**Flight Reservation System**

****

Session: 2022 – 2026

**Submitted by:**

Muhammad Tayyab 2022-CS-133

**Supervised by:**

Prof. Dr. Muhammad Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table of Contents**

**Introduction………………………………………………………………………………...……3**

**Class Responsibility Collaboration Card……………………………...………………4**

**Object Oriented Programming………………………………………………………...…6**

**Design Pattern Implementation..………………………………………………………...8**

**Class Details……........……………………………………………………………..…….......….9**

**Conclusion……........……………………………………………………………..……........…..10**

**YouTube Link:**

https://youtu.be/2H0l0Lc0HjI

* **Introduction**
* **Overview**
* The Flight Reservation System is a software application designed to facilitate the process of booking and managing flight reservations for customers. It provides a user-friendly interface for users to search for flights, make reservations, manage bookings, and perform other related tasks. The system integrates with airline databases and ensures accurate and up-to-date flight information.
* The Flight Reservation System aims to streamline the flight booking process, improve customer satisfaction, and enhance the overall efficiency of flight management. It automates many manual tasks involved in reservation handling, reducing human errors and saving time for both customers and airline staff. The system offers a seamless experience for users, enabling them to easily find and book flights based on their preferences and requirements.
* **Objectives**

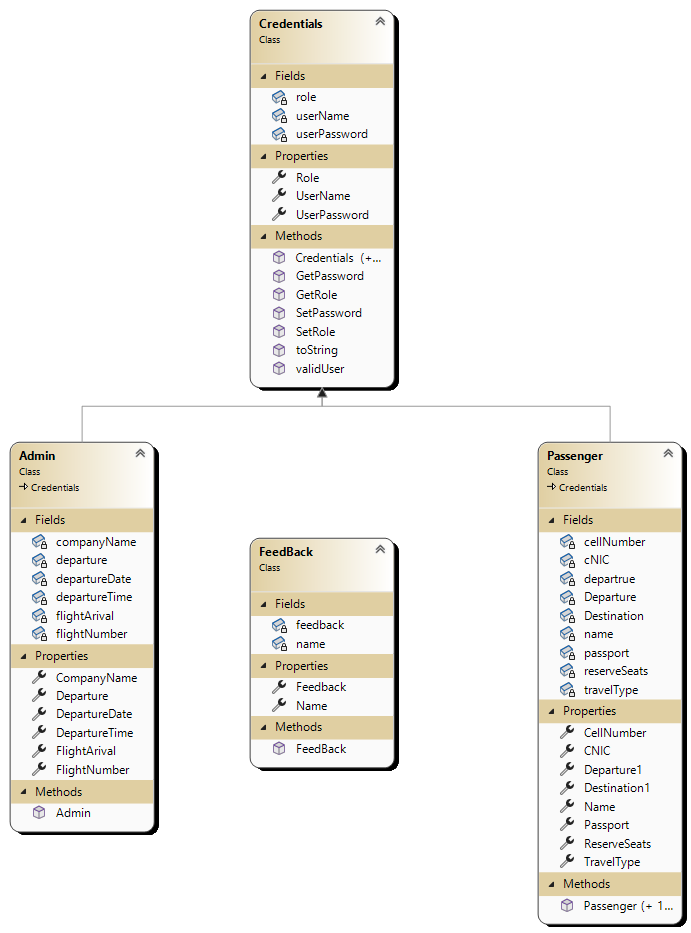
The main objective of the Flight Reservation System is to provide a reliable and efficient platform for users to book and manage flight reservations. The system aims to achieve the following goals:

Simplify Flight Booking: The system should offer an intuitive and user-friendly interface that allows users to search for flights, view available options, and make bookings with ease. It should minimize the complexity of the booking process and provide clear instructions and feedback to users.

Accurate and Up-to-Date Information: The system should integrate with reliable data sources to fetch real-time flight information, including schedules, seat availability, fares, and any updates or changes. It should ensure that users always have access to the most accurate and up-to-date information when making reservations.

* **Functionality**
* The intended functionality for the project includes the Sign in Sign Up system for the users. There is a built-in user for Admin. Only passengers can sign up for the system while the Flights are added by the admin. Its functionality also includes the Create, Retrieve, Update, Delete (CRUD) operations. It adds, removes, updates and prints the flights, passengers data. It also includes the functionality of adding and showing the statistics of the Flights

Class Responsibility Collaboration Diagram (CRC)

****

* **Object Oriented Programming**

Object-oriented programming (OOP) is a programming paradigm based on the concepts of the “objects”, which contains data and code called as attributes and behavior of the class respectively. The main concepts of OOP includes association, inheritance and polymorphism. I have used this programming paradigm in my project.

* **Association**

In Object-oriented programming, Association is a relation between two separate classes which establishes through their objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many. There are two types of Association, Aggregation and Composition. I have used Composition in my project. It is used in two places.

1. Person class have an object of Credentials. It is a relation of one-to-one because a person contains only one username, password and role. Credentials cannot exist without the object of Person, so I used composition for this particular purpose.
2. Player class contains an object of Stats. This association also is of one-to-one relation as a player can contain only one instance of runs, innings, wickets, etc. As the object of Stats should not exist without the existence of Player object, so we used composition here as well.

**Advantage**

If I compare this with my procedural programming concepts, I can observe that there is a clear advantage of OOP. There was disjoint data of Flights and Passengers in procedural programming which is rectified in OOP. Now the objects of Flights and Passengers are within the class of Flights and Passenger respectively.

* **Inheritance**

Inheritance is one of the core concepts of Object-oriented programming approach. It is a feature that allows a new class to derive from an existing class. The new class inherits all the public or protected attributes and the member functions of the base class. I have used this OOP concept in one place in this management system.

1. Flight class is a parent class.Passenger class is derived from this class.I have used the concept of inheritence here because there is similar attributes in the passenger class.

**Advantage**

Inheritance gives various advantages over procedural programming. It promotes code-reusability and reduces redundancy. It helps in organizing the program’s structure. It allows flexibility in the code as you will adjust in one place and the rest of the code will work smoothly.

* **Polymorphism**

Polymorphism is also one of the core concepts of Object-oriented programming approach. This concept refers to the ability of a function to perform multiple operation under different circumstances. There are two types of Polymorphism. The type of polymorphism used to extend the functionality of common functions in parent and child classes is called Dynamic Polymorphism. I have used Dynamic Polymorphism in few places.

In the Flight Reservation System project, polymorphism is employed in the setter and getter methods of certain classes. Polymorphism, a core concept in object-oriented programming, allows different classes to be treated as instances of a common base class or interface**.**

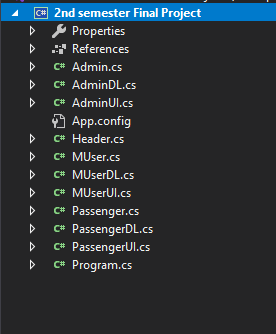
**For example**, a polymorphic getter method like GetFlightNumber() can be overridden in the derived classes to return the appropriate flight number based on the flight type. Similarly, a polymorphic setter method like SetDepartureTime() can be overridden to handle the specific validation or processing logic required for different flight types.

**Advantage**

Polymorphism allowed us to extend the functionalities from the base class to use them for the child classes. Due to polymorphism, our code has become shorter because it didn’t required us to copy paste the whole code and then make changes to it. Dynamic Polymorphism has enabled the programmer to use the same function in different manner. We lacked this when we were making our projects in procedural programming.

* **Design Pattern Implementation**

The directory structure for the project is given below :-

****

* **Business Logic (BL)**

The Business Logic folder contains the main classes of the project. It includes the classes of MUserBL, AdminBL, PassengerBL.

* **Data Layer (DL)**

The Data Layer folder contains the static Lists and functions of the project. It contains the list of the Flights , list of Passengers. There are static functions of each class as well such as storing and loading of data to and from files. It also includes the other static functions. This folder also contains a class of Interface. This class includes the static functions of interface which directs the functions depending on the user’s choice.

* **User Interface (UI)**

This folder contains all the code for printing and input of the data. This folder deals with the interaction of user with the application. It includes the menus and other functions which are used for printing and taking inputs from the user.

* **Class Details**
* **User**

The User class is a fundamental component of the Flight Reservation System project. It represents a user of the system and provides functionality for user login and sign-in. This class encapsulates user-related data and operations.

* **Properties**
* Username: Represents the username of the user. It is a unique identifier for each user.

Password: Stores the user's password, ensuring the confidentiality and security of their account.

Role: Store the Role of the user if it is Admin it login and automatically show the admin interface.

* **Admin**
* The Admin class is a vital component of the Flight Reservation System project. It represents an administrator who manages the flights and related data within the system. This class provides functionality for adding, updating, and retrieving flight information etc.
* **Passenger**
* The Passenger class is a key component of the Flight Reservation System project. It represents an individual passenger who reserves tickets for flights within the system. This class provides functionality for ticket reservation and passenger information management**.**

Match Schedule class is a child class of Schedule. Its attributes are opposition, ground and ticket price.

**Header**

The Header class is an essential component of the Flight Reservation System project. It represents the header of the application, providing functionality for displaying and managing the header content.

**Properties**

Title: Represents the title or main heading of the application.

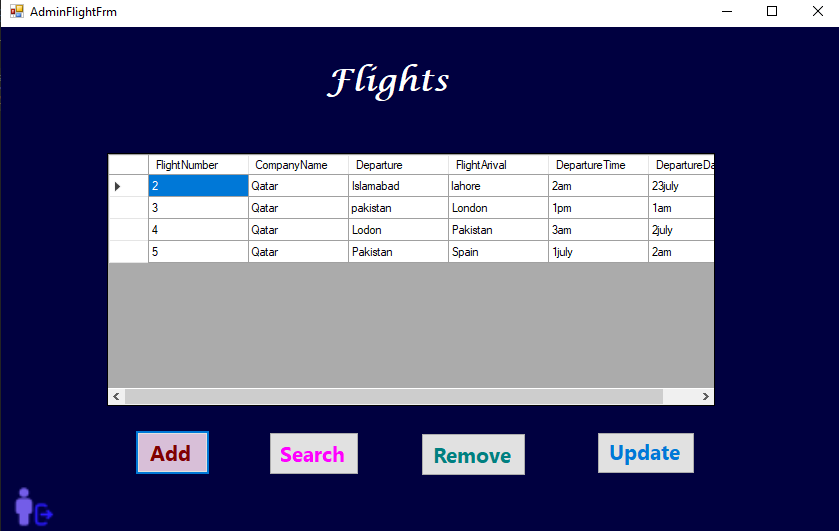
Subtitle: Stores an additional description or subtitle for the application**.**

**WireFrames:**

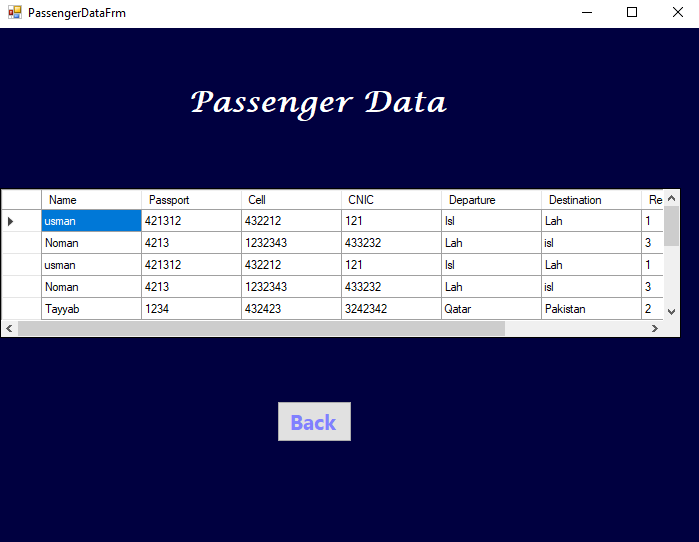
****

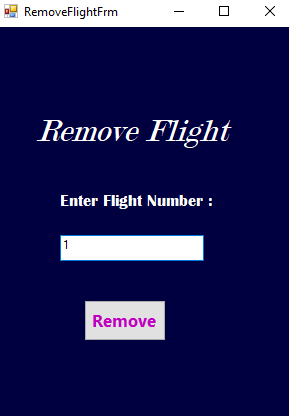
****

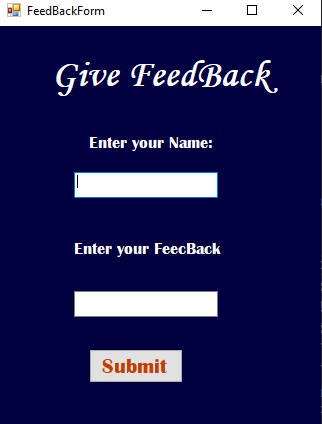
****

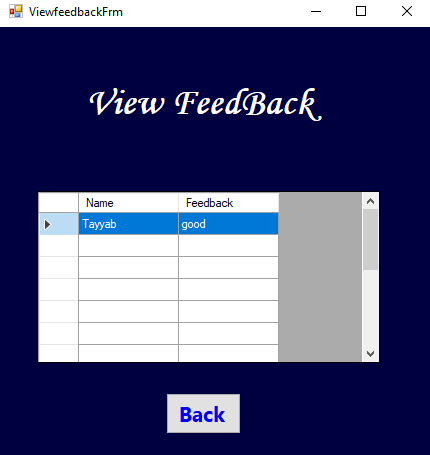
****

****

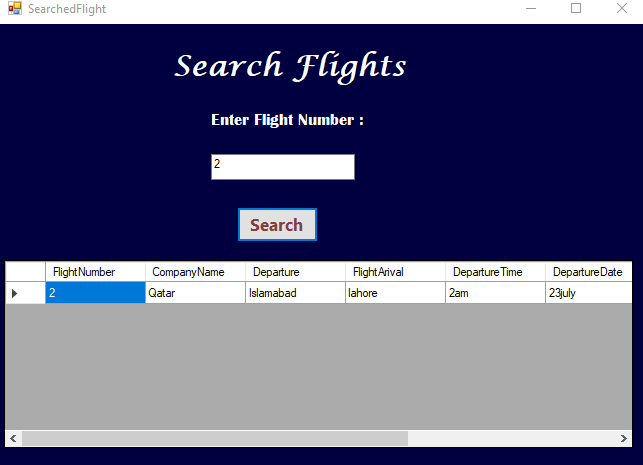
****

****

****

****

****

****

****

**Code of Application:**

Here is the code of my project…..

**Person Class:**

public class Person

{

protected Credentials crd = new Credentials();

public Person()

{

}

public Person(Credentials crd)

{

this.crd = crd;

}

public void SetUser(Credentials crd)

{

this.crd = crd;

}

public Credentials GetUser()

{

return this.crd ;

}

public string getRole()

{

return $"{ crd.GetRole(),-25}";

}

}

**Credientals Class:**

public class Credentials

{

private string userName = "";

private string userPassword = "";

private string role = "";

public string UserName { get => userName; set => userName = value; }

public string UserPassword { get => userPassword; set => userPassword = value; }

public string Role { get => role; set => role = value; }

public string GetPassword()

{

return this.userPassword;

}

public string GetRole()

{

return this.role;

}

public void SetPassword(string userPassword)

{

this.userPassword = userPassword;

}

public void SetRole(string role)

{

this.role = role;

}

public Credentials(string userName, string userPassword, string role)

{

this.userName = userName;

this.userPassword = userPassword;

this.role = role;

}

public Credentials()

{

}

public Credentials(string userPassword,string role)

{

this.userPassword=userPassword;

this.role=role;

}

public string validUser(Credentials existingUser)

{

foreach (Credentials user in MUserDL.usersList)

{

if (user.userName == existingUser.userName && user.userPassword == existingUser.userPassword)

{

return user.role;

}

}

return "User Not Found";

}

public virtual string toString()

{

return (userName + "\t");

}

}

**Admin Class:**

class Admin : Credentials

{

private int flightNumber;

private string companyName;

private string departure;

private string flightArival;

private string departureTime;

private string departureDate;

public int FlightNumber { get => flightNumber; set => flightNumber = value; }

public string CompanyName { get => companyName; set => companyName = value; }

public string Departure { get => departure; set => departure = value; }

public string FlightArival { get => flightArival; set => flightArival = value; }

public string DepartureTime { get => departureTime; set => departureTime = value; }

public string DepartureDate { get => departureDate; set => departureDate = value; }

public Admin(int FlightNumber, string CompanyName, string Departure, string FlightArival, string DepartureTime, string DepartureDate)

{

this.FlightNumber = FlightNumber;

this.CompanyName = CompanyName;

this.Departure = Departure;

this.DepartureDate = DepartureDate;

this.DepartureTime = DepartureTime;

this.FlightArival = FlightArival;

}

}

**Passenger Class:**

class Passenger : Credentials

{

private int passport;

private int cellNumber;

private int cNIC;

private int reserveSeats;

private string travelType;

private string Departure;

private string Destination;

private string name;

private string departrue;

public int Passport { get => passport; set => passport = value; }

public int CellNumber { get => cellNumber; set => cellNumber = value; }

public int CNIC { get => cNIC; set => cNIC = value; }

public int ReserveSeats { get => reserveSeats; set => reserveSeats = value; }

public string TravelType { get => travelType; set => travelType = value; }

public string Departure1 { get => Departure; set => Departure = value; }

public string Destination1 { get => Destination; set => Destination = value; }

public string Name { get => name; set => name = value; }

public Passenger(string Name,string password,string role,int Passport, int CNIC, int CellNumber, int ReserveSeats, string TravelType, string Departure, string FlightArival) : base(password,role)

{

this.Name=Name;

this.Passport = Passport;

this.CNIC = CNIC;

this.CellNumber = CellNumber;

this.ReserveSeats = ReserveSeats;

this.TravelType = TravelType;

this.Departure1 = Departure;

this.Destination1 = FlightArival;

}

public Passenger(string name, int passport, int cellNumber, int cNIC, string departrue, string destination, int reserveSeats, string travelType)

{

this.Name = name;

this.passport = passport;

this.cellNumber = cellNumber;

this.cNIC = cNIC;

this.Departure = departrue;

Destination = destination;

this.reserveSeats = reserveSeats;

this.travelType = travelType;

}

}

**Feedback Class:**

public class FeedBack

{

private string name;

private string feedback;

public string Name { get => name; set => name = value; }

public string Feedback { get => feedback; set => feedback = value; }

public FeedBack(string name,string feedback)

{

this.Name = name;

this.Feedback = feedback;

}

}

**All DL Classes:**

**Admin DL:**

class AdminDL

{

public static List<Admin> Flightdata = new List<Admin>();

public static Admin admin;

public static Admin getFlight(int flightNo)

{

foreach (Admin item in Flightdata)

{

if (flightNo == item.FlightNumber)

{

return item;

}

}

return null;

}

public static void AddFlightIntoTheList(Admin s)

{

Flightdata.Add(s);

}

public static void StoreFlight(string path)

{

StreamWriter store = new StreamWriter(path);

foreach (Admin user in Flightdata)

{

store.WriteLine(user.FlightNumber + "," + user.CompanyName + "," + user.Departure +

"," + user.FlightArival + "," + user.DepartureDate + "," + user.DepartureTime);

}

store.Flush();

store.Close();

}

public static void LoadFlight(string path)

{

if (File.Exists(path))

{

StreamReader storeFlight = new StreamReader(path);

string line;

while ((line = storeFlight.ReadLine()) != null)

{

string[] splittedRecord = line.Split(',');

int flightnumber = int.Parse(splittedRecord[0]);

string companyName = splittedRecord[1];

string departure = splittedRecord[2];

string destination = splittedRecord[3];

string date = splittedRecord[4];

string time = splittedRecord[5];

Admin newUser = new Admin(flightnumber, companyName, departure, destination, date, time);

AddFlightIntoTheList(newUser);

}

storeFlight.Close();

}

}

}

**Passenger DL:**

class PassengerDL

{

public static List<Passenger> Ticketdata = new List<Passenger>();

public static List<Passenger> Feedback = new List<Passenger>();

private static List<Admin> Flightdata = new List<Admin>();

public static Passenger GetPassport(int PassportNO)

{

foreach (Passenger item in Ticketdata)

{

if (PassportNO==item.Passport)

{

return item;

}

}

return null;

}

public static void AddTicketsIntoList(Passenger obj)

{

Ticketdata.Add(obj);

}

public static void StorePassenger(string path)

{

StreamWriter storePassenger = new StreamWriter(path,false);

foreach (Passenger passenger in PassengerDL.Ticketdata)

{

storePassenger.WriteLine(passenger.Name + "," + passenger.Passport + "," + passenger.CellNumber + "," + passenger.CNIC + "," + passenger.Departure1 + "," + passenger.Destination1 + "," + passenger.ReserveSeats + "," + passenger.TravelType);

}

storePassenger.Flush();

storePassenger.Close();

}

public static void LoadPassenger(string path)

{

if (File.Exists(path))

{

StreamReader storeUser = new StreamReader(path);

string line;

while ((line = storeUser.ReadLine()) != null)

{

string[] splittedRecord = line.Split(',');

if (splittedRecord.Length == 8)

{

string name = splittedRecord[0];

int passport = int.Parse(splittedRecord[1]);

int CellNumber=int.Parse(splittedRecord[2]);

int CNIC= int.Parse(splittedRecord[3]);

string Departrue = splittedRecord[4];

string Destination = splittedRecord[5];

int ReserveSeats= int.Parse(splittedRecord[6]);

string TravelType = splittedRecord[7];

Passenger passenger = new Passenger(name, passport, CellNumber, CNIC, Departrue, Destination, ReserveSeats, TravelType);

Ticketdata.Add(passenger);

}

else

{

MessageBox.Show("Invalid data in the file: " + line);

}

}

storeUser.Close();

}

}

}

**Credientals DL:**

public class MUserDL

{

public static List<Credentials> usersList = new List<Credentials>();

public static bool addUser(Credentials newUser, string path)

{

Person obj = new Person();

bool flag = true;

foreach (Credentials u in usersList)

{

if (obj.getRole() == "Admin")

{

if (u.GetPassword() == newUser.GetPassword() && u.UserName == newUser.UserName)

{

flag = true;

break;

}

else

{

flag = false;

}

}

else

{

flag = false;

}

}

if (obj.getRole() == "Customer")

{

foreach (Credentials s in usersList)

{

if (s.UserName == s.UserName && s.GetPassword() == s.GetPassword())

{

flag = true;

break;

}

else

{

flag = false;

}

}

}

if (flag == true)

{

usersList.Add(newUser);

storeUser(path, newUser);

}

return flag;

}

public static void storeUser(string path, Credentials newUser)

{

StreamWriter storeUser = new StreamWriter(path, true);

foreach (Credentials user in usersList)

{

storeUser.WriteLine(user.UserName + "," + user.GetPassword() + "," + user.GetRole());

}

storeUser.Close();

}

public static Credentials search(string name,string password)

{

foreach (Credentials user in MUserDL.usersList)

{

if (name==user.UserName&&password==user.UserPassword)

{

return user;

}

}

return null;

}

public static Credentials search(string name)

{

foreach (Credentials user in MUserDL.usersList)

{

if (name == user.UserName)

{

return user;

}

}

return null;

}

public static bool loadUser(string path)

{

string line;

if (File.Exists(path))

{

StreamReader storeUser = new StreamReader(path);

while ((line = storeUser.ReadLine()) != null)

{

string[] splittedRecord = line.Split(',');

if (splittedRecord.Length >= 3)

{

Credentials newUser = new Credentials();

newUser.UserName=(splittedRecord[0]);

newUser.SetPassword(splittedRecord[1]);

newUser.SetRole(splittedRecord[2]);

usersList.Add(newUser);

}

else

{

}

}

storeUser.Close();

return true;

}

else

{

return false;

}

}

}

**Feedback DL:**

class FeedbackDL

{

public static List<FeedBack> feeds = new List<FeedBack>();

public static void StoreFeedback(string path3)

{

StreamWriter storeFeedback = new StreamWriter(path3);

foreach (FeedBack user in FeedbackDL.feeds)

{

storeFeedback.WriteLine(user.Name + "," + user.Feedback);

}

storeFeedback.Flush();

storeFeedback.Close();

}

public static void LoadFeedback(string path)

{

if (File.Exists(path))

{

StreamReader storeUser = new StreamReader(path,true);

string line;

while ((line = storeUser.ReadLine()) != null)

{

string[] splittedRecord = line.Split(',');

string name = splittedRecord[0];

string feedback = splittedRecord[1];

FeedBack feedBack = new FeedBack(name, feedback);

feeds.Add(feedBack);

}

storeUser.Close();

}

}

}

**Conclusion:**

In conclusion, the Flight Reservation System project has demonstrated the successful creation of a functional, user-friendly, and robust application for managing flight reservations. By adhering to industry best practices, incorporating essential features, and employing object-oriented programming concepts, the project has laid the groundwork for a reliable and scalable solution. The system's ability to handle flight data, user authentication, ticket reservations, and passenger management reflects its practicality and real-world