|  |
| --- |
| **Flight ticket booking system**  **21CSS101J – PROGRAMMING FOR PROBLEM-SOLVING**  **Mini Project Report**  *Submitted by*  **DHAVALA V D M ADITHYA NAIDU [Reg. No.: RA2311026010448]**  **B.Tech. CSE – AI&ML**  **YANAMADALA SATYANARAYANA [Reg. No.: RA2311026010493]**  **B.Tech. CSE – AI&ML**  **SRMIST-01.jpg**  **SCHOOL OF COMPUTING**  **COLLEGE OF ENGINEERING AND TECHNOLOGY**  **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  **(Under Section 3 of UGC Act, 1956)**  S.R.M. NAGAR, KATTANKULATHUR – 603 203  CHENGALPATTU DISTRICT  **November 2023** |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Title** | **Page No.** |
| 1 | Problem Statement | 4 |
| 2 | Methodology / Procedure/ Algorithm | 6 |
| 3 | Flowchart | 7 |
| 4 | Coding (Python) | 8 |
| 5 | Front-end code | 11 |
| 6 | Modules of the proposed work | 17 |
| 7 | Results/Screenshots | 18 |
| 8 | Conclusion | 21 |
| 9 | References | 22 |

**1 . Problem Statement**

You are tasked with developing a Flight Ticket Booking System to streamline the ticket reservation process for an airline company. The system should allow users to check available seats, view destinations, explore payment methods, book tickets, and cancel reservations. The program should maintain the state of available seats, booked seats, and other relevant information.

Requirements:

1. Available Seats:

- The system should display the number of available seats for each class (Economy, Business, First Class) whenever the user chooses the corresponding option.

2. Destinations:

- Users should be able to view a list of available destinations along with their airport codes.

3. Payment Methods:

- The system should present a list of accepted payment methods (Credit Card - CC, Debit Card - DC, PayPal - PP).

4. Ticket Booking:

- Users should be able to book tickets by specifying the class (Economy/Business/First Class), the number of tickets, destination, and the preferred payment method.

- The system should check for seat availability and process the booking accordingly.

5. Ticket Cancellation:

- Users should be able to cancel booked tickets by providing the class, the number of tickets to cancel, destination, and the payment method for the refund.

- The system should verify the validity of the cancellation request and process the refund.

6. Quit:

- Users should have the option to exit the program.

Implementation Guidelines:

- Use a class-based approach to encapsulate the functionalities of the Flight Ticket Booking System.

- Implement appropriate data structures to store information about available seats, prices, booked seats, destinations, and payment methods.

- Create methods to handle the display of available seats, destinations, payment methods, booking, and cancellation.

- Implement error handling to manage invalid inputs and ensure the integrity of the booking and cancellation processes.

Example Scenario:

- The user selects "1" to display available seats and sees the current seat availability for each class.

- The user chooses "4" to book a ticket, enters the necessary details, and the system confirms the booking if seats are available.

- The user selects "5" to cancel a ticket, provides the required information, and the system processes the cancellation and issues a refund.

Implement the Flight Ticket Booking System according to the specified requirements, providing a user-friendly interface and ensuring the robustness of the booking.

**2. Methodology / Procedure/ Algorithm**

Step 1: Initialization

- Initialize dictionaries for `available\_seats`, `prices`, and `booked\_seats` for each ticket class.

- Initialize a dictionary for `destinations`.

- Initialize a list for `payment\_methods`.

Step 2: Main Loop

- Enter a loop to repeatedly display the menu and process user input.

Step 3: Display Menu

- Print the menu with options:

1. Display Available Seats

2. Display Destinations

3. Display Payment Methods

4. Book Ticket

5. Cancel Ticket

6. Quit

Step 4: User Input

- Prompt the user to enter their choice from the menu.

Step 5: Process User Choice

- If the user's choice is 1:

- Display available seats for each ticket class.

- If the user's choice is 2:

- Display available destinations.

- If the user's choice is 3:

- Display available payment methods.

- If the user's choice is 4:

- Prompt the user to enter the class, number of tickets, destination, and payment method.

- Check if there are enough available seats in the chosen class.

- Book the tickets and simulate the payment.

- If the user's choice is 5:

- Prompt the user to enter the class, number of tickets, destination, and payment method for cancellation.

- Check if the number of tickets to cancel is valid.

- Cancel the tickets and simulate the refund.

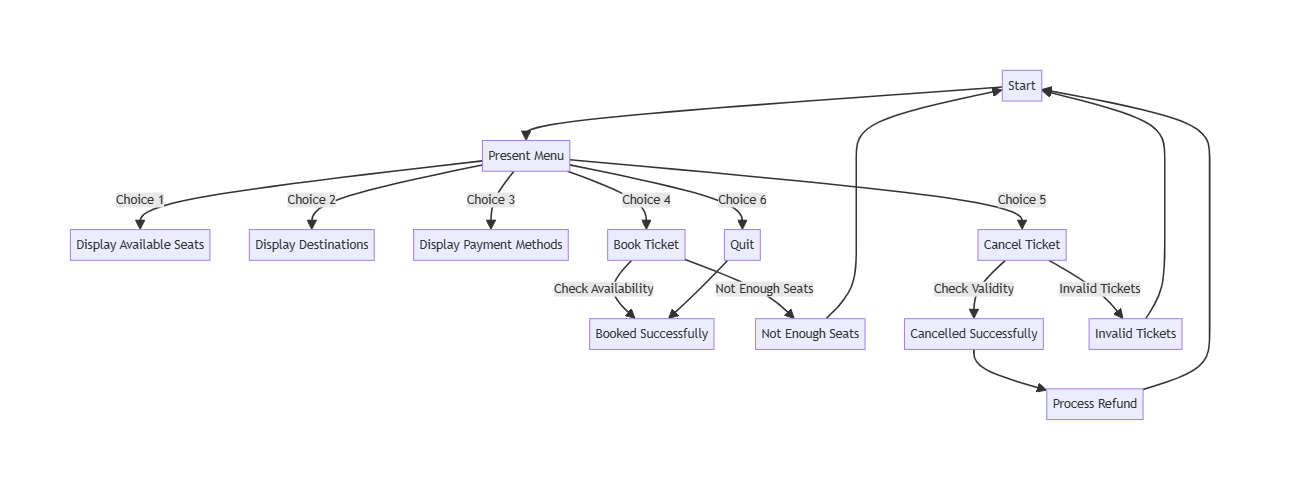
- If the user's choice is 6:

- Exit the loop and end the program.

Step 6: End

- The program ends when the user chooses to quit.

**3. Flow chart**

****

**4. Coding (Python)**

Python, a versatile language, simplifies web development and aids education. Its automation streamlines tasks, while its adaptability fuels scientific computing for data analysis and AI applications. Python's impact is evident in popular platforms like Instagram and YouTube, showcasing its integral role in our interconnected digital experiences.

# flight\_booking.py

class FlightTicketBookingSystem:

available\_seats = {

'E': 50,

'B': 20,

'FC': 10

}

prices = {

'E': 300,

'B': 600,

'FC': 1000

}

booked\_seats = {

'E': 0,

'B': 0,

'FC': 0

}

destinations = {

'New York': 'NYC',

'Los Angeles': 'LAX',

'Chicago': 'ORD',

'Miami': 'MIA'

}

payment\_methods = ['CC', 'DC', 'PP']

while True:

print("\nMenu:")

print("1. Display Available Seats")

print("2. Display Destinations")

print("3. Display Payment Methods")

print("4. Book Ticket")

print("5. Cancel Ticket")

print("6. Quit")

choice = input("Enter your choice: ")

if choice == '1':

print("\nAvailable Seats:")

for class\_name, seats in available\_seats.items():

print(f"{class\_name}: {seats} seats")

elif choice == '2':

print("\nAvailable Destinations:")

for city, code in destinations.items():

print(f"{city} ({code})")

elif choice == '3':

print("\nAvailable Payment Methods:")

for method in payment\_methods:

print(method)

elif choice == '4':

class\_name = input("Enter class (Economy/Business/First Class): ")

num\_tickets = int(input("Enter the number of tickets to book: "))

destination = input("Enter destination (NYC/LAX/ORD/MIA): ")

payment\_method = input("Enter payment method: ")

if num\_tickets <= available\_seats[class\_name]:

available\_seats[class\_name] -= num\_tickets

booked\_seats[class\_name] += num\_tickets

total\_price = prices[class\_name] \* num\_tickets

print(f"Booking successful! {num\_tickets} {class\_name} ticket(s) to {destination}. Total Price: ${total\_price}")

print(f"Payment successful using {payment\_method}.")

else:

print(f"Sorry, there are not enough available {class\_name} seats.")

elif choice == '5':

class\_name = input("Enter class (Economy/Business/First Class): ")

num\_tickets = int(input("Enter the number of tickets to cancel: "))

destination = input("Enter destination (NYC/LAX/ORD/MIA): ")

payment\_method = input("Enter payment method for refund: ")

if num\_tickets <= booked\_seats[class\_name]:

available\_seats[class\_name] += num\_tickets

booked\_seats[class\_name] -= num\_tickets

total\_refund = prices[class\_name] \* num\_tickets

print(f"Cancellation successful! {num\_tickets} {class\_name} ticket(s) to {destination} cancelled. Total Refund: ${total\_refund}")

print(f"Refund processed to your {payment\_method}.")

else:

print("Invalid number of tickets to cancel.")

elif choice == '6':

break

else:

print("Invalid choice. Please try again.")

**5. Front-end code**

import tkinter as tk

from tkinter import ttk, simpledialog, messagebox

class FlightBookingGUI:

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

self.root = root

self.root.title("Flight Ticket Booking System")

self.style = ttk.Style()

self.style.configure("TFrame", background="#f0f0f0")

self.style.configure("TLabel", background="#f0f0f0", font=("Arial", 12))

self.style.configure("TButton", background="#f0f0f0", font=("Arial", 12))

self.style.configure("TButtonBlack.TButton", foreground="black", background="#f0f0f0", font=("Arial", 12))

self.style.configure("TText", background="#e0e0e0", font=("Arial", 12))

self.style.configure("TEntry.TEntry", fieldbackground="black", foreground="white", bordercolor="black", font=("Arial", 12))

self.frame = ttk.Frame(root, style="TFrame")

self.frame.pack(fill=tk.BOTH, expand=True)

self.available\_seats\_label = ttk.Label(self.frame, text="Available Seats:", style="TLabel")

self.available\_seats\_label.pack()

self.available\_seats\_text = tk.Text(self.frame, height=10, width=50, wrap=tk.WORD, state=tk.DISABLED, bg="#c2f0c2", font=("Arial", 12))

self.available\_seats\_text.pack()

self.destinations\_label = ttk.Label(self.frame, text="Available Destinations:", style="TLabel")

self.destinations\_label.pack()

self.destinations\_text = tk.Text(self.frame, height=5, width=50, wrap=tk.WORD, state=tk.DISABLED, bg="#c2f0c2", font=("Arial", 12))

self.destinations\_text.pack()

self.payment\_methods\_label = ttk.Label(self.frame, text="Available Payment Methods:", style="TLabel")

self.payment\_methods\_label.pack()

self.payment\_methods\_text = tk.Text(self.frame, height=2, width=50, wrap=tk.WORD, state=tk.DISABLED, bg="#c2f0c2", font=("Arial", 12))

self.payment\_methods\_text.pack()

self.choice\_label = ttk.Label(self.frame, text="Enter your choice:", style="TLabel")

self.choice\_label.pack()

self.choice\_entry = ttk.Entry(self.frame, style="TEntry")

self.choice\_entry.pack()

self.submit\_button = ttk.Button(self.frame, text="Submit", command=self.handle\_choice, style="TButtonBlack.TButton")

self.submit\_button.pack()

self.output\_text = tk.Text(self.frame, height=10, width=50, wrap=tk.WORD, state=tk.DISABLED, bg="#ffc2b3", font=("Arial", 12))

self.output\_text.pack()

# Initializing the GUI with available information

self.update\_available\_seats()

self.update\_destinations()

self.update\_payment\_methods()

def update\_available\_seats(self):

seats\_text = "Available Seats:\n"

for class\_name, seats in available\_seats.items():

seats\_text += f"{class\_name}: {seats} seats\n"

self.available\_seats\_text.config(state=tk.NORMAL)

self.available\_seats\_text.delete(1.0, tk.END)

self.available\_seats\_text.insert(tk.END, seats\_text)

self.available\_seats\_text.config(state=tk.DISABLED)

def update\_destinations(self):

destinations\_text = ""

for city, code in destinations.items():

destinations\_text += f"{city} ({code})\n"

self.destinations\_text.config(state=tk.NORMAL)

self.destinations\_text.delete(1.0, tk.END)

self.destinations\_text.insert(tk.END, destinations\_text)

self.destinations\_text.config(state=tk.DISABLED)

def update\_payment\_methods(self):

methods\_text = "\n".join(payment\_methods)

self.payment\_methods\_text.config(state=tk.NORMAL)

self.payment\_methods\_text.delete(1.0, tk.END)

self.payment\_methods\_text.insert(tk.END, methods\_text)

self.payment\_methods\_text.config(state=tk.DISABLED)

def handle\_choice(self):

try:

choice = int(self.choice\_entry.get())

if 1 <= choice <= 6:

if choice == 4 or choice == 5:

self.execute\_transaction(choice)

else:

self.display\_output("Invalid choice for direct execution.\n", "red")

else:

self.display\_output("Invalid choice. Please try again.\n", "red")

except ValueError:

self.display\_output("Invalid input. Please enter a valid number.\n", "red")

def execute\_transaction(self, choice):

if choice == 4:

self.book\_ticket()

elif choice == 5:

self.cancel\_ticket()

def book\_ticket(self):

class\_name = self.show\_entry\_dialog("Enter class (Economy/Business/First Class): ")

num\_tickets = int(self.show\_entry\_dialog("Enter the number of tickets to book: "))

destination = self.show\_entry\_dialog("Enter destination (NYC/LAX/ORD/MIA): ")

payment\_method = self.show\_entry\_dialog("Enter payment method (CC/DC/PP): ")

if payment\_method == "CC" or payment\_method == "DC":

self.show\_card\_details\_window(class\_name, num\_tickets, destination)

elif payment\_method == "PP":

self.show\_paypal\_details\_window(class\_name, num\_tickets, destination)

else:

self.display\_output("Invalid payment method. Please choose CC, DC, or PP.\n", "red")

def cancel\_ticket(self):

class\_name = self.show\_entry\_dialog("Enter class (Economy/Business/First Class): ")

num\_tickets = int(self.show\_entry\_dialog("Enter the number of tickets to cancel: "))

destination = self.show\_entry\_dialog("Enter destination (NYC/LAX/ORD/MIA): ")

payment\_method = self.show\_entry\_dialog("Enter payment method for refund (CC/DC/PP): ")

if payment\_method == "CC" or payment\_method == "DC":

self.show\_card\_details\_window(class\_name, num\_tickets, destination, refund=True)

elif payment\_method == "PP":

self.show\_paypal\_details\_window(class\_name, num\_tickets, destination, refund=True)

else:

self.display\_output("Invalid payment method. Please choose CC, DC, or PP.\n", "red")

def show\_entry\_dialog(self, prompt):

return simpledialog.askstring("Input", prompt, parent=self.root)

def show\_card\_details\_window(self, class\_name, num\_tickets, destination, refund=False):

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

card\_window.title("Card Details")

card\_window.geometry("300x200")

card\_label = tk.Label(card\_window, text="Card Number:", font=("Helvetica", 12))

card\_label.pack()

card\_entry = tk.Entry(card\_window, font=("Helvetica", 12), show="\*")

card\_entry.pack()

expiry\_label = tk.Label(card\_window, text="Expiry Date (MM/YY):", font=("Helvetica", 12))

expiry\_label.pack()

expiry\_entry = tk.Entry(card\_window, font=("Helvetica", 12))

expiry\_entry.pack()

cvv\_label = tk.Label(card\_window, text="CVV:", font=("Helvetica", 12))

cvv\_label.pack()

cvv\_entry = tk.Entry(card\_window, font=("Helvetica", 12), show="\*")

cvv\_entry.pack()

confirm\_button = tk.Button(card\_window, text="Confirm Payment", command=lambda: self.confirm\_card\_payment(card\_window, class\_name, num\_tickets, destination, card\_entry, expiry\_entry, cvv\_entry, refund))

confirm\_button.pack()

def show\_paypal\_details\_window(self, class\_name, num\_tickets, destination, refund=False):

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

paypal\_window.title("PayPal Details")

paypal\_window.geometry("300x200")

paypal\_label = tk.Label(paypal\_window, text="PayPal Email:", font=("Helvetica", 12))

paypal\_label.pack()

paypal\_entry = tk.Entry(paypal\_window, font=("Helvetica", 12))

paypal\_entry.pack()

confirm\_button = tk.Button(paypal\_window, text="Confirm Payment", command=lambda: self.confirm\_paypal\_payment(paypal\_window, class\_name, num\_tickets, destination, paypal\_entry, refund))

confirm\_button.pack()

def confirm\_card\_payment(self, card\_window, class\_name, num\_tickets, destination, card\_entry, expiry\_entry, cvv\_entry, refund):

card\_number = card\_entry.get()

expiry\_date = expiry\_entry.get()

cvv = cvv\_entry.get()

if card\_number and expiry\_date and cvv:

card\_window.destroy()

if refund:

self.complete\_refund(class\_name, num\_tickets, destination)

else:

self.complete\_purchase(class\_name, num\_tickets, destination)

else:

messagebox.showerror("Error", "Incomplete card details. Please fill all fields.")

def confirm\_paypal\_payment(self, paypal\_window, class\_name, num\_tickets, destination, paypal\_entry, refund):

paypal\_email = paypal\_entry.get()

if paypal\_email:

paypal\_window.destroy()

if refund:

self.complete\_refund(class\_name, num\_tickets, destination)

else:

self.complete\_purchase(class\_name, num\_tickets, destination)

else:

messagebox.showerror("Error", "Incomplete PayPal details. Please fill the email field.")

def complete\_purchase(self, class\_name, num\_tickets, destination):

total\_price = prices[class\_name] \* num\_tickets

output = f"Booking successful! {num\_tickets} {class\_name} ticket(s) to {destination}. Total Price: ${total\_price}\n"

output += f"Payment successful.\n"

self.display\_output(output, "green")

# Updating available seats after booking

available\_seats[class\_name] -= num\_tickets

booked\_seats[class\_name] += num\_tickets

self.update\_available\_seats()

def complete\_refund(self, class\_name, num\_tickets, destination):

total\_refund = prices[class\_name] \* num\_tickets

output = f"Cancellation successful! {num\_tickets} {class\_name} ticket(s) to {destination} cancelled. Total Refund: ${total\_refund}\n"

output += f"Refund processed.\n"

self.display\_output(output, "green")

# Updating available seats after cancellation

available\_seats[class\_name] += num\_tickets

booked\_seats[class\_name] -= num\_tickets

self.update\_available\_seats()

def display\_output(self, output, color):

self.output\_text.config(state=tk.NORMAL)

self.output\_text.insert(tk.END, output, color)

self.output\_text.config(state=tk.DISABLED)

# Sample data

available\_seats = {'E': 50, 'B': 20, 'FC': 10}

prices = {'E': 300, 'B': 600, 'FC': 1000}

booked\_seats = {'E': 0, 'B': 0, 'FC': 0}

destinations = {'New York': 'NYC', 'Los Angeles': 'LAX', 'Chicago': 'ORD', 'Miami': 'MIA'}

payment\_methods = ['CC', 'DC', 'PP']

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = FlightBookingGUI(root)

root.mainloop()

**6. Modules of the proposed work**

1. Built-in Data Types:

- Dictionaries (`available\_seats`, `prices`, `booked\_seats`, `destinations`): Used to store information about available seats, prices, booked seats, and destinations.

- List (`payment\_methods`): Contains available payment methods.

2. Loops:

- `while True` loop: Used for creating an infinite loop to continuously display the menu and process user input until the user chooses to quit (`choice == '6'`).

3. Input/Output:

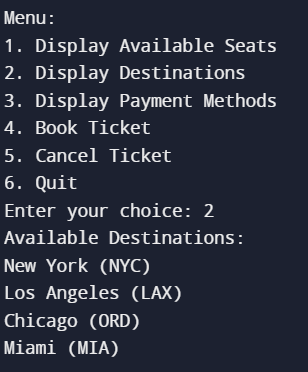
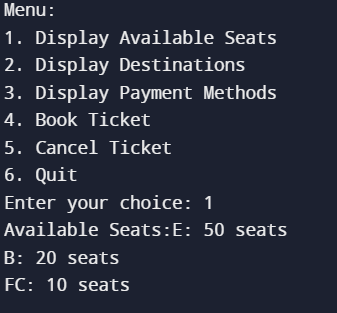
- `input()`: Used to get user input.

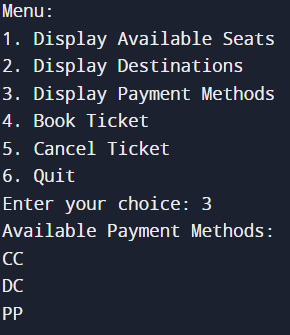
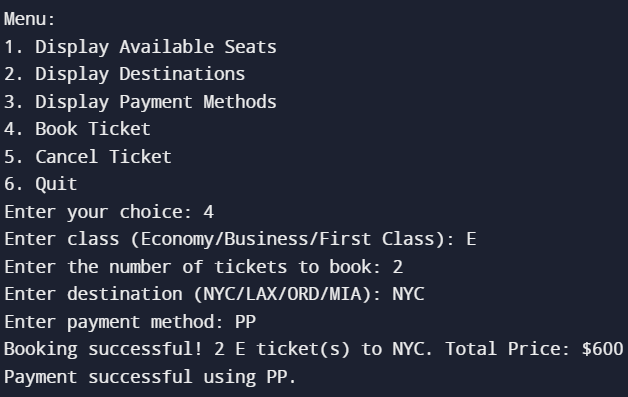
- `print()`: Used to display information and the menu.

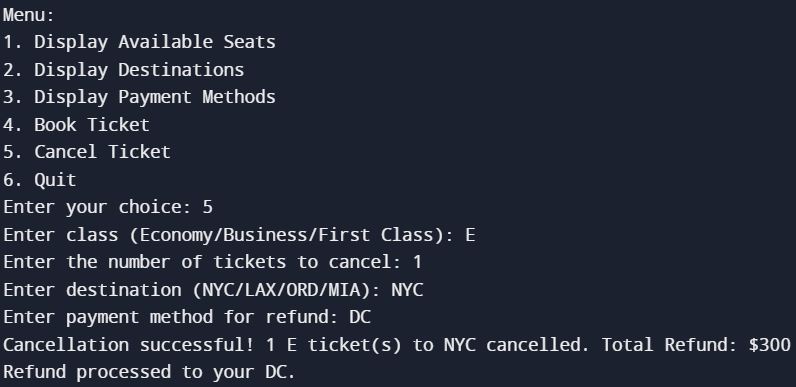
4. Conditional Statements:

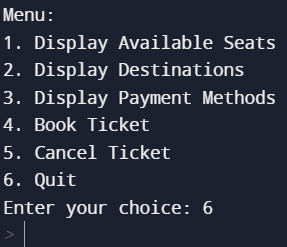
- `if-elif-else`: Used to check the user's input and execute the corresponding block of code based on their choice.

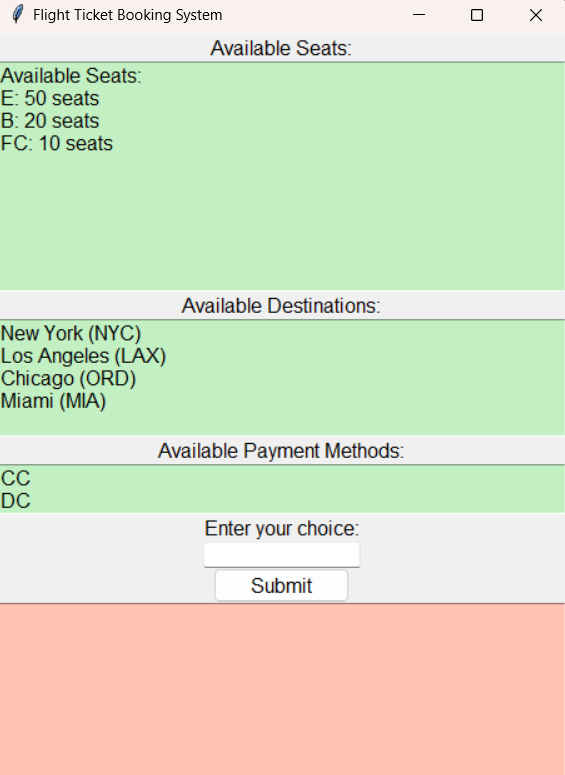
**7. Results/Screenshots**

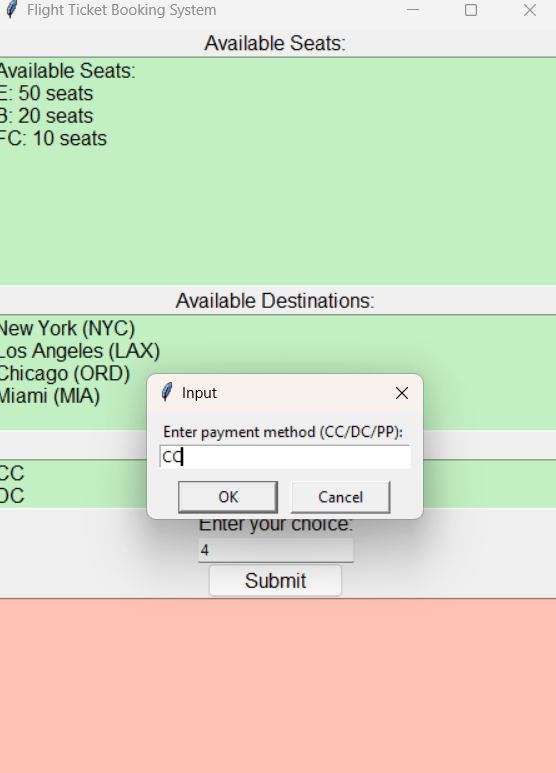
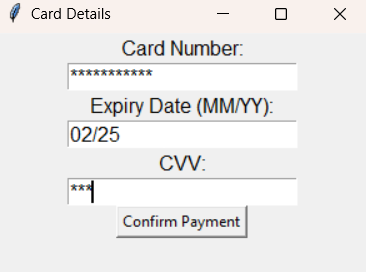
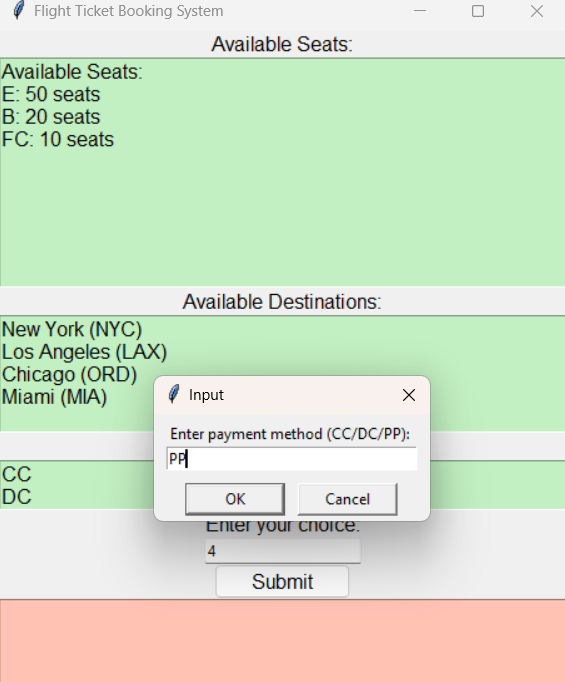
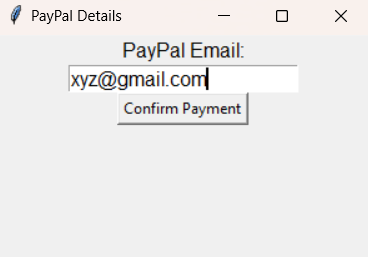




****

****

****

****

**8. Conclusion**

In conclusion, the developed Flight Ticket Booking System provides a comprehensive solution for managing the reservation process in an airline company. The program allows users to interactively explore available seats, view destinations, check accepted payment methods, book tickets, and cancel reservations. The implementation leverages a class-based structure and utilizes fundamental Python features such as dictionaries, loops, and conditional statements.

The system maintains the state of available seats, booked seats, destinations, and payment methods, ensuring that users receive accurate and timely information. The user interface is designed to be intuitive, guiding users through the various options available and responding appropriately to their inputs.

While the program currently operates without the need for external modules or libraries, it is designed to be extensible. Future enhancements could involve incorporating additional features, such as user authentication, date-based availability, or integration with external systems for a more robust flight booking experience.

In summary, the Flight Ticket Booking System addresses the specified requirements and provides a foundation for further development and refinement in response to evolving needs in the airline industry. The clean and modular design of the program facilitates ease of maintenance and future expansion, making it a practical and adaptable solution for managing flight reservations.

**9. References**

* **TKinter:** <https://realpython.com/python-gui-tkinter/>
* **Code2flow:** Used for flowchart making
* **Python programming 2E by Reema Thareja**