# AIRLINE RESERVATION SYSTEM

Aamena Elahi **410080**

Ayesha Shahid **431631**

Javeria Aaqib **421784**

Zoya **408726**

Sabeen **418804**

Maryam **431265**

# Table of Contents

# ABSTRACT:

The Airline Observation System is a software application designed using Object-Oriented Programming (OOP) principles to monitor and manage airline operations efficiently. This system focuses on tracking flight schedules, aircraft status, crew assignments, and passenger details. It ensures better coordination, timely updates, and streamlined data management across airline departments.

Each component—such as flights, aircraft, crew, and passengers—is represented as an individual class with specific attributes and behaviors. The system allows real-time observation of flights and provides alerts in case of delays, maintenance needs, or crew changes.

This project demonstrates how OOP techniques can be applied to solve real-world problems in the aviation industry through a structured and scalable software solution.

# INTRODUCTION:

The Airline Observation System is developed using Object-Oriented Programming (OOP) to simplify and automate the process of observing airline activities. This system is designed to monitor flights, manage aircraft availability, record crew assignments, and handle passenger information. It helps airline staff track current flight status, delays, and schedule changes, allowing for better decision-making and coordination.

Using OOP principles such as classes, objects, inheritance the system is structured to be modular, scalable, and easy to maintain.

# PROBLEM STATEMENT:

Airlines face numerous challenges in managing and monitoring daily operations such as flight schedules, aircraft availability, crew assignments, and passenger information. Manual tracking or poorly structured systems often lead to miscommunication, delays, data errors, and operational inefficiencies.

There is a need for an automated, organized, and user-friendly system that can observe and manage airline operations in real-time. The lack of such a system can result in flight delays, poor resource planning, and passenger dissatisfaction.

The Airline Observation System aims to solve this problem by providing a structured software solution developed using Object-Oriented Programming (OOP) principles. It will help track essential airline components efficiently, ensure timely updates, and support better decision-making in a dynamic environment.

# PROPOSED SYSTEM:

The proposed Airline Observation System is a software application designed using Object-Oriented Programming (OOP) concepts to improve the monitoring and management of airline operations. This system will automate the tracking of flight schedules, aircraft availability, crew assignments, and passenger information in a structured and efficient manner.

Each main component of the airline —such as Flight, Aircraft, Crew, and Passenger—will be represented as individual classes in the system. Through inheritance, common properties can be shared; using encapsulation, data will be protected and managed securely.

# System Analysis and Design

## 1. System Analysis

System analysis involves studying and understanding the current challenges in the airline booking process and determining the functional and non-functional requirements of the new system.

## Objectives of the New System:

- Automate flight booking and management.
- Provide real-time flight information.
- Maintain records of passengers and bookings.
- Apply OOP principles for maintainability and scalability.

## Users of the System:

# Administrator:

- Adds, updates, and deletes flight details.
- Views all passenger bookings.
- Manages pilot assignments.
- Oversees system data.

## Passenger/User:

- Register/logs in.
- Searches for available flights.
- Books and cancels tickets.
- Views booking history.

## Pilot:

- Assigned to specific flights.
- Views flight schedule.
- Updates status (e.g., ready, delayed, in-air).
- Can log work history or hours (optional feature).

# System Design

## Features and Functionality:

### User Authentication and Role Management

### Login / Logout System:

- Secure login for different user roles (Admin, Passenger, Pilot).
- Password validation and role-based access control.

### User Registration (Passenger only):

- New users can sign up and create an account.
- Required details: name, email, password, passport number, etc.

## Flight Management (Administrator)

### Add New Flights:

- Admin can add new flight entries including flight ID, source, destination, schedule, seat capacity.

### Update Flight Details:

- Modify timing, seat availability, or destination of existing flights.

### Cancel Flights:

- Remove a flight that is no longer operational.

### Assign Pilot to Flight:

- Assign a pilot object to a specific flight.
- Ensure one pilot per flight at a time.

## Flight Scheduling and Viewing (All Users)

### Search Flights:

- Passengers can search for available flights based on source, destination, and date.
- Displays available seats and timing.

### View Flight Schedule (Pilots and Admin):

- Pilot can view assigned flights.
- Admin can view all scheduled flights.

## Ticket Booking and Management (Passenger)

### Book Ticket:

- Passenger can select a flight and reserve a seat.
- System checks seat availability and confirms booking.

## Cancel Booking:

- Passenger can cancel their booking before flight departure

## 👮 Pilot Functionality

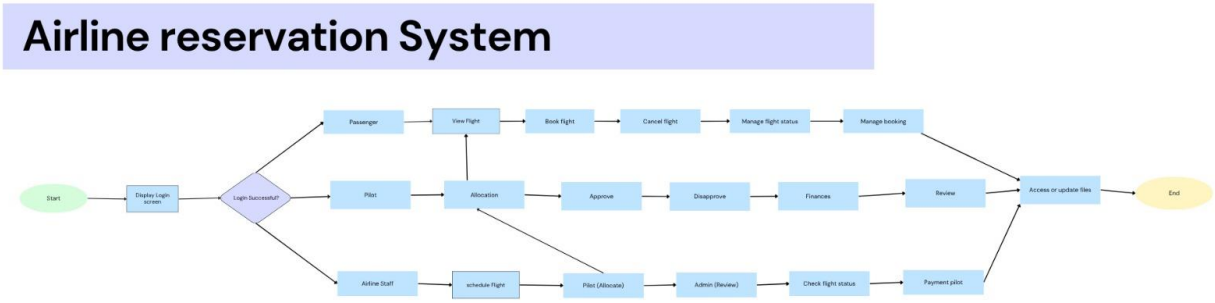
## View Assigned Flights:

- Pilot can log in and see upcoming flights they're assigned to.

## Update Flight Status:

- Pilots can update status (e.g., On-Time, Delayed, Departed).

# FLOW DIAGRAM




https://www.canva.com/design/DAGr50s9gIU/F7yU1wFvENokt-X4UbNpAA/edit?utm_content=DAGr50s9gIU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

## CODE SNIPPETS:

```python
# Flight class
class Flight:
    def __init__(self, number, origin, destination, seats, pilot):
        self.flight_number = number
        self.origin = origin
        self.destination = destination
        self.total_seats = seats
        self.available_seats = seats
        self.pilot = pilot

    def show_info(self):
        print(f"Flight: {self.flight_number} | {self.origin} → {self.destination} | Seats: {self.available_seats}/{self.total_seats}")
        print(f"Pilot: {self.pilot.name}")

    def book_seat(self):
        if self.available_seats > 0:
            self.available_seats -= 1
            return True
        return False


# Pilot class
class Pilot:
    def __init__(self, name, license_no):
        self.name = name
        self.license_no = license_no

    def save_to_file(self, filename="pilots.txt"):
        with open(filename, "a") as file:
            file.write(f"{self.name},{self.license_no}\n")
```

```python
    def cancel_my_flight(self, system, flight_number):
        system.cancel_flight_by_pilot(self.name, flight_number)


# Passenger class
class Passenger:
    def __init__(self, name):
        self.name = name

    def save_to_file(self, filename="passengers.txt"):
        with open(filename, "a") as file:
            file.write(f"{self.name}\n")

    def cancel_my_booking(self, system, flight_number):
        system.cancel_booking(self.name, flight_number)


# Booking class
class Booking:
    def __init__(self, passenger, flight):
        self.passenger = passenger
        self.flight = flight

    def save_to_file(self, filename="bookings.txt"):
        with open(filename, "a") as file:
            file.write(f"{self.passenger.name},{self.flight.flight_number}\n")

    def show(self):
        print(f"{self.passenger.name} booked flight {self.flight.flight_number}")
```

```python
# Admin class
class Admin:
    def __init__(self, name):
        self.name = name

    def add_flight(self, system, number, origin, destination, seats, pilot):
        flight = Flight(number, origin, destination, seats, pilot)
        system.flights.append(flight)
        print(f"Admin {self.name} added flight {number}")

    def view_cancellations(self, system):
        system.show_cancellations()

    def add_pilot(self, name, license_no, system=None):
        pilot = Pilot(name, license_no)
        pilot.save_to_file()
        print(f"Admin {self.name} added pilot: {name}, License: {license_no}")
        return pilot

    def add_passenger(self, name, system=None):
        passenger = Passenger(name)
        passenger.save_to_file()
        print(f"Admin {self.name} added passenger: {name}")
        return passenger

    def reassign_pilot(self, system, flight_number, new_pilot):
        for flight in system.flights:
            if flight.flight_number == flight_number:
                flight.pilot = new_pilot
                print(f"Pilot reassigned to {new_pilot.name} for flight {flight_number}")
                return
        print("Flight not found.")
```

```python
    def reassign_passenger(self, system, passenger_name, from_flight, to_flight):
        for booking in system.bookings:
            if booking.passenger.name == passenger_name and booking.flight.flight_number == from_flight:
                for f in system.flights:
                    if f.flight_number == to_flight:
                        if f.available_seats > 0:
                            f.book_seat()
                            booking.flight.available_seats += 1
                            system.bookings.remove(booking)
                            new_booking = Booking(booking.passenger, f)
                            system.bookings.append(new_booking)
                            new_booking.save_to_file()
                            system.update_bookings_file()
                            print(f"{passenger_name} reassigned from {from_flight} to {to_flight}")
                            return
                        else:
                            print("No seats available on new flight.")
                            return
# Airline System class
class AirlineSystem:
    def __init__(self):
        self.flights = []
        self.bookings = []

    def show_flights(self):
        print("\n--- Available Flights ---")
        for f in self.flights:
            f.show_info()

    def book_ticket(self, passenger, flight_num):
        for f in self.flights:
            if f.flight_number == flight_num:
```

```python
def book_ticket(self, passenger, flight_num):
    for f in self.flights:
        if f.flight_number == flight_num:
            if f.book_seat():
                b = Booking(passenger, f)
                self.bookings.append(b)
                b.save_to_file("bookings.txt")
                print("Booking successful and saved.")
                return
            else:
                print("No seats available.")
                return
    print("Flight not found.")

def cancel_booking(self, passenger_name, flight_number):
    for b in self.bookings:
        if b.passenger.name == passenger_name and b.flight.flight_number == flight_number:
            b.flight.available_seats += 1
            self.bookings.remove(b)
            self.update_bookings_file()
            self.log_cancellation(f"Passenger {passenger_name} cancelled booking on flight {flight_number}.")
            print(f"Booking cancelled for {passenger_name} on flight {flight_number}.")
            return
    print("Booking not found.")

def cancel_flight_by_pilot(self, pilot_name, flight_number):
    for f in self.flights:
        if f.flight_number == flight_number and f.pilot.name == pilot_name:
            self.bookings = [b for b in self.bookings if b.flight.flight_number != flight_number]
            self.flights.remove(f)
            self.update_bookings_file()
            self.log_cancellation(f"Pilot {pilot_name} cancelled flight {flight_number}.")
```


```python
def cancel_flight_by_pilot(self, pilot_name, flight_number):
    for f in self.flights:
        if f.flight_number == flight_number and f.pilot.name == pilot_name:
            self.bookings = [b for b in self.bookings if b.flight.flight_number != flight_number]
            self.flights.remove(f)
            self.update_bookings_file()
            self.log_cancellation(f"Pilot {pilot_name} cancelled flight {flight_number}.")
```


```python
            print(f"Flight {flight_number} cancelled by Pilot {pilot_name}.")
            return
    print("Flight not found or pilot not authorized.")

def update_bookings_file(self):
    with open("bookings.txt", "w") as file:
        for b in self.bookings:
            file.write(f"{b.passenger.name},{b.flight.flight_number}\n")

def log_cancellation(self, message):
    with open("cancellations.txt", "a") as file:
        file.write(f"{message}\n")

def show_cancellations(self):
    try:
        with open("cancellations.txt", "r") as file:
            print("\n--- Cancellation Log ---")
            for line in file:
                print(line.strip())
    except FileNotFoundError:
        print("No cancellations recorded.")
```

```python
def show_bookings_from_file(self):
    try:
        with open("bookings.txt", "r") as file:
            lines = file.readlines()
            print("\n--- All Bookings from File ---")
            for line in lines:
                name, flight = line.strip().split(",")
                print(f"{name} booked flight {flight}")
    except FileNotFoundError:
        print("No bookings found in file.")

def show_saved_pilots(self):
    try:
        with open("pilots.txt", "r") as file:
            print("\n--- All Pilots ---")
            for line in file:
                name, license_no = line.strip().split(",")
                print(f"Pilot: {name}, License: {license_no}")
    except FileNotFoundError:
        print("No pilots found in file.")

def show_saved_passengers(self):
    try:
        with open("passengers.txt", "r") as file:
            print("\n--- All Passengers ---")
            for line in file:
                print(f"Passenger: {line.strip()}")
    except FileNotFoundError:
        print("No passengers found in file.")
```

```python
# Main Program
if __name__ == "__main__":
    system = AirlineSystem()

    # Predefined flight and pilot setup for testing
    predefined_pilots = [
        Pilot("Capt. Ayesha", "AY001"),
        Pilot("Capt. Imran", "IM002"),
        Pilot("Capt. Rizwan", "RZ003"),
        Pilot("Capt. Sana", "SN004"),
        Pilot("Capt. Kamal", "KM005"),
    ]

    predefined_flights = [
        Flight("PA101", "Lahore", "Karachi", 3, predefined_pilots[0]),
        Flight("PA102", "Islamabad", "Lahore", 2, predefined_pilots[1]),
        Flight("PA103", "Karachi", "Peshawar", 4, predefined_pilots[2]),
        Flight("PA104", "Quetta", "Islamabad", 5, predefined_pilots[3]),
        Flight("PA105", "Multan", "Faisalabad", 3, predefined_pilots[4]),
    ]

    for pilot in predefined_pilots:
        pilot.save_to_file()
    system.flights.extend(predefined_flights)

    print("🎉 Welcome to Airline Reservation System 🎉\n")

    while True:
        print("\nSelect your role:\n1. Passenger\n2. Admin\n3. Pilot\n4. Exit")
        option = input("Enter your choice: ").strip()
```

```python
if option == "1":
    # Passenger menu without loop
    print("\n📋 Passenger Menu:\n1. View Flights\n2. Book Flight\n3. Cancel Booking\n4. Exit")
    p_choice = input("Enter your choice: ").strip()

    if p_choice == "1":
        system.show_flights()

    elif p_choice == "2":
        system.show_flights()
        name = input("Enter your name: ").strip()
        flight_id = input("Enter flight number to book: ").strip()
        passenger = Passenger(name)
        passenger.save_to_file()
        system.book_ticket(passenger, flight_id)

    elif p_choice == "3":
        name = input("Enter your name: ").strip()
        flight_id = input("Enter flight number to cancel: ").strip()
        passenger = Passenger(name)
        passenger.cancel_my_booking(system, flight_id)

    elif p_choice == "4":
        print("Exiting to main menu.")

    else:
        print("Invalid choice. Returning to main menu.")

elif option == "2":
    admin = Admin("Mr. Arshad")
    while True:
        print("\n🛠 Admin Menu:\n1. Add Flight\n2. View Flights\n3. Exit")
```

```python
        print("\n🛠 Admin Menu:\n1. Add Flight\n2. View Flights\n3. Exit")
        a_choice = input("Enter your choice: ").strip()

        if a_choice == "1":
            number = input("Flight number: ")
            origin = input("Origin: ")
            destination = input("Destination: ")
            seats = int(input("Seats: "))
            pname = input("Pilot name: ")
            license = input("Pilot license: ")
            pilot = admin.add_pilot(pname, license)
            admin.add_flight(system, number, origin, destination, seats, pilot)

        elif a_choice == "2":
            system.show_flights()

        elif a_choice == "3":
            print("Exiting Admin menu.")
            break

        else:
            print("Invalid choice. Try again.")

elif option == "3":
    name = input("Enter your pilot name: ")
    license_no = input("Enter your license no: ")
    pilot = Pilot(name, license_no)

    while True:
        print("\n🛫 Pilot Menu:\n1. Cancel Flight\n2. Exit")
        p_choice = input("Enter your choice: ").strip()
```

```
            if p_choice == "1":
                flight_id = input("Enter flight number to cancel: ")
                pilot.cancel_my_flight(system, flight_id)

            elif p_choice == "2":
                print("Exiting Pilot menu.")
                break

            else:
                print("Invalid choice. Try again.")

    elif option == "4":
        print("👏 Thank you for using Airline Reservation System.")
        break

    else:
        print("Invalid choice. Please select from 1 to 4.")
```

# CONCLUSION:

Thus, this Airline System helps create a system that manages flights, helps passenger book flight with convenience and saves time as they can schedule flights by sitting at home.

This project successfully demonstrates how real-world airline functions—like flight tracking, passenger information, and scheduling—can be modeled into software components. The structure not only improves data handling but also simplifies maintenance and future upgrades.