SU5
Travel Agency App

Table of Contents

Acknowledgement

We are here to express our deepest gratitude to **Dr.Cynthia** and **Mrs.Autumn** for your invaluable support and guidance throughout our Python Programming Essential Batch 1 and final project on Python Programming and MySQL database (**SU5 Travel Agency Application**). Your unwavering dedication and expertise provided us with a golden opportunity to delve deeply into the subject matter, significantly enhancing our knowledge and skills.

Your patience, encouragement, and willingness to help us understand complex concepts were instrumental in the successful completion of our project. The insights we gained from your teachings and the rigorous research process have been incredibly beneficial, and we are confident that these will serve us well in our future endeavors.

Once again, thank you for your mentorship and for inspiring us to strive for excellence. We are truly grateful for the opportunity to learn from you.

Last but not least, We are here to express our heartfelt appreciation for the incredible effort and dedication each of you put into our final project on Python Programming and MySQL database (**SU5 Travel Agency Application**).

Working alongside such a committed and talented team has been a rewarding experience. Your hard work, creative ideas, and unwavering determination were key factors in finalizing the project within our limited time frame. Each of you contributed significantly, and it was truly a collaborative effort that led to our success.

Thank you for your exceptional teamwork and for making this project a memorable and educational journey. I am proud of what we have achieved together and look forward to any future collaborations.

With sincere and special thanks to -

Student ID - SSM (14796,15441,15444, 15445,15449)

Introduction

SU5 Travel Agency Application is an application platform designed to streamline and enhance the travel booking and management process for travel agents and their customers. These applications typically offer a variety of features to facilitate different aspects of travel planning, booking, and customer service. Here's an overview of what a comprehensive travel agency application include:

Key Features

1. **Customers and Booking**
    - **Flight Reservations:** Integration with global distribution systems (GDS) and airlines for booking flights.
    - **Hotel Bookings:** Access to a wide range of hotels, allowing users to compare prices and amenities.
    - **Car Rentals:** Options to book rental cars from various providers.
    - **Package Deals:** Combination of flights, hotels, and car rentals into travel packages.
    - **Cruises and Tours:** Booking options for cruises and guided tours.
2. **Customer Management with MySQL Database**
    - **User Profiles:** Store customer information, preferences, and past travel history.
    - **Personalized Recommendations:** Suggest travel options based on user preferences and past behavior.
    - **Communication Tools:** Email and messaging systems for communicating with clients.
    - **Trend Analysis:** Able to tabulate traveling trend analysis based only on the database record.

Benefits

- **Efficiency:** Automates many aspects of travel planning and booking, saving time for both agents and customers.
- **Accuracy:** Reduces errors in booking and management with automated systems and real-time data.
- **Customer Satisfaction:** Enhances the customer experience with personalized service and streamlined communication.
- **Competitive Edge:** Helps travel agencies remain competitive by offering a wide range of services and modern conveniences.

System Requirements

- Operating System: Windows 7 or higher
- Python 3.6 or higher
- MySQL Server 5.7 or higher
- Required Python packages: tkinter, PIL (Pillow), mysql-connector-python

Installation Method

Step-by-Step Installation Guide

1. **Install Python**: Download and install Python from the official Python website.
2. **Install MySQL Server**: Download and install MySQL Server from the official MySQL website.
3. **Install Required Python Packages**: Open your command prompt and run the following commands:

```sh
pip install tkinter
pip install Pillow
pip install mysql-connector-python
```

4. **Set Up Database**:

- Open MySQL Workbench or MySQL command line.
- Create a new database named ‘ SUM5TA’ :

```sql
CREATE DATABASE SUM5TA;
```

Application Overview

The **SU5 Travel Agency Application** allows users to manage customers, trips, and bookings. The main features include adding customers, adding trips, creating bookings, and confirming bookings.

Using the Application

Main Page

1. **Start the Application**: Run the `TravelAgencyApp` by executing the `main()` function in the script.
2. **Main Page**: The main page displays a welcome image. Click "Enter" to access the main application tabs.

Customers Tab

1. **Add Customer**: Fill in the customer's details (Name, Email, Phone, Passport, Address) and click "Add Customer".
2. **Customer List**: View the list of added customers.

Trips Tab

1. **Add Trip**: Fill in the trip details (Destination, Option, Duration, Price) and click "Add Trip".
2. **Trip List**: View the list of added trips.

Bookings Tab

1. **Create Booking**:
    - Select a customer from the dropdown.
    - Select a trip from the dropdown.
    - Click "Create Booking" to create the booking.
2. **Confirm Booking**:
    - Select a booking from the list.
    - Click "Confirm Booking" to confirm the booking.

Imported Files and Functions

The script imports several modules and functions:

1. **tkinter**:
    - `tk`
    - `ttk`
    - `messagebox`
2. **PIL**:
    - `Image`
    - `ImageTk`
3. **mysql.connector**:
    - `connect`
    - `Error`

Database Schema

The database consists of three main tables:

1. **Customers**:
    - `id`: INT, AUTO_INCREMENT, PRIMARY KEY
    - `name`: VARCHAR(255), NOT NULL
    - `email`: VARCHAR(255), NOT NULL, UNIQUE
    - `phone`: VARCHAR(50), NOT NULL
    - `passport`: VARCHAR(50), NOT NULL
    - `address`: TEXT, NOT NULL
2. **Trips**:
    - `id`: INT, AUTO_INCREMENT, PRIMARY KEY
    - `destination`: VARCHAR(255), NOT NULL
    - `option`: VARCHAR(255), NOT NULL
    - `duration`: INT, NOT NULL
    - `price`: FLOAT, NOT NULL
3. **Bookings**:
    - `id`: INT, AUTO_INCREMENT, PRIMARY KEY
    - `customer_id`: INT, NOT NULL, FOREIGN KEY
    - `trip_id`: INT, NOT NULL, FOREIGN KEY
    - `status`: VARCHAR(50), DEFAULT 'Not Confirmed'

Error Handling and Troubleshooting:

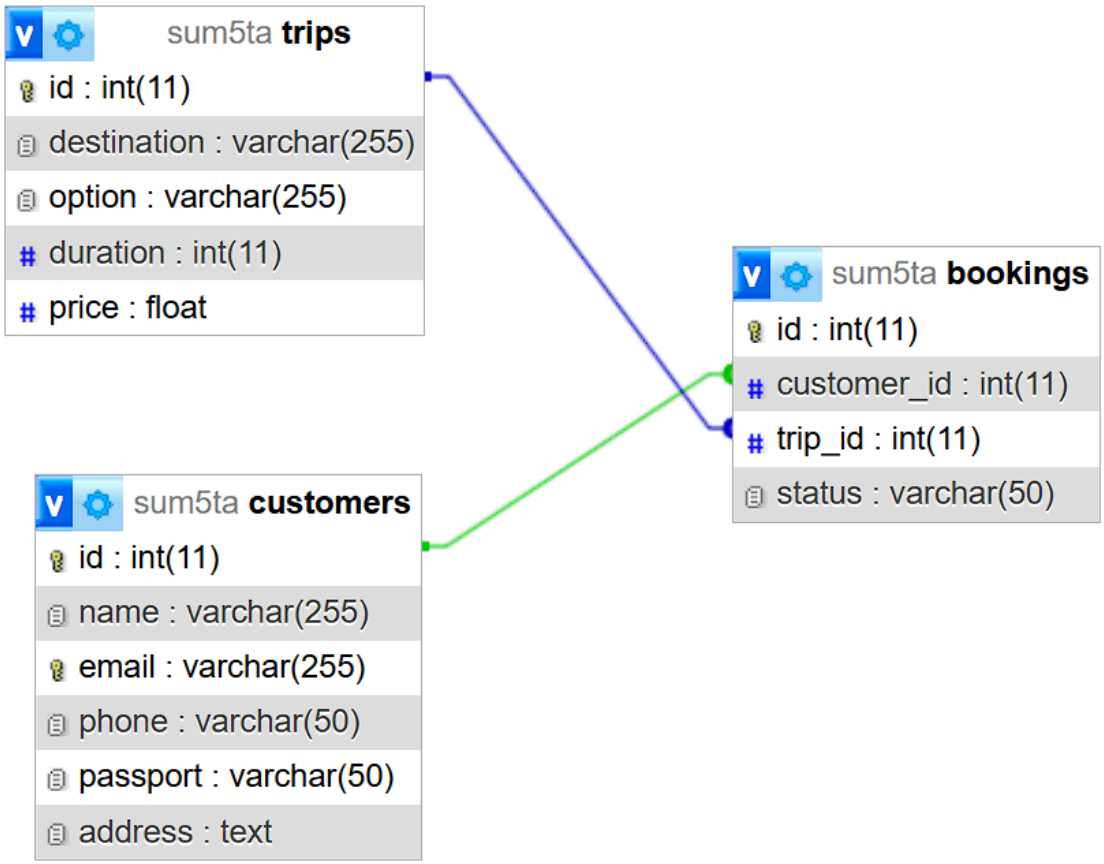
- **Database Connection Errors**: Ensure MySQL server is running and credentials are correct.
- **Input Validation Errors**: Ensure all fields are filled in before submitting.
- **Booking Errors**: Ensure valid customers and trips are selected before creating or confirming a booking.

Contact Information

For further assistance, please contact:

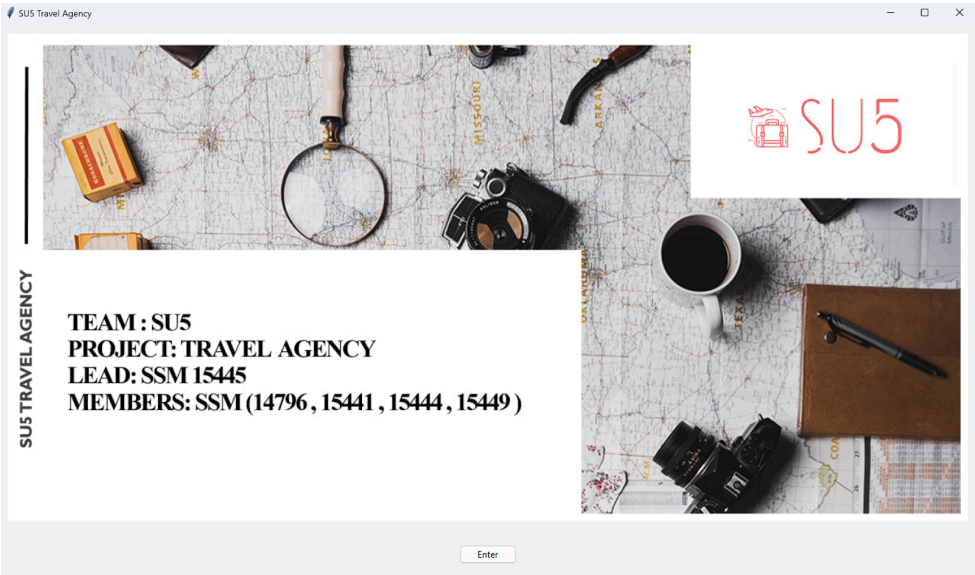
- **Support Telegram**: @Spring , @Sc_arlet , @Catkilled , @mr_n0m3rcy , @Anonymous2513
- **Phone**: TBA

Database Designer

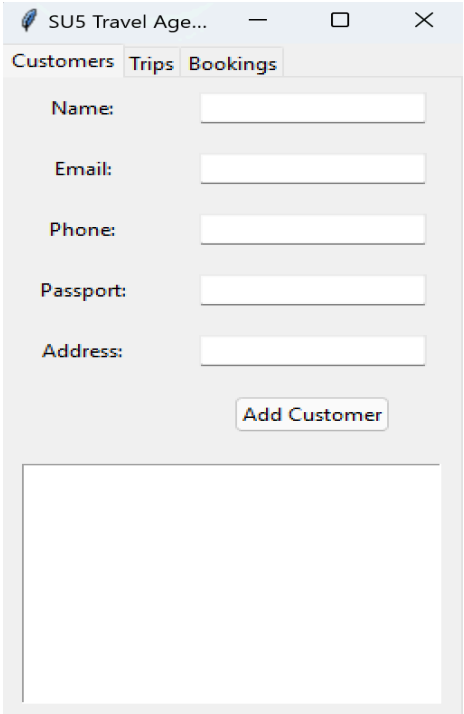


User Interface

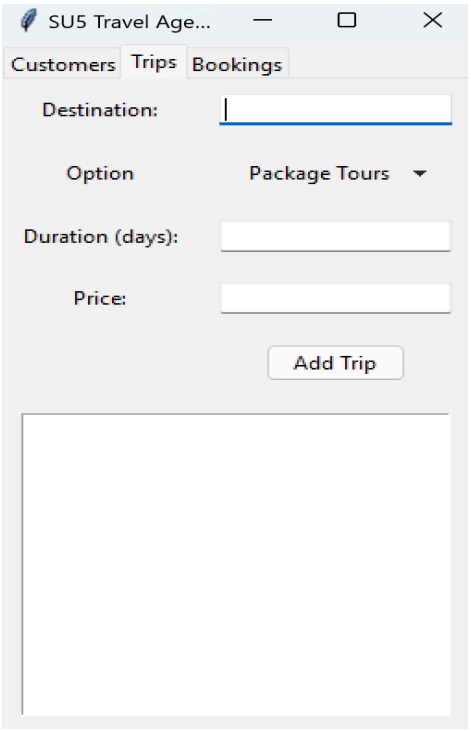
Main Page



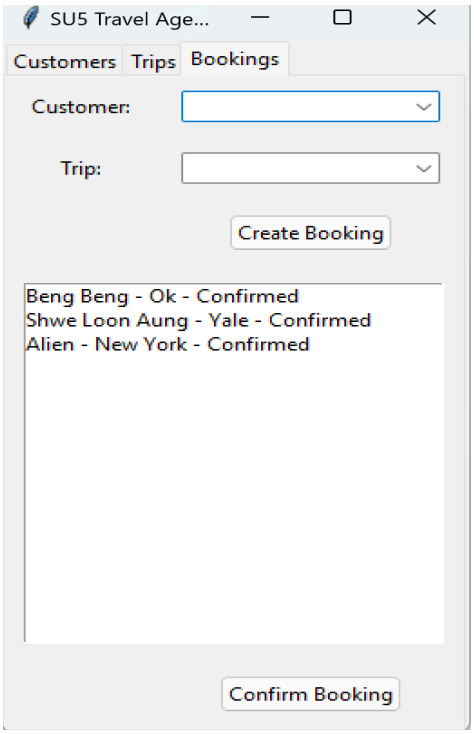
Customer Tab



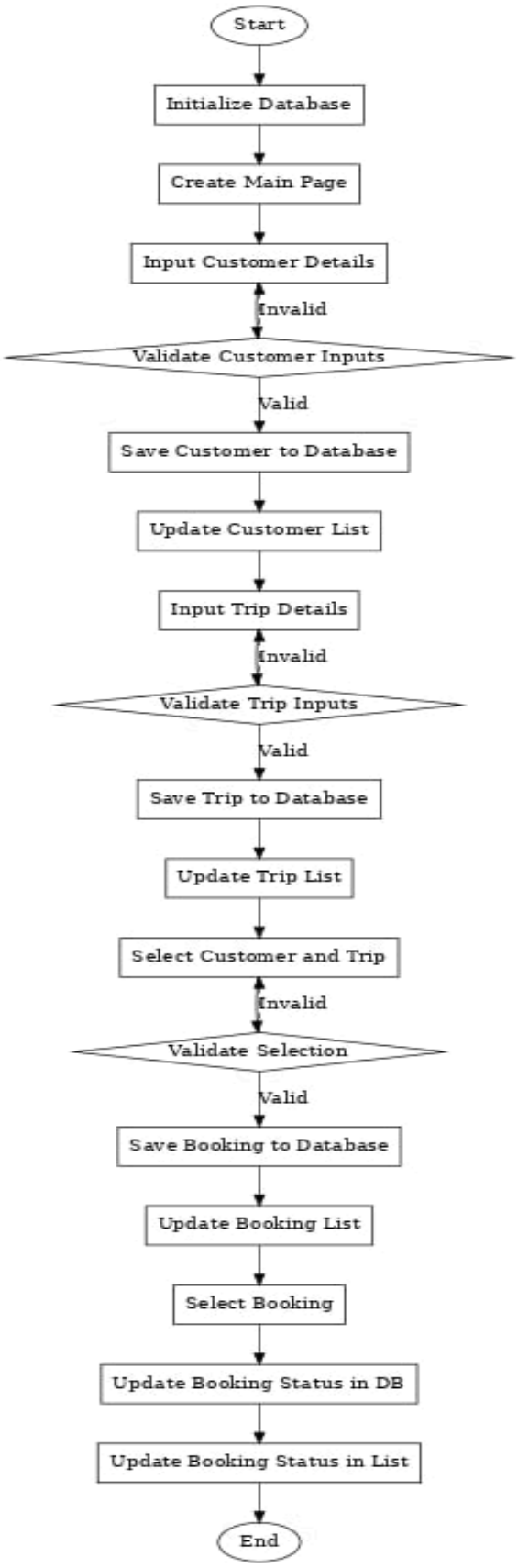
Trips Tab



Booking Tab



FlowChart


Start
Initialize Database
Create Main Page
Input Customer Details
Invalid
Validate Customer Inputs
Valid
Save Customer to Database
Update Customer List
Input Trip Details
Invalid
Validate Trip Inputs
Valid
Save Trip to Database
Update Trip List
Select Customer and Trip
Invalid
Validate Selection
Valid
Save Booking to Database
Update Booking List
Select Booking
Update Booking Status in DB
Update Booking Status in List
End

Code

Here's the complete code:

```
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import mysql.connector
from mysql.connector import Error

class Customer:
    def __init__(self, name, email, phone, passport, address):
        self.name = name
        self.email = email
        self.phone = phone
        self.passport = passport
        self.address = address

    def __str__(self):
        return f"Customer(name={self.name}, email={self.email},
phone={self.phone}, passport={self.passport}, address={self.address})"

class Trip:
    def __init__(self, destination, option, duration, price):
        self.destination = destination
        self.option = option
        self.duration = duration
        self.price = price

    def __str__(self):
        return f"Trip(destination={self.destination},
option={self.option}, duration={self.duration} days,
price=${self.price})"

class Booking:
```

```
    def __init__(self, customer, trip):
        self.customer = customer
        self.trip = trip

    def __str__(self):
        return f"Booking(Customer={self.customer.name},
Trip={self.trip.destination})"

class TravelAgencyApp:

    def __init__(self, root):
        self.root = root
        self.root.title("SU5 Travel Agency")

        self.customers = []
        self.trips = []
        self.bookings = []

        self.database_init()
        self.create_main_page()

    def database_init(self):
        try:
            self.mydb = mysql.connector.connect(
                host='localhost',
                user='root',
                password='',
                database='SUM5TA'
            )

            if self.mydb.is_connected():
                self.cursor = self.mydb.cursor()

                self.cursor.execute('''
```

```
            CREATE TABLE IF NOT EXISTS customers (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                email VARCHAR(255) NOT NULL UNIQUE,
                phone VARCHAR(50) NOT NULL,
                passport VARCHAR(50) NOT NULL,
                address TEXT NOT NULL
            )
            ''')

            self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS trips (
                id INT AUTO_INCREMENT PRIMARY KEY,
                destination VARCHAR(255) NOT NULL,
                option VARCHAR(255) NOT NULL,
                duration INT NOT NULL,
                price FLOAT NOT NULL
            )
            ''')

            self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS bookings (
                id INT AUTO_INCREMENT PRIMARY KEY,
                customer_id INT NOT NULL,
                trip_id INT NOT NULL,
                status VARCHAR(50) DEFAULT 'Not Confirmed',
                FOREIGN KEY (customer_id) REFERENCES customers(id),
                FOREIGN KEY (trip_id) REFERENCES trips(id)
            )
            ''')

            self.mydb.commit()
        except Error as e:
            print(f"Error: {e}")
```

```
            if self.mydb.is_connected():
                self.mydb.close()

    def create_main_page(self):
        self.main_frame = ttk.Frame(self.root)
        self.main_frame.pack(expand=1, fill='both')
        self.load_image(self.main_frame)
        enter_button = ttk.Button(self.main_frame, text="Enter",
command=self.create_widgets)
        enter_button.pack(pady=20)

    def load_image(self, frame):
        # Path to the image file
        image_path = "C:/Users/leish/OneDrive/Desktop/SUM_Python/SU5
Cover Page.jpg"
        try:
            # Load the image using PIL
            image = Image.open(image_path)
            photo = ImageTk.PhotoImage(image)

            # Create a label widget to display the image
            image_label = tk.Label(frame, image=photo)
            image_label.image = photo  # Keep a reference to avoid
garbage collection
            image_label.pack(padx=10, pady=10)
        except IOError:
            print(f"Unable to load image at path: {image_path}")

    def create_widgets(self):
        self.main_frame.destroy()  # Remove the main frame

        self.tab_control = ttk.Notebook(self.root)  # Changed to
self.tab_control
```

```
        self.customer_tab = ttk.Frame(self.tab_control)
        self.trip_tab = ttk.Frame(self.tab_control)
        self.booking_tab = ttk.Frame(self.tab_control)

        self.tab_control.add(self.customer_tab, text='Customers')
        self.tab_control.add(self.trip_tab, text='Trips')
        self.tab_control.add(self.booking_tab, text='Bookings')

        self.tab_control.pack(expand=1, fill='both')

        self.create_customer_tab()
        self.create_trip_tab()
        self.create_booking_tab()

    def create_customer_tab(self):
        ttk.Label(self.customer_tab, text="Name:").grid(column=0, row=0,
padx=10, pady=10)
        self.customer_name = ttk.Entry(self.customer_tab)
        self.customer_name.grid(column=1, row=0, padx=10, pady=10)

        ttk.Label(self.customer_tab, text="Email:").grid(column=0,
row=1, padx=10, pady=10)
        self.customer_email = ttk.Entry(self.customer_tab)
        self.customer_email.grid(column=1, row=1, padx=10, pady=10)

        ttk.Label(self.customer_tab, text="Phone:").grid(column=0,
row=2, padx=10, pady=10)
        self.customer_phone = ttk.Entry(self.customer_tab)
        self.customer_phone.grid(column=1, row=2, padx=10, pady=10)

        ttk.Label(self.customer_tab, text="Passport:").grid(column=0,
row=3, padx=10, pady=10)
        self.customer_passport = ttk.Entry(self.customer_tab)
        self.customer_passport.grid(column=1, row=3, padx=10, pady=10)
```

```
        ttk.Label(self.customer_tab, text="Address:").grid(column=0,
row=4, padx=10, pady=10)
        self.customer_address = ttk.Entry(self.customer_tab)
        self.customer_address.grid(column=1, row=4, padx=10, pady=10)

        ttk.Button(self.customer_tab, text="Add Customer",
command=self.add_customer).grid(column=1, row=5, padx=10, pady=10)

        self.customer_list = tk.Listbox(self.customer_tab)
        self.customer_list.grid(column=0, row=6, columnspan=2, padx=10,
pady=10, sticky='nsew')

        self.customer_tab.grid_columnconfigure(0, weight=1)
        self.customer_tab.grid_columnconfigure(1, weight=1)
        self.customer_tab.grid_rowconfigure(6, weight=1)

    def create_trip_tab(self):
        ttk.Label(self.trip_tab, text="Destination:").grid(column=0,
row=0, padx=10, pady=10)
        self.trip_destination = ttk.Entry(self.trip_tab)
        self.trip_destination.grid(column=1, row=0, padx=10, pady=10)

        ttk.Label(self.trip_tab, text="Option").grid(column=0, row=1,
padx=10, pady=10)
        self.option_var = tk.StringVar(value="Select an option")
        options = ["Bus Express ", "Package Tours", "Cruises", "Flight
Ticket", "Hotel", "Trekking"]
        self.trip_option = ttk.OptionMenu(self.trip_tab,
self.option_var, *options)
        self.trip_option.grid(column=1, row=1, padx=10, pady=10)

        ttk.Label(self.trip_tab, text="Duration (days):").grid(column=0,
row=3, padx=10, pady=10)
```

```
        self.trip_duration = ttk.Entry(self.trip_tab)
        self.trip_duration.grid(column=1, row=3, padx=10, pady=10)

        ttk.Label(self.trip_tab, text="Price:").grid(column=0, row=4,
padx=10, pady=10)
        self.trip_price = ttk.Entry(self.trip_tab)
        self.trip_price.grid(column=1, row=4, padx=10, pady=10)

        ttk.Button(self.trip_tab, text="Add Trip",
command=self.add_trip).grid(column=1, row=5, padx=10, pady=10)

        self.trip_list = tk.Listbox(self.trip_tab)
        self.trip_list.grid(column=0, row=6, columnspan=2, padx=10,
pady=10, sticky='nsew')

        self.trip_tab.grid_columnconfigure(0, weight=1)
        self.trip_tab.grid_columnconfigure(1, weight=1)
        self.trip_tab.grid_rowconfigure(6, weight=1)

    def create_booking_tab(self):
        #Create Custome label adn combobox
        ttk.Label(self.booking_tab, text="Customer:").grid(column=0,
row=0, padx=10, pady=10)
        self.booking_customer = ttk.Combobox(self.booking_tab)
        self.booking_customer.grid(column=1, row=0, padx=10, pady=10)

        #Create Trip laabel and cobobox
        ttk.Label(self.booking_tab, text="Trip:").grid(column=0, row=1,
padx=10, pady=10)
        self.booking_trip = ttk.Combobox(self.booking_tab)
        self.booking_trip.grid(column=1, row=1, padx=10, pady=10)

        #Create - Create Booking button
        ttk.Button(self.booking_tab, text="Create Booking",
```

```
command=self.create_booking).grid(column=1, row=2, padx=10, pady=10)

        #Create Booking list
        self.booking_list = tk.Listbox(self.booking_tab)
        self.booking_list.grid(column=0, row=3, columnspan=2, padx=10,
pady=10, sticky='nsew')

        #Create Confirm Booking button
        ttk.Button(self.booking_tab, text="Confirm Booking",
command=self.confirm_booking).grid(column=1, row=4, padx=10, pady=10)

        #Create grid weights for responsiveness
        self.booking_tab.grid_columnconfigure(0, weight=1)
        self.booking_tab.grid_columnconfigure(1, weight=1)
        self.booking_tab.grid_rowconfigure(3, weight=1)

        #Load initial booking data
        self.load_booking_data()

    def on_tab_selected(self, event):
        selected_tab = event.widget.tab('current')['text']
        if selected_tab == 'Bookings':
            self.load_booking_data()

    def load_booking_data(self):
        # Load customers from the database
        trip_destinations = []
        try:
            self.cursor.execute('SELECT * FROM customers')
            customer_rows = self.cursor.fetchall()
            for customer_row in customer_rows: #
structure(self.customers) = [Customer, Customer]
                self.customers.append(Customer(name=customer_row[1],
email=customer_row[2], phone=customer_row[3], passport=customer_row[4],
```

```
address=customer_row[5]))
            customer_names = [customer.name for customer in
self.customers]  # Assuming name is the second element
            self.booking_customer['values'] = customer_names
        except Exception as e:
            print(f"Error loading customers: {e}")

    # Load trips from the database

        try:
            self.cursor.execute('SELECT * FROM trips')
            trip_rows = self.cursor.fetchall()
            for row in trip_rows:
                self.trips.append(Trip(row[1], row[2], row[3], row[4]))
                trip_destinations.append(row[1])  # Assuming destination
is the second element
            self.booking_trip['values'] = trip_destinations
        except Exception as e:
            print(f"Error loading trips: {e}")

    # Clear and reload the bookings list
        self.booking_list.delete(0, tk.END)
        try:
            self.cursor.execute('''
            SELECT c.name, t.destination, b.status
            FROM bookings b
            JOIN customers c ON b.customer_id = c.id
            JOIN trips t ON b.trip_id = t.id
            ''')
            booking_rows = self.cursor.fetchall()
            self.bookings = [list(row) for row in booking_rows]  # Store
as array (list of lists)
            for booking in self.bookings:
                booking_info = f"{booking[0]} - {booking[1]} -
```

```
{booking[2]}"  # Format booking info
                self.booking_list.insert(tk.END, booking_info)
        except Exception as e:
            print(f"Error loading bookings: {e}")

    def add_customer(self):
        name = self.customer_name.get()
        email = self.customer_email.get()
        phone = self.customer_phone.get()
        passport = self.customer_passport.get()
        address = self.customer_address.get()

        if name and email and phone and passport and address:
            customer = Customer(name, email, phone, passport, address)
            self.customers.append(customer)

            #insert into table
            try:
                self.cursor.execute('''
                INSERT INTO customers (name, email, phone, passport,
address)
                VALUES (%s, %s, %s, %s, %s)
                ''', (name, email, phone, passport, address))
                self.mydb.commit()
                print(f"- Customer Added : {name}, {passport}.")
            except Error as e:
                print(f"Error: {e}")

            self.customer_list.insert(tk.END, str(customer))
            self.booking_customer['values'] = [customer.name for
customer in self.customers]
            self.customer_name.delete(0, tk.END)
            self.customer_email.delete(0, tk.END)
            self.customer_phone.delete(0, tk.END)
```

```
            self.customer_passport.delete(0, tk.END)
            self.customer_address.delete(0, tk.END)

            # Move to the next tab
            self.tab_control.select(self.trip_tab)
        else:
            messagebox.showwarning("Input Error", "Please fill all
fields")

    def add_trip(self):
        destination = self.trip_destination.get()
        option = self.option_var.get()
        duration = self.trip_duration.get()
        price = self.trip_price.get()

        if destination and duration and price and option != "Select an
option":
            trip = Trip(destination, option, int(duration),
float(price))
            self.trips.append(trip)

            #insert into table
            try:
                self.cursor.execute('''
                INSERT INTO trips (destination, option, duration, price)
                VALUES (%s, %s, %s, %s)
                ''', (destination, option, duration, price))
                self.mydb.commit()
                print(f"- Trip Added : {destination}, {price}.")
            except Error as e:
                print(f"Error: {e}")
            self.trip_list.insert(tk.END, str(trip))
            self.booking_trip['values'] = [trip.destination for trip in
self.trips]
```

```
            self.trip_destination.delete(0, tk.END)
            self.trip_duration.delete(0, tk.END)
            self.trip_price.delete(0, tk.END)

            # Move to the next tab
            self.tab_control.select(self.booking_tab)
        else:
            messagebox.showwarning("Input Error", "Please fill all
fields")

    def create_booking(self):
        customer_name = self.booking_customer.get()
        trip_destination = self.booking_trip.get()
        customer = next((c for c in self.customers if c.name ==
customer_name), None)
        trip = next((t for t in self.trips if t.destination ==
trip_destination), None)
        if customer and trip:
            #print(customer_name, trip_destination)
            booking = Booking(customer, trip)
            self.bookings.append(booking)

            try:
                self.cursor.execute('''
                INSERT INTO bookings (customer_id, trip_id, status)
                VALUES (
                    (SELECT id FROM customers WHERE name = %s),
                    (SELECT id FROM trips WHERE destination = %s),
                    'Not Confirmed'
                )
                ''', (customer_name, trip_destination))
                self.mydb.commit()
                print(f"- Booking created: {customer_name},
{trip_destination}.")
```

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


            self.booking_list.insert(tk.END, f"{str(booking)} - Not
Confirmed")
        else:
            messagebox.showwarning("Input Error", "Please select valid
customer and trip")

    def confirm_booking(self):
        try:
            selected_index = self.booking_list.curselection()[0]
            selected_booking = self.bookings[selected_index]
            self.cursor.execute('''
            SELECT id FROM bookings WHERE customer_id = (SELECT id FROM
customers WHERE name = %s)
            AND trip_id = (SELECT id FROM trips WHERE destination = %s)
            ''', (selected_booking.customer.name,
selected_booking.trip.destination))
            result = self.cursor.fetchone()

            if result is None:
                raise ValueError("Booking not found in the database")

            booking_id= result[0]

            self.cursor.execute('''
            UPDATE bookings SET status = 'Confirmed' WHERE id = %s
            ''', (booking_id,))
            self.mydb.commit()

            # Update the display in the listbox
            self.booking_list.delete(selected_index)
```

```
            self.booking_list.insert(selected_index,
f"{str(selected_booking)} - Confirmed")

            self.show_booking_confirmation(selected_booking)
            print(f"- Booking confirmed:
{selected_booking.customer.name}, {selected_booking.trip.destination}.")
        except IndexError:
            messagebox.showwarning("Selection Error", "Please select a
booking to confirm")

    def show_booking_confirmation(self, booking):
        customer_info = f"Customer: {booking.customer.name}, Email:
{booking.customer.email}, Phone: {booking.customer.phone}, Passport:
{booking.customer.passport}, Address: {booking.customer.address}"
        trip_info = f"Trip: {booking.trip.destination}, Option:
{booking.trip.option}, Duration: {booking.trip.duration} days, Price:
${booking.trip.price}"

        confirmation_message = f"Booking
Confirmed!\n\n{customer_info}\n{trip_info}"
        messagebox.showinfo("Booking Confirmation",
confirmation_message)
        confirmation_window = tk.Toplevel(self.root)
        confirmation_window.title("Booking Confirmation")

        tk.Label(confirmation_window, text=confirmation_message,
padx=20, pady=20).pack()
        ttk.Button(confirmation_window, text="Confirm", command=lambda:
self.close_confirmation_window(confirmation_window)).pack(pady=10)

    def close_confirmation_window(self, window):
        window.destroy()
        self.reset_main_page()
```

```
    def reset_main_page(self):
        self.tab_control.destroy()
        self.create_main_page()

def main():
    root = tk.Tk()
    app = TravelAgencyApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```