

TRABAJO PRÁCTICO N° 3:

El gestor de pasajeros simula ser una aplicación para los empleados de una aerolínea la cual gestiona la carga , baja y modificación de los pasajeros. Los que estén a cargo del manejo de la app podrán cargar pasajeros en los 4 vuelos que tiene a disposición la empresa y podrá modificar los datos del pasajero en caso de que haya alguna equivocación o eliminarla en caso de que cancele su pasaje.

1- Ventana de inicio:



La ventana de inicio sirve de bienvenida a la aplicación , en donde se podrá hacer uso de las 4 opciones que tiene incorporadas. Internamente funciona con carga de archivos donde se cargaran los datos que hayan sido guardados previamente.

```
1 referencia
public Inicio()
{
    InitializeComponent();

    try
    {
        List<Avion> aviones;
        if (!new Json<List<Avion>>().Read(@"\Vuelos.json", out aviones))
        {
            throw new Exception();
        }
        else
        {
            Vuelos.vuelos = aviones;
        }
    }
    catch (Exception ex)
    {
        new Text().Save("logError.txt", LogErrors.LogError(ex, "form Inicio"));
        MessageBox.Show("Error fatal.Por favor comunicarse con el área de sistemas", "ATENCIÓN!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

2- Ventana de carga:

Al hacer click en “Cargar pasajero” se abre un nuevo formulario con la información de los aviones: Su destino y los asientos disponibles de cada avión:



Cargar pasajero

Modificar pasajero

Borrar pasajero

Gestionar tickets

Salir

Vuelo: 1
Destino: Cordoba
Asientos: 1

Reservar lugar

Vuelo: 2
Destino: Aeroparque
Asientos: 3

Reservar lugar

Vuelo: 3
Destino: Mendoza
Asientos: 1

Reservar lugar

Vuelo: 4
Destino: Aeroparque
Asientos: Completo

Reservar lugar

Al hacer click en uno de los botones de “Reservar lugar” se accede a la parte baja de la ventana donde se mostrará mediante botones los lugares disponibles. De no tener lugares disponibles o particularmente el número de asiento buscado está ocupado el botón queda deshabilitado.



Cargar pasajero
Modificar pasajero
Borrar pasajero
Gestionar tickets
Salir

Vuelo: 1
Destino: Cordoba
Asientos: 1

Vuelo: 2
Destino: Aeroparque
Asientos: 3

Vuelo: 3
Destino: Mendoza
Asientos: 1

Vuelo: 4
Destino: Aeroparque
Asientos: Completo

Reservar lugar

Reservar lugar

Reservar lugar

Reservar lugar

Asiento 1


Asiento 2

Asiento 3

Asiento 4

Atrás

Luego una vez clickeado el asiento disponible se accede a la última instancia del formulario donde se cargan los datos del pasajero y luego de cargarlo , un mensaje anuncia la carga del mismo.



Cargar pasajero
Buscar pasajero
Modificar pasajero
Borrar pasajero
Salir

Vuelo: 1
Destino: Cordoba
Asientos: 1

Vuelo: 2
Destino: Aeroparque
Asientos: 3

Vuelo: 3
Destino: Mendoza
Asientos: 1

Vuelo: 4
Destino: Aeroparque
Asientos: Completo

Reservar lugar

Reservar lugar

Asiento 3

Asiento 4

Atrás

Prueba

Pdf

90678543

Cargar pasajero


Carga correcta !

i
Pasajero Prueba cargado correctamente !
Ticket:Numero de vuelo: 1
Destino: Cordoba
Pasajero: Prueba Pdf
DNI: 90678543
N° Asiento: 4

Aceptar

3- Ventana modificar pasajero:

En esta ventana al igual que en la ventana de carga primero se muestran todos los vuelos con su información respectiva. Luego una vez que elegimos en qué avión queremos hacer la modificación mediante el botón que se encuentra abajo del listBox , se accede al datagrid que muestra los pasajeros a bordo. Para poder modificar los pasajeros se tiene que hacer doble click en cualquier lugar de la fila, este evento permite acceder a la última instancia de la ventana que es donde se modifican los datos del pasajero.



Cargar pasajero

Modificar pasajero

Borrar pasajero

Gestionar tickets

Salir

Vuelo: 1
Destino: Cordoba
Asientos: Completo

Vuelo: 2
Destino: Aeroparque
Asientos: 3

Vuelo: 3
Destino: Mendoza
Asientos: 1

Vuelo: 4
Destino: Aeroparque
Asientos: Completo

Modificar vuelo

Modificar vuelo

Modificar vuelo

Modificar vuelo

Nombre	Apellido	Dni	IdAsiento
Prueba	Pdf	78945612	4
Takeshi	Kovacs	12345678	3
Milagros	Patane	38860480	2
Denis	Roldan	38797877	1

Atrás

38797877

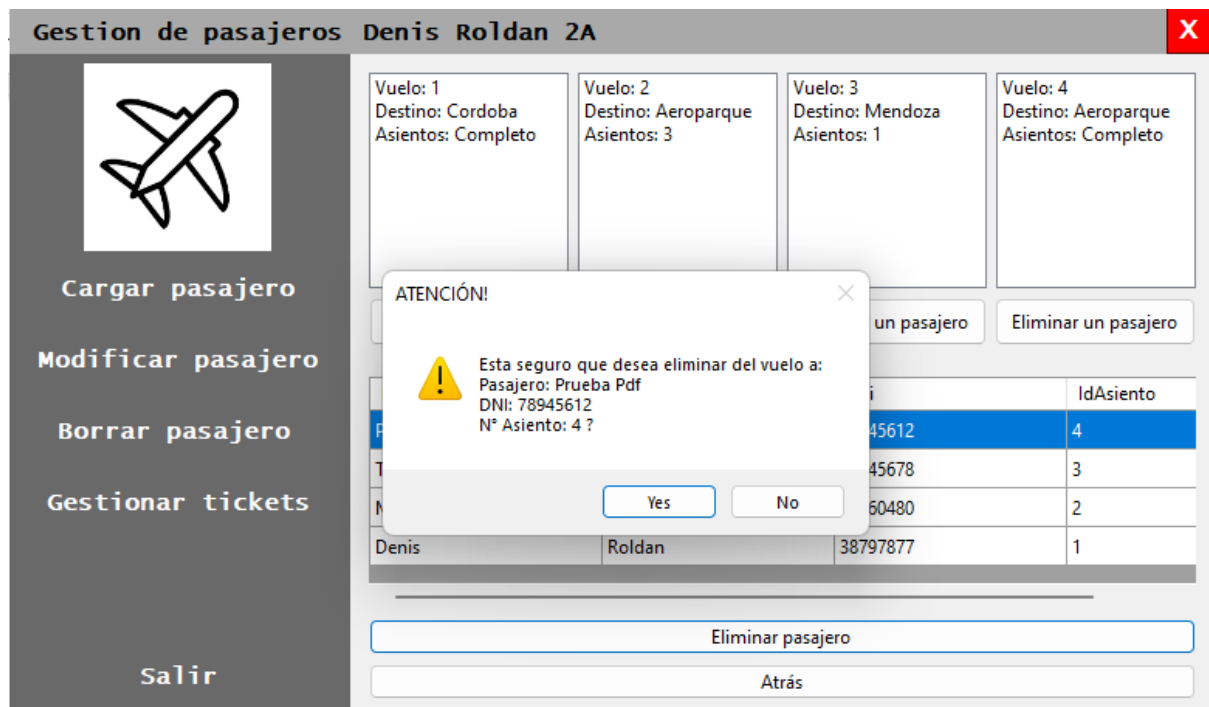
Denis

Roldan

Aceptar cambios

4- Ventana eliminar pasajero:

Misma mecánica que modifica al pasajero. En esta ventana se debe hacer un solo click sobre la persona a eliminar y luego clicar el botón Eliminar pasajero, saldrá un mensaje preguntando si se quiere hacer y dependiendo la respuesta se elimina o no.



5- Consideraciones:

Por último se listara lo requerido en el trabajo práctico:

1 - Archivos y tipos genéricos: La gestión de archivos esta dada por uno de tipo json con su clase correspondiente:

```
namespace Archivos
{
    7 referencias
    public class Json<T> : IArchivos<T>
    {
        string folder;

        6 referencias
        public Json()
        {
            folder = string.Format("{0}{1}", AppDomain.CurrentDomain.BaseDirectory, @"VuelosDB");
        }

        6 referencias
        public bool Save(string file, T data)
        {
            try
            {
                if (!Directory.Exists(folder))
                {
                    Directory.CreateDirectory(folder);
                }

                file = folder + file;

                string jsonObject = JsonSerializer.Serialize(data);
                File.WriteAllText(file, jsonObject);
                return true;
            }
            catch (Exception ex)
            {
                throw new Exception($"No se pudo guardar el archivo: {file}", ex);
            }
        }

        2 referencias
        public bool Read(string file, out T data)
        {
            try
            {
                if (!Directory.Exists(folder))
                {
                    Directory.CreateDirectory(folder);
                }

                file = folder + file;
                string jsonObject = File.ReadAllText(file);

                data = JsonSerializer.Deserialize<T>(jsonObject);

                return true;
            }
            catch (Exception ex)
            {
                throw new Exception($"No se pudo leer el archivo: {file}", ex);
            }
        }
    }
}
```

Esta clase también implementa una interface de lectura y guardado en json , y en archivo de texto que es donde se guardan los logs con los errores.

```

namespace Archivos
{
    public class Text : IArchivos<string>
    {
        string folder;

        public Text()
        {
            folder = string.Format("{0}{1}", AppDomain.CurrentDomain.BaseDirectory, @"\Texto");
        }

        public bool Save(string file, string data)
        {
            try
            {
                if (!Directory.Exists(folder))
                {
                    Directory.CreateDirectory(folder);
                }

                file = Path.Combine(folder, file);
                using (StreamWriter sw = new StreamWriter(file, true))
                {
                    sw.Write(data);
                    return true;
                }
            }
            catch (Exception)
            {
                throw new Exception($"Problemas al guardar el archivo: {file}");
            }
        }

        public bool Read(string file, out string data)
        {
            try
            {
                if (!Directory.Exists(folder))
                {
                    Directory.CreateDirectory(folder);
                }

                file = Path.Combine(folder, file);
                using (StreamReader sr = new StreamReader(file))
                {
                    data = sr.ReadToEnd();
                    return true;
                }
            }
            catch (Exception)
            {
                throw new Exception($"Problemas para leer el archivo: {file}");
            }
        }
    }
}

```

Y finalmente la interface con la implementación de los tipos genéricos:

```

6
7 namespace Archivos
8 {
9     2 referencias
10    public interface IArchivos<T>
11    {
12        18 referencias
13        bool Save(string file, T data);
14
15        3 referencias
16        bool Read(string file, out T data);
17    }
18 }

```

2 - Test: Se testean una serie de métodos de la clase Avión:

```
8
9 namespace Test
10 {
11     [TestClass]
12     0 referencias
13     public class AvionTest
14     {
15         [TestMethod]
16         0 referencias
17         public void ChequearPasajeroRepetido_CuandoElPasajeroExiste_DeberiaRetornarTrue()
18         {
19             //Arrange - preparar el caso de prueba
20             List<Avion> lista = HardcodearAviones();
21             Pasajero prueba = new Pasajero("prueba", "unitaria", 37089123, 2);
22             bool resultadoEsperado = true;
23
24             //Act - Invocar el método a probar
25             bool resultado = Avion.ChequearPasajeroRepetido(lista, prueba);
26
27             //Assert - Verifico que el resultado sea el esperado.
28             Assert.AreEqual(resultadoEsperado, resultado);
29         }
30
31         [TestMethod]
32         0 referencias
33         public void ChequearPasajeroRepetido_CuandoElPasajeroNoExiste_DeberiaRetornarFalse()
34         {
35             List<Avion> lista = HardcodearAviones();
36             Pasajero prueba = new Pasajero("prueba", "unitaria", 37797877, 2);
37             bool resultadoEsperado = false;
38
39             //Act - Invocar el método a probar
40             bool resultado = Avion.ChequearPasajeroRepetido(lista, prueba);
41
42             //Assert - Verifico que el resultado sea el esperado.
43             Assert.AreEqual(resultadoEsperado, resultado);
44         }
45
46         [TestMethod]
47         0 referencias
48         public void ObtenerPasajeros_CuandoElIdEsDistinto_DeberiaRetornarFalse()
49         {
50             List<Pasajero> listaP = new List<Pasajero>();
51             List<Avion> lista = HardcodearAviones();
52             bool resultadoEsperado = false;
53
54             bool resultado = Avion.ObtenerPasajeros(3, lista[1], out _);
55             Assert.AreEqual(resultadoEsperado, resultado);
56         }
57
58         [TestMethod]
59         0 referencias
60         public void ObtenerPasajeros_CuandoElIdEsIgual_DeberiaRetornarTrue()
61         {
62             List<Pasajero> listaP = new List<Pasajero>();
63             List<Avion> lista = HardcodearAviones();
64             bool resultadoEsperado = true;
65
66             bool resultado = Avion.ObtenerPasajeros(2, lista[1], out _);
67             Assert.AreEqual(resultadoEsperado, resultado);
68         }
69     }
70 }
```


3 - Excepciones: En gran parte del código de los formularios se utilizan manejo de excepciones , notese que se usan también excepciones propias:

```
1 referencia
private void ModificarPasajeroConAvionLleno()
{
    if (string.IsNullOrEmpty(txtBox_Nombre.Text) || string.IsNullOrEmpty(txtBox_Apellido.Text) || string.IsNullOrEmpty(txtBox_DNI.Text))
    {
        throw new FormatException("Debe completar todos los campos del formulario");
    }
    else if (!Regex.IsMatch(txtBox_Nombre.Text, @"^[a-zA-Z]+$") || !Regex.IsMatch(txtBox_Apellido.Text, @"^[a-zA-Z]+$") || !long.TryParse(txtBox_DNI.Text, out _))
    {
        throw new FormatException("Hay errores en los campos del formulario");
    }
    else if ((txtBox_DNI.Text.Length < 7) || (txtBox_DNI.Text.Length > 8))
    {
        throw new FormatException("Ingreso un Dni con la cantidad de numeros inválida");
    }
    else if (txtBox_Nombre.Text == pasajeroOriginal.Nombre && txtBox_Apellido.Text == pasajeroOriginal.Apellido)
    {
        throw new FormatException("Para realizar una modificación se debe realizar al menos un cambio en el pasajero (nombre o apellido)");
    }
    else
    {
    }
}
```