**customer\_login.py**

import tkinter as tk

from tkinter import ttk, messagebox

from database import get\_db\_connection

import re

class PasswordValidationError(Exception):

pass

def validate\_password(password):

pattern = r'^(?=.\*[!@#$%^&\*(),.?":{}|<>])[A-Za-z\d!@#$%^&\*(),.?":{}|<>]{8,}$'

if not re.match(pattern, password):

raise PasswordValidationError("Password must be at least 8 characters long and contain at least one special character (!@#$%^&\* etc.)")

def validate\_email(email):

pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

if not re.match(pattern, email):

raise ValueError("Invalid email format")

def customer\_login(email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM customer WHERE email = %s AND password = %s", (email, password))

customer = cursor.fetchone()

conn.close()

return customer

def register\_customer(name, email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("INSERT INTO customer (name, email, password) VALUES (%s, %s, %s)",(name, email, password))

conn.commit()

return True

except Exception as e:

return str(e)

finally:

conn.close()

class CustomerLogin:

def \_\_init\_\_(self, on\_login\_success):

self.on\_login\_success = on\_login\_success

self.root = tk.Tk()

self.root.title("Customer Login")

self.root.geometry("500x500")

self.style = ttk.Style()

self.style.theme\_use('clam')

self.style.configure('.', background="#FFFFFF", foreground="#333333")

self.style.configure('TButton', font=('Arial', 10, 'bold'), padding=8, background="#0E2A5B", foreground="white")

self.create\_login\_page()

def create\_login\_page(self):

for widget in self.root.winfo\_children():

widget.destroy()

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame, text="Customer Login", font=('Arial', 18, 'bold'), foreground="#0E2A5B").pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Email:").pack(pady=5)

self.email\_entry = ttk.Entry(main\_frame)

self.email\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password:").pack(pady=5)

self.password\_entry = ttk.Entry(main\_frame, show="\*")

self.password\_entry.pack(pady=5)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Login", command=self.login, style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Register", command=self.show\_register\_page, style='TButton').pack(side=tk.RIGHT, padx=5)

def login(self):

email = self.email\_entry.get()

password = self.password\_entry.get()

try:

customer = customer\_login(email, password)

if customer:

messagebox.showinfo("Login Successful", "Welcome!")

self.root.destroy()

self.on\_login\_success(customer[0], customer[1])

else:

messagebox.showerror("Login Failed", "Invalid credentials")

except Exception as e:

messagebox.showerror("Error", f"Login failed: {str(e)}")

def register(self):

name = self.register\_name\_entry.get()

email = self.register\_email\_entry.get()

password = self.register\_password\_entry.get()

try:

validate\_email(email)

validate\_password(password)

result = register\_customer(email, password)

if result is True:

messagebox.showinfo("Registration Successful", "You can now log in.")

self.create\_login\_page()

else:

messagebox.showerror("Registration Failed", result)

except (PasswordValidationError, ValueError) as e:

messagebox.showerror("Registration Failed", str(e))

def show\_register\_page(self):

for widget in self.root.winfo\_children():

widget.destroy()

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame, text="Customer Registration", font=('Arial', 18, 'bold'), foreground="#0E2A5B").pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Name:").pack()

self.register\_name\_entry = ttk.Entry(main\_frame)

self.register\_name\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Email:").pack(pady=5)

self.register\_email\_entry = ttk.Entry(main\_frame)

self.register\_email\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password:").pack(pady=5)

self.register\_password\_entry = ttk.Entry(main\_frame, show="\*")

self.register\_password\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password must be at least 8 characters with special characters", font=('Arial', 8), foreground="#666666").pack(pady=5)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Register", command=self.register, style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Back to Login", command=self.create\_login\_page, style='TButton').pack(side=tk.RIGHT, padx=5)

def run(self):

self.root.mainloop()

**admin\_login.py**

import tkinter as tk

from tkinter import ttk, messagebox

from database import get\_db\_connection

import re

class PasswordValidationError(Exception):

pass

def validate\_password(password):

pattern = r'^(?=.\*[!@#$%^&\*(),.?":{}|<>])[A-Za-z\d!@#$%^&\*(),.?":{}|<>]{8,}$'

if not re.match(pattern, password):

raise PasswordValidationError("Password must be at least 8 characters long and contain at least one special character (!@#$%^&\* etc.)")

def validate\_email(email):

pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

if not re.match(pattern, email):

raise ValueError("Invalid email format")

def customer\_login(email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM customer WHERE email = %s AND password = %s", (email, password))

customer = cursor.fetchone()

conn.close()

return customer

def register\_customer(email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("INSERT INTO customer (email, password) VALUES (%s, %s)", (email, password))

conn.commit()

return True

except Exception as e:

return str(e)

finally:

conn.close()

def admin\_login(email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM admin WHERE email = %s AND password = %s", (email, password))

admin = cursor.fetchone()

conn.close()

return admin

class CustomerLogin:

def \_\_init\_\_(self, on\_login\_success):

self.on\_login\_success = on\_login\_success

self.root = tk.Tk()

self.root.title("Customer Login")

self.root.geometry("400x350")

self.style = ttk.Style()

self.style.theme\_use('clam')

self.style.configure('.', background="#FFFFFF", foreground="#333333")

self.style.configure('TButton', font=('Arial', 10, 'bold'), padding=8, background="#0E2A5B", foreground="white")

self.create\_login\_page()

def create\_login\_page(self):

for widget in self.root.winfo\_children():

widget.destroy()

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame, text="Customer Login", font=('Arial', 18, 'bold'), foreground="#0E2A5B").pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Email:").pack(pady=5)

self.email\_entry = ttk.Entry(main\_frame)

self.email\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password:").pack(pady=5)

self.password\_entry = ttk.Entry(main\_frame, show="\*")

self.password\_entry.pack(pady=5)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Login", command=self.login, style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Register", command=self.show\_register\_page, style='TButton').pack(side=tk.RIGHT, padx=5)

def login(self):

email = self.email\_entry.get()

password = self.password\_entry.get()

try:

validate\_email(email)

validate\_password(password)

except (PasswordValidationError, ValueError) as e:

messagebox.showerror("Login Failed", str(e))

customer = customer\_login(email, password)

if customer:

messagebox.showinfo("Login Successful", "Welcome, Customer!")

self.root.destroy()

self.on\_login\_success(customer[0])

else:

messagebox.showerror("Login Failed", "Invalid email or password.")

def register(self):

email = self.register\_email\_entry.get()

password = self.register\_password\_entry.get()

try:

validate\_email(email)

validate\_password(password)

result = register\_customer(email, password)

if result is True:

messagebox.showinfo("Registration Successful", "You can now log in.")

self.create\_login\_page()

else:

messagebox.showerror("Registration Failed", result)

except (PasswordValidationError, ValueError) as e:

messagebox.showerror("Registration Failed", str(e))

def show\_register\_page(self):

for widget in self.root.winfo\_children():

widget.destroy()

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame, text="Customer Registration", font=('Arial', 18, 'bold'), foreground="#0E2A5B").pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Email:").pack(pady=5)

self.register\_email\_entry = ttk.Entry(main\_frame)

self.register\_email\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password:").pack(pady=5)

self.register\_password\_entry = ttk.Entry(main\_frame, show="\*")

self.register\_password\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password must be at least 8 characters with special characters", font=('Arial', 8), foreground="#666666").pack(pady=5)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Register", command=self.register, style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Back to Login", command=self.create\_login\_page, style='TButton').pack(side=tk.RIGHT, padx=5)

def run(self):

self.root.mainloop()

class AdminLogin:

def \_\_init\_\_(self, on\_login\_success):

self.on\_login\_success = on\_login\_success

self.root = tk.Tk()

self.root.title("Admin Login")

self.root.geometry("500x500")

self.style = ttk.Style()

self.style.theme\_use('clam')

self.style.configure('.', background="#FFFFFF", foreground="#333333")

self.style.configure('TButton', font=('Arial', 10, 'bold'), padding=8, background="#0E2A5B", foreground="white")

self.create\_login\_page()

def create\_login\_page(self):

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame, text="Admin Login", font=('Arial', 18, 'bold'), foreground="#0E2A5B").pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Email:").pack(pady=5)

self.email\_entry = ttk.Entry(main\_frame)

self.email\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Password:").pack(pady=5)

self.password\_entry = ttk.Entry(main\_frame, show="\*")

self.password\_entry.pack(pady=5)

ttk.Button(main\_frame, text="Login", command=self.login, style='TButton').pack(pady=20)

def login(self):

email = self.email\_entry.get()

password = self.password\_entry.get()

try:

validate\_email(email)

validate\_password(password)

except (PasswordValidationError, ValueError) as e:

messagebox.showerror("Login Failed", str(e))

return

admin = admin\_login(email, password)

if admin:

messagebox.showinfo("Login Successful", "Welcome, Admin!")

self.root.destroy()

self.on\_login\_success()

else:

messagebox.showerror("Login Failed", "Invalid email or password.")

def run(self):

self.root.mainloop()

**main.py**

import tkinter as tk

import requests

import datetime

from tkinter import ttk, messagebox

from tkcalendar import Calendar

from fpdf import FPDF

import os

from database import \*

from admin\_login import AdminLogin

from customer\_login import CustomerLogin

from datetime import datetime

class CelestiaAirlinesApp:

API\_KEY = '606a419df38bff95986c9f4745721ac5' # Weather API key

CRITICAL\_CONDITIONS = ['thunderstorm', 'heavy rain', 'snow', 'hurricane', 'tornado']

def \_\_init\_\_(self):

self.root = tk.Tk()

self.root.title("Celestia Airlines")

self.root.geometry("800x600") # Smaller window size

self.setup\_styles()

self.book\_btn = None

self.seat\_buttons = {} # To store seat button references

self.current\_selected\_seat = None # To track currently selected seat

self.selected\_seat = tk.StringVar() # Initialize book\_btn

self.show\_main\_menu()

def setup\_styles(self):

self.style = ttk.Style()

self.style.theme\_use('clam')

self.primary\_color = "#0E2A5B" # Dark blue

self.secondary\_color = "#FF6B00" # Orange

self.bg\_color = "#FFFFFF" # White

self.text\_color = "#333333" # Dark gray

self.highlight\_color = "#F5F5F5" # Light gray

self.style.configure('.', background=self.bg\_color,foreground=self.text\_color)

self.style.configure('TButton', font=('Arial', 10, 'bold'), padding=8, background=self.primary\_color,foreground="white",borderwidth=0)

self.style.map('TButton', background=[('active', self.secondary\_color),('disabled', '#CCCCCC')])

self.style.configure('Header.TLabel', font=('Arial', 18, 'bold'),foreground=self.primary\_color)

self.style.configure('Subheader.TLabel',font=('Arial', 14),foreground=self.primary\_color)

self.style.configure('TFrame',background=self.bg\_color)

self.style.configure('TEntry',fieldbackground="white",foreground=self.text\_color,

bordercolor=self.primary\_color,lightcolor=self.primary\_color,padding=5)

self.style.configure('TCombobox',selectbackground=self.highlight\_color)

self.style.configure('Treeview',background="white",fieldbackground="white",foreground=self.text\_color)

self.style.map('Treeview',background=[('selected', self.secondary\_color)])

def show\_main\_menu(self):

for widget in self.root.winfo\_children():

widget.destroy()

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

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(20, 10))

ttk.Label(header\_frame,

text="✈ Celestia Airlines",

style='Header.TLabel').pack(pady=10)

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=40, pady=20)

ttk.Label(main\_frame, text="Book, Manage, Fly", style='Subheader.TLabel').pack(pady=10)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=30)

admin\_btn = ttk.Button(btn\_frame, text="Admin Login", command=self.show\_admin\_login,style='TButton',width=20)

admin\_btn.pack(pady=15, ipady=10)

customer\_btn = ttk.Button(btn\_frame, text="Customer Login",

command=self.show\_customer\_login,style='TButton',width=20)

customer\_btn.pack(pady=15, ipady=10)

footer\_frame = ttk.Frame(self.root)

footer\_frame.pack(fill=tk.X, pady=(10, 20))

ttk.Label(footer\_frame, text="© 2023 Celestia Airlines. All rights reserved.", font=('Arial', 8)).pack()

def show\_admin\_login(self):

admin\_login = AdminLogin(self.show\_admin\_dashboard)

admin\_login.run()

def show\_customer\_login(self):

customer\_login = CustomerLogin(lambda id, name: self.show\_customer\_dashboard(id, name))

customer\_login.run()

def show\_admin\_dashboard(self):

self.root.geometry("1200x800") # Larger window for better layout

for widget in self.root.winfo\_children():

widget.destroy()

main\_container = ttk.Frame(self.root)# Main container

main\_container.pack(fill=tk.BOTH, expand=True)

sidebar\_frame = ttk.Frame(main\_container, width=200)# Sidebar Frame

sidebar\_frame.pack(side=tk.LEFT, fill=tk.Y, padx=5, pady=5)

self.admin\_content\_frame = ttk.Frame(main\_container)# Main Content Frame

self.admin\_content\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

ttk.Label(sidebar\_frame, text="Flight Management", style='Subheader.TLabel').pack(pady=10)# Sidebar Buttons

buttons = [("Add Flight", self.show\_add\_flight),

("View Flights", self.show\_view\_flights),

("Cancel Flight", self.show\_cancel\_flight),

("Weather Check", self.show\_weather\_check),

("Crew Management", self.show\_crew\_management),

("Pilot Management", self.show\_pilot\_management)]

for text, command in buttons:

btn = ttk.Button(sidebar\_frame, text=text, command=command, width=20)

btn.pack(pady=5)

ttk.Button(sidebar\_frame,text="Logout",command=self.show\_main\_menu).pack(side=tk.BOTTOM, pady=20)

self.show\_view\_flights()

def clear\_admin\_content(self):

for widget in self.admin\_content\_frame.winfo\_children():

widget.destroy()

def show\_add\_flight(self):

self.clear\_admin\_content()

form\_frame = ttk.Frame(self.admin\_content\_frame)

form\_frame.pack(pady=20, padx=20, fill=tk.BOTH, expand=True)

cities\_of\_india = ["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Ahmedabad",

"Chennai", "Kolkata", "Surat", "Pune", "Jaipur","Thane", "Bhopal",

"Lucknow", "Kanpur", "Nagpur", "Visakhapatnam", "Indore","Patna", "Vadodara", "Ghaziabad"]

left\_frame = ttk.Frame(form\_frame)

left\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10)

right\_frame = ttk.Frame(form\_frame)

right\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10)

ttk.Label(left\_frame, text="Source:").pack(pady=5)

source\_var = tk.StringVar(value=cities\_of\_india[0])

ttk.OptionMenu(left\_frame, source\_var, \*cities\_of\_india).pack(fill=tk.X)

ttk.Label(left\_frame, text="Destination:").pack(pady=5)

dest\_var = tk.StringVar(value=cities\_of\_india[1])

ttk.OptionMenu(left\_frame, dest\_var, \*cities\_of\_india).pack(fill=tk.X)

ttk.Label(left\_frame, text="Departure Date:").pack(pady=5)

cal = Calendar(left\_frame, selectmode='day', date\_pattern='yyyy-mm-dd')

cal.pack(pady=10)

ttk.Label(left\_frame, text="Departure Time (HH:MM):").pack()

departure\_time\_entry = ttk.Entry(left\_frame)

departure\_time\_entry.pack(fill=tk.X)

ttk.Label(left\_frame, text="Arrival Time (HH:MM):").pack()

arrival\_time\_entry = ttk.Entry(left\_frame)

arrival\_time\_entry.pack(fill=tk.X)

ttk.Label(right\_frame, text="Economy Price (₹):").pack()

economy\_price\_entry = ttk.Entry(right\_frame)

economy\_price\_entry.pack(fill=tk.X, pady=5)

ttk.Label(right\_frame, text="Business Price (₹):").pack()

business\_price\_entry = ttk.Entry(right\_frame)

business\_price\_entry.pack(fill=tk.X, pady=5)

ttk.Label(right\_frame, text="First Class Price (₹):").pack()

first\_class\_price\_entry = ttk.Entry(right\_frame)

first\_class\_price\_entry.pack(fill=tk.X, pady=5)

btn\_frame = ttk.Frame(right\_frame)

btn\_frame.pack(pady=20, fill=tk.X)

ttk.Button(btn\_frame, text="Add Flight", command=lambda: perform\_add()).pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Cancel", command=self.clear\_admin\_content).pack(side=tk.RIGHT, padx=5)

def perform\_add():

source = source\_var.get()

destination = dest\_var.get()

date = cal.get\_date()

dep\_time = departure\_time\_entry.get()

arr\_time = arrival\_time\_entry.get()

try:

economy = float(economy\_price\_entry.get())

business = float(business\_price\_entry.get())

first = float(first\_class\_price\_entry.get())

except ValueError:

messagebox.showerror("Error", "Please enter valid prices.")

return

if not all([source, destination, date, dep\_time, arr\_time]):

messagebox.showerror("Error", "All fields must be filled!")

return

if source == destination:

messagebox.showerror("Error", "Source and Destination cannot be the same.")

return

if economy <= 0 or business <= 0 or first <= 0:

messagebox.showerror("Error", "Prices must be positive.")

return

success, error = add\_flight(source=source,destination=destination,departure\_date=date,

departure\_time=dep\_time,arrival\_time=arr\_time,

economy\_price=economy,business\_price=business,first\_class\_price=first) # simulate success

if success:

messagebox.showinfo("Success", "Flight added successfully!")

self.clear\_admin\_content()

else:

messagebox.showerror("Error", error or "Failed to add flight.")

def generate\_ticket\_from\_selection(self):

selected\_item = self.bookings\_tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a booking first!")

return

booking\_id = self.bookings\_tree.item(selected\_item)['values'][0]

self.generate\_ticket\_pdf\_by\_booking(booking\_id)

def cancel\_selected\_booking(self):

selected\_item = self.bookings\_tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a booking first!")

return

booking\_id = self.bookings\_tree.item(selected\_item)['values'][0]

success = cancel\_ticket(booking\_id)

if success:

messagebox.showinfo("Success", "Booking cancelled successfully!")

self.view\_bookings(self.current\_customer\_id)

else:

messagebox.showerror("Error", "Failed to cancel booking")

def show\_view\_flights(self):

self.clear\_admin\_content()

view\_frame = ttk.Frame(self.admin\_content\_frame)

view\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

tree\_frame = ttk.Frame(view\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,show="headings",

columns=("id", "source", "dest", "date", "dep\_time", "arriv\_time", "econ\_price", "bus\_price", "first\_price"))

scrollbar.config(command=tree.yview)

tree.heading("id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.heading("dep\_time", text="Departure")

tree.heading("arriv\_time", text="Arrival")

tree.heading("econ\_price", text="Economy Price")

tree.heading("bus\_price", text="Business Price")

tree.heading("first\_price", text="First Class Price")

tree.column("id", width=80)

tree.column("source", width=100)

tree.column("dest", width=100)

tree.column("date", width=100)

tree.column("dep\_time", width=100)

tree.column("arriv\_time", width=100)

tree.column("econ\_price", width=100)

tree.column("bus\_price", width=100)

tree.column("first\_price", width=100)

tree.pack(fill=tk.BOTH, expand=True) # Load data

flights = get\_all\_flights()

for flight in flights:

tree.insert("", tk.END, values=(flight[0], flight[1], flight[2], flight[3],

flight[4], flight[5], f"₹{flight[8]}", f"₹{flight[9]}", f"₹{flight[10]}"))

def show\_cancel\_flight(self):

self.clear\_admin\_content()

cancel\_frame = ttk.Frame(self.admin\_content\_frame)

cancel\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

tree\_frame = ttk.Frame(cancel\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame,yscrollcommand=scrollbar.set,

columns=("id", "source", "dest", "date"),show="headings",selectmode="browse")

scrollbar.config(command=tree.yview)

tree.heading("id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.pack(fill=tk.BOTH, expand=True)

flights = get\_all\_flights()

for flight in flights:

tree.insert("", tk.END, values=(flight[0], flight[1], flight[2], flight[3]))

def perform\_cancellation():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a flight from the list!")

return

flight\_id = tree.item(selected\_item)['values'][0]

success, error\_message = cancel\_flight(flight\_id)

if success:

messagebox.showinfo("Success", f"Flight {flight\_id} cancelled successfully!")

self.show\_view\_flights() # Refresh the view

else:

messagebox.showerror("Error", error\_message)

btn\_frame = ttk.Frame(cancel\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Cancel Selected Flight",

command=perform\_cancellation,style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Close", command=self.clear\_admin\_content).pack(side=tk.RIGHT, padx=5)

def show\_weather\_check(self):

self.clear\_admin\_content()

weather\_frame = ttk.Frame(self.admin\_content\_frame)

weather\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

ttk.Label(weather\_frame, text="Flight Weather Monitoring", style='Subheader.TLabel').pack(pady=10)

tree\_container = ttk.Frame(weather\_frame)

tree\_container.pack(fill=tk.BOTH, expand=True)

x\_scrollbar = ttk.Scrollbar(tree\_container, orient=tk.HORIZONTAL)

x\_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)

y\_scrollbar = ttk.Scrollbar(tree\_container)

y\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

self.weather\_flight\_tree = ttk.Treeview(tree\_container,xscrollcommand=x\_scrollbar.set,

yscrollcommand=y\_scrollbar.set,columns=("id", "source", "date"),show="headings",height=8 )

self.weather\_flight\_tree.pack(fill=tk.BOTH, expand=True)

x\_scrollbar.config(command=self.weather\_flight\_tree.xview)

y\_scrollbar.config(command=self.weather\_flight\_tree.yview)

self.weather\_flight\_tree.heading("id", text="Flight ID", anchor=tk.CENTER)

self.weather\_flight\_tree.heading("source", text="Source City", anchor=tk.CENTER)

self.weather\_flight\_tree.heading("date", text="Date", anchor=tk.CENTER)

self.weather\_flight\_tree.column("id", width=150, anchor=tk.CENTER)

self.weather\_flight\_tree.column("source", width=200, anchor=tk.CENTER)

self.weather\_flight\_tree.column("date", width=200, anchor=tk.CENTER)

self.weather\_labels = {'city': ttk.Label(weather\_frame, text="", font=('Arial', 10)),

'temp': ttk.Label(weather\_frame, text="", font=('Arial', 10)),

'condition': ttk.Label(weather\_frame, text="", font=('Arial', 10)),

'warning': ttk.Label(weather\_frame, text="", foreground="red", font=('Arial', 10, 'bold'))}

for label in self.weather\_labels.values():

label.pack(pady=5)

self.weather\_btn\_frame = ttk.Frame(weather\_frame)

self.weather\_btn\_frame.pack(pady=10)

ttk.Button(self.weather\_btn\_frame, text="Check Weather", command=self.check\_flight\_weather).pack(side=tk.LEFT, padx=5)

self.delay\_btn = ttk.Button(self.weather\_btn\_frame, text="Delay Flight",

command=self.delay\_flight,state=tk.DISABLED)

self.cancel\_btn = ttk.Button(self.weather\_btn\_frame, text="Cancel Flight",

command=self.cancel\_flight\_weather,state=tk.DISABLED)

self.delay\_btn.pack(side=tk.LEFT, padx=5)

self.cancel\_btn.pack(side=tk.LEFT, padx=5)

self.load\_upcoming\_flights()

def show\_crew\_management(self):

self.clear\_admin\_content()

crew\_frame = ttk.Frame(self.admin\_content\_frame)

crew\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

ttk.Label(crew\_frame, text="Crew Management", style='Subheader.TLabel').pack(pady=10)

ttk.Button(crew\_frame, text="Add Crew", command=self.add\_crew).pack(pady=5)

ttk.Button(crew\_frame, text="View Crew", command=self.view\_crew).pack(pady=5)

ttk.Button(crew\_frame, text="Remove Crew", command=self.remove\_crew).pack(pady=5)

def show\_pilot\_management(self):

self.clear\_admin\_content()

pilot\_frame = ttk.Frame(self.admin\_content\_frame)

pilot\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

ttk.Label(pilot\_frame, text="Pilot Management", style='Subheader.TLabel').pack(pady=10)

ttk.Button(pilot\_frame, text="Add Pilot", command=self.add\_pilots).pack(pady=5)

ttk.Button(pilot\_frame, text="View Pilots", command=self.view\_pilots).pack(pady=5)

ttk.Button(pilot\_frame, text="Remove Pilot", command=self.remove\_pilots).pack(pady=5)

def show\_customer\_dashboard(self, customer\_id, customer\_name):

self.current\_customer\_id = customer\_id

self.customer\_name = customer\_name

self.root.geometry("1000x800")

for widget in self.root.winfo\_children():

widget.destroy()

header\_frame = ttk.Frame(self.root)

header\_frame.pack(fill=tk.X, pady=(10, 20))

ttk.Label(header\_frame, text="Customer Dashboard", style='Header.TLabel').pack(pady=10)

ttk.Label(header\_frame, text=f"Welcome, {customer\_name}", style='Header.TLabel').pack(pady=10)

notebook = ttk.Notebook(self.root)

notebook.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

search\_book\_frame = ttk.Frame(notebook)# ===== Search & Book Tab =====

notebook.add(search\_book\_frame, text="Search & Book")

sb\_container = ttk.Frame(search\_book\_frame)# Scrollable Container

sb\_container.pack(fill=tk.BOTH, expand=True)

sb\_canvas = tk.Canvas(sb\_container)

sb\_scrollbar = ttk.Scrollbar(sb\_container, orient="vertical", command=sb\_canvas.yview)

sb\_scrollable\_frame = ttk.Frame(sb\_canvas)

sb\_scrollable\_frame.bind("<Configure>", lambda e: sb\_canvas.configure(scrollregion=sb\_canvas.bbox("all")))

sb\_canvas.create\_window((0, 0), window=sb\_scrollable\_frame, anchor="nw")

sb\_canvas.configure(yscrollcommand=sb\_scrollbar.set)

sb\_canvas.pack(side="left", fill="both", expand=True)

sb\_scrollbar.pack(side="right", fill="y")

search\_frame = ttk.LabelFrame(sb\_scrollable\_frame, text="Search Flights", padding=10)# Search Section

search\_frame.pack(fill=tk.X, padx=20, pady=10)

trip\_type\_frame = ttk.Frame(search\_frame)# Trip type

trip\_type\_frame.pack(fill=tk.X, pady=5)

ttk.Label(trip\_type\_frame, text="Trip Type:").pack(side=tk.LEFT)

self.trip\_type = tk.StringVar(value="ONE WAY")

ttk.Radiobutton(trip\_type\_frame, text="ONE WAY", variable=self.trip\_type, value="ONE WAY").pack(side=tk.LEFT, padx=5)

ttk.Radiobutton(trip\_type\_frame, text="ROUND TRIP", variable=self.trip\_type, value="ROUND TRIP").pack(side=tk.LEFT, padx=5)

ttk.Radiobutton(trip\_type\_frame, text="MULTI CITY", variable=self.trip\_type, value="MULTI CITY").pack(side=tk.LEFT, padx=5)

loc\_frame = ttk.Frame(search\_frame) # From/To

loc\_frame.pack(fill=tk.X, pady=5)

ttk.Label(loc\_frame, text="From:").grid(row=0, column=0, sticky=tk.W)

self.from\_city = ttk.Combobox(loc\_frame, values=["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Chennai",

"Kolkata", "Ahmedabad", "Pune", "Jaipur", "Lucknow"])

self.from\_city.grid(row=1, column=0, padx=5, sticky=tk.W+tk.E)

ttk.Label(loc\_frame, text="To:").grid(row=0, column=1, sticky=tk.W)

self.to\_city = ttk.Combobox(loc\_frame, values=["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Chennai",

"Kolkata", "Ahmedabad", "Pune", "Jaipur", "Lucknow"])

self.to\_city.grid(row=1, column=1, padx=5, sticky=tk.W+tk.E)

ttk.Button(search\_frame,text="SEARCH FLIGHTS",command=lambda: self.perform\_search(customer\_id),style='TButton').pack(pady=10)

results\_frame = ttk.LabelFrame(sb\_scrollable\_frame, text="Available Flights", padding=10)# Results Section

results\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

tree\_frame = ttk.Frame(results\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

self.flights\_tree = ttk.Treeview(tree\_frame, show="headings",yscrollcommand=scrollbar.set,

columns=("id", "source", "dest", "date", "dep\_time", "arriv\_time", "econ\_price", "bus\_price"))

scrollbar.config(command=self.flights\_tree.yview)

for col in ["id", "source", "dest", "date", "dep\_time", "arriv\_time", "econ\_price", "bus\_price"]:

self.flights\_tree.heading(col, text=col.replace('\_', ' ').title())

self.flights\_tree.column(col, width=100)

self.flights\_tree.pack(fill=tk.BOTH, expand=True)

self.book\_btn = ttk.Button(results\_frame, text="BOOK SELECTED FLIGHT", style='TButton',

command=lambda: self.book\_selected\_flight(customer\_id))

self.book\_btn.pack(pady=10)

book\_frame = ttk.Frame(notebook)# ===== Direct Booking Tab =====

notebook.add(book\_frame, text="Direct Booking")

db\_container = ttk.Frame(book\_frame)# Scrollable Container

db\_container.pack(fill=tk.BOTH, expand=True)

db\_canvas = tk.Canvas(db\_container)

db\_scrollbar = ttk.Scrollbar(db\_container, orient="vertical", command=db\_canvas.yview)

db\_scrollable\_frame = ttk.Frame(db\_canvas)

db\_scrollable\_frame.bind("<Configure>", lambda e: db\_canvas.configure(scrollregion=db\_canvas.bbox("all")))

db\_canvas.create\_window((0, 0), window=db\_scrollable\_frame, anchor="nw")

db\_canvas.configure(yscrollcommand=db\_scrollbar.set)

db\_canvas.pack(side="left", fill="both", expand=True)

db\_scrollbar.pack(side="right", fill="y")

flight\_id\_frame = ttk.Frame(db\_scrollable\_frame)# Flight ID Section

flight\_id\_frame.pack(fill=tk.X, padx=20, pady=10)

ttk.Label(flight\_id\_frame, text="Flight ID:").pack(side=tk.LEFT)

self.book\_flight\_entry = ttk.Entry(flight\_id\_frame)

self.book\_flight\_entry.pack(side=tk.LEFT, padx=5, fill=tk.X, expand=True)

ttk.Button(flight\_id\_frame, text="Show Available Seats", style='TButton',

command=lambda: self.show\_seats\_for\_booking(book\_frame, customer\_id)).pack(side=tk.RIGHT, padx=5)

class\_frame = ttk.LabelFrame(db\_scrollable\_frame, text="Class Selection", padding=10)# Class Selection

class\_frame.pack(fill=tk.X, padx=20, pady=10)

self.class\_var = tk.StringVar(value="economy")

ttk.Radiobutton(class\_frame, text="Economy", variable=self.class\_var, value="economy").pack(anchor=tk.W)

ttk.Radiobutton(class\_frame, text="Business", variable=self.class\_var, value="business").pack(anchor=tk.W)

ttk.Radiobutton(class\_frame, text="First Class", variable=self.class\_var, value="first\_class").pack(anchor=tk.W)

self.seat\_frame = ttk.LabelFrame(db\_scrollable\_frame, text="Seat Selection", padding=10)# Seat Selection

self.seat\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

self.canvas = tk.Canvas(self.seat\_frame)

scrollbar = ttk.Scrollbar(self.seat\_frame, orient="vertical", command=self.canvas.yview)

self.scrollable\_frame = ttk.Frame(self.canvas)

self.scrollable\_frame.bind("<Configure>", lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))

self.canvas.create\_window((0, 0), window=self.scrollable\_frame, anchor="nw")

self.canvas.configure(yscrollcommand=scrollbar.set)

self.canvas.pack(side="left", fill="both", expand=True)

scrollbar.pack(side="right", fill="y")

self.seat\_grid\_frame = ttk.Frame(self.scrollable\_frame)

self.seat\_grid\_frame.pack(fill=tk.BOTH, expand=True)

self.selected\_seat = tk.StringVar()

details\_frame = ttk.LabelFrame(db\_scrollable\_frame, text="Passenger Details", padding=10)# Passenger Details

details\_frame.pack(fill=tk.X, padx=20, pady=10)

ttk.Label(details\_frame, text="Passenger Name:").pack()

self.name\_entry = ttk.Entry(details\_frame)

self.name\_entry.pack(pady=5)

ttk.Button(db\_scrollable\_frame, text="Confirm Booking",

command=lambda: self.perform\_booking\_from\_tab(customer\_id),style='TButton').pack(pady=10)

bookings\_frame = ttk.Frame(notebook)# ===== My Bookings Tab =====

notebook.add(bookings\_frame, text="My Bookings")

mb\_container = ttk.Frame(bookings\_frame)# Scrollable Container

mb\_container.pack(fill=tk.BOTH, expand=True)

mb\_canvas = tk.Canvas(mb\_container)

mb\_scrollbar = ttk.Scrollbar(mb\_container, orient="vertical", command=mb\_canvas.yview)

mb\_scrollable\_frame = ttk.Frame(mb\_canvas)

mb\_scrollable\_frame.bind("<Configure>", lambda e: mb\_canvas.configure(scrollregion=mb\_canvas.bbox("all")))

mb\_canvas.create\_window((0, 0), window=mb\_scrollable\_frame, anchor="nw")

mb\_canvas.configure(yscrollcommand=mb\_scrollbar.set)

mb\_canvas.pack(side="left", fill="both", expand=True)

mb\_scrollbar.pack(side="right", fill="y")

bookings\_tree\_frame = ttk.Frame(mb\_scrollable\_frame)# Bookings Treeview

bookings\_tree\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

scrollbar = ttk.Scrollbar(bookings\_tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

self.bookings\_tree = ttk.Treeview(bookings\_tree\_frame,

columns=("id", "flight\_id", "source", "dest", "date", "seat", "class"),show="headings",yscrollcommand=scrollbar.set)

scrollbar.config(command=self.bookings\_tree.yview)

for col in ["id", "flight\_id", "source", "dest", "date", "seat", "class"]:

self.bookings\_tree.heading(col, text=col.title())

self.bookings\_tree.column(col, width=100)

self.bookings\_tree.pack(fill=tk.BOTH, expand=True)

bookings\_btn\_frame = ttk.Frame(mb\_scrollable\_frame)

bookings\_btn\_frame.pack(fill=tk.X, pady=10)

ttk.Button(bookings\_btn\_frame, text="Refresh Bookings", command=lambda: self.view\_bookings(customer\_id)).pack(side=tk.LEFT)

ttk.Button(bookings\_btn\_frame, text="Generate PDF Ticket", command=self.generate\_ticket\_from\_selection).pack(side=tk.LEFT)

ttk.Button(bookings\_btn\_frame, text="Cancel Booking", command=self.cancel\_selected\_booking).pack(side=tk.LEFT)

footer\_frame = ttk.Frame(self.root)

footer\_frame.pack(fill=tk.X, pady=(10, 20))

ttk.Button(footer\_frame, text="Logout", command=self.show\_main\_menu).pack(pady=10)

self.view\_bookings(customer\_id)

def \_bind\_to\_mousewheel(self, event, canvas):

canvas.bind\_all("<MouseWheel>", lambda e: canvas.yview\_scroll(int(-1\*(e.delta/120)), "units"))

def \_unbind\_from\_mousewheel(self, event, canvas):

canvas.unbind\_all("<MouseWheel>")

def load\_upcoming\_flights(self):

for item in self.weather\_flight\_tree.get\_children():

self.weather\_flight\_tree.delete(item)

try:

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("""SELECT id, source, departure\_date FROM flights

WHERE departure\_date >= CURDATE() ORDER BY departure\_date ASC""")

flights = cursor.fetchall()

for flight in flights:

self.weather\_flight\_tree.insert("", tk.END, values=(flight[0], flight[1], flight[2]))

except Exception as e:

messagebox.showerror("Error", f"Database error: {str(e)}")

finally:

if conn.is\_connected():

conn.close()

def check\_flight\_weather(self):

selected = self.weather\_flight\_tree.selection()

if not selected:

messagebox.showerror("Error", "Please select a flight first!")

return

flight\_id = self.weather\_flight\_tree.item(selected)['values'][0]

city = self.weather\_flight\_tree.item(selected)['values'][1]

try:

url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&units=metric&appid={self.API\_KEY}"

response = requests.get(url)

data = response.json()

if data['cod'] != 200:

raise Exception(data['message'])

weather = {'city': data['name'],

'temp': data['main']['temp'],

'condition': data['weather'][0]['main'].lower(),

'description': data['weather'][0]['description'].lower()}

self.weather\_labels['city'].config(text=f"City: {weather['city']}")

self.weather\_labels['temp'].config(text=f"Temperature: {weather['temp']}°C")

self.weather\_labels['condition'].config(text=f"Condition: {weather['description'].title()}")

is\_critical = any(cond in weather['description'] for cond in self.CRITICAL\_CONDITIONS)

if is\_critical:

self.weather\_labels['warning'].config(

text="WARNING: Critical weather detected! Consider delaying or canceling the flight.")

self.delay\_btn.config(state=tk.NORMAL)

self.cancel\_btn.config(state=tk.NORMAL)

else:

self.weather\_labels['warning'].config(text="Weather conditions are normal")

self.delay\_btn.config(state=tk.DISABLED)

self.cancel\_btn.config(state=tk.DISABLED)

except Exception as e:

messagebox.showerror("Weather Error", f"Failed to get weather data: {str(e)}")

def delay\_flight(self):

selected = self.weather\_flight\_tree.selection()

flight\_id = self.weather\_flight\_tree.item(selected)['values'][0]

messagebox.showinfo("Flight Delayed", f"Flight {flight\_id} has been delayed")

self.load\_upcoming\_flights()

self.clear\_weather\_display()

def cancel\_flight\_weather(self):

selected = self.weather\_flight\_tree.selection()

flight\_id = self.weather\_flight\_tree.item(selected)['values'][0]

success, error\_message = cancel\_flight(flight\_id)

if success:

messagebox.showinfo("Success", f"Flight {flight\_id} cancelled successfully!")

self.load\_upcoming\_flights()

self.clear\_weather\_display()

else:

messagebox.showerror("Error", error\_message)

def clear\_weather\_display(self):

for label in self.weather\_labels.values():

label.config(text="")

self.weather\_labels['warning'].config(text="")

self.delay\_btn.config(state=tk.DISABLED)

self.cancel\_btn.config(state=tk.DISABLED)

def add\_flight(self):

add\_window = tk.Toplevel(self.root)

add\_window.title("Add Flight")

add\_window.geometry("500x600")

main\_frame = ttk.Frame(add\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

cities\_of\_india = ["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Ahmedabad","Nagpur", "Visakhapatnam",

"Chennai", "Kolkata", "Surat", "Pune", "Jaipur","Lucknow", "Kanpur","Indore",

"Thane", "Bhopal", "Patna", "Vadodara", "Ghaziabad"]

ttk.Label(main\_frame, text="Source:").pack(pady=5)

source\_var = tk.StringVar(value=cities\_of\_india[0])

ttk.OptionMenu(main\_frame, source\_var, \*cities\_of\_india).pack()

ttk.Label(main\_frame, text="Destination:").pack(pady=5)

dest\_var = tk.StringVar(value=cities\_of\_india[0])

ttk.OptionMenu(main\_frame, dest\_var, \*cities\_of\_india).pack()

ttk.Label(main\_frame, text="Departure Date:").pack(pady=5)

cal = Calendar(main\_frame, selectmode='day',date\_pattern='yyyy-mm-dd',year=2025, month=4, day=3)

cal.pack(pady=10)

ttk.Label(main\_frame, text="Departure Time (HH:MM):").pack()

departure\_time\_entry = ttk.Entry(main\_frame)

departure\_time\_entry.pack()

ttk.Label(main\_frame, text="Arrival Time (HH:MM):").pack()

arrival\_time\_entry = ttk.Entry(main\_frame)

arrival\_time\_entry.pack()

ttk.Label(main\_frame, text="Economy Price (₹):").pack()

economy\_price\_entry = ttk.Entry(main\_frame)

economy\_price\_entry.pack()

ttk.Label(main\_frame, text="Business Price (₹):").pack()

business\_price\_entry = ttk.Entry(main\_frame)

business\_price\_entry.pack()

ttk.Label(main\_frame, text="First Class Price (₹):").pack()

first\_class\_price\_entry = ttk.Entry(main\_frame)

first\_class\_price\_entry.pack()

def perform\_add():

source = source\_var.get()

dest = dest\_var.get()

date = cal.get\_date()

departure\_time = departure\_time\_entry.get()

arrival\_time = arrival\_time\_entry.get()

try:

economy\_price = float(economy\_price\_entry.get())

business\_price = float(business\_price\_entry.get())

first\_class\_price = float(first\_class\_price\_entry.get())

except ValueError:

messagebox.showerror("Error", "Prices must be valid numbers")

return

if not all([source, dest, date,departure\_time,arrival\_time]):

messagebox.showerror("Error", "All fields are required!")

return

if any(price <= 0 for price in [economy\_price, business\_price, first\_class\_price]):

messagebox.showerror("Error", "Prices must be positive numbers")

return

result = add\_flight(source=source,destination=dest,departure\_date=date,departure\_time=departure\_time,

arrival\_time=arrival\_time,economy\_price=economy\_price,business\_price=business\_price,first\_class\_price=first\_class\_price)

if result is True:

messagebox.showinfo("Success", "Flight added successfully!")

add\_window.destroy()

else:

messagebox.showerror("Error", result)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=20)

ttk.Button(btn\_frame, text="Add Flight", command=perform\_add).pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Cancel", command=add\_window.destroy).pack(side=tk.RIGHT, padx=5)

def view\_flights(self):

view\_window = tk.Toplevel(self.root)

view\_window.title("View Flights")

view\_window.geometry("800x500")

main\_frame = ttk.Frame(view\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

flights = get\_all\_flights()

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,show="headings",

columns=("id", "source", "dest", "date", "dep\_time", "arriv\_time", "econ\_price", "bus\_price", "first\_price"))

scrollbar.config(command=tree.yview)

tree.heading("id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.heading("dep\_time", text="Departure Time")

tree.heading("arriv\_time", text="Arrival Time")

tree.heading("econ\_price", text="Economy Price")

tree.heading("bus\_price", text="Business Price")

tree.heading("first\_price", text="First Class Price")

tree.column("id", width=80)

tree.column("source", width=100)

tree.column("dest", width=100)

tree.column("date", width=120)

tree.column("econ\_price", width=100)

tree.column("bus\_price", width=100)

tree.column("first\_price", width=100)

tree.pack(fill=tk.BOTH, expand=True)

for flight in flights:

tree.insert("", tk.END, values=(flight[0], flight[1], flight[2], flight[3], f"₹{flight[7]}", f"₹{flight[8]}", f"₹{flight[9]}"))

def add\_pilots(self):

add\_pilots = tk.Toplevel(self.root)

add\_pilots.title("Add Pilots")

add\_pilots.geometry("400x400")

main\_frame = ttk.Frame(add\_pilots)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

ttk.Label(main\_frame, text="Name:").pack()

name\_entry = ttk.Entry(main\_frame)

name\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Age:").pack()

age\_entry = ttk.Entry(main\_frame)

age\_entry.pack(pady=5)

def perform\_add\_pilots():

name = name\_entry.get()

age = age\_entry.get()

result = add\_pilot(name, age)

if result is True:

messagebox.showinfo("Success", "Pilot added successfully!")

add\_pilots.destroy()

else:

messagebox.showerror("Error", result)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Add", command=perform\_add\_pilots).pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Cancel", command=add\_pilots.destroy).pack(side=tk.RIGHT, padx=5)

def add\_crew(self):

add\_crew\_window = tk.Toplevel(self.root)

add\_crew\_window.title("Add Crew Members")

add\_crew\_window.geometry("400x400")

main\_frame = ttk.Frame(add\_crew\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

ttk.Label(main\_frame, text="Name:").pack()

name\_entry = ttk.Entry(main\_frame)

name\_entry.pack(pady=5)

ttk.Label(main\_frame, text="Age:").pack()

age\_entry = ttk.Entry(main\_frame)

age\_entry.pack(pady=5)

def perform\_add\_crew():

name = name\_entry.get()

age = age\_entry.get()

result = add\_crew(name, age)

if result is True:

messagebox.showinfo("Success", "Crew added successfully!")

add\_crew\_window.destroy()

else:

messagebox.showerror("Error", result)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Add", command=perform\_add\_crew).pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Cancel", command=add\_crew\_window.destroy).pack(side=tk.RIGHT, padx=5)

def cancel\_flight(self):

cancel\_window = tk.Toplevel(self.root)

cancel\_window.title("Cancel Flight")

cancel\_window.geometry("500x400")

main\_frame = ttk.Frame(cancel\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

flights = get\_all\_flights()

if not flights:

messagebox.showinfo("No Flights", "No flights available to cancel.")

cancel\_window.destroy()

return

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,columns=("id", "source", "dest", "date"),

show="headings",selectmode="browse")

scrollbar.config(command=tree.yview)

tree.heading("id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.pack(fill=tk.BOTH, expand=True)

for flight in flights:

tree.insert("", tk.END, values=(flight[0], flight[1], flight[2], flight[3]))

def perform\_cancellation():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a flight from the list!")

return

flight\_id = tree.item(selected\_item)['values'][0]

success, error\_message = cancel\_flight(flight\_id)

if success:

messagebox.showinfo("Success", f"Flight {flight\_id} cancelled successfully!")

cancel\_window.destroy()

else:

messagebox.showerror("Error", f"Failed to cancel flight: {error\_message}")

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Cancel Selected Flight", command=perform\_cancellation,style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Close", command=cancel\_window.destroy).pack(side=tk.RIGHT, padx=5)

def remove\_crew(self):

remove\_window = tk.Toplevel(self.root)

remove\_window.title("Remove Crew Member")

remove\_window.geometry("700x500")

main\_frame = ttk.Frame(remove\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

crew\_members = get\_all\_crew()

if not crew\_members:

messagebox.showinfo("No Crew", "No crew members available to remove.")

remove\_window.destroy()

return

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,columns=("id", "name", "flight\_id", "age"),

show="headings",selectmode="browse")

scrollbar.config(command=tree.yview)

tree.heading("id", text="ID")

tree.heading("name", text="Name")

tree.heading("age", text="Age")

tree.heading("flight\_id", text="Flight ID")

tree.column("id", width=50)

tree.column("name", width=150)

tree.column("flight\_id", width=50)

tree.column("age", width=100)

tree.pack(fill=tk.BOTH, expand=True)

for crew in crew\_members:

tree.insert("", tk.END, values=(crew[0], crew[1], crew[2], crew[3] or "None"))

def perform\_removal():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a crew member to remove!")

return

crew\_id = tree.item(selected\_item)['values'][0]

result = remove\_crew(crew\_id)

if result is True:

messagebox.showinfo("Success", "Crew member removed successfully!")

remove\_window.destroy()

else:

messagebox.showerror("Error", result)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Remove Selected Crew",

command=perform\_removal,style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame, text="Close", command=remove\_window.destroy).pack(side=tk.RIGHT, padx=5)

def remove\_pilots(self):

remove\_window = tk.Toplevel(self.root)

remove\_window.title("Remove Pilot")

remove\_window.geometry("700x500")

main\_frame = ttk.Frame(remove\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

pilots = get\_all\_pilots()

if not pilots:

messagebox.showinfo("No Pilots", "No pilots available to remove.")

remove\_window.destroy()

return

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,columns=("id", "name", "age", "flight\_id"),

show="headings",selectmode="browse")

scrollbar.config(command=tree.yview)

tree.heading("id", text="ID")

tree.heading("name", text="Name")

tree.heading("age", text="Age")

tree.heading("flight\_id", text="Flight ID")

tree.column("id", width=50)

tree.column("name", width=150)

tree.column("age", width=50)

tree.column("flight\_id", width=100)

tree.pack(fill=tk.BOTH, expand=True)

for pilot in pilots:

tree.insert("", tk.END, values=(pilot[0], pilot[1], pilot[2], pilot[3] or "None"))

def perform\_removal():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a pilot to remove!")

return

pilot\_id = tree.item(selected\_item)['values'][0]

result = remove\_pilot(pilot\_id)

if result is True:

messagebox.showinfo("Success", "Pilot removed successfully!")

remove\_window.destroy()

else:

messagebox.showerror("Error", result)

btn\_frame = ttk.Frame(main\_frame)

btn\_frame.pack(pady=10)

ttk.Button(btn\_frame, text="Remove Selected Pilot",

command=perform\_removal,style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(btn\_frame,text="Close", command=remove\_window.destroy).pack(side=tk.RIGHT, padx=5)

def view\_pilots(self):

view\_window = tk.Toplevel(self.root)

view\_window.title("All Pilots")

view\_window.geometry("800x500")

main\_frame = ttk.Frame(view\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

pilots = view\_pilots()

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,

columns=("id", "name", "age", "flight\_id"),show="headings")

scrollbar.config(command=tree.yview)

tree.heading("id", text="ID")

tree.heading("name", text="Name")

tree.heading("age", text="Age")

tree.heading("flight\_id", text="Flight ID")

tree.column("id", width=50)

tree.column("name", width=150)

tree.column("age", width=50)

tree.column("flight\_id", width=100)

tree.pack(fill=tk.BOTH, expand=True)

if pilots:

for pilot in pilots:

tree.insert("", tk.END, values=(pilot[0], pilot[1], pilot[2], pilot[3] or "Not assigned"))

else:

tree.insert("", tk.END, values=("No pilots found in the system", "", "", ""))

def view\_crew(self):

view\_window = tk.Toplevel(self.root)

view\_window.title("All Crew Members")

view\_window.geometry("800x500")

main\_frame = ttk.Frame(view\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

crew\_members = view\_crew()

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,columns=("id", "name", "flight\_id", "age"),show="headings")

scrollbar.config(command=tree.yview)

tree.heading("id", text="ID")

tree.heading("name", text="Name")

tree.heading("age", text="Age")

tree.heading("flight\_id", text="Flight ID")

tree.column("id", width=50)

tree.column("name", width=150)

tree.column("age", width=50)

tree.column("flight\_id", width=100)

tree.pack(fill=tk.BOTH, expand=True)

if crew\_members:

for crew in crew\_members:

tree.insert("", tk.END, values=(crew[0], crew[1], crew[2], crew[3] or "Not assigned"))

else:

tree.insert("", tk.END, values=("No crew members found in the system", "", "", ""))

def search\_flights(self, customer\_id=None):

search\_window = tk.Toplevel(self.root)

search\_window.title("Search Flights")

search\_window.geometry("800x600")

main\_frame = ttk.Frame(search\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

criteria\_frame = ttk.LabelFrame(main\_frame, text="Search Criteria", padding=10)

criteria\_frame.pack(fill=tk.X, pady=10)

loc\_frame = ttk.Frame(criteria\_frame)

loc\_frame.pack(fill=tk.X, pady=5)

ttk.Label(loc\_frame, text="From:").grid(row=0, column=0, sticky=tk.W)

from\_entry = ttk.Entry(loc\_frame)

from\_entry.grid(row=1, column=0, padx=5, sticky=tk.W+tk.E)

ttk.Label(loc\_frame, text="To:").grid(row=0, column=1, sticky=tk.W)

to\_entry = ttk.Entry(loc\_frame)

to\_entry.grid(row=1, column=1, padx=5, sticky=tk.W+tk.E)

date\_frame = ttk.Frame(criteria\_frame)

date\_frame.pack(fill=tk.X, pady=5)

ttk.Label(date\_frame, text="Departure Date:").grid(row=0, column=0, sticky=tk.W)

cal = Calendar(date\_frame, selectmode='day', date\_pattern='yyyy-mm-dd')

cal.grid(row=1, column=0, padx=5, sticky=tk.W)

results\_frame = ttk.LabelFrame(main\_frame, text="Flight Results", padding=10)

results\_frame.pack(fill=tk.BOTH, expand=True, pady=10)

tree\_frame = ttk.Frame(results\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,

columns=("id", "source", "dest", "date", "departure", "arrival", "price"),show="headings")

scrollbar.config(command=tree.yview)

tree.heading("id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.heading("departure", text="Departure")

tree.heading("arrival", text="Arrival")

tree.heading("price", text="Price")

tree.column("id", width=80)

tree.column("source", width=100)

tree.column("dest", width=100)

tree.column("date", width=100)

tree.column("departure", width=100)

tree.column("arrival", width=100)

tree.column("price", width=100)

tree.pack(fill=tk.BOTH, expand=True)

def perform\_search(self, customer\_id):

source = self.from\_city.get().strip()

destination = self.to\_city.get().strip()

trip\_type = self.trip\_type.get()

if not (source and destination):

messagebox.showerror("Error", "Please enter both source and destination!")

return

for item in self.flights\_tree.get\_children():

self.flights\_tree.delete(item)

try:

if trip\_type == "ONE WAY":

flights = search\_flights(source, destination)

self.display\_flights(flights, one\_way=True)

elif trip\_type == "ROUND TRIP":

outbound\_flights = search\_flights(source, destination)

return\_flights = search\_flights(destination, source)

self.display\_flights(outbound\_flights, return\_flights)

elif trip\_type == "MULTI CITY":

messagebox.showinfo("Info", "Please specify additional cities in the multi-city form")

except Exception as e:

messagebox.showerror("Error", f"Database error: {str(e)}")

def display\_flights(self, outbound\_flights, return\_flights=None, one\_way=False):

for item in self.flights\_tree.get\_children():

self.flights\_tree.delete(item)

if outbound\_flights:

for flight in outbound\_flights:

self.flights\_tree.insert("", tk.END, values=(flight[0], flight[1], flight[2], flight[3],

flight[4], flight[5],f"₹{flight[6]}" if flight[6] else "N/A", f"₹{flight[7]}" if flight[7] else "N/A", "Outbound"))

if return\_flights and not one\_way:

for flight in return\_flights:

self.flights\_tree.insert("", tk.END,

values=(flight[0], flight[1], flight[2], flight[3],

flight[4], flight[5], f"₹{flight[6]}" if flight[6] else "N/A",

f"₹{flight[7]}" if flight[7] else "N/A", "Return"))

if not outbound\_flights and (not return\_flights or one\_way):

self.flights\_tree.insert("", tk.END, values=("No flights found", "", "", "", "", "", "", ""))

if hasattr(self, 'book\_btn'):

self.book\_btn.config(state=tk.NORMAL if outbound\_flights else tk.DISABLED)

def add\_multi\_city\_leg(self, parent\_frame):

leg\_frame = ttk.Frame(parent\_frame)

leg\_frame.pack(fill=tk.X, pady=5)

ttk.Label(leg\_frame, text="From:").grid(row=0, column=0, padx=5)

from\_entry = ttk.Combobox(leg\_frame, values=["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Chennai","Kolkata", "Ahmedabad", "Pune", "Jaipur", "Lucknow"])

from\_entry.grid(row=0, column=1, padx=5)

ttk.Label(leg\_frame, text="To:").grid(row=0, column=2, padx=5)

to\_entry = ttk.Combobox(leg\_frame, values=["Mumbai", "Delhi", "Bangalore", "Hyderabad", "Chennai","Kolkata", "Ahmedabad", "Pune", "Jaipur", "Lucknow"])

to\_entry.grid(row=0, column=3, padx=5)

ttk.Label(leg\_frame, text="Date:").grid(row=0, column=4, padx=5)

date\_entry = Calendar(leg\_frame, selectmode='day', date\_pattern='yyyy-mm-dd')

date\_entry.grid(row=0, column=5, padx=5)

self.multi\_city\_legs.append({'from\_entry': from\_entry,

'to\_entry': to\_entry,'date\_entry': date\_entry,'frame': leg\_frame})

def search\_multi\_city\_flights(self):

for item in self.flights\_tree.get\_children():

self.flights\_tree.delete(item)

try:

all\_flights = []

for leg in self.multi\_city\_legs:

source = leg['from\_entry'].get().strip()

destination = leg['to\_entry'].get().strip()

date = leg['date\_entry'].get\_date()

if not (source and destination):

messagebox.showerror("Error", "Please fill all fields for each leg!")

return

flights = search\_flights(source, destination)

if flights:

for flight in flights:

all\_flights.append({'flight': flight,

'type': f"Leg {self.multi\_city\_legs.index(leg) + 1}"})

if all\_flights:

for flight\_info in all\_flights:

flight = flight\_info['flight']

self.flights\_tree.insert("", tk.END,

values=(flight[0], flight[1], flight[2], flight[3],

flight[4], flight[5], f"₹{flight[6]}" if flight[6] else "N/A",

f"₹{flight[7]}" if flight[7] else "N/A", flight\_info['type']))

if hasattr(self, 'book\_btn'):

self.book\_btn.config(state=tk.NORMAL)

else:

self.flights\_tree.insert("", tk.END, values=("No flights found", "", "", "", "", "", "", ""))

if hasattr(self, 'book\_btn'):

self.book\_btn.config(state=tk.DISABLED)

except Exception as e:

messagebox.showerror("Error", f"Database error: {str(e)}")

def view\_bookings(self, customer\_id):

bookings\_window = tk.Toplevel(self.root)

bookings\_window.title("My Bookings")

bookings\_window.geometry("800x500")

main\_frame = ttk.Frame(bookings\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame,yscrollcommand=scrollbar.set,

columns=("id", "name", "flight\_id", "source", "dest", "date", "seat", "class"),show="headings")

scrollbar.config(command=tree.yview)

tree.heading("id", text="Booking ID")

tree.heading("name", text="Name")

tree.heading("flight\_id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.heading("seat", text="Seat")

tree.heading("class", text="Class")

tree.column("id", width=80)

tree.column("name", width=80)

tree.column("flight\_id", width=80)

tree.column("source", width=100)

tree.column("dest", width=100)

tree.column("date", width=100)

tree.column("seat", width=80)

tree.column("class", width=100)

tree.pack(fill=tk.BOTH, expand=True)

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("""SELECT b.id, b.passenger\_name, b.flight\_id, f.source, f.destination,

f.departure\_date, b.seat\_id, b.seat\_class FROM bookings b

JOIN flights f ON b.flight\_id = f.id WHERE b.customer\_id = %s""", (customer\_id,))

bookings = cursor.fetchall()

if bookings:

for booking in bookings:

tree.insert("", tk.END, values=(booking[0], booking[1], booking[2], booking[3],

booking[4], booking[5], booking[6], booking[7]))

else:

tree.insert("", tk.END, values=("No bookings found", "", "", "", "", "", "", ""))

except Exception as e:

messagebox.showerror("Error", f"Database error: {str(e)}")

finally:

conn.close()

button\_frame = ttk.Frame(main\_frame)

button\_frame.pack(fill=tk.X, pady=10)

def generate\_pdf\_ticket():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a booking first!")

return

booking\_id = tree.item(selected\_item)['values'][0]

self.generate\_ticket\_pdf\_by\_booking(booking\_id)

ttk.Button(button\_frame, text="Generate PDF Ticket",

command=generate\_pdf\_ticket,style='TButton').pack(side=tk.LEFT, padx=5)

def cancel\_booking():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a booking first!")

return

booking\_id = tree.item(selected\_item)['values'][0]

result = cancel\_ticket(booking\_id)

if result is True:

messagebox.showinfo("Success", "Booking cancelled successfully!")

bookings\_window.destroy()

else:

messagebox.showerror("Error", result)

ttk.Button(button\_frame, text="Cancel Booking",

command=cancel\_booking,style='TButton').pack(side=tk.LEFT, padx=5)

def generate\_ticket\_pdf(self, customer\_id, flight\_id, seat\_id, seat\_class):

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("SELECT email FROM customer WHERE id = %s", (customer\_id,))

customer\_email = cursor.fetchone()[0]

cursor.execute("""SELECT source, destination, departure\_date FROM flights WHERE id = %s""", (flight\_id,))

flight = cursor.fetchone()

if not flight:

messagebox.showerror("Error", "Flight not found!")

return source, destination, departure\_date = flight

pdf = FPDF()

pdf.add\_page()

except Exception as e:

messagebox.showerror("Error", f"Failed to generate ticket: {str(e)}")

finally:

conn.close()

def generate\_ticket\_pdf\_by\_booking(self, booking\_id):

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("""SELECT b.passenger\_name, f.source, f.destination,

f.departure\_date, f.departure\_time, f.arrival\_time, b.seat\_id, b.seat\_class, b.id

FROM bookings b JOIN flights f ON b.flight\_id = f.id JOIN customer c ON b.customer\_id = c.id WHERE b.id = %s """, (booking\_id,))

booking = cursor.fetchone()

if not booking:

messagebox.showerror("Error", "Booking not found!")

return (passenger\_name, source, destination, departure\_date,

departure\_time, arrival\_time, seat\_number, seat\_class, booking\_id) = booking

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", 'B', 16)

pdf.cell(0, 10, "Celestia Airlines - E-Ticket", 0, 1, 'C')

pdf.ln(10)

pdf.set\_font("Arial", 'B', 12)

pdf.cell(0, 10, f"Passenger: {passenger\_name}", 0, 1)

pdf.cell(0, 10, f"Flight: {source} to {destination}", 0, 1)

pdf.cell(0, 10, f"Date: {departure\_date}", 0, 1)

pdf.cell(0, 10, f"Time: {departure\_time} - {arrival\_time}", 0, 1)

pdf.ln(5)

pdf.cell(0, 10, f"Seat: {seat\_number} ({seat\_class})", 0, 1)

pdf.cell(0, 10, f"Booking ID: {booking\_id}", 0, 1)

pdf.ln(10)

pdf.set\_font("Arial", 'I', 10)

pdf.cell(0, 10, "Thank you for choosing Celestia Airlines!", 0, 1, 'C')

filename = f"tickets/ticket\_booking\_{booking\_id}.pdf"

pdf.output(filename)

messagebox.showinfo("Success", f"Ticket saved as {filename}")

except Exception as e:

messagebox.showerror("Error", f"Failed to generate ticket: {str(e)}")

finally:

conn.close()

def cancel\_ticket(self, customer\_id):

cancel\_window = tk.Toplevel(self.root)

cancel\_window.title("Cancel Ticket")

cancel\_window.geometry("600x500")

main\_frame = ttk.Frame(cancel\_window)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

tree\_frame = ttk.Frame(main\_frame)

tree\_frame.pack(fill=tk.BOTH, expand=True)

scrollbar = ttk.Scrollbar(tree\_frame)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree = ttk.Treeview(tree\_frame, yscrollcommand=scrollbar.set,show="headings",selectmode="browse",

columns=("id", "flight\_id", "source", "dest", "date", "seat", "class"))

scrollbar.config(command=tree.yview)

tree.heading("id", text="Booking ID")

tree.heading("name", text="Passenger Name")

tree.heading("flight\_id", text="Flight ID")

tree.heading("source", text="From")

tree.heading("dest", text="To")

tree.heading("date", text="Date")

tree.heading("seat", text="Seat")

tree.heading("class", text="Class")

tree.column("id", width=80)

tree.column("name" , width=80)

tree.column("flight\_id", width=80)

tree.column("source", width=100)

tree.column("dest", width=100)

tree.column("date", width=100)

tree.column("seat", width=80)

tree.column("class", width=100)

tree.pack(fill=tk.BOTH, expand=True)

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("""SELECT b.id, b.flight\_id, f.source, f.destination,

f.departure\_date, b.seat\_id, b.seat\_class FROM bookings b

JOIN flights f ON b.flight\_id = f.id WHERE b.customer\_id = %s""", (customer\_id,))

bookings = cursor.fetchall()

if bookings:

for booking in bookings:

tree.insert("", tk.END, values=(booking[0], booking[1], booking[2],

booking[3], booking[4], booking[5], booking[6]))

except Exception as e:

messagebox.showerror("Error", f"Database error: {str(e)}")

finally:

conn.close()

def perform\_cancellation():

selected\_item = tree.focus()

if not selected\_item:

messagebox.showerror("Error", "Please select a booking to cancel!")

return booking\_id = tree.item(selected\_item)['values'][0]

result = cancel\_ticket(booking\_id)

if result is True:

messagebox.showinfo("Success", "Booking cancelled successfully!")

cancel\_window.destroy()

else:

messagebox.showerror("Error", result)

button\_frame = ttk.Frame(main\_frame)

button\_frame.pack(fill=tk.X, pady=10)

ttk.Button(button\_frame, text="Cancel Selected Ticket",command=perform\_cancellation,style='TButton').pack(side=tk.LEFT, padx=5)

ttk.Button(button\_frame, text="Close", command=cancel\_window.destroy).pack(side=tk.RIGHT, padx=5)

def book\_selected\_flight(self, customer\_id):

selected\_items = self.flights\_tree.selection()

if not selected\_items:

messagebox.showerror("Error", "Please select at least one flight!")

return

booking\_window = tk.Toplevel(self.root)

booking\_window.title("Book Flight")

booking\_window.geometry("600x400")

flight\_info\_frame = ttk.Frame(booking\_window)

flight\_info\_frame.pack(pady=10)

flight\_id = self.flights\_tree.item(selected\_items[0])['values'][0]

source = self.flights\_tree.item(selected\_items[0])['values'][1]

destination = self.flights\_tree.item(selected\_items[0])['values'][2]

date = self.flights\_tree.item(selected\_items[0])['values'][3]

ttk.Label(flight\_info\_frame, text=f"Flight {flight\_id}: {source} to {destination} on {date}").pack()

class\_frame = ttk.LabelFrame(booking\_window, text="Class Selection")

class\_frame.pack(pady=10)

seat\_class\_var = tk.StringVar(value="economy")

ttk.Radiobutton(class\_frame, text="Economy", variable=seat\_class\_var, value="economy").pack(anchor=tk.W)

ttk.Radiobutton(class\_frame, text="Business", variable=seat\_class\_var, value="business").pack(anchor=tk.W)

ttk.Radiobutton(class\_frame, text="First Class", variable=seat\_class\_var, value="first\_class").pack(anchor=tk.W)

details\_frame = ttk.LabelFrame(booking\_window, text="Passenger Details")

details\_frame.pack(pady=10)

ttk.Label(details\_frame, text="Passenger Name:").pack()

name\_entry = ttk.Entry(details\_frame)

name\_entry.pack()

ttk.Label(booking\_window, text="Seat Number:").pack()

seat\_entry = ttk.Entry(booking\_window)

seat\_entry.pack()

def perform\_booking():

seat\_id = seat\_entry.get()

seat\_class = seat\_class\_var.get()

passenger\_name = name\_entry.get()

if not all([seat\_id, passenger\_name]):

messagebox.showerror("Error", "Please fill all fields!")

return result = book\_ticket(customer\_id,flight\_id,seat\_id,

seat\_class,passenger\_name,id)

if result is True:

messagebox.showinfo("Success", "Booking successful!")

booking\_window.destroy()

self.view\_bookings(customer\_id) # Refresh bookings view

else:

messagebox.showerror("Error", result)

ttk.Button(booking\_window, text="Confirm Booking", command=perform\_booking).pack(pady=10)

def show\_seats\_for\_booking(self, parent\_frame, customer\_id):

for widget in self.seat\_grid\_frame.winfo\_children():

widget.destroy()

try:

flight\_id = int(self.book\_flight\_entry.get())

except ValueError:

messagebox.showerror("Error", "Invalid Flight ID! Must be a number")

return

if not flight\_exists(flight\_id):

messagebox.showerror("Error", "Flight ID does not exist!")

return

try:

booked\_seats = get\_booked\_seats(flight\_id)

self.seat\_buttons = {} # Dictionary to store seat buttons

header\_frame = ttk.Frame(self.seat\_grid\_frame)

header\_frame.grid(row=0, column=0, columnspan=7, sticky='ew')

for col in range(6):

ttk.Label(header\_frame, text=str(col+1)).grid(row=0, column=col+1, padx=5)

for row\_idx in range(7): # Rows A-G

row\_label = ttk.Label(self.seat\_grid\_frame, text=chr(65 + row\_idx))

row\_label.grid(row=row\_idx + 1, column=0, padx=5, pady=5)

for col\_idx in range(6):

seat = f"{chr(65 + row\_idx)}{col\_idx + 1}"

is\_booked = seat in booked\_seats

btn = tk.Button(self.seat\_grid\_frame,text=seat,command=lambda s=seat: self.select\_seat(s),

width=4,state='disabled' if is\_booked else 'normal',

bg='#ff9999' if is\_booked else '#99ff99',fg='black',relief='raised')

btn.grid(row=row\_idx + 1, column=col\_idx + 1, padx=5, pady=5)

self.seat\_buttons[seat] = btn # Store button reference

self.seat\_grid\_frame.update\_idletasks()

self.canvas.config(scrollregion=self.canvas.bbox("all"))

self.current\_selected\_seat = None # Track currently selected seat

except Exception as e:

messagebox.showerror("Error", f"Failed to load seats: {str(e)}")

def select\_seat(self, seat):

if self.current\_selected\_seat and self.current\_selected\_seat in self.seat\_buttons:

btn = self.seat\_buttons[self.current\_selected\_seat]

if btn['state'] == 'normal': # Only reset available seats

btn.config(bg='#99ff99') # Reset to green

if seat in self.seat\_buttons:

btn = self.seat\_buttons[seat]

if btn['state'] == 'normal': # Only allow selection of available seats

btn.config(bg='#9999ff') # Blue for selected

self.current\_selected\_seat = seat

self.selected\_seat.set(seat) # Update the selected seat variable

else:

self.current\_selected\_seat = None

self.selected\_seat.set("")

def perform\_booking\_from\_tab(self, customer\_id):

try:

flight\_id = int(self.book\_flight\_entry.get())

if not flight\_exists(flight\_id):

messagebox.showerror("Error", "Flight ID does not exist!")

return

except ValueError:

messagebox.showerror("Error", "Please enter a valid Flight ID")

return

seat\_class = self.class\_var.get()

seat\_id = self.selected\_seat.get()

passenger\_name = self.name\_entry.get().strip()

if not passenger\_name:

messagebox.showerror("Error", "Please enter passenger name")

return

if not seat\_id:

messagebox.showerror("Error", "Please select a seat")

return

result = book\_ticket(customer\_id=customer\_id,flight\_id=flight\_id,

seat\_id=seat\_id,seat\_class=seat\_class,passenger\_name=passenger\_name)

if result is True:

messagebox.showinfo("Success", f"Seat {seat\_id} ({seat\_class}) booked successfully!")

booking\_id = self.get\_last\_booking\_id(customer\_id)

if booking\_id:

self.generate\_ticket\_pdf\_by\_booking(booking\_id)

self.selected\_seat.set("")

self.name\_entry.delete(0, tk.END)

self.show\_seats\_for\_booking(None, customer\_id)

self.view\_bookings(customer\_id)

else:

messagebox.showerror("Error", result)

def get\_last\_booking\_id(self, customer\_id):

conn = get\_db\_connection()

cursor = conn.cursor()

try:

cursor.execute("""SELECT id FROM bookings WHERE customer\_id = %s ORDER BY id DESC LIMIT 1""", (customer\_id,))

result = cursor.fetchone()

return result[0] if result else None

except Exception as e:

print(f"Error getting last booking ID: {e}")

return None

finally:

conn.close()

def run(self):

self.root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

app = CelestiaAirlinesApp()

app.run()

**database.py**

import mysql.connector

from tkinter import messagebox

from fpdf import FPDF

def get\_db\_connection():

try:

conn = mysql.connector.connect(host="localhost",user="root",password="", database="celestia\_airlines")

return conn

except mysql.connector.Error as err:

print(f"Database connection error: {err}")

messagebox.showerror("Database Error", f"Failed to connect to database: {err}")

return None

def get\_connection\_and\_cursor():

conn = get\_db\_connection()

if conn:

return conn, conn.cursor()

else:

return None, None

def customer\_login(email, password):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT id, name FROM customer WHERE email = %s AND password = %s", (email, password))

customer = cursor.fetchone()

conn.close()

return customer

def search\_flights(source, destination):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

query = """

SELECT id, source, destination, departure\_date, departure\_time, arrival\_time,economy\_price, business\_price, first\_class\_price

FROM flights WHERE source LIKE %s AND destination LIKE %s"""

cursor.execute(query, (f"%{source}%", f"%{destination}%"))

flights = cursor.fetchall()

conn.close()

return flights

def get\_booked\_seats(flight\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("SELECT seat\_id FROM bookings WHERE flight\_id = %s", (flight\_id,))

booked\_seats = [row[0] for row in cursor.fetchall()]

return booked\_seats

except Exception as e:

return str(e)

finally:

conn.close()

def book\_ticket(customer\_id, flight\_id, seat\_id, seat\_class, passenger\_name):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("SELECT id FROM flights WHERE id = %s", (flight\_id,))# Check if flight exists

if not cursor.fetchone(): # Make sure to fetch the result

return "Flight not found"

cursor.execute("SELECT id FROM bookings WHERE flight\_id = %s AND seat\_id = %s", (flight\_id, seat\_id))

if cursor.fetchone(): # Make sure to fetch the result

return "Seat already booked"

price\_column = f"{seat\_class}\_price"

cursor.execute(f"SELECT {price\_column} FROM flights WHERE id = %s", (flight\_id,))

price\_result = cursor.fetchone() # Make sure to fetch the result

if not price\_result:

return "Could not determine price"

price = price\_result[0]

cursor.execute("""INSERT INTO bookings (id, customer\_id, flight\_id, seat\_id, seat\_class, passenger\_name, booking\_date)

VALUES (%s, %s, %s, %s, %s, %s, CURDATE())""", (id,customer\_id, flight\_id, seat\_id, seat\_class, passenger\_name))

conn.commit()

return True

except mysql.connector.Error as err:

conn.rollback()

return f"Database error: {err}"

finally:

conn.close()

def cancel\_ticket(booking\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("SELECT customer\_id, flight\_id FROM bookings WHERE id = %s", (booking\_id,))

booking = cursor.fetchone()

if booking:

customer\_id, flight\_id = booking

cursor.execute("DELETE FROM bookings WHERE id = %s", (booking\_id,))

cursor.execute("""UPDATE customer SET flight\_id = NULL WHERE id = %s AND flight\_id = %s

AND EXISTS (SELECT 1 FROM information\_schema.columns WHERE table\_name = 'customer' AND column\_name = 'flight\_id'

)""", (customer\_id, flight\_id))

conn.commit()

return True

except Exception as e:

conn.rollback()

return str(e)

finally:

conn.close()

def get\_all\_flights():

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return []

cursor.execute("SELECT \* FROM flights")

flights = cursor.fetchall()

conn.close()

return flights

def get\_all\_crew():

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return []

cursor.execute("SELECT \* FROM crew")

crew = cursor.fetchall()

conn.close()

return crew

def get\_all\_pilots():

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return []

cursor.execute("SELECT id, name, age, flight\_id FROM pilots")

pilots = cursor.fetchall()

conn.close()

return pilots

def add\_pilot(name, age):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("""SELECT f.id FROM flights f LEFT JOIN pilots p ON f.id = p.flight\_id GROUP BY f.id HAVING COUNT(p.id) < 2 LIMIT 1""")

flight\_result = cursor.fetchone()

flight\_id = flight\_result[0] if flight\_result else None

if flight\_id:

cursor.execute("""INSERT INTO pilots (name, age, flight\_id) VALUES (%s, %s, %s)""", (name, age, flight\_id))

else:

cursor.execute("""INSERT INTO pilots (name, age, flight\_id) VALUES (%s, %s, NULL)""", (name, age))

conn.commit()

return True

except Exception as e:

conn.rollback()

return str(e)

finally:

conn.close()

def add\_crew(name, age):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("""SELECT flights.id FROM flights LEFT JOIN crew ON flights.id = crew.flight\_id

GROUP BY flights.id HAVING COUNT(crew.id) < 4 LIMIT 1""")

flight\_result = cursor.fetchone()

flight\_id = flight\_result[0] if flight\_result else None

if flight\_id:

cursor.execute("INSERT INTO crew (name, age, flight\_id) VALUES (%s, %s, %s)", (name, age, flight\_id))

else:

cursor.execute("INSERT INTO crew (name, age, flight\_id) VALUES (%s, %s, NULL)", (name, age))

conn.commit()

return True

except Exception as e:

conn.rollback()

return str(e)

finally:

conn.close()

def add\_flight(source, destination, departure\_date, departure\_time, arrival\_time,first\_class\_seats=10,

economy\_price, business\_price, first\_class\_price,economy\_seats=60, business\_seats=20):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return False, "Database connection error"

try:

cursor.execute('''INSERT INTO flights

(source, destination, departure\_date, departure\_time, arrival\_time,economy\_price, business\_price, first\_class\_price,

economy\_seats, business\_seats, first\_class\_seats)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''', (source, destination, departure\_date, departure\_time, arrival\_time,

economy\_price, business\_price, first\_class\_price,economy\_seats, business\_seats, first\_class\_seats))

conn.commit()

return True, None

except mysql.connector.Error as err:

conn.rollback()

return False, f"Database error: {err}"

finally:

conn.close()

def cancel\_flight(flight\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return False, "Database connection error"

try:

cursor.execute("UPDATE customer SET flight\_id = NULL WHERE flight\_id = %s", (flight\_id,))

cursor.execute("UPDATE crew SET flight\_id = NULL WHERE flight\_id = %s", (flight\_id,))

cursor.execute("UPDATE pilots SET flight\_id = NULL WHERE flight\_id = %s", (flight\_id,))

cursor.execute("DELETE FROM bookings WHERE flight\_id = %s", (flight\_id,))

cursor.execute("DELETE FROM flights WHERE id = %s", (flight\_id,))

conn.commit()

return True, None

except Exception as e:

conn.rollback()

return False, str(e)

finally:

conn.close()

def view\_bookings(customer\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return []

try:

cursor.execute("""SELECT b.id, b.flight\_id, f.source, f.destination,

f.departure\_date, b.seat\_id, b.seat\_class FROM bookings b

JOIN flights f ON b.flight\_id = f.id WHERE b.customer\_id = %s """, (customer\_id,))

bookings = cursor.fetchall()

return bookings

except Exception as e:

return str(e)

finally:

conn.close()

def remove\_crew(crew\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("DELETE FROM crew WHERE id = %s", (crew\_id,))

conn.commit()

return True

except Exception as e:

conn.rollback()

return str(e)

finally:

conn.close()

def remove\_pilot(pilot\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute("DELETE FROM pilots WHERE id = %s", (pilot\_id,))

conn.commit()

return True

except Exception as e:

conn.rollback()

return str(e)

finally:

conn.close()

def view\_pilots():

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT id, name, age, flight\_id FROM pilots")

pilots = cursor.fetchall()

conn.close()

return pilots

def view\_crew():

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return []

cursor.execute("SELECT id, name,flight\_id, age FROM crew")

crew = cursor.fetchall()

conn.close()

return crew

def flight\_exists(flight\_id):

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT EXISTS(SELECT 1 FROM flights WHERE id = %s)", (flight\_id,))

exists = cursor.fetchone()[0]

conn.close()

return bool(exists)

def get\_booking\_details(booking\_id):

conn, cursor = get\_connection\_and\_cursor()

if not conn:

return "Database connection error"

try:

cursor.execute(""" SELECT b.id, f.source, f.destination, f.departure\_date, f.departure\_time, f.arrival\_time,

b.seat\_id, b.seat\_class FROM bookings b JOIN flights f ON b.flight\_id = f.id WHERE b.id = %s""", (booking\_id,))

return cursor.fetchone()

except Exception as e:

return str(e)

finally:

conn.close()

def generate\_booking\_pdf(booking\_id, filename):

booking = get\_booking\_details(booking\_id)

if not booking or isinstance(booking, str):

return False, booking if isinstance(booking, str) else "Booking not found"

try:

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

labels = ["Booking ID", "Source", "Destination", "Date", "Seat ID", "Class"]

for i, value in enumerate(booking):

pdf.cell(200, 10, txt=f"{labels[i]}: {value}", ln=1)

pdf.output(filename)

return True, None

except Exception as e:

return False, str(e)

**Output(snaps shots):-**