**HO CHI MINH UNIVERSITY OF EDUCATION AND TECHNOLOGY**

**FACULTY OF INTERNATIONAL EDUCATION**



**SUBJECT: OBJECT-ORIENTED PROGRAMMING**

**TOPIC: FLIGHT TICKET SALE SYSTEM**

**GROUP 17**

**Semester 01 – Academic year: 2025 – 2026**

**Class ID:**

**Lecturer: Mr. Le Van Vinh**

**Group of students:**

-Nguyen Chi Dung Student ID: 23110010

- Nguyen Do Minh Quan Student ID: 23110059

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student ID | Task | Completed (%) |
| Nguyen Chi Dung | 23110010 | Design database | 100 |
| Design function add, delete, update | 100 |
| Nguyen Do Minh Quan | 23110059 | Design GUI | 100 |
| Design book, cancel tickets | 100 |
| All members of the group | | Study about Winform C# | 100 |
| Learn some references about iTextSharp and .Net.Mail libraries | 100 |
| Fix errors of the program | 100 |

Contents

[**CHAPTER 1. FUNCTIONS USED FOR PROGRAM** 6](#_Toc211804662)

[**1.1. Data transfer objects (DTO)** 6](#_Toc211804663)

[**1.2.Class DBHelper** 7](#_Toc211804664)

[**1.3. Add, delete, update** 9](#_Toc211804665)

[**1.4. Some other functions** 15](#_Toc211804666)

[**1.4.1. Book and cancel flight tickets** 15](#_Toc211804667)

[**1.4.2. Export Ticket to PDF using iTextSharp library** 19](#_Toc211804668)

[**1.4.3. Send ticket to customer via email using System.Net.Mail library** 21](#_Toc211804669)

[**1.5. Object-oriented characteristics applied in program** 23](#_Toc211804670)

[**CHAPTER 2. USER INTERFACE (WINFORM C#)** 25](#_Toc211804671)

[**CHAPTER 3. CONCLUSION** 33](#_Toc211804672)

**ACKNOWLEDMENT**

We would like to express our deepest gratitude to Mr. Le Van Vinh, our lecturer of the Object-Oriented Programming course, for his dedicated guidance, valuable feedback, and continuous support throughout the development of this project. His instructions and encouragement have been essential in helping us understand and apply object-oriented programming principles effectively in our work.

We are also thankful to our classmates and friends for their helpful suggestions and collaboration during the implementation process.

Finally, we would like to acknowledge the contributions of each member of our team. Through teamwork, discussion, and mutual effort, we successfully completed the Flight Ticket Sale System project, gaining not only technical knowledge but also valuable experience in software development and problem-solving.

**PREFACE**

We would like to express our deepest gratitude to Mr. Le Van Vinh, our lecturer of the Object-Oriented Programming course, for his dedicated guidance, valuable feedback, and continuous support throughout the development of this project. His instructions and encouragement have been essential in helping us understand and apply object-oriented programming principles effectively in our work.

We are also thankful to our classmates and friends for their helpful suggestions and collaboration during the implementation process.

Finally, we would like to acknowledge the contributions of each member of our team. Through teamwork, discussion, and mutual effort, we successfully completed the Flight Ticket Sale System project, gaining not only technical knowledge but also valuable experience in software development and problem-solving.

# **CHAPTER 1. FUNCTIONS USED FOR PROGRAM**

## **1.1. Data transfer objects (DTO)**

In this part we have 4 classes, we will use data in this part to connect to DTA and BLL layer:

- Flight.cs

public class Flight

{

    public string flightID { get; set; }

    public string departAirport { get; set; }

    public *DateTime* departTime { get; set; }

    public string arriveAirport { get; set; }

    public *DateTime* arriveTime { get; set; }

    public int remainSeat { get; set; }

    public decimal price { get; set; }

}

- Ticket.cs

    public class Ticket

    {

        public string ticketID { get; set; }

        public string flightID { get; set; }

        public decimal price { get; set; }

        public string username { get; set; }

        public *DateTime* bookingDate { get; set; }

        public int quantity { get; set; }

        public decimal totalPrice { get; set; }

    }

- User.cs:

public class User

    {

        public string Username { get; set; }

        public string Password { get; set; }

        public string Role { get; set; }

    }

- Session.cs : this class is used for log-in session later , fuction ClearSession() is used when user logs out.

public class Session

    {

        public static string CurrentUsername { get; set; }

        public static string CurrentUserRole { get; set; }

        public static void ClearSession()

        {

            CurrentUsername = null;

            CurrentUserRole = null;

        }

    }

## **1.2.Class DBHelper**

- In Database Access Layer (DAL), we have class DBHelper. This is the base class for inheritance. In DBHealper, we have some functions:

1. GetConnection() : this function returns the connection to database through ConnectionString

protected string connectionString = "Data Source= ;Initial Catalog=Flight\_Ticket\_2;Integrated Security=True";

protected *SqlConnection* GetConnection()

{

    return new *SqlConnection*(connectionString);

}

2. ExecuteNonQuery() : this function helps the program run INSERT, UPDATE, DELETE query in database

protected int ExecuteNonQuery(string *query*, *SqlParameter*[] *parameters* = null)

{

    using (*SqlConnection* conn = GetConnection())

    {

        using (*SqlCommand* cmd = new *SqlCommand*(*query*, conn))

        {

            if (*parameters* != null)

                cmd.Parameters.AddRange(*parameters*);

            conn.Open();

            return cmd.ExecuteNonQuery();

        }

    }

}

3. ExecuteQuery : this function helps to run SELECT query in database and returns a table

protected *DataTable* ExecuteQuery(string *query*, *SqlParameter*[] *parameters* = null)

{

    using (*SqlConnection* conn = GetConnection())

    {

        using (*SqlCommand* cmd = new *SqlCommand*(*query*, conn))

        {

            if (*parameters* != null)

                cmd.Parameters.AddRange(*parameters*);

            conn.Open();

*DataTable* dt = new *DataTable*();

            dt.Load(cmd.ExecuteReader());

            return dt;

        }

    }

}

4. ExecuteScalar : this function returns a number.

protected object ExecuteScalar(string *query*, *SqlParameter*[] *parameters* = null)

{

    using (*SqlConnection* conn = GetConnection())

    {

        using (*SqlCommand* cmd = new *SqlCommand*(*query*, conn))

        {

            if (*parameters* != null)

                cmd.Parameters.AddRange(*parameters*);

            conn.Open();

            return cmd.ExecuteScalar();

        }

    }

}

## **1.3. Add, delete, update**

To show this function, we choose object Flight to make examples:

- In Data Access Layer (DAL) we have class FlightDAL, here we have:

1. Add function: in this layer, we will have SQL Query and Parameters to insert into database by ExecuteNonQuery function.

public int AddFLight(*Flight* *f*)

{

    string query = "INSERT INTO Flights (FlightID, DepartAirport, DepartTime, ArriveAirport, ArriveTime, RemainSeat, Price) " +

                   "VALUES (@flightID, @departAirport, @departTime, @arriveAirport, @arriveTime, @remainSeat, @price)";

*SqlParameter*[] prms =

    {

        new *SqlParameter*("@flightID", *f*.flightID),

        new *SqlParameter*("@departAirport", *f*.departAirport),

        new *SqlParameter*("@departTime", *f*.departTime),

        new *SqlParameter*("@arriveAirport", *f*.arriveAirport),

        new *SqlParameter*("@arriveTime", *f*.arriveTime),

        new *SqlParameter*("@remainSeat", *f*.remainSeat),

        new *SqlParameter*("@price", *f*.price)

    };

    return ExecuteNonQuery(query, prms);

}

2. Delete function:

public int DeleteFlight(*Flight* *f*)

        {

            string query = "DELETE FROM Flights WHERE FlightID = @flightID";

*SqlParameter*[] prms =

            {

                new *SqlParameter*("@flightID", *f*.flightID)

            };

            return ExecuteNonQuery(query, prms);

        }

3. Update function:

public int updateFlight(*Flight* *f*)

        {

            string query = "UPDATE Flights SET DepartAirport = @departAirport, DepartTime = @departTime, " +

                           "ArriveAirport = @arriveAirport, ArriveTime = @arriveTime, RemainSeat = @remainSeat, Price = @price " +

                           "WHERE FlightID = @flightID";

*SqlParameter*[] prms =

            {

                new *SqlParameter*("@flightID", *f*.flightID),

                new *SqlParameter*("@departAirport", *f*.departAirport),

                new *SqlParameter*("@departTime", *f*.departTime),

                new *SqlParameter*("@arriveAirport", *f*.arriveAirport),

                new *SqlParameter*("@arriveTime", *f*.arriveTime),

                new *SqlParameter*("@remainSeat", *f*.remainSeat),

                new *SqlParameter*("@price", *f*.price)

            };

            return ExecuteNonQuery(query, prms);

        }

- In Business Logic Layer (BLL): we have class FlightBLL

1. Add function:

public void Add(*Flight* *f*)

        {

            if (string.IsNullOrEmpty(*f*.flightID))

            {

                throw new *ArgumentException*("Flight ID cannot be empty.");

            }

            if (*f*.departAirport == *f*.arriveAirport)

            {

                throw new *ArgumentException*("Departure and arrival airports cannot be the same.");

            }

            if (*f*.price <= 0)

            {

                throw new *ArgumentException*("Price must be greater than zero.");

            }

            fDAL.addFLight(*f*);

        }

2. Delete function:

public void Delete(*Flight* *f*)

{

if (string.IsNullOrEmpty(*f*.flightID))

      {

        throw new *ArgumentException*("Flight ID cannot be empty.");

       }

       fDAL.DeleteFlight(*f*);

}

3. Update function:

public void UpdateFlight(*Flight* *f*)

        {

            if (string.IsNullOrEmpty(*f*.flightID))

            {

                throw new *ArgumentException*("Flight ID cannot be empty.");

            }

            if (*f*.departAirport == *f*.arriveAirport)

            {

                throw new *ArgumentException*("Departure and arrival airports cannot be the same.");

            }

            if (*f*.price <= 0)

            {

                throw new *ArgumentException*("Price must be greater than zero.");

            }

            fDAL.updateFlight(*f*);

        }

- Next, here is the application of add, delete and update in User Interface (UI)

1. Add:

private void btnAddFlight\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

                var f = new *Flight*

                {

                    flightID = txtFlightID.Text,

                    departAirport = txtDepart.Text,

                    departTime = dtpDepart.Value,

                    arriveAirport = txtArrive.Text,

                    arriveTime = dtpArrive.Value,

                    remainSeat = Convert.ToInt32(txtSeat.Text),

                    price = Convert.ToDecimal(txtPrice.Text)

                };

                fBll.Add(f);

                MessageBox.Show("Flight added successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

                ResetForm();

                LoadFlights();

            }

            catch (*Exception* ex)

            {

                MessageBox.Show("Error adding flight: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

            }

        }

2. Delete:

private void btnDeleteFlight\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

                var f = new *Flight*

                {

                    flightID = txtFlightID.Text

                };

                fBll.Delete(f);

                MessageBox.Show("Flight deleted successfully!");

                ResetForm();

                LoadFlights();

            }

            catch (*Exception* ex)

            {

                MessageBox.Show("Error deleting flight: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

            }

        }

3. Update:

private void btnUpdate\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

                var f = new *Flight*

                {

                    flightID = txtFlightID.Text,

                    departAirport = txtDepart.Text,

                    departTime = dtpDepart.Value,

                    arriveAirport = txtArrive.Text,

                    arriveTime = dtpArrive.Value,

                    remainSeat = Convert.ToInt32(txtSeat.Text),

                    price = Convert.ToDecimal(txtPrice.Text)

                };

                fBll.UpdateFlight(f);

                MessageBox.Show("Flight updated successfully!");

                ResetForm();

                LoadFlights();

            }

            catch (*Exception* ex)

            {

                MessageBox.Show("Error updating flight: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

            }

        }

## **1.4. Some other functions**

### **1.4.1. Book and cancel flight tickets**

The nature of these functions is to apply add/delete function for tickets and then reduce/increase the number of remaining seats of the flight

Function to reduce

- DAL:

public int AddRemainSeat(string *flightID*, string *ticketID*)

        {

            string query = @"

                UPDATE Flights

                SET RemainSeat = RemainSeat +

                    (SELECT Quantity FROM Tickets WHERE TicketID = @ticketID)

                WHERE FlightID = @flightID

            ";

*SqlParameter*[] prms =

            {

                new *SqlParameter*("@flightID", *flightID*),

                new *SqlParameter*("@ticketID", *ticketID*)

            };

            return ExecuteNonQuery(query, prms);

        }

It is the same for ReduceRemainingSeat() but change the query to:

string query = @"

                UPDATE Flights

                SET RemainSeat = RemainSeat -

                    (SELECT Quantity FROM Tickets WHERE TicketID = @ticketID)

                WHERE FlightID = @flightID

            ";

- BLL :

public void DecreaseSeat(string *flightID*, string *ticketID*)

        {

            tDAL.ReduceRemainSeat(*flightID*, *ticketID*);

        }

public void IncreaseSeat(string *flightID*, string *ticketID*)

        {

            tDAL.AddRemainSeat(*flightID*, *ticketID*);

        }

- UI:

+ booking

private void btnBook\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

                var t = new *Ticket*

                {

                    ticketID = GenerateTicketID(),

                    flightID = txtFlightID.Text,

                    price = Convert.ToDecimal(txtPrice.Text),

                    username = Session.CurrentUsername,

                    bookingDate = DateTime.Now,

                    quantity = Convert.ToInt32(txtQuantity.Text),

                    totalPrice = Convert.ToDecimal(txtPrice.Text) \* Convert.ToDecimal(txtQuantity.Text)

                };

                tBll.Add(t);

                tBll.DecreaseSeat (t.flightID, t.ticketID);

                MessageBox.Show("Flight booked successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

                ResetForm();

                LoadFlights();

            }

            catch (*Exception* ex)

            {

                MessageBox.Show(

                    "Error booking flight:\n" +

                    ex.Message + "\n\n" +

                    "Source: " + ex.Source + "\n" +

                    "StackTrace:\n" + ex.StackTrace

                );

            }

        }

+ Cancel:

private void btnCancel\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

*DataGridViewRow* row = dataGridViewTickets.SelectedRows[0];

                string ticketID = row.Cells["TicketID"].Value.ToString();

                string flightID = row.Cells["FlightID"].Value.ToString();

                var t = new *Ticket*

                {

                    ticketID = ticketID,

                    flightID = flightID,

                };

                tBLL.IncreaseSeat(flightID, ticketID);

                tBLL.Delete(t);

                MessageBox.Show("Ticket cancelled successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

            }

            catch

            {

                MessageBox.Show("Please select a ticket to cancel.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

                return;

            }

        }

### **1.4.2. Export Ticket to PDF using iTextSharp library**

- Function in BLL:

public void ExportPDF(*Ticket* *t*, string *filePath*)

        {

*Document* doc = new *Document*(PageSize.A6);

            PdfWriter.GetInstance(doc, new *System*.*IO*.*FileStream*(*filePath*, System.IO.FileMode.Create));

            doc.Open();

*Paragraph* title = new *Paragraph*("Ticket Information", FontFactory.GetFont(FontFactory.HELVETICA\_BOLD, 16));

            title.Alignment = Element.ALIGN\_CENTER;

            doc.Add(title);

            doc.Add(new *Paragraph*("\n"));

*PdfPTable* table = new *PdfPTable*(2);

            table.AddCell("Ticket ID");

            table.AddCell(*t*.ticketID);

            table.AddCell("Flight ID");

            table.AddCell(*t*.flightID);

            table.AddCell("Price");

            table.AddCell(*t*.price.ToString("C"));

            table.AddCell("Username");

            table.AddCell(*t*.username);

            table.AddCell("Booking Date");

            table.AddCell(*t*.bookingDate.ToString("g"));

            table.AddCell("Quantity");

            table.AddCell(*t*.quantity.ToString());

            table.AddCell("Total Price");

            table.AddCell(*t*.totalPrice.ToString("C"));

            doc.Add(table);

            doc.Add(new *Paragraph*("Thanks for choosing our service!", new *Font*(Font.FontFamily.HELVETICA, 12, Font.ITALIC)));

            doc.Close();

        }

- Application in UI

private void btnExportPDF\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

*DataGridViewRow* row = dataGridViewTickets.SelectedRows[0];

                var t = new *Ticket*

                {

                    ticketID = row.Cells["TicketID"].Value.ToString(),

                    flightID = row.Cells["FlightID"].Value.ToString(),

                    price = Convert.ToDecimal(row.Cells["Price"].Value),

                    username = row.Cells["Username"].Value.ToString(),

                    bookingDate = Convert.ToDateTime(row.Cells["BookingDate"].Value),

                    quantity = Convert.ToInt32(row.Cells["Quantity"].Value),

                    totalPrice = Convert.ToDecimal(row.Cells["TotalPrice"].Value)

                };

*SaveFileDialog* saveFileDialog = new *SaveFileDialog*

                {

                    Filter = "PDF file |\*.pdf",

                    Title = "Save Ticket as PDF",

                    FileName = $"Ticket\_{t.ticketID}.pdf"

                };

                if (saveFileDialog.ShowDialog() == DialogResult.OK)

                {

                    tBLL.ExportPDF(t, saveFileDialog.FileName);

                    MessageBox.Show("Ticket exported to PDF successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

                }

            }

            catch

            {

                MessageBox.Show("Please choose ticket to export to PDF File ", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

            }

        }

### **1.4.3. Send ticket to customer via email using System.Net.Mail library**

- In Data access layer (DAL), we have EmailHelper

public void SendTicketEmail(string *toEmail*, string *username*, string *ticketID*, string *flightID*, string *departure*, string *destination*, *DateTime* *date*,*DateTime* *datebuy* ,string *quantity*, decimal *price*, decimal *totalprice*)

        {

            string subject = $"Flight Ticket - Flight Code {*flightID*}";

            // Nội dung email

            string body = $@"

               // Format body by html and use parameters

            ";

            using (*MailMessage* mail = new *MailMessage*())

            {

                mail.From = new *MailAddress*(sendEmail, "Flight Ticket Saling");

                mail.To.Add(*toEmail*);

                mail.Subject = subject;

                mail.Body = body;

                mail.IsBodyHtml = true;

                using (*SmtpClient* smtp = new *SmtpClient*("smtp.gmail.com", 587))

                {

                    smtp.Credentials = new *NetworkCredential*(sendEmail, appPass);

                    smtp.EnableSsl = true;

                    smtp.Send(mail);

                }

            }

        }

In BLL we don’t have code for this function, we apply it straightly to UI

private void btnEmail\_Click(object *sender*, *EventArgs* *e*)

        {

            try

            {

*DataGridViewRow* row = dataGridViewTickets.SelectedRows[0];

                var t = new *Ticket*

                {

                    ticketID = row.Cells["TicketID"].Value.ToString(),

                    flightID = row.Cells["FlightID"].Value.ToString(),

                    price = Convert.ToDecimal(row.Cells["Price"].Value),

                    username = row.Cells["Username"].Value.ToString(),

                    bookingDate = Convert.ToDateTime(row.Cells["BookingDate"].Value),

                    quantity = Convert.ToInt32(row.Cells["Quantity"].Value),

                    totalPrice = Convert.ToDecimal(row.Cells["TotalPrice"].Value)

                };

*FlightBLL* fBLL = new *FlightBLL*();

*Flight* flight = fBLL.GetFlightByID(t.flightID);

*EmailHelper* emailHelper = new *EmailHelper*();

                string pdfPath = $"Ticket\_{t.ticketID}.pdf";

                emailHelper.SendTicketEmail(

*toEmail*: txtEmail.Text,

*username*: t.username,

*ticketID*: t.ticketID,

*flightID*: t.flightID,

*departure*: flight.departAirport,

*destination*: flight.arriveAirport,

*date*: flight.departTime,

*datebuy*: t.bookingDate,

*price*:  t.price,

*quantity*: t.quantity.ToString(),

*totalprice*: t.totalPrice

                );

                MessageBox.Show("Email sent successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

            }

            catch (*Exception* ex)

            {

                MessageBox.Show("Error sending email" + ex.Message);

            }

        }

## **1.5. Object-oriented characteristics applied in program**

**1.** **Encapsulation**: This characteristic is used nearly everywhere in our program, we have private/protected objects/variables nearly everywhere

**2. Inheritance** : we apply inheritance in 2 places

- Firstly in DA Layer, we have a base class is DBHelper and 3 subclasses: FlightDAL, TicketDAl and UserDAL. These 3 subclasses reused the functions of their base class : ExecuteQuery,…

- Secondly, in BL Layer we have Interface Imanager.

public interface *IManager*<*T*>

    {

        void Add(*T* *obj*);

        void Delete(*T* *obj*);

    }

This is the same as base class and its 2 children is FlightBLL and TicketBLL because both of them have and use again Add and Delete functions of Interface

**3. Abstract** : we apply abstract in DbHelper.cs

public abstract class DbHelper

The 3 subclasses just give the parameters for Execute functions, then DBHelper will work with SQL then return the correct result for 3 classes, but they don’t really need to understand what DBHelper does.

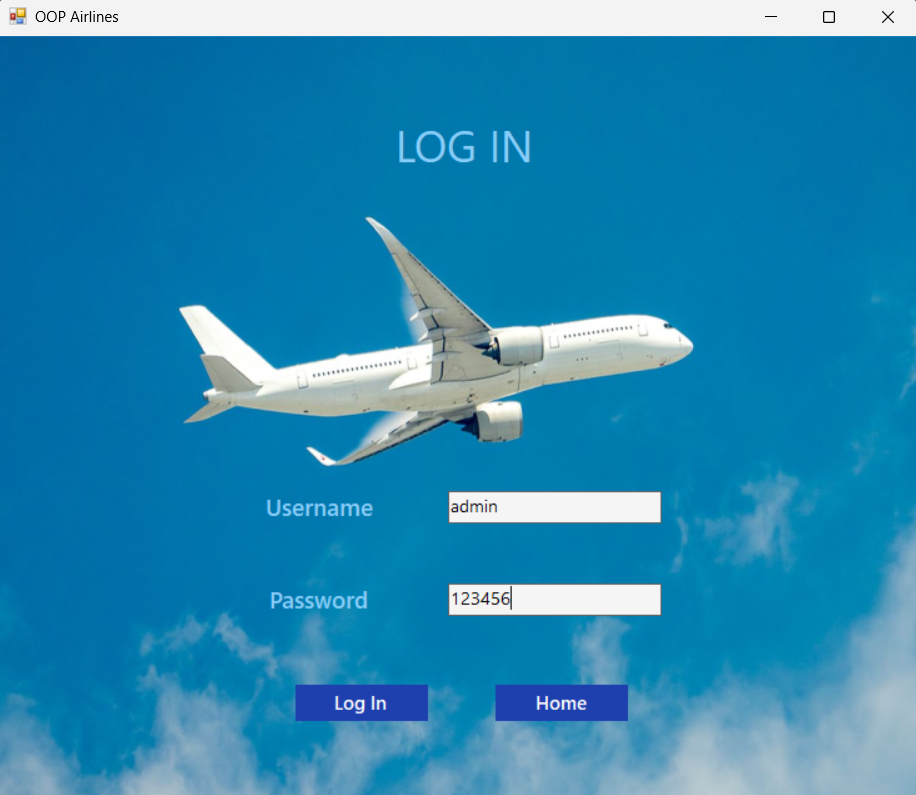
**4. Polymorphism**: we apply in 2 places

- In the DAL class, DBHelper, it creates functions that its subclasses use for many different purposes. For example, ExecuteNonQuery, for example, is often used for adding, deleting, and editing. But all 3 of the subclasses have the function of adding. Then there is only 1 ExecuteNonQuery that can do many different tasks.

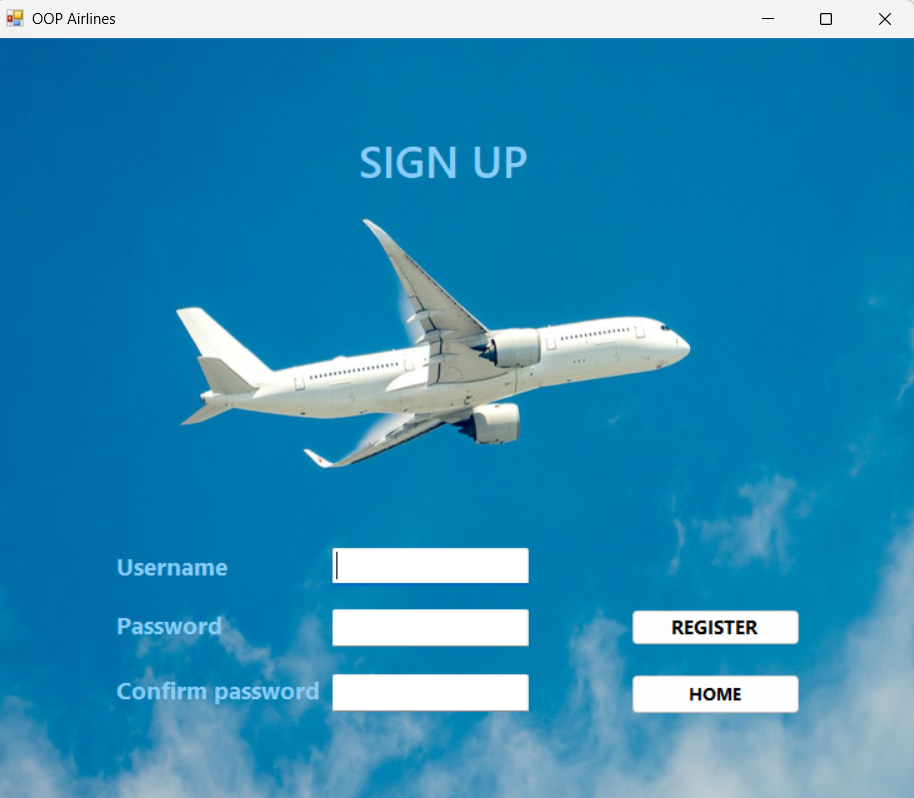
- In the BLL class, adding an Interface here to see the polymorphism more clearly, in which there are 2 functions: Add and Delete, it is the same one in DBHelper, the same function performs both the function of adding flights, adding tickets, and deleting as well.

# **CHAPTER 2. USER INTERFACE (WINFORM C#)**

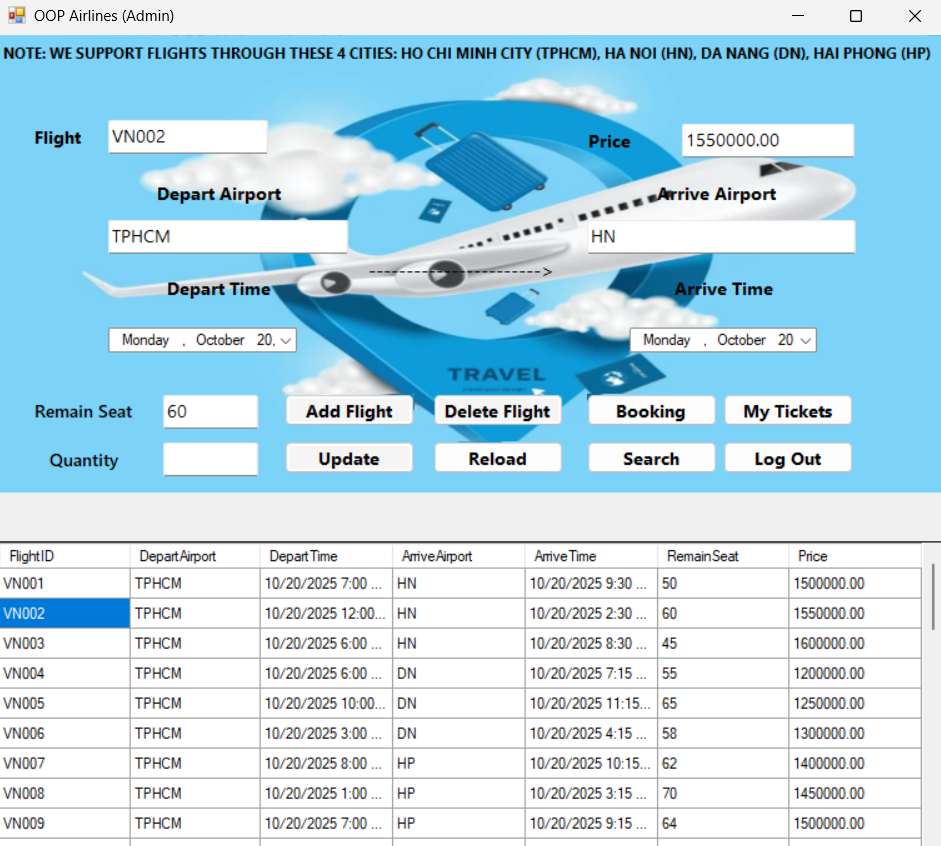
**-** Login form:



- Register:



- Control form for Admin



Here we have some functions add, delete, update, search, booking, see bought ticket, log out

- add : insert data in textboxs except quantity, then click “Add Flight”

- delete: click on DataGridView to choose flight then click “Delete Flight”

- update : click on DataGridView to choose flight then change information in textbox then click on “Update”

- book flights: click on DataGridView to choose flight then insert quantity of seats we need to book then click on “Booking”.

- search : insert depature place and destination in textbox departure and destination then click button “Search”.

- view booked ticket by CurrentUsername in Session.cs. I have mentioned about Session.cs before and now we will show clearly about it

This is the code parts about session in program:

In data transfer object:

public class Session

    {

        public static string CurrentUsername { get; set; }

        public static string CurrentUserRole { get; set; }

        public static void ClearSession()

        {

            CurrentUsername = null;

            CurrentUserRole = null;

        }

    }

In UI:

private void LoadTickets()

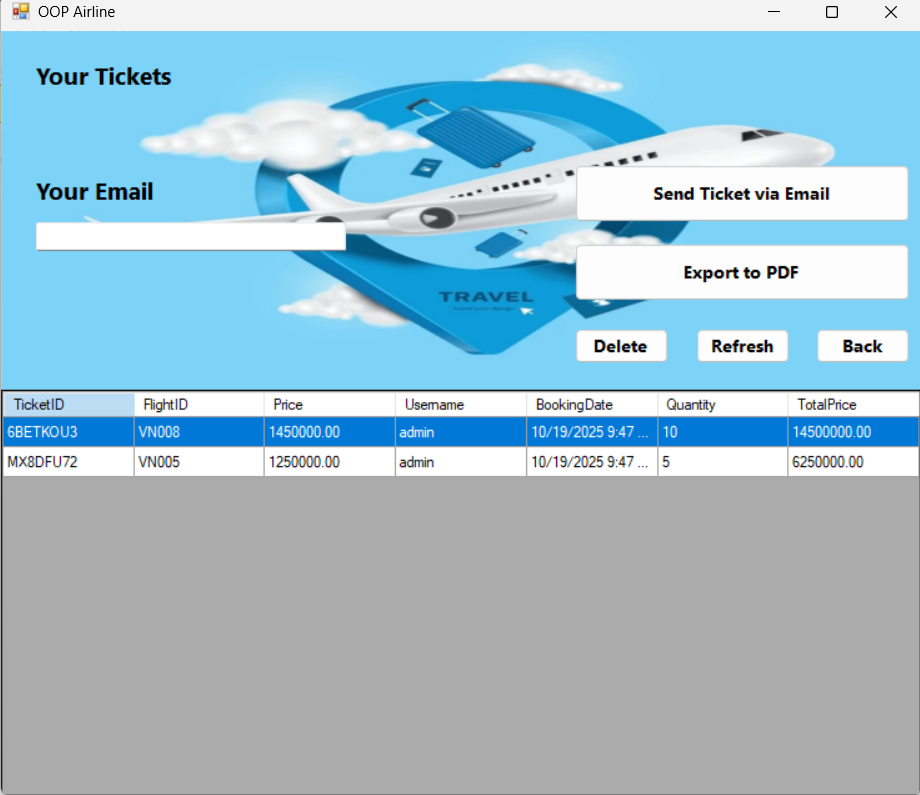
        {

            string username = Session.CurrentUsername;

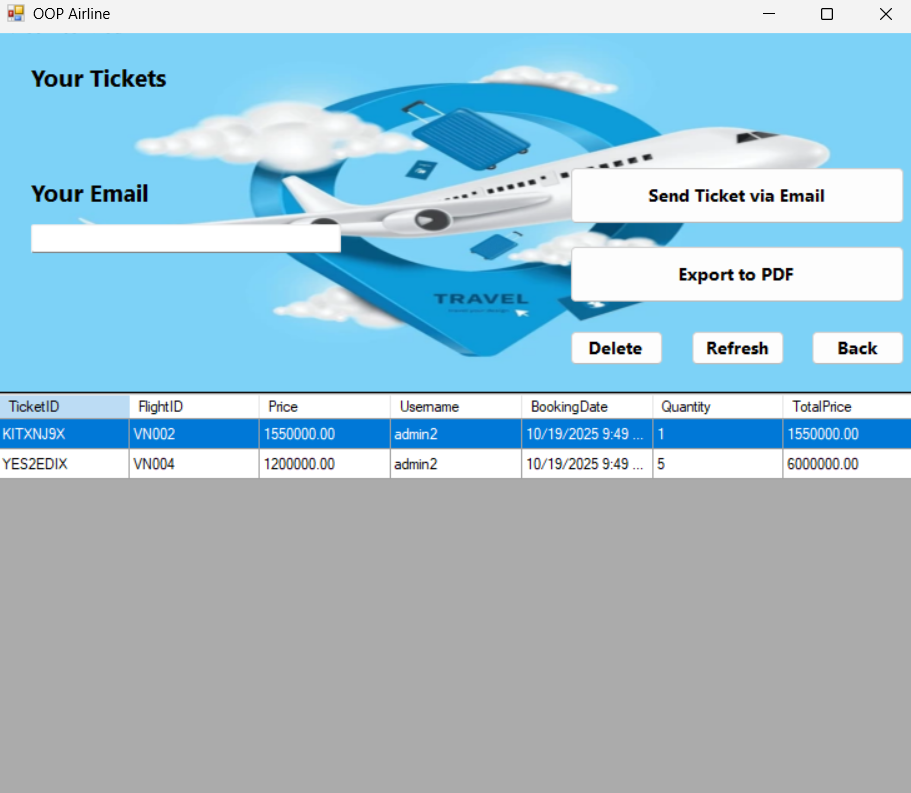
            dataGridViewTickets.DataSource = new *TicketBLL*().getTicketsByUsers(username);

        }

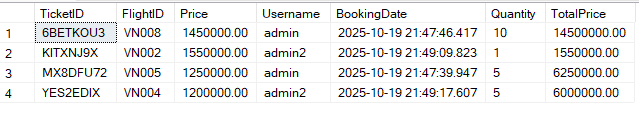
This is the tickets for username admin:



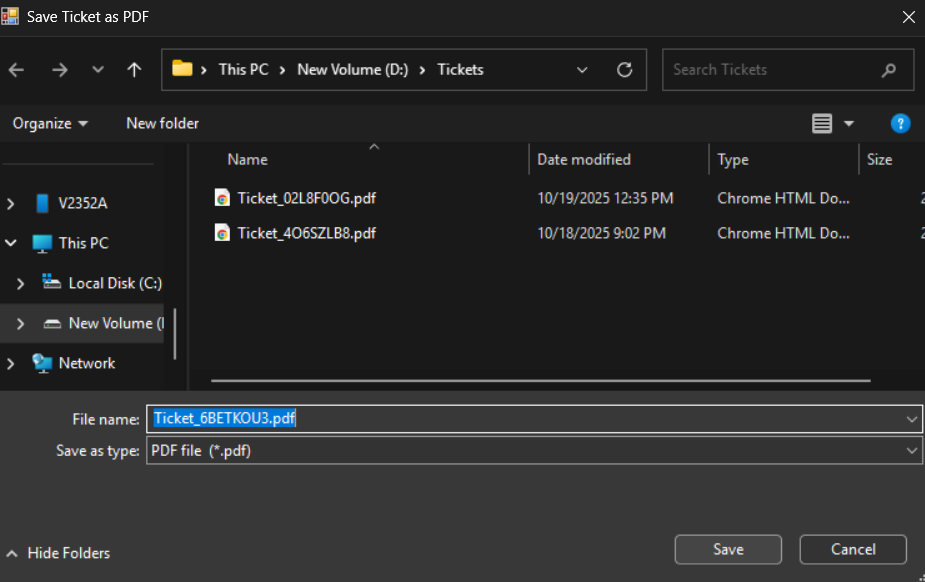
And these are tickets for username admin2:



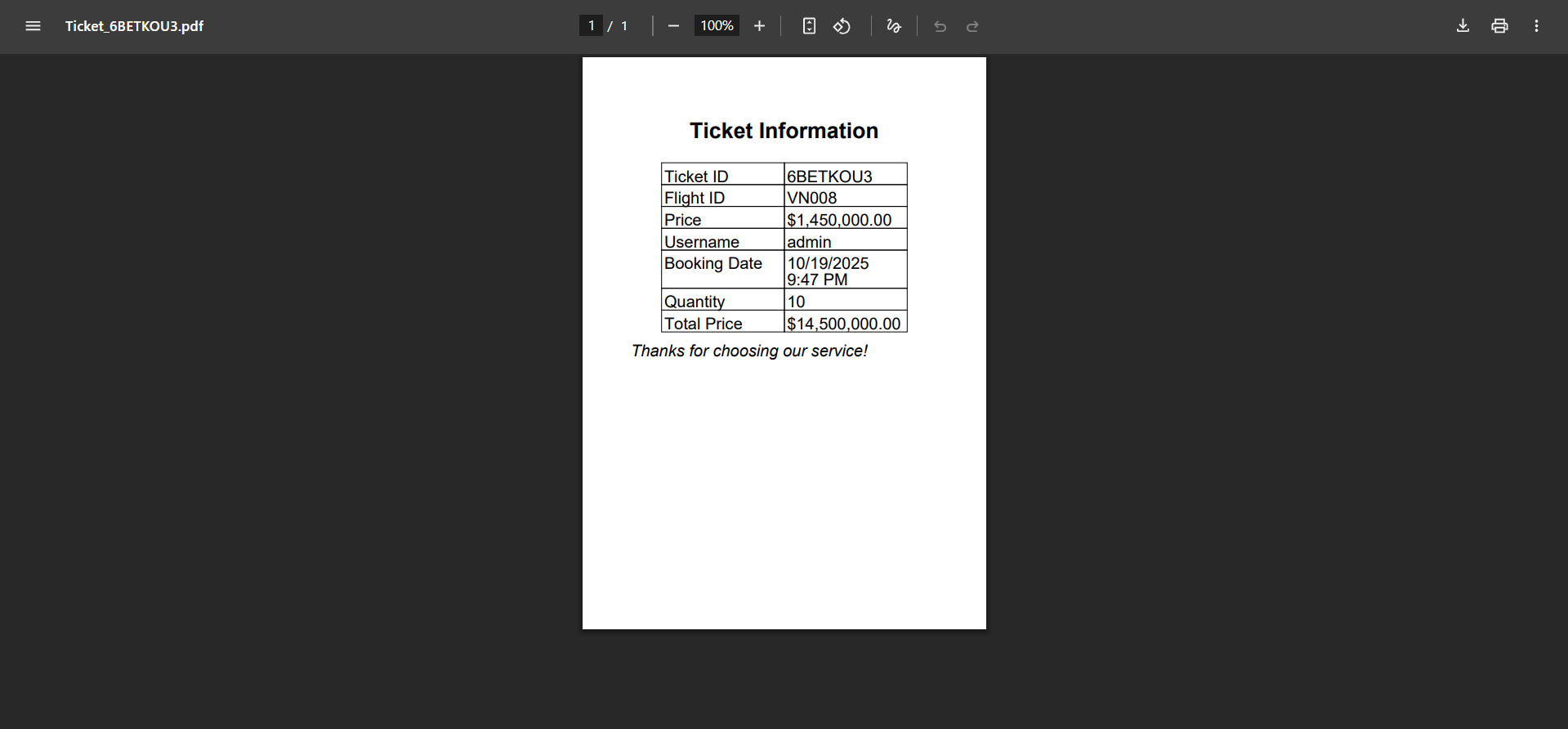
While data in database, table Tickets:



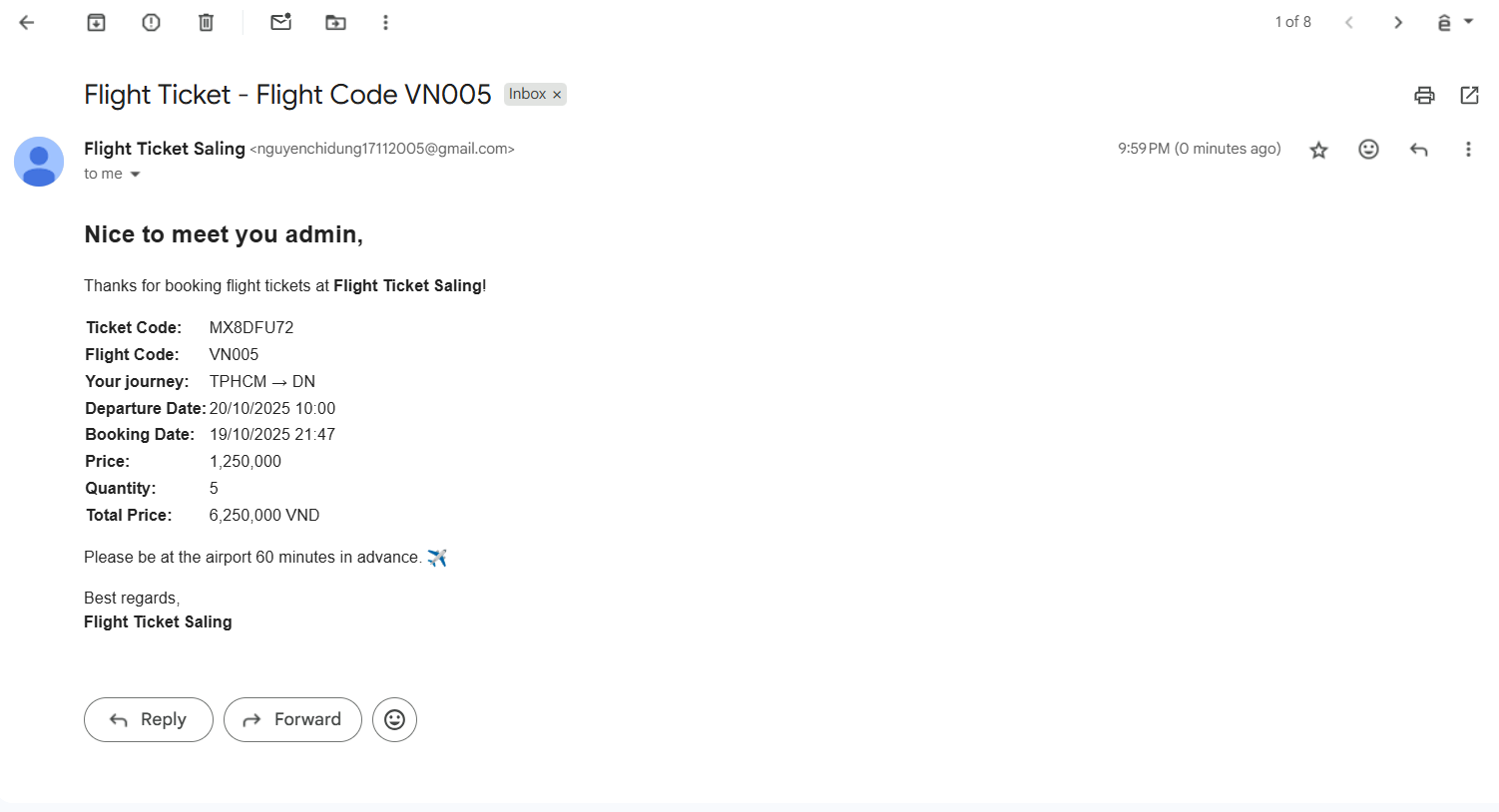
- export to PDF: user click on ticket in datagridview then click on “Export to PDF”:



The default name of file is Ticket\_TicketID.pdf. This is the pdf file.



- send ticket Via Email: user click on ticket in datagridview, input their email into textbox then click on “Send Ticket Via Email”, this is the sample email:



# **CHAPTER 3. CONCLUSION**

In conclusion, the Flight Ticket Sale System project successfully demonstrates the practical application of object-oriented programming in developing a structured and functional software solution. By separating the program into different layers — Data Access Layer (DAL), Business Logic Layer (BLL), and User Interface (UI) — we ensured that the system is modular, easy to maintain, and expandable in the future.

* The system effectively implements major features such as:
* Managing flight information (add, delete, update).
* Booking and canceling flight tickets with automatic seat adjustment.
* Exporting tickets to PDF files.
* Sending tickets via email to customers.

Through this project, we gained valuable insights into software design, database management, and user interface development. We also learned the importance of debugging, error handling, and teamwork in completing a software project successfully.

Although the system currently meets the basic requirements, there is room for improvement. Future development could include features such as online payment integration, user authentication with role-based access control, and improved UI design for a more intuitive user experience.

Overall, this project has provided us with a strong foundation in object-oriented programming and real-world software development, which will be beneficial for our future studies and professional careers.