# 

**Project Report**

**Project Title:** Airline Reservation System

# Submitted by: Hazal Group

|  |  |
| --- | --- |
| **M. Saqib Imran** | **SU92-BSSEM-F24-109** |
| **M. Umar** | **SU92-BSSEM-F24-088** |
| **M. Ahmad Naveed** | **SU92-BSSEM-F24-149** |

# Submitted To: Ma’am Ayesha Karim

# Course Name: Object Oriented Programming

# Course Code: CS-1206

# Date of Submission: 30/06/2025

**ACKNOWLEDGEMENTS**

I would like to express my deepest gratitude to **Professor Ayesha Karim** for her invaluable guidance, insightful feedback, and constant encouragement throughout the development of our final year project, ***"Hazel Blue Airline – Airline Reservation System."*** Her mentorship played a key role in shaping the direction and success of this project, and we are truly grateful for her unwavering support.

As the **Project Lead**, I had the privilege of guiding an exceptional team comprising **Saqib, Umer, Unbreen, and Ukasha**. Their dedication, cooperation, and relentless efforts were crucial in bringing this project to life. From early brainstorming sessions to final deployment, we worked collaboratively to develop a complete and functional airline reservation system. Each member contributed unique ideas and skills that enriched the project and made the journey both productive and memorable.

I am especially proud of the teamwork we demonstrated in overcoming technical challenges, managing time effectively, and ensuring that our work met the expectations we set. Leading this team has been a rewarding experience that allowed me to enhance not only my technical expertise but also my leadership and coordination skills.

A heartfelt thank you also goes to our **families and friends** for their continuous encouragement, emotional support, and patience throughout this demanding academic phase. Their belief in us kept us motivated, especially during long coding sessions, testing cycles, and unexpected bugs.

I would also like to acknowledge the vital role played by **online learning resources, technical communities, documentation platforms, and open-source contributors**. These resources helped us troubleshoot complex problems and refine the quality of our code, making the development process smoother and more efficient.

Furthermore, I express appreciation to our **institute’s faculty and technical staff** for maintaining an environment conducive to learning and innovation. The availability of computer labs, tools, and development support greatly contributed to the success of this project.

Completing *Hazel Blue Airline* has been more than just a technical achievement. It has been a journey of **learning, growth, collaboration, and personal development**. I am proud of the commitment and perseverance our team showed and honored to have led such a driven and passionate group.

This project has not only strengthened my foundation in C++ and system design but has also prepared me for future challenges in both academic and professional settings. I look forward to applying the experiences and lessons gained from this project in my career as a software developer.

**ABSTRACT**

The **Hazel Blue Airline Reservation System** is a comprehensive, console-based software application developed in **C++**, designed to streamline and automate the process of booking and managing airline tickets. This system addresses the limitations of traditional manual booking methods by offering a structured digital platform for both **passengers** and **administrators** to interact with airline services efficiently.

The system features two main user roles: **Admin** and **Passenger**. Administrators are equipped with functionalities such as adding flights, managing schedules, viewing all bookings, and monitoring ticket information. Passengers, on the other hand, can register an account, log in securely, browse available flights, book tickets, view or cancel bookings, and print ticket details.

Key features of the system include:

* Flight management (add, view, and search flights)
* Unique ticket generation with booking history
* Seat availability checks and auto-updates
* Secure login and role-based access control
* File-based data persistence for flights, passengers, and tickets

The project was developed with the goal of improving operational efficiency, reducing manual errors, and enhancing the passenger experience by offering a simple yet powerful interface. The use of **object-oriented programming (OOP)** principles such as inheritance and encapsulation enhance code organization, maintainability, and scalability.

This project demonstrates not only technical proficiency in C++ but also an understanding of real-world problem-solving, software lifecycle management, and user-centric design. It serves as a practical model for small to medium-scale airline systems and forms a solid foundation for further enhancement into a full-fledged GUI or web-based application.

**Chapter 1**

**PROJECT BACKGROUND**

**1.1 Introduction**

The airline industry is one of the most dynamic and complex sectors, requiring efficient and user-friendly systems to manage ticketing, flight information, and customer interaction. Traditional manual processes often result in booking errors, scheduling issues, and poor customer experience. In response to these challenges, this project presents the **Hazel Blue Airline Reservation System**, a console-based application developed in **C++** aimed at streamlining the process of airline ticket reservation and management.

This system enables both **passengers** and **administrators** to perform core tasks such as flight booking, ticket cancellation, flight management, and profile handling through an interactive command-line interface. It demonstrates real-world application of **object-oriented programming (OOP)** concepts like encapsulation, inheritance, and polymorphism.

**1.2 Problem Statement**

Manual airline reservation processes are inefficient, prone to human error, and lack scalability. Existing systems often fail to offer a smooth and secure experience for users due to poor design, limited functionality, or high complexity. There is a need for a lightweight, easy-to-use application that:

* Allows users to book and manage flights efficiently.
* Helps admins manage flights, view tickets, and monitor data.
* Offers secure and organized handling of user data and bookings.

**1.3 Project Scope**

The project focuses on building a **C++-based desktop airline reservation system** with the following scope:

* Registration and login for passengers.
* Admin login with elevated privileges.
* Adding, editing, and viewing flight details.
* Booking and cancellation of tickets.
* Generating ticket IDs and showing complete booking details.
* Data storage through file handling (no external database).

This system is best suited for small to medium-sized airlines or academic demonstration purposes.

**1.4 Project Objectives**

The main objectives of this project are:

* To design and implement a structured and user-friendly airline reservation system.
* To ensure secure access and role-based functionality for passengers and administrators.
* To apply object-oriented design principles for maintainability and extensibility.
* To handle persistent data storage using file I/O.
* To improve user experience through a clean, guided console UI

**1.5 Project Schedule**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Task** | **Duration** |
| Phase 1 | **Requirements Gathering & Analysis** | 1 week |
| Phase 2 | **System Design & Diagrams** | 2 weeks |
| Phase 3 | **Backend Implementation (C++ Logic)** | 3 weeks |
| Phase 4 | |  | | --- | |  |  |  | | --- | | **File Handling, Testing & Debugging** | | 2 weeks |
| Phase 5 | **Documentation & Finalization** | 1 week |

**1.6 Project Tools and Technologies**

* **Programming Language:** C++
* **IDE:** Dev C++ / Visual Studio
* **Paradigm:** Object-Oriented Programming (OOP)
* **Data Storage:** Text files (File Handling)
* **Diagrams:** Draw.io, Lucidchart, or Manual Drafting Tools, plant uml
* **Platform:** Windows Console Application

**Chapter 2**

**Related Work**

**2.1 Overview**

The advancement of digital systems in the travel and transportation industry has led to the development of various **airline reservation systems**. These systems are designed to automate the process of booking, managing, and cancelling airline tickets, and they have become a critical component of modern airline operations. In this chapter, we review related systems and technologies to understand existing solutions, identify gaps, and draw comparisons with our project — *Hazel Blue Airline Reservation System*.

**2.2 Existing Airline Reservation Systems**

Numerous reservation systems exist in both commercial and academic contexts. Prominent examples include:

* **Amadeus**, **Sabre**, and **Galileo** — widely used in commercial aviation for managing airline inventory, passenger bookings, and global distribution.
* **Open-source systems** such as *Flight Reservation System in Java* or *Python-based Booking Systems* are often built for academic use or prototypes.
* University-level projects developed in **Java, C#, or Python** demonstrate basic features like flight booking, cancellation, and admin control, but often lack modularity or scalability.

While these systems serve various needs, many of them are complex, web-based, or require third-party APIs and databases. This makes them less ideal for environments where simplicity, offline operation, or C++-based development is desired.

**2.3 Comparison with Our System**

The **Hazel Blue Airline Reservation System** was developed with the goal of offering a compact, offline, console-based solution for flight reservation using **C++ and file handling** — without requiring external databases. Key differences include:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Commercial Systems** | **Hazel Blue Airline (Our System)** |
| Platform | Web/Desktop/Cloud | Console (Desktop) |
| Language | Java, .NET, PHP | C++ |
| Data Storage | Cloud Databases (SQL/Oracle) | Local File System (Text files) |
| User Roles | Advanced (CRM, Agents) | Admin & Passenger |
| Customization | Limited / Subscription-based | Fully customizable (Open source) |
| Internet Dependency | High | None (Offline-capable) |
| Learning Focus | Business functionality | Academic OOP Practice + Functional System |

**2.4 Key Observations**

* Most commercial systems are not suitable for academic learning due to complexity.
* Existing C++ airline systems often lack polish in UI, code design, or documentation.
* Very few systems balance **object-oriented design**, **file handling**, and a complete feature set like booking, cancellation, and admin management — which our system provides.

**2.5 Conclusion**

The *Hazel Blue Airline Reservation System* addresses the gap between overly complex commercial solutions and underdeveloped academic models. By focusing on a **clean console interface**, **OOP principles**, and **file-based persistence**, it serves as a learning model as well as a practical airline reservation prototype for small-scale environments.

**Chapter 3**

**System Analysis and Design**

**3.1 Requirement Analysis and Specification**

**3.1.1 Functional Requirements**

The functional requirements specify the expected behaviour’s and features of the system. These are written using the standard **"The system shall..."** format:

1. **The system shall** allow new users (passengers) to register with a unique username, password, full name, and email address.
2. **The system shall** allow registered passengers to log in using their credentials.
3. **The system shall** allow the administrator to log in using predefined credentials (admin / admin123).
4. **The system shall** allow the administrator to add new flights with relevant details including flight name, source, destination, date, time, total seats, and ticket price.
5. **The system shall** allow the administrator to view all available flights.
6. **The system shall** allow passengers to view only those flights which have available seats.
7. **The system shall** allow passengers to book tickets for selected flights.
8. **The system shall** generate a unique ticket ID and display detailed ticket information upon successful booking.
9. **The system shall** allow passengers to view their personal booking history.
10. **The system shall** allow passengers to cancel previously booked flights.
11. **The system shall** update flight seat availability in real-time upon booking or cancellation.
12. **The system shall** allow the administrator to view a list of all booked tickets.
13. **The system shall** persist all data related to flights, users, and tickets using text files (file handling).

**3.1.2 Non-Functional Requirements**

* **Usability:**  
  The system shall offer a simple and intuitive text-based interface for ease of navigation.
* **Maintainability:**  
  The code shall be written using object-oriented principles such as inheritance and encapsulation, ensuring modularity and easier future modifications.
* **Reliability:**  
  The system shall accurately maintain data integrity in terms of login credentials, booking operations, and ticket cancellations.
* **Portability:**  
  The system shall be platform-independent and runnable on any machine with a C++ compiler and console access.
* **Security:**  
  The system shall implement basic authentication to prevent unauthorized access to admin or passenger features.
* **Offline Operation:**  
  The system shall operate completely offline using local file storage instead of a database.

**3.2 Class Design**

This section describes the object-oriented structure used in the system:

**3.2.1 User (Abstract Base Class)**

* **Purpose:** Represents a general user (Admin or Passenger).
* **Attributes:**

**username**: string — Unique login name(public)

**password:** string — Account password(public)

**name**: string — Full name(public)

**email**: string — Email address(public)

* **Methods:**
  + Constructor: User(uname, pwd, n, e)
  + virtual viewProfile() const
* **Responsibilities:**
  + Serves as the base class for Admin and Passenger
  + Provides a template for profile viewing

**3.2.2 Passenger (Derived from User)**

* **Purpose:** Passenger who can book and cancel flights.
* **Attributes:**
  + int bookings[MAX\_BOOKINGS]
  + int bookingCount
* **Methods:**
  + Constructor: Passenger(uname, pwd, n, e)
  + viewProfile() const override
* **Responsibilities:**
  + Manages personal bookings
  + Inherits basic user details

**3.2.3 Admin (Derived from User)**

* **Purpose:** Admin who manages the system.
* **Attributes:** Inherits all from User.
* **Methods:**
  + Admin() — Initializes default admin credentials
  + viewProfile() const override
* **Responsibilities:**
  + Adds flights and views tickets
  + Performs system management

**3.2.4 Flight**

* **Purpose:** Represents a flight record.
* **Attributes:**
  + id, name, source, destination, date, time
  + totalSeats, availableSeats, price
* **Methods:**
  + Constructor: Flight(...)
* **Responsibilities:**
  + Stores and manages flight information

**3.2.5 Ticket**

* **Purpose:** Represents a flight ticket.
* **Attributes:**
  + ticketId, flightId, passengerUsername, bookingDate, totalPrice, isCancelled
* **Methods:**
  + Constructor: Ticket(...)
* **Responsibilities:**
  + Tracks bookings
  + Enables logical deletion (cancellation)

**3.2.6 AirlineSystem**

* **Purpose:** Manages all core functionality and data.
* **Attributes:**

Passenger passengers[MAX\_PASSENGERS]

int passengerCount

Flight flights[MAX\_FLIGHTS]

int flightCount

Ticket tickets[MAX\_TICKETS]

int ticketCount

Passenger\* currentPassenger

bool adminLoggedIn

int flightIdCounter

**Public Methods:**

void mainMenu()

void adminLogin()

void adminMenu()

void registerPassenger()

void passengerLogin()

void passengerMenu()

void bookTicket()

void viewMyBookings()

void cancelBooking()

void addFlightMenu()

void viewAllFlights()

void viewAllTickets()

void viewTicketDetails()

void aboutUs()

* **Private Helper Methods:**

Passenger\* findPassenger(string username)

Flight\* findFlight(int id)

bool hasBookedFlight(int id)

bool removeBooking(int flightId)

string generateTicketId()

string getCurrentDate()

void displayAvailableFlights()

void displayTicketDetails(const Ticket&)

void saveTicketToFile(const Ticket&)

void saveFlightsToFile()

void loadFlightsFromFile()

void savePassengersToFile()

void loadPassengersFromFile()

void saveTicketsToFile()

void loadTicketsFromFile()

* **Responsibilities:**
  + Central control of operations
  + Handles session logic, booking logic, and persistence

A diagram of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

A computer screen shot of a diagram

Description automatically generated

A computer screen shot of a diagram

Description automatically generated

**Chapter 4**

**Project Implementation**

**4.1 Overview**

This chapter describes the implementation details of the *Hazel Blue Airline Reservation System*. The system was developed using **C++** with a **console-based user interface**, leveraging **object-oriented programming (OOP)** principles and **file handling** for data persistence. The project consists of two primary roles: **Admin** and **Passenger**, each with specific access to system functionalities.

The implementation process followed a modular approach, with each component such as user management, flight operations, and ticket handling developed independently and integrated progressively.

**4.2 Development Environment**

|  |  |
| --- | --- |
| **Component** | **Details** |
| Programming Language | C++ |
| Compiler/IDE | Dev C++ / Visual Studio Code |
| Platform | Windows (Console Application) |
| Data Storage | Local Text Files (I/O operations) |
| Programming Paradigm | Object-Oriented Programming (OOP) |

**4.3 System Modules**

**4.3.1 User Authentication Module**

* Allows passenger registration and login with credential validation.
* Provides secure admin login with predefined credentials.
* Verifies user existence from passengers.txt.

**4.3.2 Flight Management Module (Admin)**

* Admin can add flights with required data fields.
* Flight records are stored in flights.txt.
* Admin can view all added flights in tabular format.

**4.3.3 Booking and Ticketing Module (Passenger)**

* Allows passengers to book available flights.
* Generates unique ticket IDs using a random string.
* Ticket information is saved in tickets.txt.

**4.3.4 Cancellation and History Module**

* Passengers can cancel flights by entering the flight ID.
* Seat availability is automatically restored on cancellation.
* Booking history is maintained for each user.

**4.3.5 File Handling Module**

* All user data (passenger details), flight data, and ticket data are stored using standard file I/O.
* Files:
  + passengers.txt — stores registered users and their bookings
  + flights.txt — stores flight schedules and seat availability
  + tickets.txt — stores booked tickets and cancellation status

**4.4 Key Features Implemented**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Role** | **Status** |
| Register / Login | Passenger | ✅ Implemented |
| Admin Login | Admin | ✅ Implemented |
| Add / View Flights | Admin | ✅ Implemented |
| Book Tickets | Passenger | ✅ Implemented |
| View / Cancel Bookings | Passenger | ✅ Implemented |
| View All Tickets | Admin | ✅ Implemented |
| File Storage for Data Persistence | System-wide | ✅ Implemented |
| Dynamic Ticket ID Generation | System-wide | ✅ Implemented |

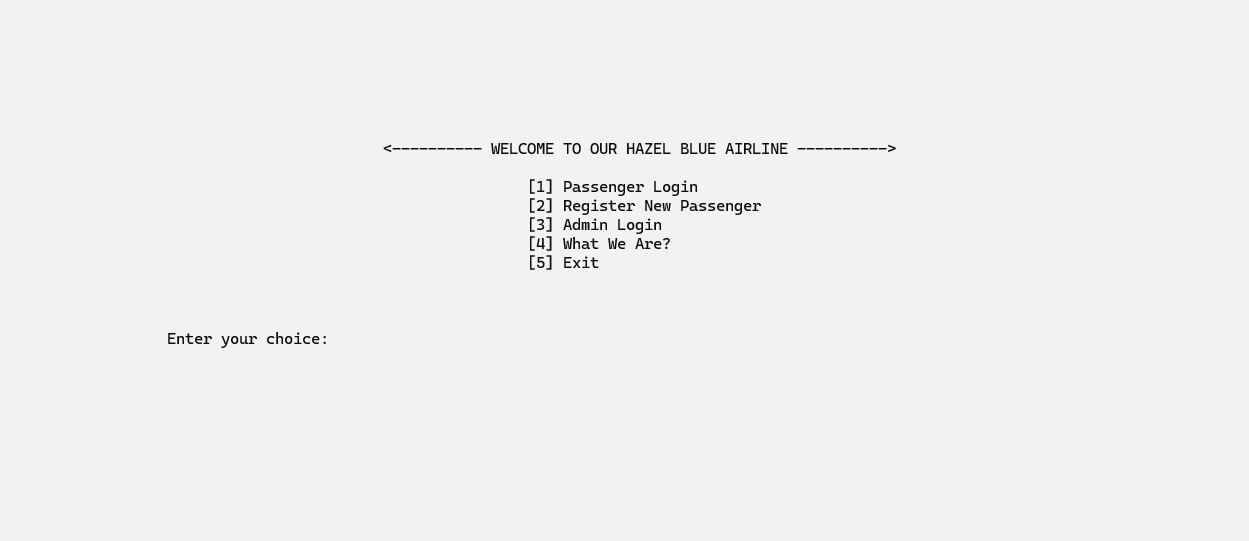
**4.5 Challenges Faced**

* Ensuring proper **data integrity** during booking and cancellation through file synchronization.
* Preventing **duplicate usernames** and invalid flight selections.
* Designing a flexible structure to accommodate both **admin and passenger flows** in the same system loop.

**4.6 Screenshots and Demonstrations**

screenshots of:

* Main Menu

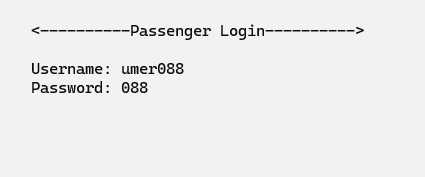


**4.7 Passenger Login and Menu Flow**

When the user selects **option 1** from the main menu (Passenger Login), the system performs the following steps:

**Step 1: Prompt for Login**

* The system asks the user to enter their **username** and **password**.
* These credentials are checked against the data stored in passengers.txt.



* If the credentials are correct, the system logs in the passenger and displays the **Passenger Menu**.
* If the credentials are invalid, the system displays an error message.

#### ****Step 2: Passenger Menu Display****

Once successfully logged in, the following menu is shown to the passenger:

A screenshot of a computer screen

Description automatically generated

Each option in the menu leads to the corresponding function:

* **[1] View Profile:** Shows the passenger's name, email, and username.

A screenshot of a computer

Description automatically generated

* **[2] View Available Flights:** Lists all flights with available seats.

A close-up of a computer screen

Description automatically generated

* **[3] Book Ticket:** Prompts for a flight ID and books a ticket if seats are available.

A screenshot of a computer

Description automatically generated

A screen shot of a ticket

Description automatically generated

* **[4] View My Bookings:** Lists all flights booked by the passenger.

A close-up of a message

Description automatically generated

* **[5] View Ticket Details:** Shows full ticket information for each booking.

A screen shot of a computer

Description automatically generated

A screen shot of a ticket

Description automatically generated

* **[6] Cancel Booking:** Allows the passenger to cancel a booking by entering the flight ID.

A screenshot of a computer screen

Description automatically generated

* **[7] Logout:** Ends the session and returns to the main menu.

A black text on a white background

Description automatically generated

**4.8 Passenger Registration Flow**

When the user selects **option 2** from the main menu, the system enters the **registration process**, allowing a new passenger to create an account.

**Step 1: Input Required Information**

The system prompts the user to enter the following details:

* **Username** – must be unique (checked against existing records)
* **Password** – secure login credential
* **Full Name** – passenger’s real name
* **Email Address** – for future contact or ticket details

**Step 2: Validation**

* The system checks if the **username already exists** in passengers.txt.
  + If it does, a message is shown: "Username already exists."
  + If not, the registration proceeds.

**Step 3: Saving Data**

* Upon successful validation, the new passenger data is:
  + Stored in the **passengers array**
  + Written to the **passengers.txt** file in this format:
* A confirmation message is displayed:

Registration successful.

**Step 4: Return to Main Menu**

* After registration, the system pauses briefly and returns the user to the main menu where they can now login using the newly created account.

A screen shot of a computer

Description automatically generated

**4.9 Admin Login and Menu Flow**

When the user selects **option 3** from the main menu, the system initiates the **Admin Login process**. The admin has elevated privileges to manage flights and view system-wide bookings.

**Step 1: Prompt for Admin Credentials**

The credentials are matched against the **predefined values:**

* Username: admin
* Password: admin123



**Step 2: Validation**

* If the entered credentials match:
  + The admin is logged in.
  + The **Admin Menu** is displayed.
* If credentials are incorrect:
  + An error message appears:  
    "Invalid admin credentials."
  + Control returns to the main menu.

**Step 3: Admin Menu Display**

Once successfully logged in, the admin is presented with the following options:

A screenshot of a computer program

Description automatically generated

**[1] View Profile**

* Displays the admin's profile, including:

A close-up of a computer screen

Description automatically generated

**[2] Add Flight**

* Prompts the admin to input complete flight details:



* Saves the new flight record into flights.txt.
* Each flight gets a unique auto-incremented ID.

**[3] View Flights**

* Displays a formatted list of **all flights**, including:

A screenshot of a computer screen

Description automatically generated

**[4] View All Tickets**

* Shows a detailed list of **all active (non-cancelled)** tickets booked by passengers.

A screen shot of a computer screen

Description automatically generated A screen shot of a computer screen

Description automatically generated A screen shot of a computer

Description automatically generated

**[5] Logout**

* Ends the admin session.
* Returns control to the main menu.

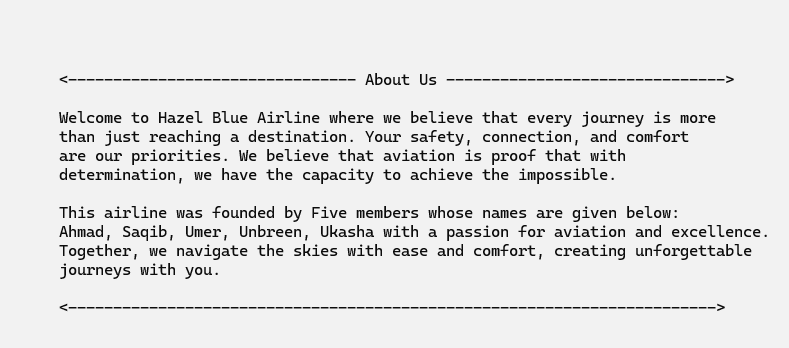
A close-up of a white background

Description automatically generated

**4.10 About Us Section (Option 4 in Main Menu)**

The **"About Us"** section provides background information about the system and its creators. It reflects the mission and vision behind the *Hazel Blue Airline Reservation System* and adds a personal touch to the user experience.

When the user selects **option 4** from the **main menu**, the system displays a formatted message that highlights the purpose of the system and the development team behind it.



**4.11 Exit Option (Option 5 in Main Menu)**

The **Exit option** allows the user to terminate the program gracefully. When the user selects **option 5** from the **main menu**, the system closes all active sessions, ends the program loop, and displays a farewell message.

**Functionality:**

* Ends the main menu loop (while(choice != 5)).
* Displays a **thank you message** to the user.
* Properly closes the application without any forced shutdowns or errors.

A black text on a white background

Description automatically generated

**Chapter 5**

**Key Concepts and Conclusion**

**5.1 Key Programming Concepts Used**

The development of the *Hazel Blue Airline Reservation System* relied heavily on core C++ concepts, particularly those related to object-oriented programming (OOP). Each concept was carefully chosen and implemented to address specific challenges in system design, data management, and user interaction.

Below is an in-depth explanation of the key concepts used, along with how and why they were applied in the system:

**1. Object-Oriented Programming (OOP)**

**Why Used:**  
To model real-world entities like users, flights, and tickets using objects that encapsulate both data and behavior.

**How Used:**

* Each entity (User, Passenger, Admin, Flight, Ticket, AirlineSystem) is modeled as a class.
* Methods are encapsulated within relevant classes (e.g., bookTicket() in AirlineSystem).
* The project follows modular and object-centric architecture.

**Benefit:**  
Improves code structure, reusability, clarity, and future scalability.

**2. Inheritance**

**Why Used:**  
To avoid duplication and reuse shared logic.

**How Used:**

* Passenger and Admin both inherit from the User base class.
* They reuse the username, password, name, and email attributes.
* Each child class overrides the viewProfile() method.

**Benefit:**  
Minimizes code duplication and increases code manageability.

**3. Polymorphism (Runtime / Virtual Functions)**

**Why Used:**  
To allow different user types to behave differently using the same interface.

**How Used:**

* viewProfile() is declared virtual in the User class and overridden in Passenger and Admin.
* Enables calling the correct version depending on the object type at runtime.

**Benefit:**  
Adds flexibility and supports extensibility of user roles in the system.

**4. Encapsulation**

**Why Used:**  
To protect internal state and expose only necessary functionality.

**How Used:**

* Class data like password, flight details, and ticket info are handled within the class scope.
* Sensitive fields are never directly accessed outside their class.

**Benefit:**  
Improves security and helps enforce class-level data control.

**5. Abstraction**

**Why Used:**  
To hide implementation details and expose only necessary behavior.

**How Used:**

* The user interacts only through methods like bookTicket(), cancelBooking(), etc., without needing to know internal logic.
* For example, generateTicketId() encapsulates ticket ID logic from the rest of the system.

**Benefit:**  
Simplifies the interface for both users and developers while maintaining complexity internally.

**6. Constructors**

**Why Used:**  
To initialize objects cleanly at the time of creation.

**How Used:**

* User, Passenger, Admin, Flight, and Ticket classes all use constructors to set initial values.
* The Admin class uses a default constructor with predefined credentials.

**Benefit:**  
Ensures objects are created in a valid, consistent state.

**7. Composition**

**Why Used:**  
To model "has-a" relationships between classes.

**How Used:**

* AirlineSystem contains arrays of Passenger, Flight, and Ticket objects.
* Demonstrates how one class is composed of several others to manage the entire system.

**Benefit:**  
Enables structured and logical organization of complex data relationships.

**8. Static Arrays**

**Why Used:**  
To store multiple records in memory in a fast and efficient manner.

**How Used:**

* Arrays like passengers[100], flights[100], tickets[200] are used.
* Counters like passengerCount and ticketCount are maintained manually.

**Benefit:**  
Improves lookup speed and simplifies logic for beginner-level projects.

**9. File Handling (Persistence)**

**Why Used:**  
To permanently store user, flight, and ticket data across sessions.

**How Used:**

* Text files like passengers.txt, flights.txt, and tickets.txt are read/written using fstream, ifstream, and ofstream.

**Benefit:**  
Allows data to be saved and restored without databases.

**10. Randomization**

**Why Used:**  
To generate unique ticket IDs.

**How Used:**

* The generateTicketId() method uses rand() and a character set to create a 6-character alphanumeric code.

**Benefit:**  
Ensures each booking is uniquely traceable.

**11. Menu-Driven Navigation**

**Why Used:**  
To create a clean, guided user experience for Admin and Passenger roles.

**How Used:**

* mainMenu(), adminMenu(), and passengerMenu() provide a structured flow using do-while and switch-case.

**Benefit:**  
User-friendly interaction in a console-based interface.

**12. Defensive Programming**

**Why Used:**  
To prevent invalid or dangerous user actions.

**How Used:**

* Checks like "if username already exists", "if seat is not available", "if ticket is already cancelled", etc.

**Benefit:**  
Improves system reliability and user trust.

**13. Control Structures**

**Why Used:**  
To control logic based on user input.

**How Used:**

* Uses if, else, while, do-while, and switch extensively.

**Benefit:**  
Handles program decisions and branching effectively.

**14. Code Modularity**

**Why Used:**  
To separate responsibilities into small, manageable parts.

**How Used:**

* Methods like addFlightMenu(), cancelBooking(), viewMyBookings() each handle one responsibility.

**Benefit:**  
Easy to test, debug, and maintain.

**5.2 Testing and Validation**

Effective testing is crucial to ensure the reliability, accuracy, and performance of any software system. The *Hazel Blue Airline Reservation System* was thoroughly tested using **manual test cases** and simulated user interactions to validate the behavior of all features across different roles (Admin and Passenger).

**1. Functional Testing**

Each function was tested independently to confirm it performs the expected action.

|  |  |  |
| --- | --- | --- |
| **Feature Tested** | **Test Case** | **Result** |
| Passenger Registration | Register with unique vs. duplicate username | ✅ Pass |
| Passenger Login | Login with correct/incorrect credentials | ✅ Pass |
| Admin Login | Login with correct/incorrect credentials | ✅ Pass |
| Add Flight | Add valid flight data | ✅ Pass |
| View Flights | Show list of all added flights | ✅ Pass |
| Book Ticket | Book ticket on available flights | ✅ Pass |
| Cancel Booking | Cancel existing booking | ✅ Pass |
| View Ticket Details | Show details for all booked tickets | ✅ Pass |
| Data Persistence | Check if files save/load data correctly | ✅ Pass |

**2. Edge Case Handling**

|  |  |  |
| --- | --- | --- |
| **Edge Case** | **Expected Result** | **Status** |
| Booking when no seats are available | Show error, prevent overbooking | ✅ Pass |
| Cancelling an already-cancelled ticket | Display appropriate message | ✅ Pass |
| Login with blank input | Show error message | ✅ Pass |
| Adding duplicate flight ID (manually edited) | Prevented by ID auto-increment | ✅ Pass |
| Cancelling with invalid flight ID | Show "Flight not found" | ✅ Pass |

**3. File Validation**

Tested file handling by:

* Opening files (flights.txt, passengers.txt, tickets.txt) after operations
* Verifying correct formatting and consistency of data
* Ensuring that cancelled tickets are preserved with cancellation status

**4. User Experience Testing**

* Menus displayed cleanly and clearly
* System guides users with prompts and messages
* After each action, user is returned to appropriate menu without crashes

**5. Final Result**

The system passed **all critical functional and non-functional tests** under different user roles and data conditions. It performs reliably across sessions and stores data persistently through file handling.

**5.3 Future Enhancements / Scope for Improvement**

1. **Add GUI:** Replace console with a user-friendly graphical interface (e.g., Qt or WinForms).
2. **Database Integration:** Use SQLite/MySQL instead of text files for better performance and scalability.
3. **Email Notifications:** Send booking/cancellation emails to passengers.
4. **Search & Filters:** Allow passengers to search flights by date, destination, or price.
5. **Profile Editing & Recovery:** Enable users to update info and recover passwords.
6. **Payment Simulation:** Add mock payment options for realistic booking experience.
7. **Admin Controls:** Allow admin to cancel/reschedule flights and notify users.
8. **Printable Tickets:** Generate PDF or printable ticket formats.
9. **Analytics Dashboard:** Show booking stats and trends to Admin.
10. **Mobile App Support:** Future-proof the system for Android/iOS compatibility.

**5.4 Source Code**

The complete source code for the *Hazel Blue Airline Reservation System* is presented below. It demonstrates the full implementation of the project using object-oriented programming in **C++**, including user role management, flight operations, ticket booking, file handling, and menu navigation.

Each class and function is modularly organized for clarity and maintainability. The system can be compiled and executed using any C++ compiler that supports standard input/output and file operations.

**5.5 Conclusion**

The *Hazel Blue Airline Reservation System* marks the successful completion of a fully functional, console-based airline ticket booking application developed in **C++**. This project was designed to replace manual and paper-based ticketing with a structured digital solution that improves accuracy, efficiency, and user experience.

The system caters to two primary roles — **Admin** and **Passenger** — each having clearly defined access to core features like flight management, ticket booking, viewing and cancelling reservations. Through the use of **object-oriented principles**, the system maintains modularity, reusability, and logical separation between components. **File handling** replaces the need for a database, making it lightweight and ideal for academic use or small-scale operations.

In addition to implementing a complete booking lifecycle, the project demonstrates real-world software development practices such as **menu-driven UI design**, **data validation**, and **persistent storage**. The structure not only meets the functional requirements but also ensures the system can be extended or upgraded in the future — for example, integrating a graphical interface or online payment gateway.

Overall, this project served as an excellent hands-on learning experience. It deepened the understanding of **object-oriented programming**, system architecture, and the importance of user-centric design. It also showcased how theoretical knowledge can be translated into a practical, working application.

The successful execution of this system reflects the teamwork, planning, and commitment put forth by all contributors — and stands as a solid foundation for more advanced software projects in the future.