PRÁCTICA VUELOS
EN
WINDOWS FORMS

DISEÑO DE INTERFACES

2º DAM

ANDREA CENICEROS

ÍNDICE

Introducción 3

Vuelos 3.0 4

Vuelos 3.1 7

Vuelos 3.2 8

Vuelos 3.3 9

Vuelos 3.4 11

Vuelos 4.0 12

Conclusiones 17

Fuentes 18

Introducción

En este documento se recogen todas las respuestas a las preguntas formuladas en los ejercicios anteriores además de a las de la práctica actual.

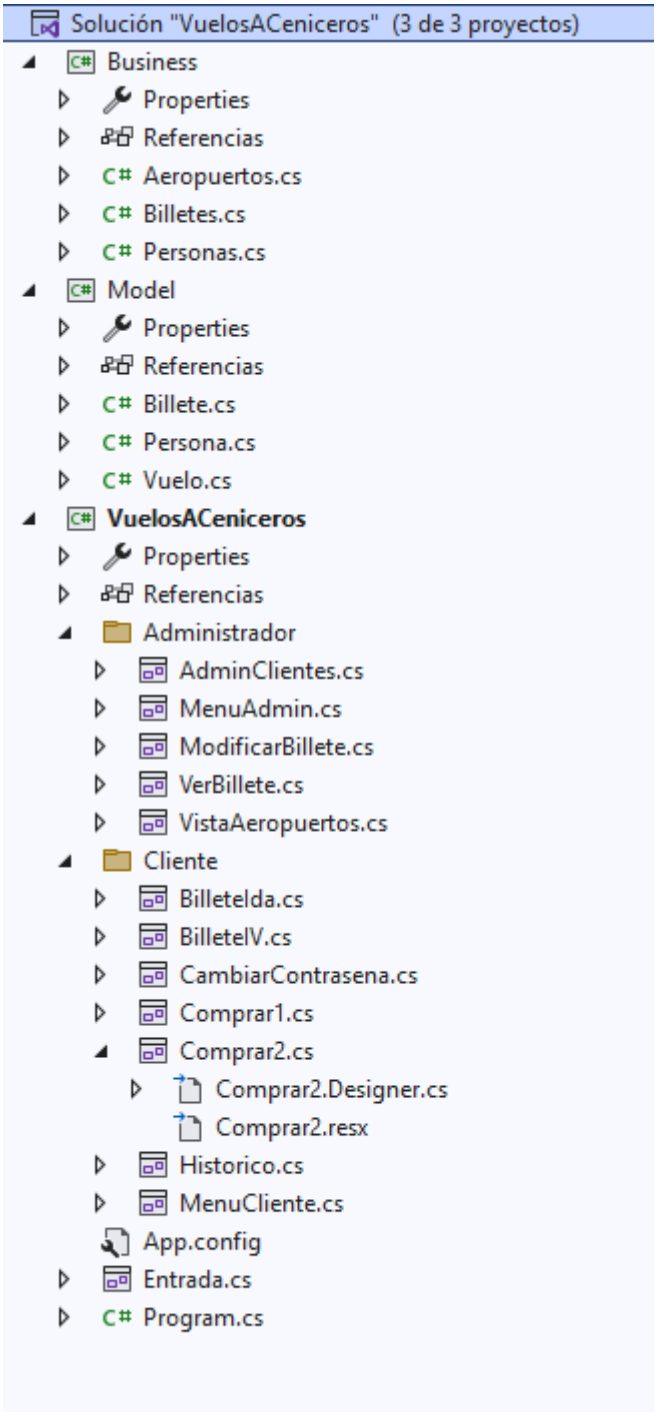
También se han incorporado comentarios en el propio código que ayuden a su comprensión y al programador que en un futuro tenga que corregirlo / actualizarlo.

La imagen de la derecha muestra la estructura completa de la aplicación, que está dividida en diferentes proyectos / capas con la interfaz, la lógica, y los objetos (Modelo / *Model* - Vista / *VuelosACeniceros* - Controlador/ Business).

Lo primero que se carga al acceder a la aplicación de *VuelosACeniceros* es un formulario de Acceso que solicita un usuario y contraseña.

Hay dos tipos de usuarios diferentes, que tienen acceso a funcionalidades diferentes, uno como Administrador del sistema y el otro, como Cliente.

Aquí se facilitan las claves para poder acceder a la demo del programa:



ACCESO (Usuario - contraseña)

- Administrador: admin - admin
- Usuarios:

11111111A - 1111 / 22222222B - 2222 / 33333333C - 3333 / 44444444D - 4444 /55555555E - 5555

Vuelos 3.0

PREGUNTA 7

No nos interesa que permanezca abierto el formulario de logueo por ello debemos cerrarlo.
- this.Close();
¿Pero qué ocurre? ¿Se nos cierra la aplicación? Explica que ha sucedido en este punto y como lo has solucionado.

Para que no se cierre la aplicación, en *Program*, he quitado cualquier formulario que apareciese en *Application.Run()*, y he añadido el siguiente código antes que ese:

```
Entrada entrada = new Entrada();
entrada.Show();
Application.Run();
```

Luego he creado el método *MainForm_Closed* y lo he asociado al form principal, y cuando se cierre, si quedan formularios abiertos se asociará con el propio código con el formulario siguiente que esté abierto (el de la posición 0, cualquiera que sea).

```
private static void MainForm_Closed(object sender, FormClosedEventArgs e)
{
    ((Form)sender).FormClosed -= MainForm_Closed; //Elimina la subscripción porque ya está
    if (Application.OpenForms.Count == 0) // Si no hay ningún form abierto entra
    {
        Application.ExitThread(); // Cierra la aplicación
    }
    else
    {
        Application.OpenForms[0].FormClosed += MainForm_Closed; // Vuelve añadir la subscripción
al evento para que cierre
    }
}
```

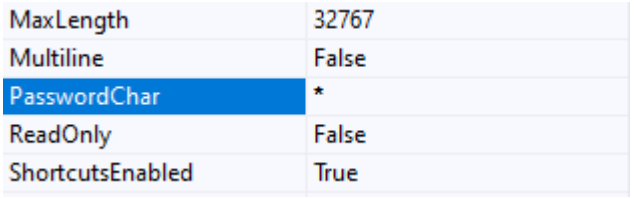
El main de Program quedará así:

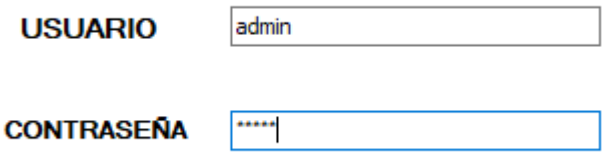
```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Entrada entrada = new Entrada();
    entrada.Show();
    entrada.FormClosed += MainForm_Closed;
    Application.Run();
}
```

PREGUNTA 8

A continuación, haz que la caja de texto en la cual metemos la contraseña, no muestre los caracteres, si no asteriscos. (Mira a ver en propiedades del *textBox*). ¿Cómo lo has solucionado? Explica qué has hecho.

Los *textBox* tienen una propiedad que se llama *PasswordChar*, en la que puedes indicar que caracter quieres que se muestre en lugar de la contraseña. En este caso he indicado (*) para que al escribir se muestre ese carácter por cada keypress.





PREGUNTA 9

Enlaza a los menús todos los formularios generados hasta el momento a través de sus botones. Los botones que se han hecho y aun no tenemos capacidad para trabajarlos se quedarán sin uso. ¿Nos interesa que el menú se quede abierto? Si se queda abierto ¿Nos interesa que se quede bloqueado o no? Razona la respuesta y explícame cómo has realizado una opción u otra bajo tu criterio.

Al tener los menús de *Administrador* y *Cliente* una opción de *Salir*, lo interesante sería dejarlos abiertos y hasta que no se seleccione ese botón no se cierre. Por ello yo recomendaría dejarlos abiertos pero bloquearlos hasta que se haya terminado con los formularios a los que se accede desde cada uno de ellos, y así no poder hacer click en salir teniendo acciones sin finalizar / formularios sin cerrar.

Por eso en mi proyecto los he abierto con *.ShowDialog* para que *MenuAdmin* y *MenuCliente* se queden bloqueados hasta que se cierren los secundarios.

PREGUNTA 10

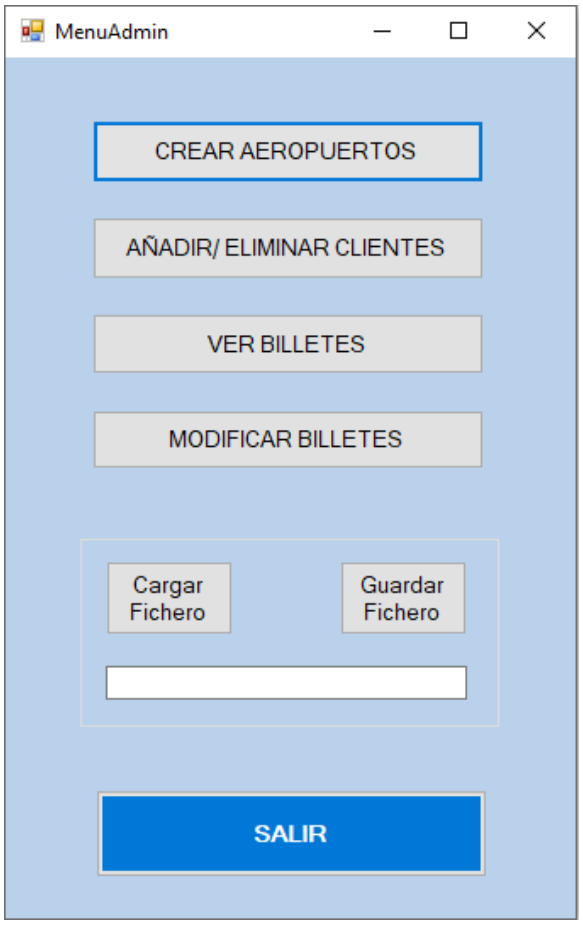
¿Por qué los formularios de cliente no es necesario referenciarlos a través del nombre de su carpeta, pero en cambio los creados con el administrador si necesitas el nombre de la carpeta? Consigue que todos trabajen en el mismo espacio de nombres, es decir, sin necesidad de referenciar a la carpeta que lo contienen. Razona la respuesta.

En mi caso no he tenido que poner el nombre de la carpeta al inicializar los formularios, porque los he creado fuera y los he arrastrado a cada carpeta (En *Administrador* el tercer formulario *AdminClientes* para administrar clientes y en *Cliente* los formularios *Comprar1* y *Comprar2* para la compra de billetes).

En el namespace, los formularios que he creado directamente dentro de la carpeta tienen el nombre del proyecto*.Carpeta* y los que he creado fuera no. Para poder usarlos sin tener que referenciar a la carpeta se puede añadir el *using namespace.Carpeta.*

Ej: En mi formulario de Entrada:

using VuelosACeniceros.Administrador
using VuelosACeniceros.Cliente



Vuelos 3.1

PREGUNTA 2

No vamos a tener una conexión a la BBDD pueda alimentar nuestra aplicación detrás, deberemos crear un método elementos los objetos creados de personas y billetes para poder hacerlas pruebas. ¿Qué forma has elegido para crear dicho método? ¿Por qué?

Teniendo en cuenta que los formularios "hijos" tienen acceso más sencillo a la información del "padre", creo que la mejor opción es declarar estáticas las clases contenedoras en el primer formulario (*Entrada*) y rellenar en ellas las listas. Así se pueden usar en el resto de formularios sin tener que pasarlas como parámetro de entrada, solo llamando *Entrada.billetes* o *Entrada.personas.*

PREGUNTA 4

Mi vuelo posee un botón y una caja de texto, con ese botón lo que haremos es buscar en la lista de billetes algún billete que posea un número identificativo, idéntico al que hayamos introducido en la caja de texto, y nos mostrará la información en una ventana de diálogo. ¿Tenemos número identificativo de billetes? ¿Cómo lo solucionas? ¿Qué solución se te ocurre para que los genere la aplicación?

Hasta ahora no teníamos identificativo, pero se puede añadir un atributo id al billete, que se autoincremente cada vez que se crea un nuevo billete y que no pueda repetirse. El contador debe ser *static* para que coja siempre el último valor utilizado y asegurarse de ello. Se autoincrementa en los constructores y ahí le da valor al atributo id.

PREGUNTA 5

La siguiente funcionalidad será la de históricos billetes de cada cliente, solo debes tener en cuenta que únicamente nos mostrará el histórico de los billetes del cliente logueado y no del resto. Muestra la información en un diálogo. ¿Necesitas algún método nuevo que no se haya contemplado en el diagrama de clases que te ha sido pasado en este documento? ¿Por qué?

Ya hay un método en la clase contenedora Billetes que busca billetes por DNI, por lo que se podría llamar a ese método para que el cliente vea el histórico de sus billetes sin necesidad de crear uno nuevo.

* Para el tema contraseña, al principio cree un atributo llamado password que cuando se añade a la lista por defecto es su dni, pero que cuando se loguea, si su contraseña es igual al dni, se abra un nuevo formulario para que la modifique (se puede probar con la primera persona, las demás tienen una contraseña diferente). En el método Rellenar de las clases contenedoras, ya añade contraseñas distintas para que no haya que cambiarlas.

Vuelos 3.2

PREGUNTA 1

Lo primero que debemos conseguir es que cuando carguemos este form nos muestre el listado completo de nuestra clase Personas. Como vimos en la práctica Vuelos 2.2 utilizamos para mostrar una tabla con la información de personas el elemento "DataGridView" ¿Cómo podemos mostrar la información que contiene nuestra clase Personas en el DatGriedView?

Podemos mostrar la información de nuestra clase Personas en el *DataGridView* creando un BindingSource, indicándole que el tipo de dato que va a coger es Persona, pasándole el Binding como fuente al *DataGrid* y pasando la lista de personas que tenemos como fuente de datos a ese Binding.

PREGUNTA 2

Cuando añado un cliente, lo que estoy haciendo realmente es construir una persona y con un objeto persona construido… ¿Cómo hago para añadirlo a mi clase Personas y también en mí DataGriedView? (Una vez que presionamos el botón añadir, ya debe estar añadida la persona a la lista y no esperar a cerrar el formulario).

En el método del botón añadir recojo los datos de los *textBox* y los uso para crear una nueva persona con parámetros. Esa persona se la paso a la lista que he asociado previamente al *Binding* y actualizo el *DataGridView* para que se cargue en la lista nuevamente, sin necesidad de cerrar el formulario.

PREGUNTA 3

A continuación, crearemos un método en la clase billetes que nos borre los billetes que tenga un cliente en concreto a través de su DNI.
- public bool Eliminar_billete_dni(string dni)
Cuando llamamos a este método ¿ocurre algún posible error o excepción? ¿Por qué?

El método public bool *Buscar_dni_billete(string dni)* no necesito crearlo porque tengo el de *Buscar_dni* que me devuelve la lista de billetes asociados a ese dni. Si la lista no es null, salta el mensaje de aviso, si lo es, elimina directamente.

Por otro lado, al eliminar personas no me ha saltado ningún error ni excepción, pero en el método cuando borraba el billete tenía un break, por lo que al eliminar el primero asociado a esa persona ya no borraba los demás.

Utilizando el método dado, me aseguro de que borra todos los asociados a ese DNI. (Lo compruebo añadiendo de nuevo otra persona con el mismo DNI).

Vuelos 3.3

PREGUNTA 1

Lo que vamos a intentar ahora, es mostrar únicamente los campos que creamos convenientes. No olvidemos que debemos seguir cumpliendo todos los requisitos previos. Mira a ver qué te parece esta sentencia:

this.dataGriedClientes.Columns["Nombre"].Visible = false;

¿Qué has hecho para evitar mostrar todos los campos?

En mi caso no quería que se viese la columna de la *Password* ni la del boolean *nuevaContrasena*, y lo que he hecho ha sido borrar el Get en la clase Person de este segundo caso, para que no se pueda mostrar pero he mantenido el Set para que se pueda modificar. Y con la Password, he hecho invisible la columna en el código del método del *DataGridView*:

dataClientes.Columns[4].Visible = false;

PREGUNTA 2

La contraseña del campo pass que tiene como atributo la clase persona o algún atributo parecido que lo que nos permita es generar la contraseña. La intentaremos realizar de forma aleatoria sin la necesidad de tener que meterla nosotros. Tendremos la capacidad de generarla nosotros o de decidir que sea realizada de forma aleatoria. Para ello tenemos que tener presente la clase random:

```
public string generador_pass()
{
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
var stringChars = new char[8];
var random = new Random();
for (int i = 0; i < stringChars.Length; i++)
{
stringChars[i] = chars[random.Next(chars.Length)];
}
return (new String(stringChars));
}
```

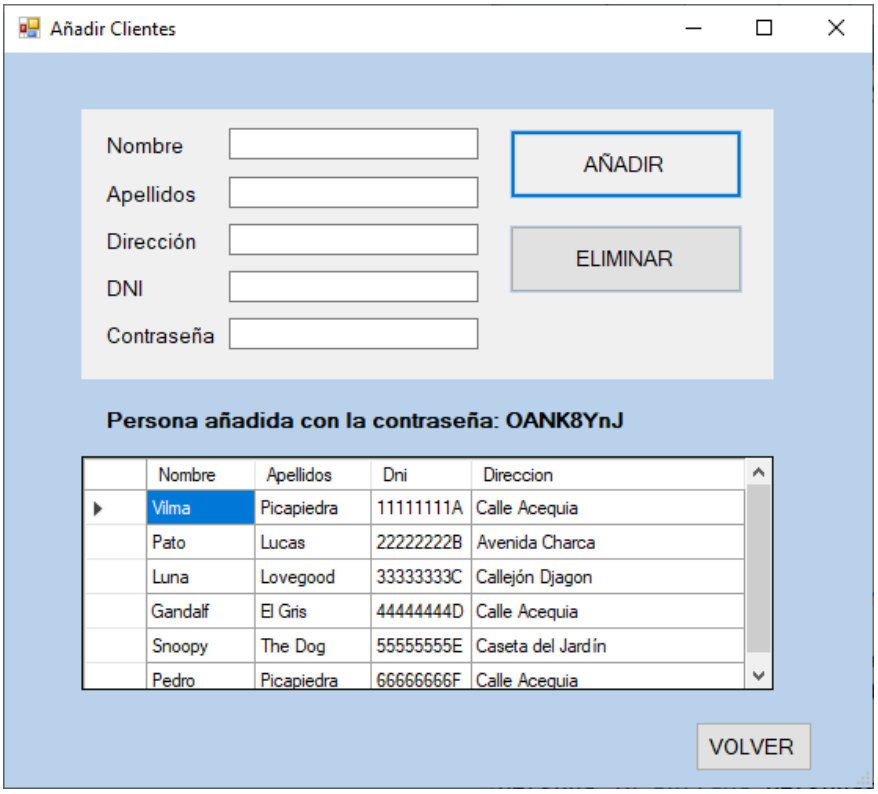
¿Cómo funciona este método? Explícalo de la forma más detallada que puedas. Y además modifícalo para que genere el número de caracteres que nosotros deseemos pasado como parámetro de entrada a la función.

Lo que hace es llenar aleatoriamente un string con la longitud que se le ha pasado anteriormente, en este caso [8], utilizando el carácter que ocupa la posición random en el string de referencia. El del ejemplo contiene el abecedario de letras mayúsculas y minúsculas y los números del 0 al 9.

La modificación para poder pasarle el parámetro sería así:

```
public string generador_pass(int totalCaract)
{
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
var stringChars = new char[totalCaract];
var random = new Random();
for (int i = 0; i < stringChars.Length; i++)
{
stringChars[i] = chars[random.Next(chars.Length)];
}
return (new String(stringChars));
```

En realidad no hay nada aleatorio de verdad en programación, utiliza el tiempo en el que se le hace la petición, por lo que hay que tener cuidado al generar muchas contraseñas seguidas, pues el resultado podría ser el mismo.



* Para actualizar mi método de cambiar contraseña, he creado un nuevo atributo *nuevaContrasena,* para verificar si es la primera vez que se accede al sistema. De ser así, se abre una ventana que obliga a cambiar de contraseña por seguridad del usuario.

Vuelos 3.4

PREGUNTA 1

Para crear aeropuerto lo que deseamos es que nos aparezca un listado con los aeropuertos actuales y podremos modificarlos, es decir, añadir o eliminar destinos y orígenes donde trabaja nuestra empresa. Describe brevemente cómo lo has hecho. Ten en cuenta que esta lista de aeropuertos alimentará la compra de billetes de los clientes, los cuales deben poder elegir los nuevos destinos y no les deben aparecer los destinos eliminados.

He creado una clase contenedora Aeropuertos que tiene una lista de Strings que serán tanto orígenes como destinos, con constructor lleno y vacío, y he creado los métodos de *AnadirAeropuerto, EliminarAeropuerto* y *Rellenar*, para pasarle que por defecto tenga los 4 aeropuertos con los que ya contábamos.

¿Qué cambios has realizado en la compra de billetes de los clientes?

En el apartado de *Cliente*, *Comprar1*, le paso la lista estática de Entrada con los aeropuertos como fuente para los *combobox*, así si se esa lista se actualiza, tanto origen como destino se actualizan a su vez y muestran los elementos añadidos, y no los eliminados.

PREGUNTA 2

A través del botón modificar billetes, podremos cambiar los datos de los billetes ya creados. Para ello podremos realizarlo de dos maneras diferentes.

a. Introduciendo un Id del billete que nos permita modificar los campos
b. Aparecerá un listado en el cual podremos seleccionar el billete que queramos modificar

Comenta a continuación qué clases y complejidad has encontrado para llevar a buen término el punto 2.

Lo primero ha sido seleccionar si la opción *a* o *b*. Me parecía más sencilla de usar la a, que traiga solo los datos del id que corresponda, y que se muestren en elementos editables como *textBox* etc.

Al final al usar un *NumericUpDown* he conseguido combinar las dos cosas, ya que va cargando automáticamente los billetes que se pueden modificar y avisa de los que no. El botón *Modificar* llama al método del mismo nombre que he creado en la clase *Billetes*, y que recibe por parámetros los nuevos vuelos y la nueva persona que creo para guardar los datos de la modificación.

Tengo también un método en *Billetes* al que llamo en el *Load* del formulario, y devuelvo la lista con fechas posteriores a la actual, por lo que si el id no está en la lista, el admin recibirá un mensaje de que el billete solicitado no está disponible para modificar.

Vuelos 4

PREGUNTA 2

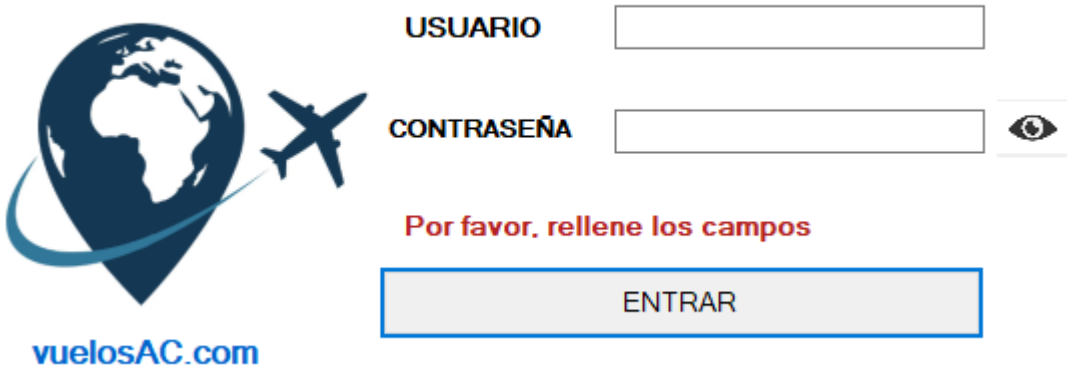
Ahora mismo céntrate únicamente en la usabilidad y en la interacción con el usuario final, tanto el cliente como el administrador. Esto es simple, juega con tu aplicación y mejora la usabilidad y la experiencia de usuario dando información donde creas que es necesario, mejorando interacción de botones, o lo que consideres oportuno. A continuación, detalla esos cambios o mejoras de forma escrita y explicando el porqué de las mismas.

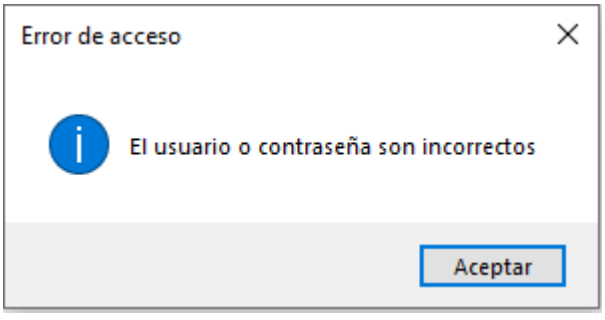
Usabilidad

Entre las mejoras que he implementado, he sustituido la mayoría de avisos emergentes por mensajes en el *form* y bloqueo de los botones de acción en caso de que, por ejemplo, falten datos, o se haga alguna acción que no necesite ser confirmada dos veces.

En los iconos de los messegeBox he diferenciado al administrador del usuario: al usuario le aviso todo por icon infomation, ya que es más amigable, y al administrador si le paso avisos de exclamación o error.

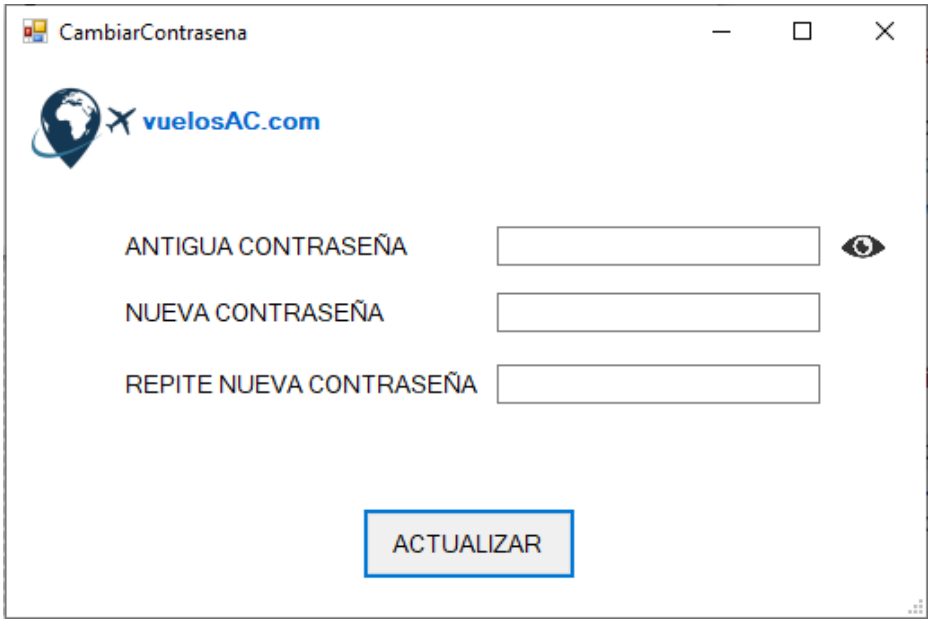
He utilizado el keypress para impedir que los textBox tengan un formato incorrecto que luego no se pudiese parsear para crear los objetos.



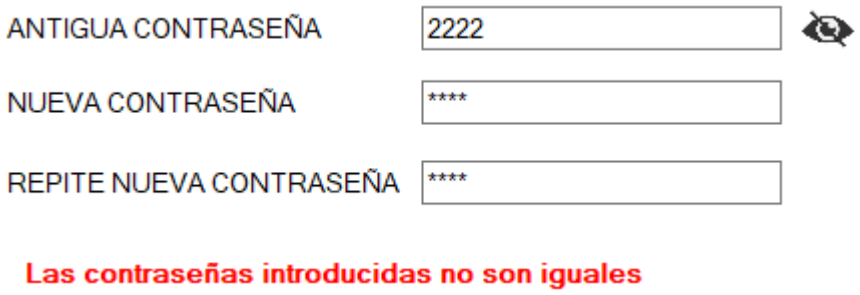


Seguridad

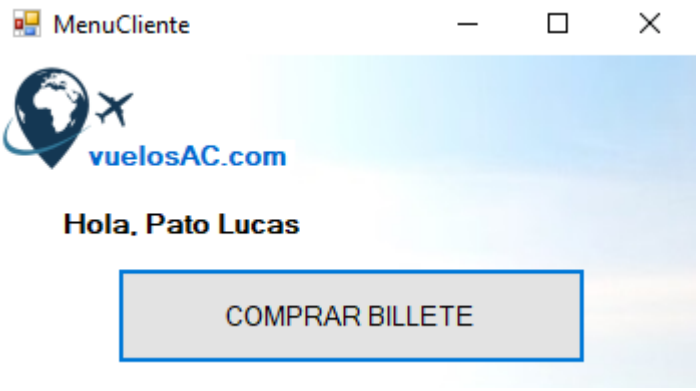
La primera vez que se entra como usuario, se le pide cambiar la contraseña para que el usuario se asegure de que solo él la conoce.



Con el símbolo del ojo, se puede activar o desactivar la visibilidad de la contraseña antigua, y en el caso de la nueva, de no ser iguales, salta un aviso.

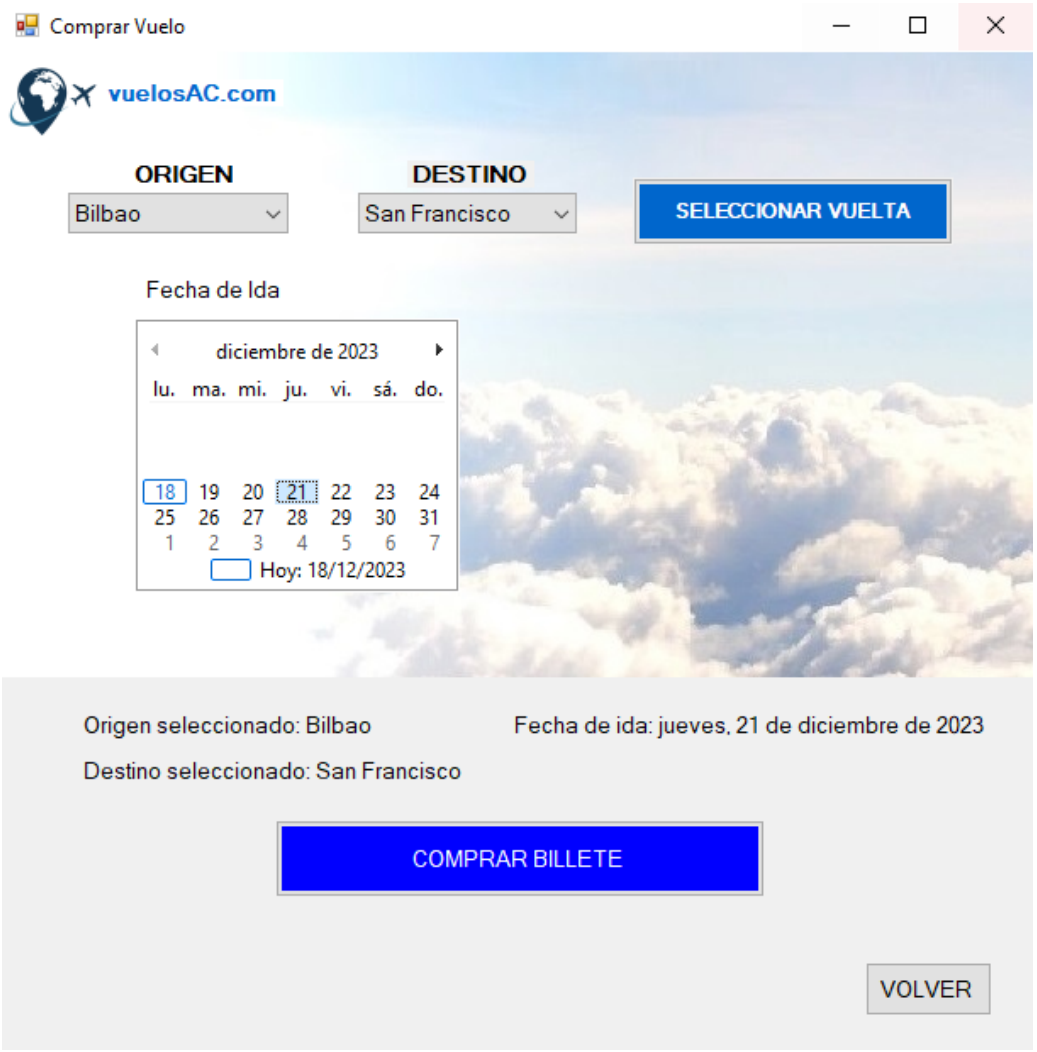


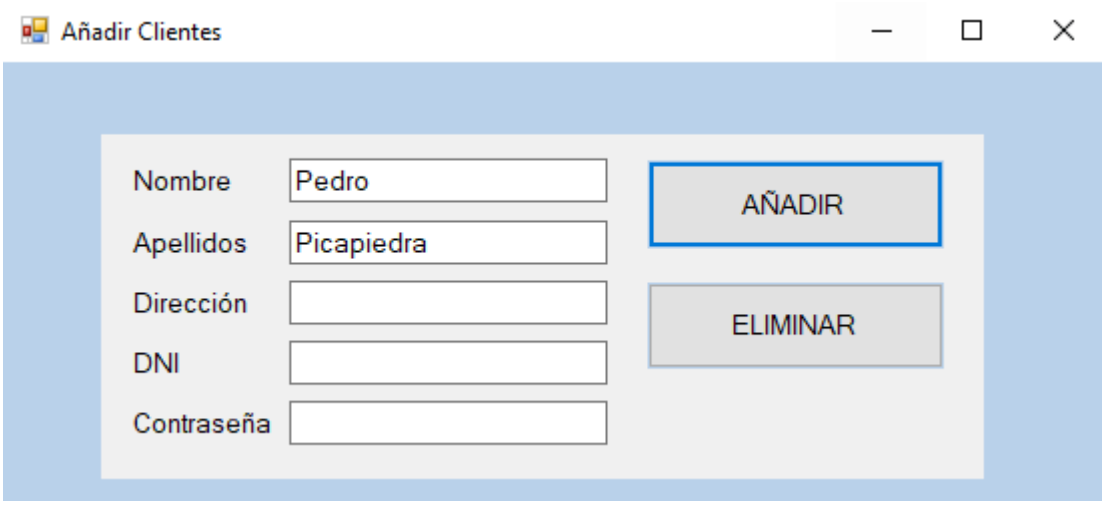
Además, cuando el usuario accede, recibe un saludo de bienvenida con su nombre y apellidos.



Accesibilidad

Para mejorar la visibilidad de los elementos y la información, he ampliado los tamaños de letra que vienen por defecto, los de los botones y los contrastes de colores de fondo, así como marcar en negrita algunos elementos a destacar como los títulos o los avisos.





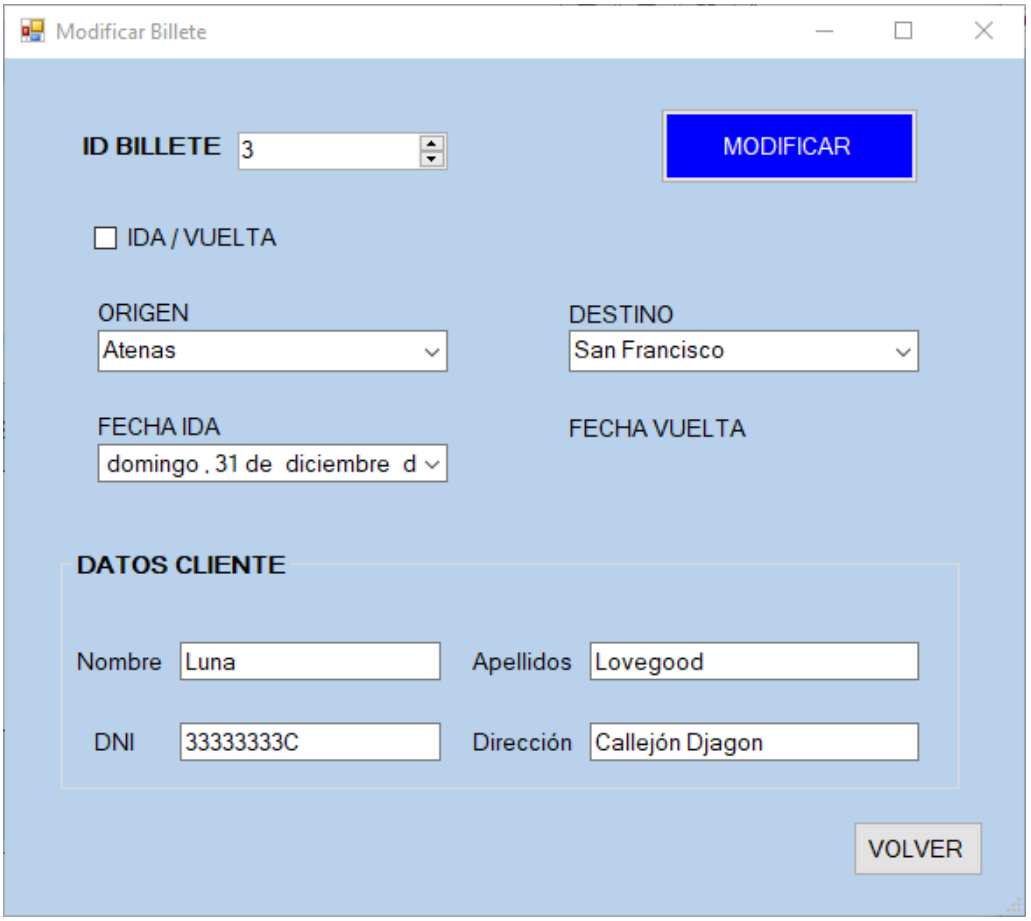
Diseño

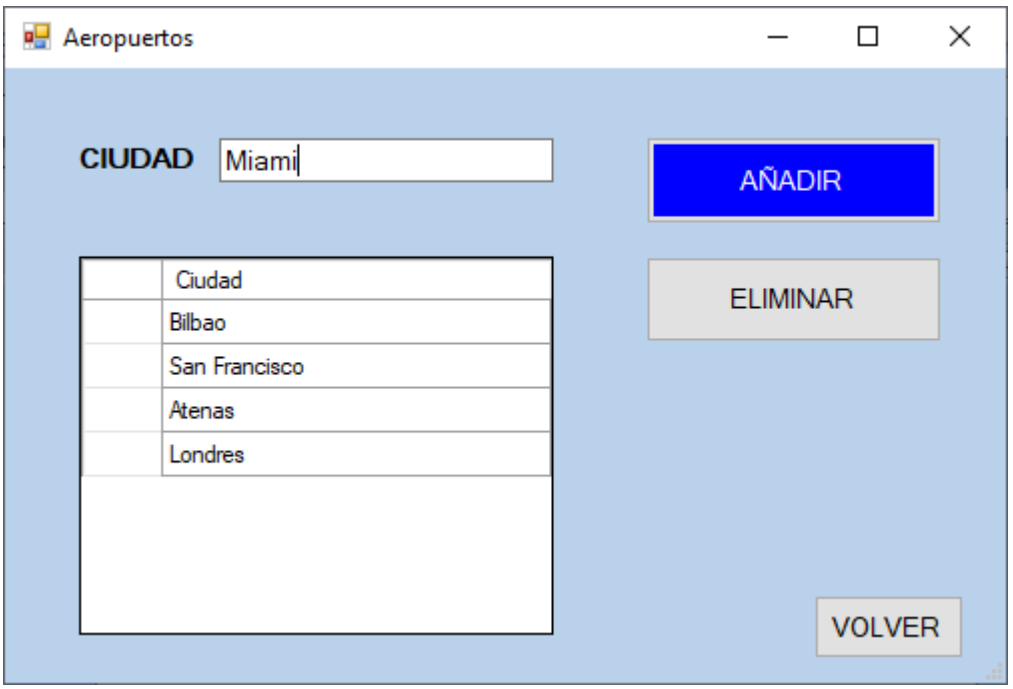
El color elegido para la marca y los elementos de la aplicación ha sido el azul, relacionado con el cielo, la tranquilidad, la confianza y la seguridad, elementos que se quiere se vinculen a una compañía aérea.

En los formularios a los que acceden los clientes, se ha incorporado el logotipo para mantener la presencia de la marca.

Además se han incorporado imágenes de nubes para ambientar la compra, y se han añadido elementos visuales a la hora de mostrar los datos, como el formato de billete para ver los datos en lugar de mostrarlos en un *messegeBox* con texto únicamente.

En el caso de las ventanas que utiliza el administrador, se ha optado por un diseño más austero, ya que lo van a utilizar los trabajadores de la empresa, manteniendo el color azul pero eliminado la marca y primando la sencillez y el rápido uso.





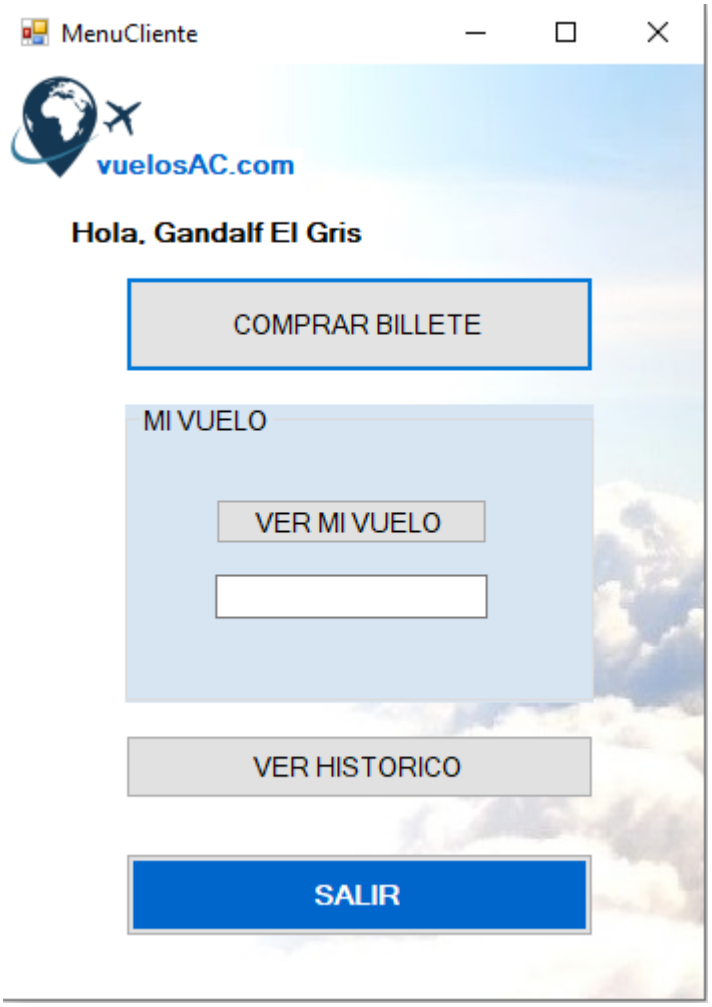
Conclusiones

En este programa se han aplicado los elementos que brinda Windows Forms, de manera que a los dos tipos de usuario (Administrador y Cliente) les resulte intuitiva, educativa, accesible y les permita realizar las operaciones que necesiten lo más fácil posible.

La intención es que el uso de esta herramienta no genere insatisfacción y que cumpla con la finalidad para la que está creada, que en este caso es la venta y administración de billetes de avión.

Así mismo, está programada de forma escalable, para que, a futuro, se le puedan seguir añadiendo nuevas funcionalidades a medida que evolucione el negocio.

Igualmente, se aceptan y agradecen sugerencias para la mejora de la aplicación, también aplicables a futuros desarrollos.



Fuentes

INFORMACIÓN

- Apuntes Diseño de Interfaces, 2º DAM, Salesianos los Boscos, C. Peña.

- Windows Forms

https://learn.microsoft.com/es-es/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022

- Combo Box

https://learn.microsoft.com/es-es/dotnet/desktop/winforms/controls/combobox-control-overview-windows-forms?view=netframeworkdesktop-4.8

- Binding Source

https://learn.microsoft.com/es-es/dotnet/desktop/winforms/controls/bindingsource-component-overview?view=netframeworkdesktop-4.8

- DataGridView

https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.datagridview?view=windowsdesktop-8.0

- Chart

https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1

IMÁGENES

Logotipo, fondos y billete

https://www.freepik.es/

Capturas de la app