. Pickup <input type="text"/> Drop <input type="text"/> Pooling <input type="text"/> 1 <input checked="" type="checkbox"/> Base Charge * <input type="text"/> <input checked="" type="checkbox"/> Distance(KMs) * <input type="text"/> <input checked="" type="checkbox"/> Travelling Insurance * <input type="text"/> <input checked="" type="checkbox"/> Extra Luggage <input type="text"/>	Receipt <div style="text-align: right;"> <input type="button" value="Total"/> <input type="button" value="Receipt"/> <input type="button" value="Reset"/> <input type="button" value="Exit"/> </div>
<input type="radio"/> Standard Cab <input type="text"/> 0 <input type="radio"/> Single <input type="radio"/> Ford Galaxy Cab <input type="text"/> 0 <input type="radio"/> Return <input type="radio"/> Ford Mondeo Cab <input type="text"/> 0 <input type="radio"/> SpecialNeeds	Paid Tax <input type="text"/> Sub Total <input type="text"/> Total Cost <input type="text"/>	

7. CONCLUSION

In conclusion, while the provided code serves the basic functionality of a taxi booking system, there is room for refinement in terms of code structure, documentation, and user interface enhancements to make it more robust and user-friendly.

The provided code appears to be a simple taxi booking system implemented in Python using the Tkinter library for the graphical user interface. The program consists of two main classes, 'user' and 'travel', representing the login interface and the cab booking system, respectively. The login interface enables users to either log in with existing credentials or create a new account, storing user information in a SQLite database.

In the 'travel' class, the main graphical user interface for the cab booking system is established. Users can input their personal details, choose pickup and drop-off locations, select cab options, and view the total cost of their booking. The code incorporates various Tkinter widgets such as labels, entry fields, buttons, and checkboxes to facilitate user interaction..

Several functions are implemented to handle different aspects of the system, including calculating costs based on user selections, generating receipts, and resetting the form for a new booking. The program also includes error handling to notify users of invalid inputs..

However, the code could benefit from improvements and enhancements. For instance, it lacks proper documentation and comments, making it challenging for someone else to understand the code easily. Additionally, the structure of the code could be refined by organizing functions more systematically and adhering to best practices. Furthermore, there are opportunities to enhance the user interface and user experience, such as implementing input validation and improving the overall design..

the provided code establishes a foundation for a basic Cab Booking System with user authentication. However, there is room for improvement in terms of code structure, documentation, and error handling to enhance its overall reliability and maintainability.

REFERENCES

- D. Santani, R. K. Balan and C. J. Woodard, "Spatio-temporal efficiency in a taxi dispatch system", 6th International Conference on Mobile Systems Applications and Services MobiSys, October, 2008.
- > C. Wang, W. K. Ng and H. Chen, "From data to knowledge to action: A taxi business intelligence system", 15th International Conference on Information Fusion (FUSION), pp. 1623-1628, July 2012.
- > H. Wang, D. H. Lee and R. Cheu, "PDPTW based taxi dispatch modeling for booking service", 5th International Conference on Natural Computation, vol. 1, pp. 242-247, August, 2009.