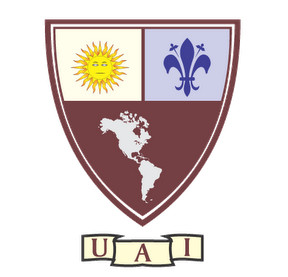
Universidad Abierta Interamericana

Facultad de Tecnología Informática

Modelos Computacionales de Gestión Administrativa



Trabajo Práctico Integrador

Docente: Milio, Roberto Claudio

Alumnos:

Dome, Martin

Rodriguez Arata, Damian

# **Índice**

[**Índice**](#_alqssrsm948s) **1**

[**Descripción del Negocio**](#_7unuhad2i483) **2**

[**Casos de Uso**](#_pmibjvi16u8) **3**

[2.1. LOGIN](#_yl8hfdjwfsy) 4

[2.1.1 Ciclo de vida](#_yyfx66mqpppp) 4

[Secuencia de navegación:](#_tx5ik6wps67s) 5

[2.1.2 Código](#_inzkb4hfuyzj) 5

[2.2 BACKUP & RESTORE](#_y00nw8mcdzqp) 21

[2.2.1 Backup](#_vp1mabtusp49)

[Frecuencia de BackUp](#_vp1mabtusp49) 22

[2.2.1.1 Ciclo de Vida](#_409ndmqdkdqf) 22

[Secuencia de navegación:](#_yxkvu5z97uq5) 23

[2.2.2 Restore](#_stop5emn0xvu) 28

[2.2.2.1 Ciclo de Vida](#_na99dqsutvt9) 28

[Secuencia de navegación:](#_ax128152zwbc) 29

[**3- Dígito Verificador**](#_tpb5gg7xqeva) **33**

[3.1 Ciclo de vida](#_quttzm5m2iey) 33

[Secuencia de navegación:](#_ls5np1tf9xe8) 34

[3.2 Código](#_un3e1yko14vg) 35

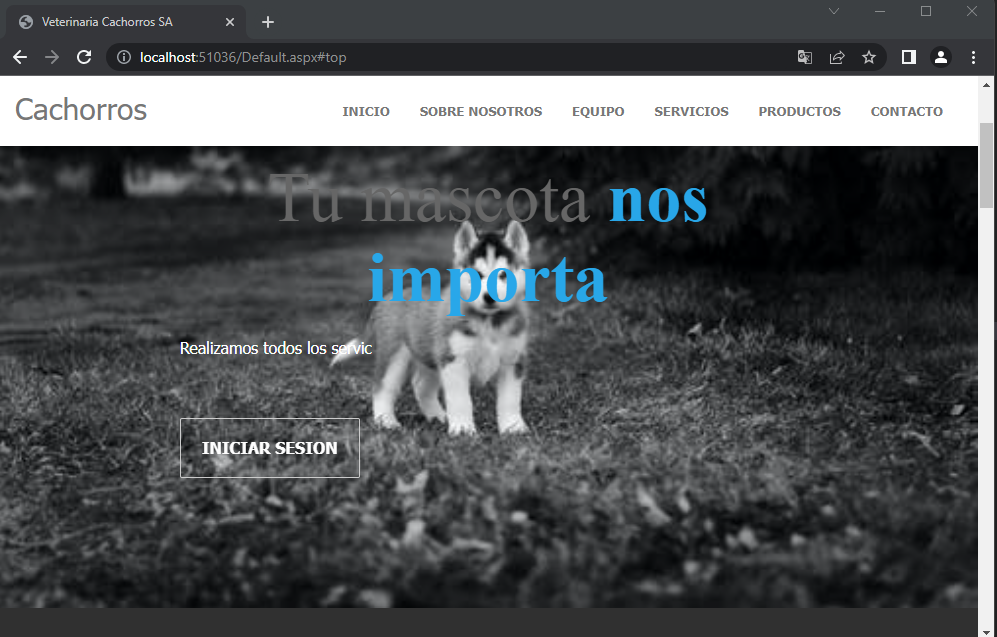
[**5.1-Validaciones de control**](#_act5r7kl0d) **47**

[6.1.2 - CONTROL DE INTEGRIDAD](#_b7lgqdtqxm5) 49

# 

# **Descripción del Negocio**

**Veterinaria “Cachorros SA”,** es una empresa familiar ubicada en Burzaco, con sucursales en Quilmes y Wilde, fundada en 1996 por el doctor Carlos A. Gomez y Juan Pablo Viña, quienes actualmente siguen brindando su atención especializada en perros y gatos. El uso de internet se realiza principalmente para expandir más su mercado, dar a conocer sus servicios y captar nuevos clientes de zonas nuevas zonas geográficas.











# **Casos de Uso**

## 

## **2.1. LOGIN**

## **2.1.1 Ciclo de vida**

### **Secuencia de navegación:**

1. El usuario selecciona opción de login en la página de inicio. (UI)
2. Se hace una redirección a la página de login, y el usuario ingresa los datos.(UI)
3. Se verifican los datos ingresados. (BLL)
4. Se procede a encriptar la password ingresada por el usuario. Se envía la consulta y devuelve los datos del usuario, si los datos son correctos. (DAL) (BE)
5. Se envía consulta para buscar las acciones del usuario. (BLL)
6. Se devuelven los datos de las acciones del usuario. (DAL) (BE)
7. Se guardan en las variables de sesión los datos del usuario. Login.aspx.cs (UI)
8. Se redirecciona a la página de Respuesta.aspx. (UI)
9. Se recuperan los datos del usuario de la variable de sesion.(UI)
10. Se envían los datos a la bitácora de la acción realizada por el usuario. (BLL)
11. Se persisten los datos de la bitácora. (DAL) (BE)
12. Se muestra al usuario los datos de tipos de acceso en la pantalla de Respuesta.aspx y si es el administrador además se muestra el botón de “Bitácora” (UI).
13. El usuario administrador presiona el botón “Mostrar Bitácora”. (UI)
14. Se envía consulta para buscar la bitácora. (BLL)
15. Se recuperan los datos de la bitácora. (DAL) (BE)
16. Se carga el objeto Bitácora que alimentara la grilla. (UI)
17. Se muestra por pantalla una grilla, con la bitácora ordenada por fecha, con un filtro para Tipo de Usuario, y con paginado para ver muchos resultados.(UI)

### **2.1.2 Código**

**1. Default.aspx (Capa UI):** El usuario selecciona opción de login en la página de inicio.

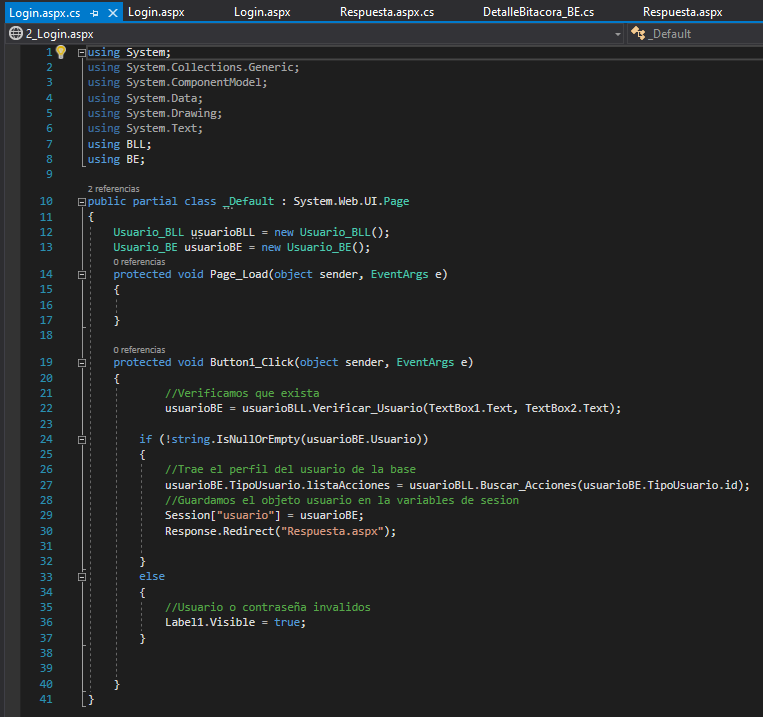


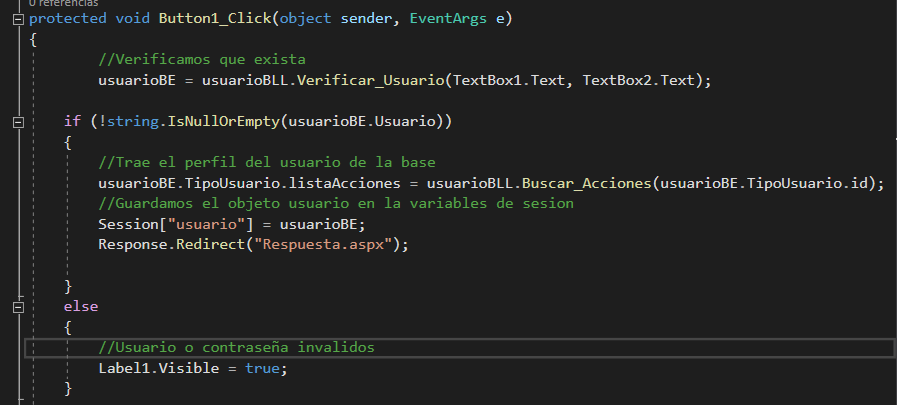
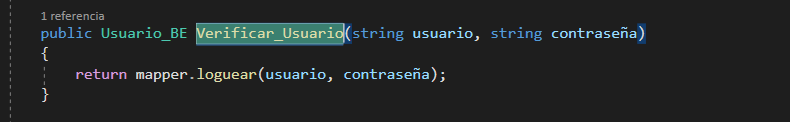
**Default.aspx**

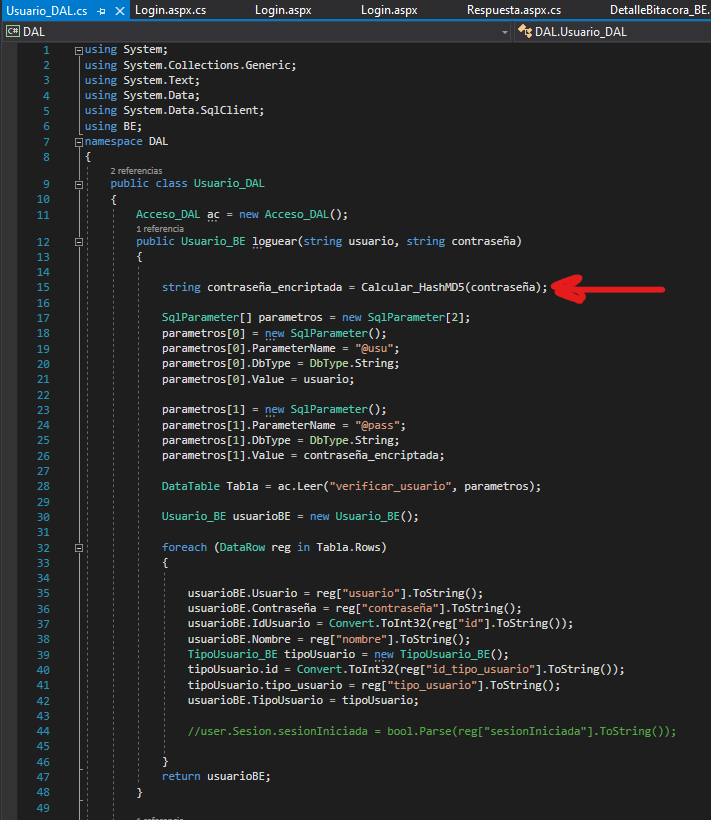


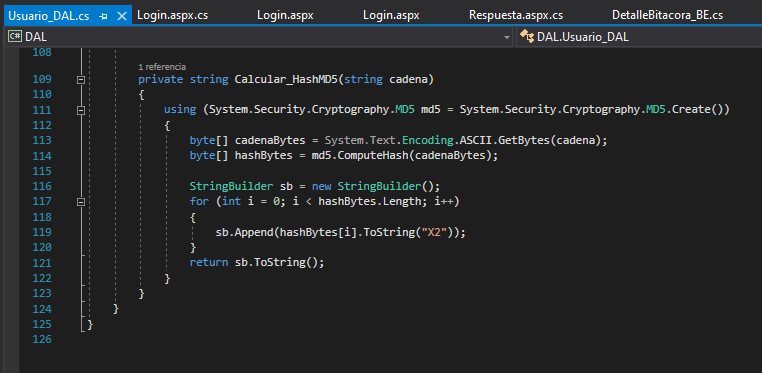
**2. Login.aspx (Capa UI):** Se hace una redirección a página de login, y el usuario ingresa los datos.

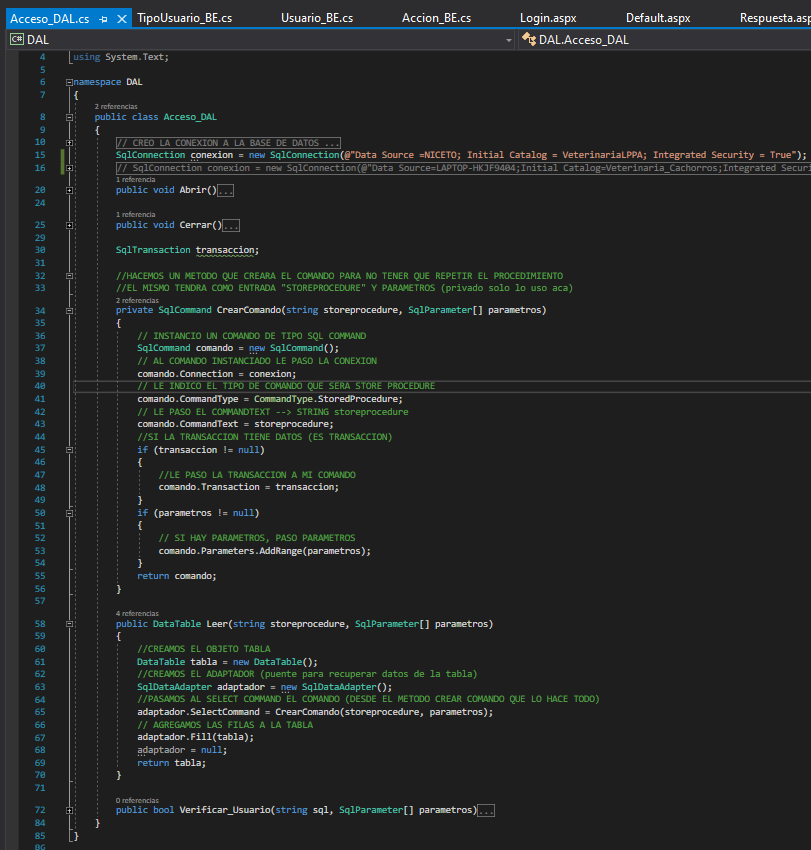


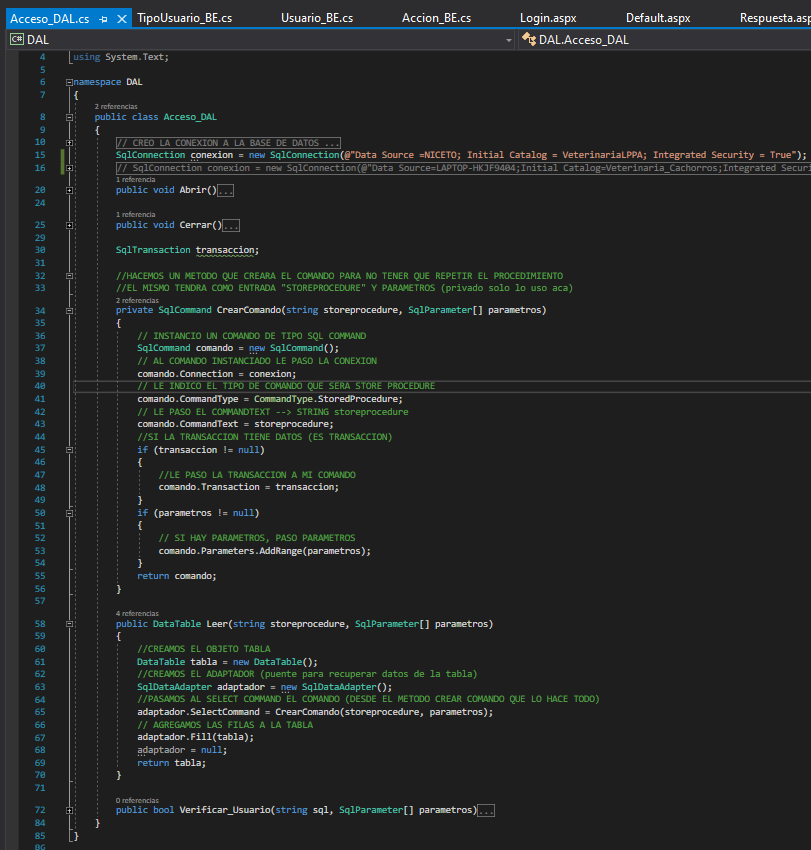
**Login.aspx.cs**

**3. Verificar\_Usuario (BLL):** Se verifican los datos ingresados.   
**4. Usuario\_DAL.cs (Capa DAL):** Se procede a encriptar la password ingresada por el usuario. Se envía la consulta y devuelve los datos del usuario, si los datos son correctos.

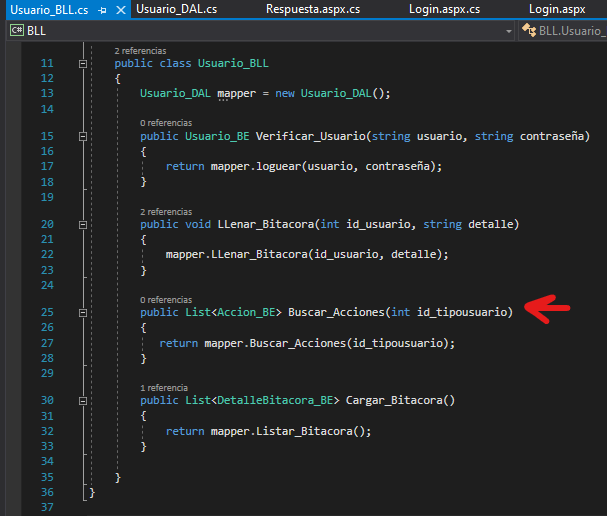




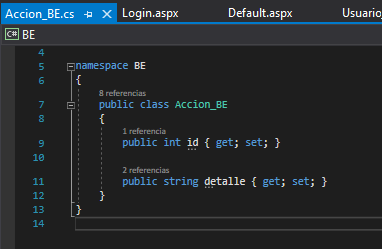
**Acceso\_DAL.cs (Capa DAL):** 



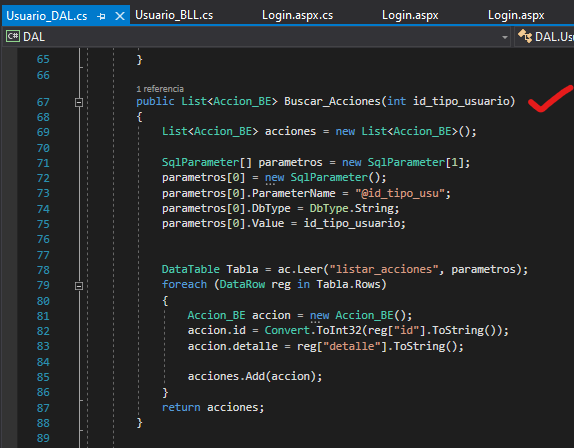
**5. Usuario\_BLL (Capa BLL):** Se envía consulta para buscar las acciones del usuario.



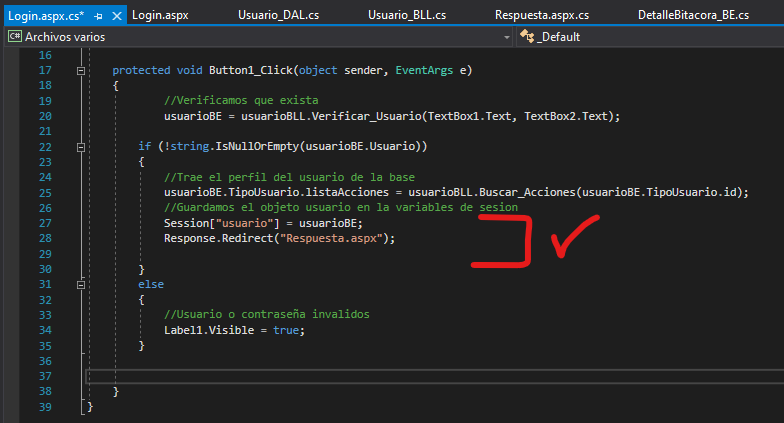
**Accion.BE.cs (Capa BE):**



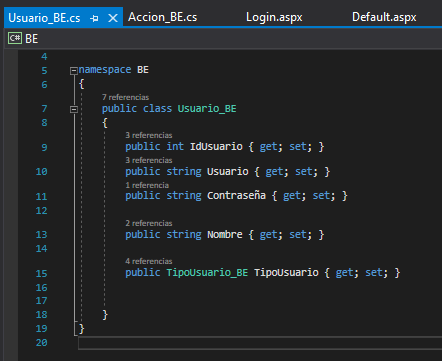
**6. Usuario\_DAL.cs (Capa DAL**): Se devuelven los datos de las acciones del usuario



**7. Loging.aspx.cs (Capa UI):** Se guardan en las variables de sesión los datos del usuario.



**Usuario\_BE.cs (Capa BE):**

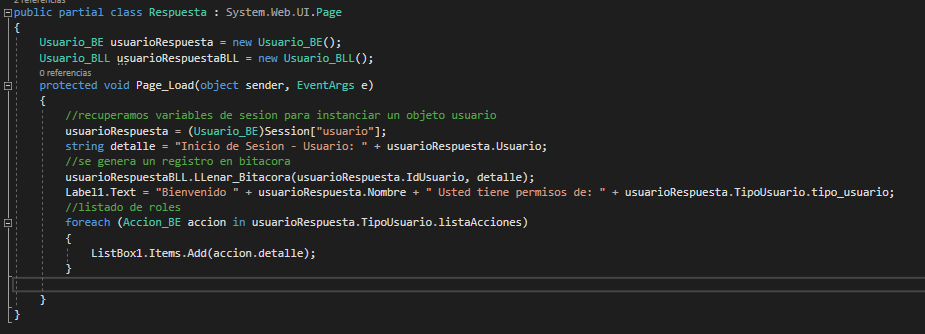


**TipoUsuario\_BE.cs (Capa BE):**

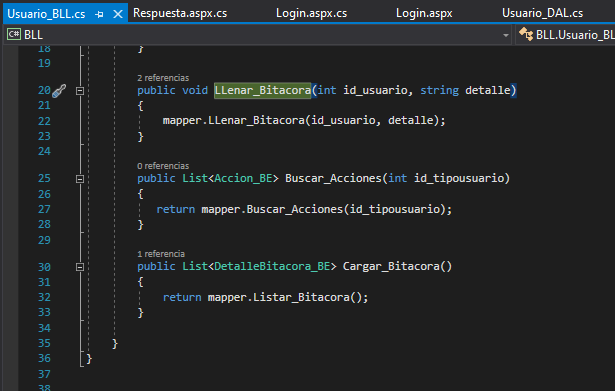


**8. Respuesta.aspx.cs (Capa UI):** Se redirecciona a la página de Respuesta.



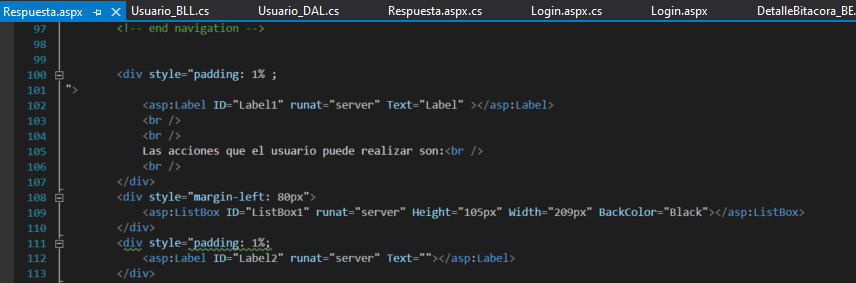
**9. Respuesta.aspx.cs (Capa UI):** se recuperan los datos del usuario de la variable de sesión.

**10. Usuario\_BLL.cs (Capa BLL):** Se envían los datos a la bitácora de la acción realizada por el usuario.

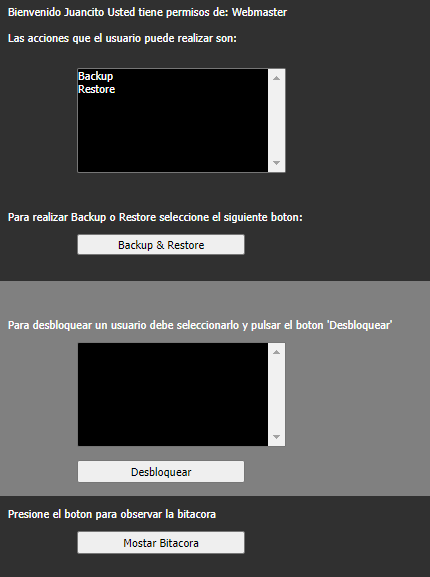


**11. Usuario\_DAL.cs(Capa DAL):** Se persisten los datos de la bitácora. ****

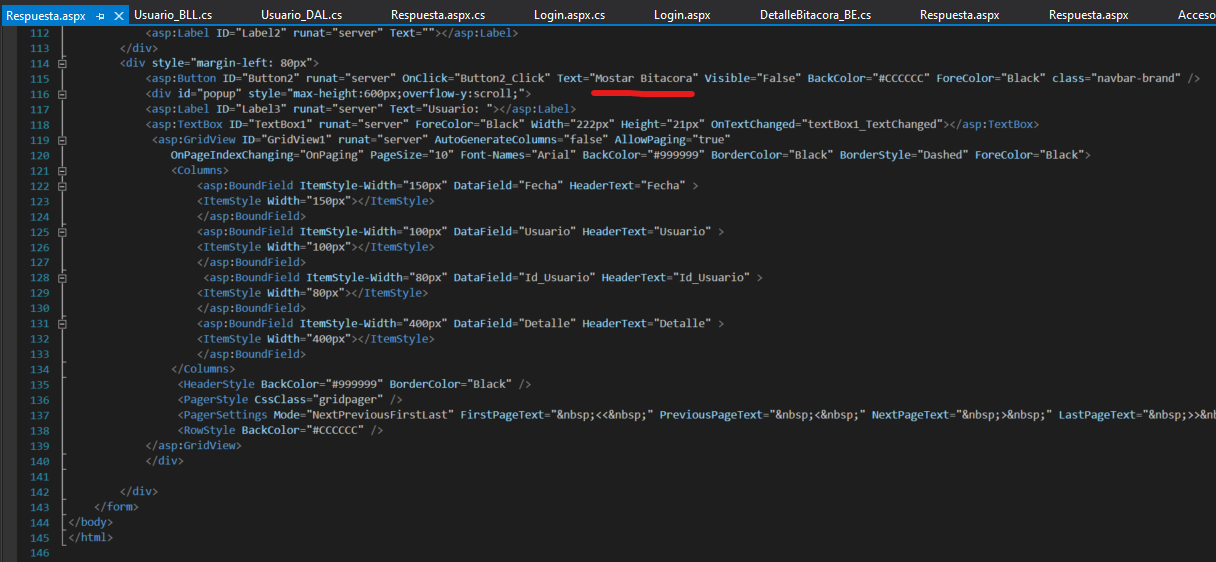
**12. Respuesta.aspx (Capa UI):** Se muestra al usuario los datos de tipos de acceso en la pantalla y si es el administrador además se muestra el botón de “Mostrar Bitácora”.



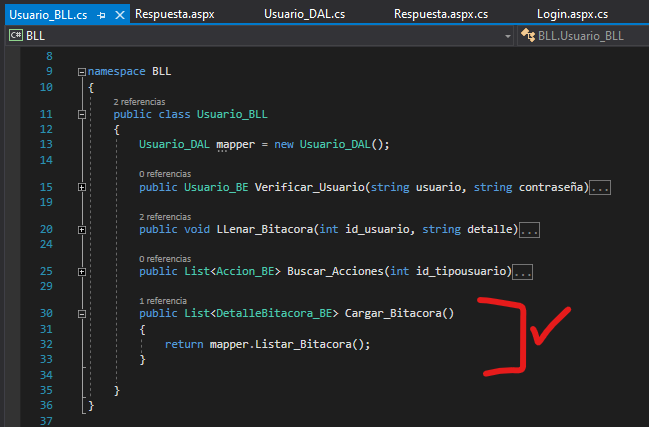
**Respuesta.aspx**

****

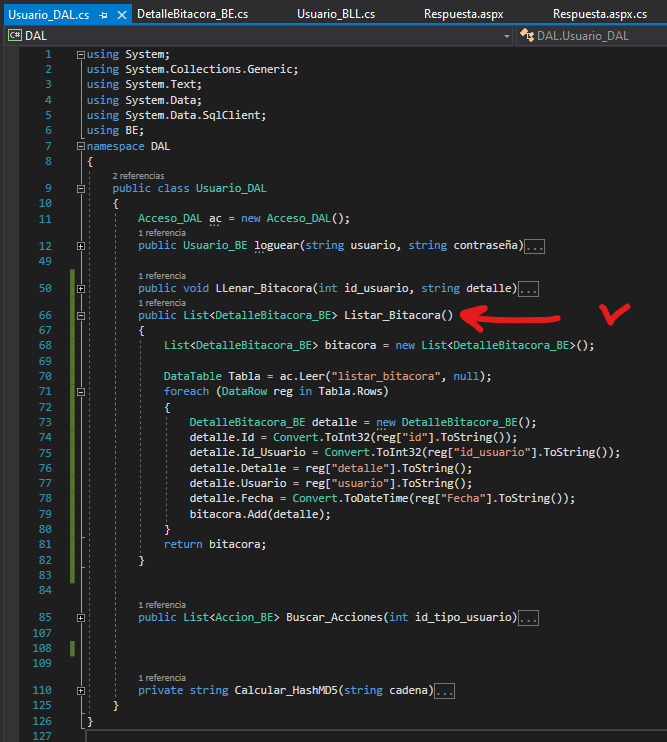
13. **Respuesta.aspx (Capa UI):** El usuario administrador presiona el botón “Mostrar Bitácora”.



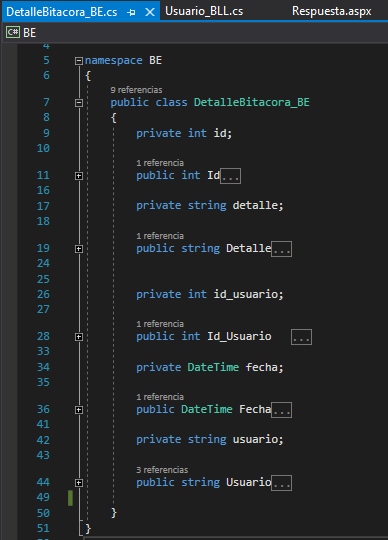
14. **Usuario\_BLL.cs (Capa BLL):** Se envía consulta para buscar la bitácora.



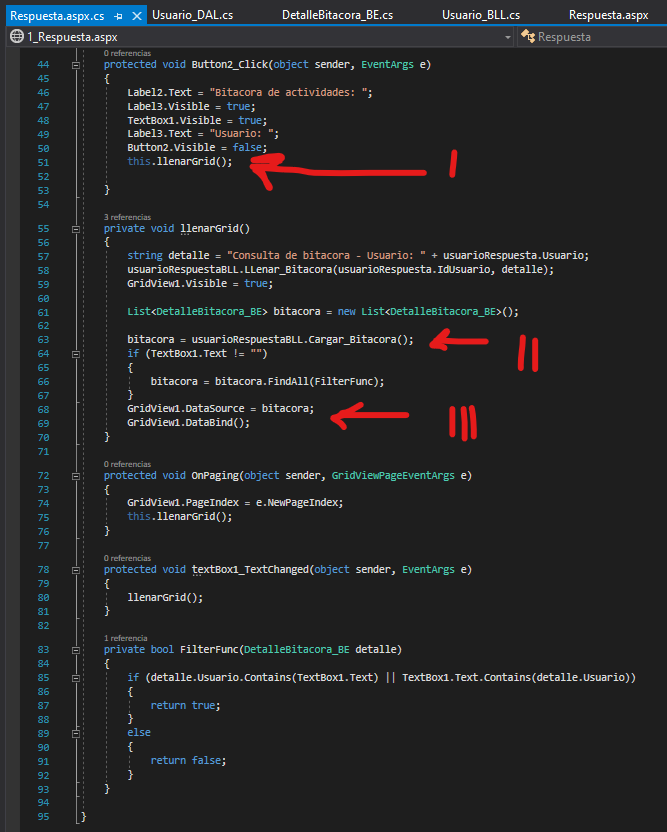
**15. Usuario\_DAL.cs (Capa DAL):** Se recuperan los datos de la bitácora.



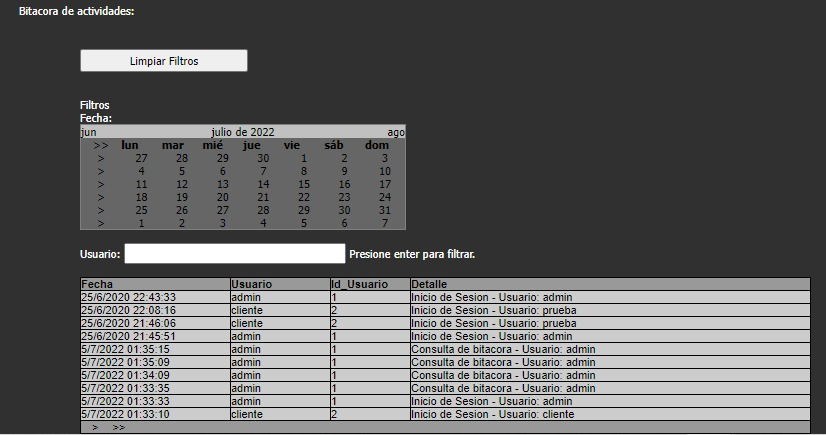
**DetalleBitacora\_BE.cs (Capa BE):**



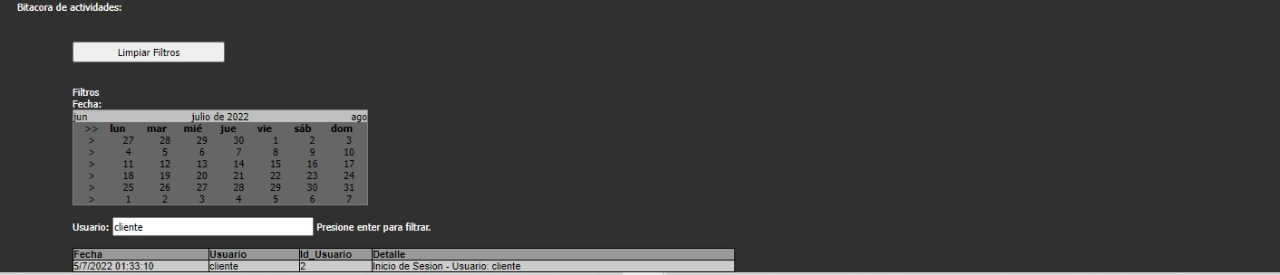
**16- Respuesta.ASPX.cs (Capa UI):** Se carga el objeto Bitácora que alimentara a la grilla.



**17- Respuesta.ASPX (Capa UI):** Se muestra por pantalla una grilla, con la bitácora ordenada por fecha, con un filtro para Tipo de Usuario, calendario para fecha, y con paginado para ver muchos resultados.



**Con un filtro por fecha y cliente:**



Se utilizó un calendario para tomar las fechas como filtro, lleva por defecta la fecha actual. Cuando para la fecha seleccionada no hayan registros la grilla no se mostrará.

## **2.2 BACKUP & RESTORE**

## **2.2.1 Backup**

## **2.2.1.1 Ciclo de Vida**

## 

**Frecuencia del Backup:**

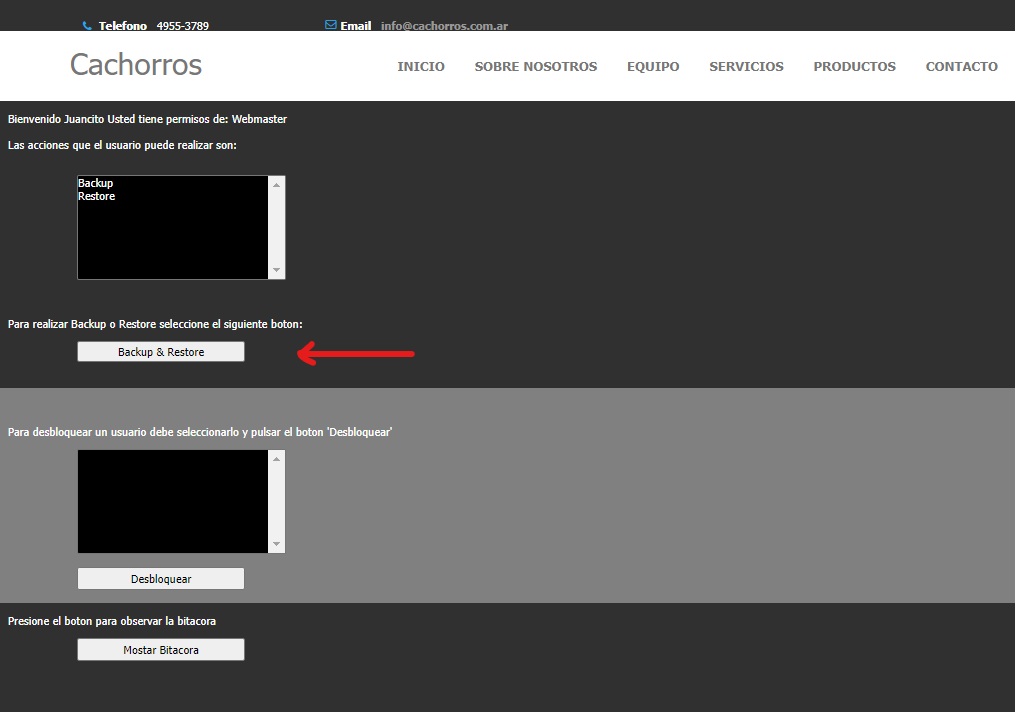
El backup lo realiza el usuario ADMIN, a demanda cada vez que lo considera necesario o lo necesite. Siendo el mismo un backup total de la base de datos.

### **Secuencia de navegación:**

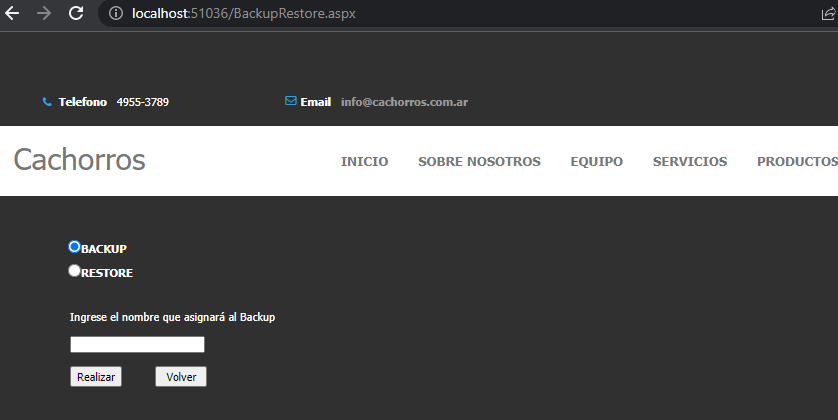
1. Como precondición se debe realizar el Login (2.1) con usuario webmaster (ADMIN).
2. Una vez realizado el Login e ingresado en la pantalla principal se presiona el botón de Backup & Restore, esto nos redireccionará a la interfaz correspondiente: BackupRestore.aspx. ( Capa UI)
3. Se debe ingresar un nombre de archivo dentro del Textbox ( Capa UI)
4. Para finalizar la acción se pulsa sobre el botón “Realizar” (Capa UI)
5. Se envían los datos a la acción correspondiente del usuario para la realización del Backup (Capa BLL)
6. Se procede a realizar el respaldo correspondiente de los datos (Capa DAL). Se retorna a la pantalla de BackupRestore.aspx

**2.2.1.2 Código**

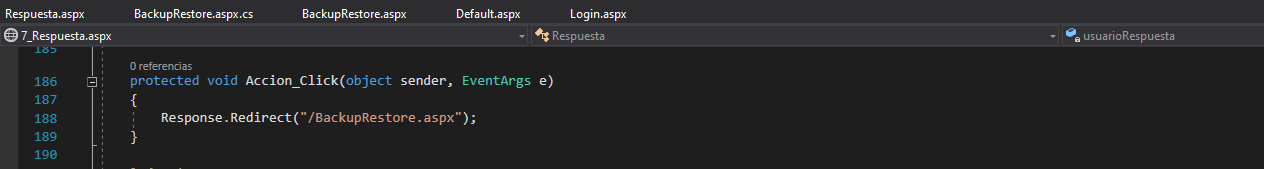
**Respuesta.aspx [Diseño]**



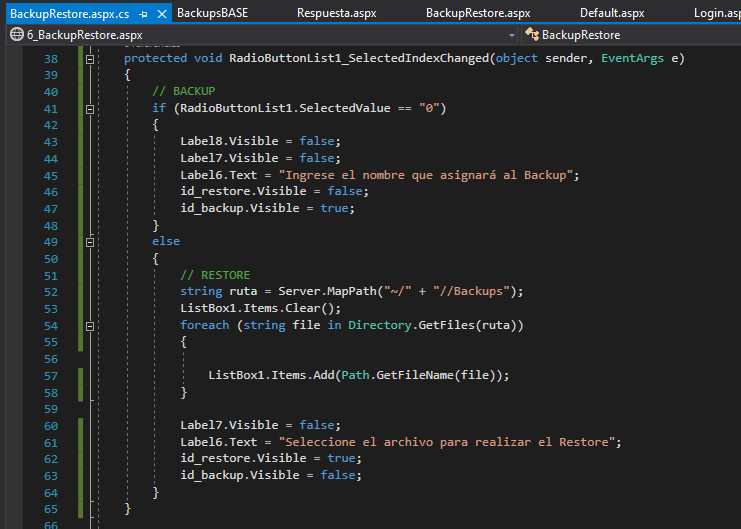
**BackupRestore.aspx [Diseño]**

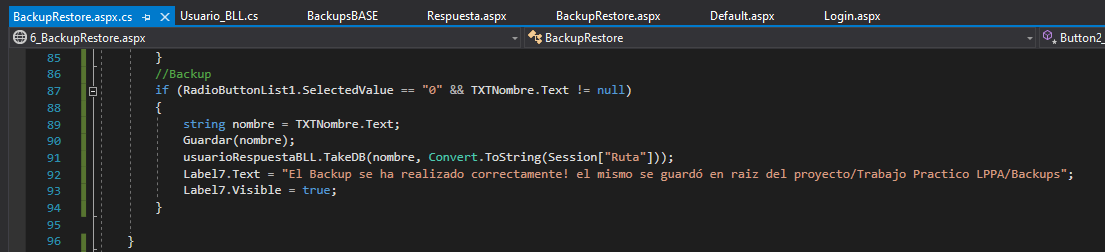


**Respuesta.aspx.cs (Capa BLL):**

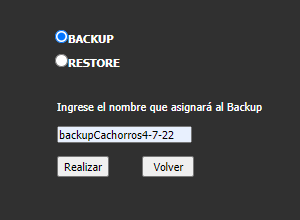


**BackupRestore.aspx.cs (Capa BLL):**





3.Se debe ingresar un nombre de archivo dentro del Textbox (UI), el cual será el nombre que se le asigne al Backup.

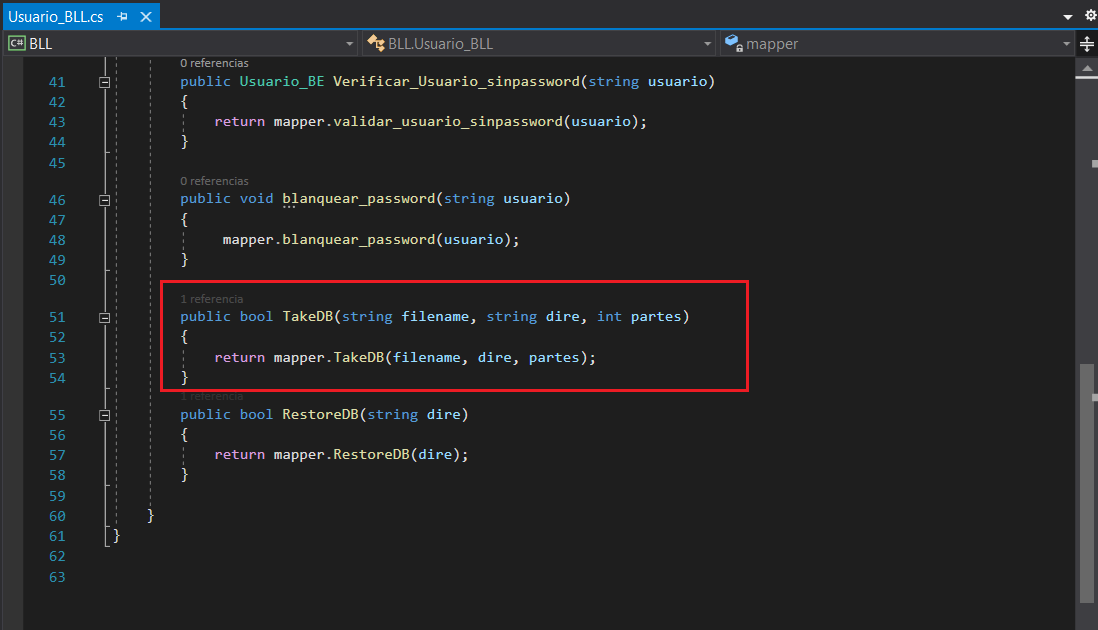


4. Para finalizar la acción se pulsa sobre el botón “Realizar” (Capa UI)

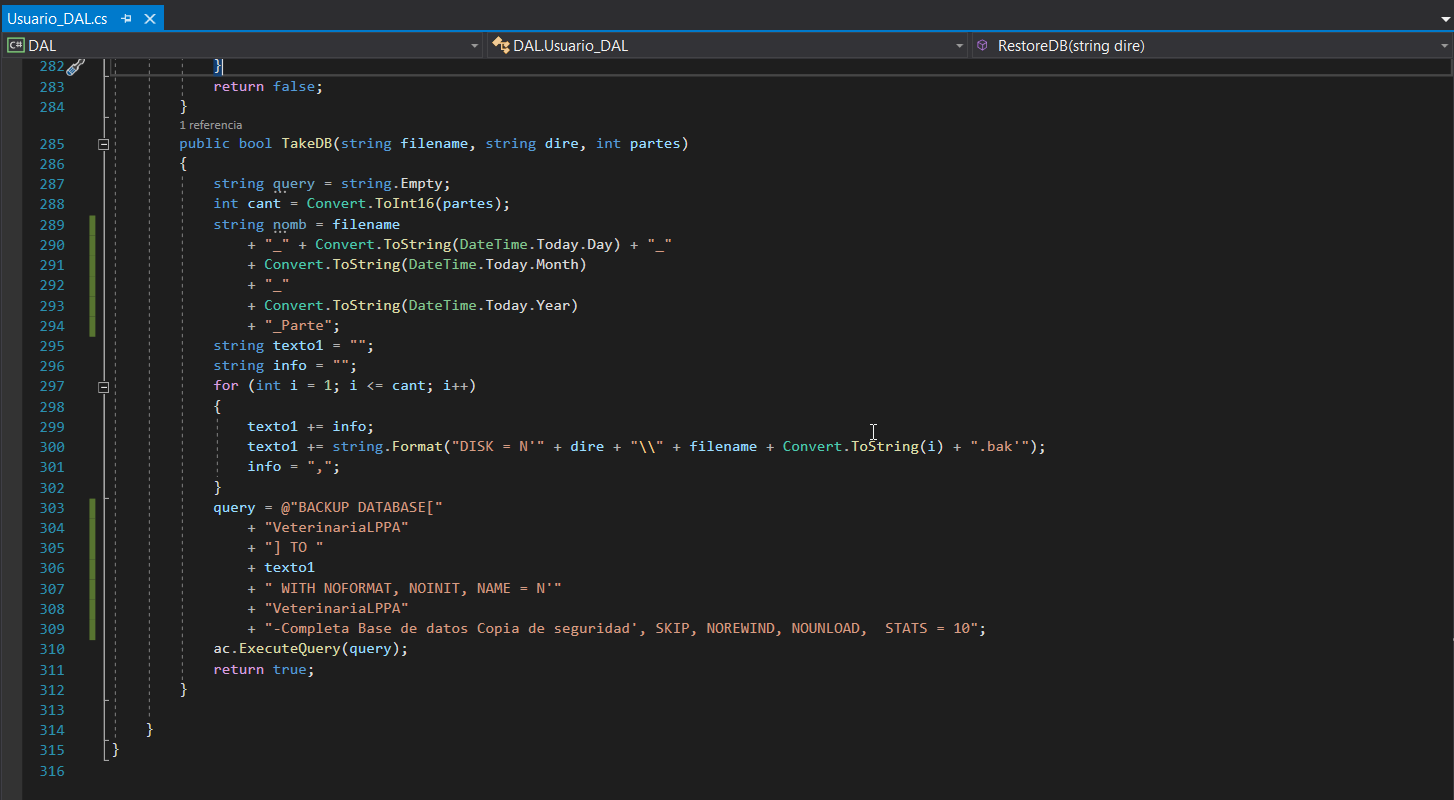
5. Se envían los datos a la acción correspondiente del usuario para la realización del Backup (Capa BLL).

También tenemos un botón “Volver”, que nos regresa al menú de “Respuesta.aspx” donde tenemos las acciones del administrador si está logueado.

**Usuario\_BLL.cs**



Allí se genera el nombre de la ruta donde se dejará el archivo .bak. Además de concatenar el nombre del mismo. La ruta por defecto es:\lppa-tp-main-V”*número de versión del aplicativo final*”\Trabajo Practico LPPA\Backups\”*nombre ingresado*”.

**Usuario\_DAL.cs**

6. Se procede a realizar el respaldo correspondiente de los datos (Capa DAL). Se retorna a la pantalla de BackupRestore.aspx (Capa UI)



### **2.2.2 Restore**

### **2.2.2.1 Ciclo de Vida**

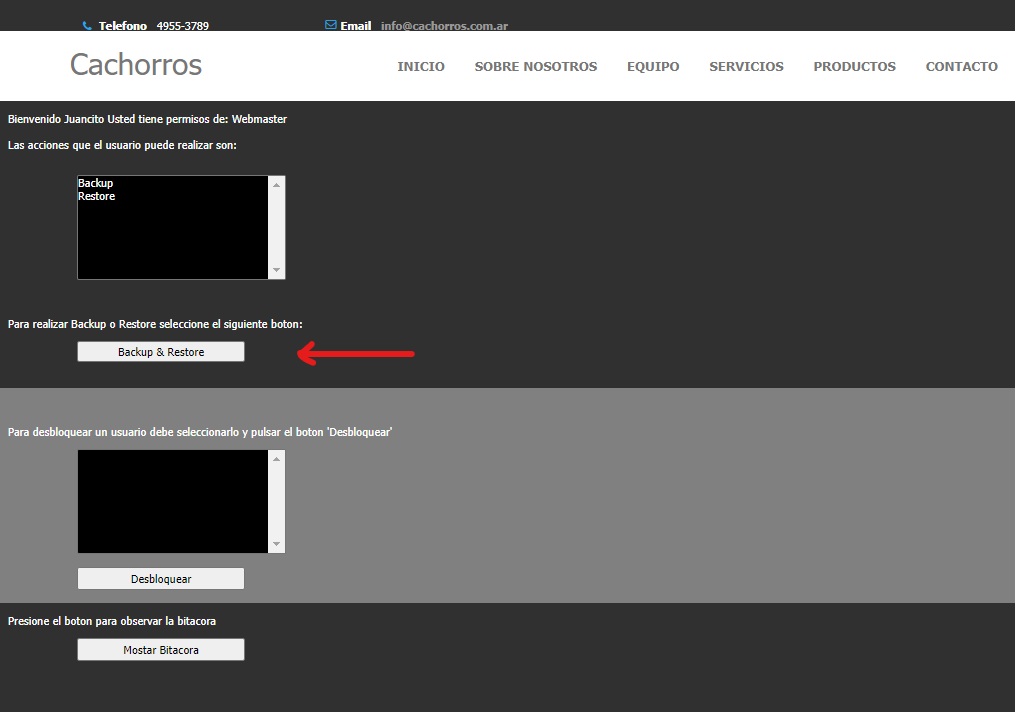
### 

### **Secuencia de navegación:**

1. Como precondición se debe realizar el Login (2.1) con usuario webmaster (ADMIN).
2. Una vez realizado el Login e ingresado en la pantalla principal se presiona el botón de Backup & Restore, esto nos redireccionará a la interfaz correspondiente: BackupRestore.aspx. (UI)
3. Se debe seleccionar el archivo correspondiente al Restore que se desea realizar (UI)
4. Para finalizar la acción se pulsa sobre el botón “Realizar” (UI)
5. Se envían los datos a la acción correspondiente del usuario para la realización del Restore (BLL)
6. Se procede a realizar el Restore correspondiente de los datos (DAL). Se retorna a la pantalla de BackupRestore.aspx

**2.2.2.2 Código:**

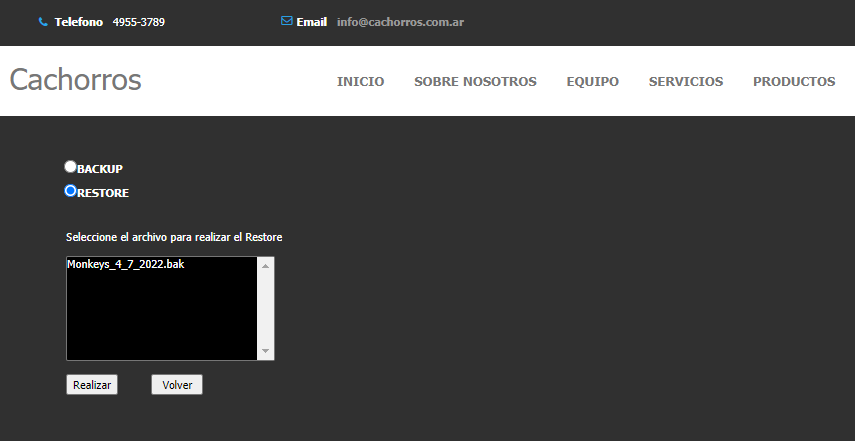
**Respuesta.aspx [Diseño]**



2. Como precondición se debe realizar el Login (2.1) con usuario webmaster (ADMIN).

3. Una vez realizado el Login e ingresado en la pantalla principal se presiona el botón de Backup & Restore, esto nos redireccionará a la interfaz correspondiente: BackupRestore.aspx. (Capa UI)

**BackupRestore.apsx**

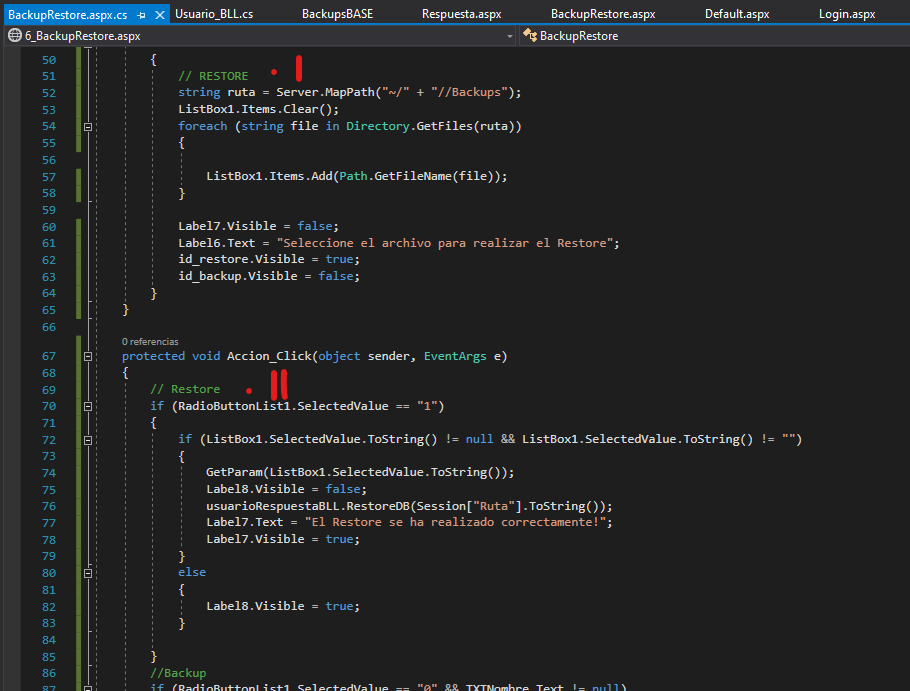


4. Se debe seleccionar el archivo correspondiente al Restore que se desea realizar (Capa UI).

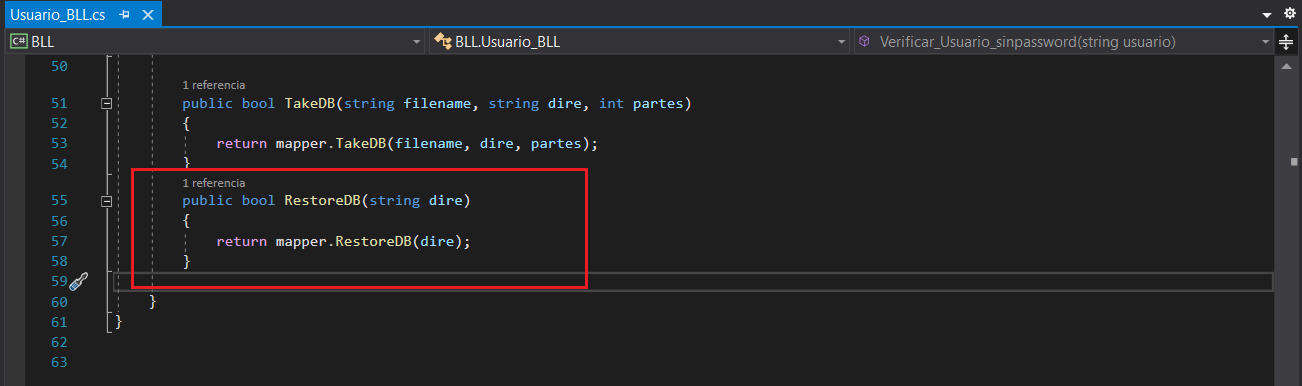
5. Para finalizar la acción se pulsa sobre el botón “Realizar” (UI) **BackupRestore.apsx**

En la siguiente imagen se ven los pasos I y II en el código, así desde la línea 52 hasta la 63 (paso I), se recuperan los archivos .bak para el Restore. Y luego desde la línea 69 a 78 se validan y ejecutan los pasos del restore (paso II). **BackupRestore.apsx.cs.**

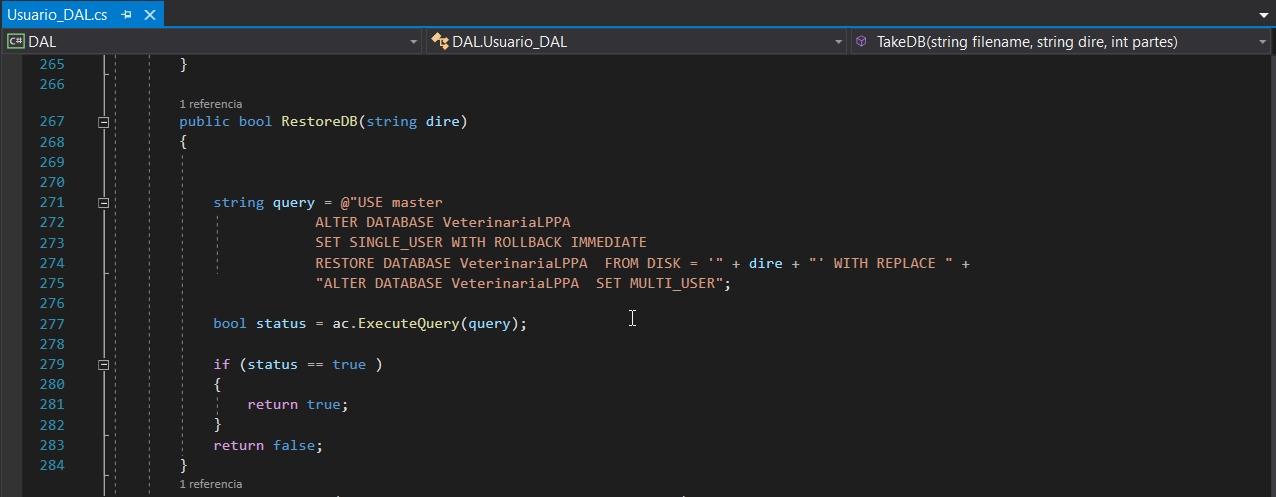
Se pasa el flujo del programa a **Usuario\_BLL.cs** donde se invoca al método “RestoreDB”, que recibe la ruta que construimos antes, y de ahí se continua el flujo **Usuario\_DAL.cs,** quien finalmente, ejecuta el código de la base que restituye el backup.



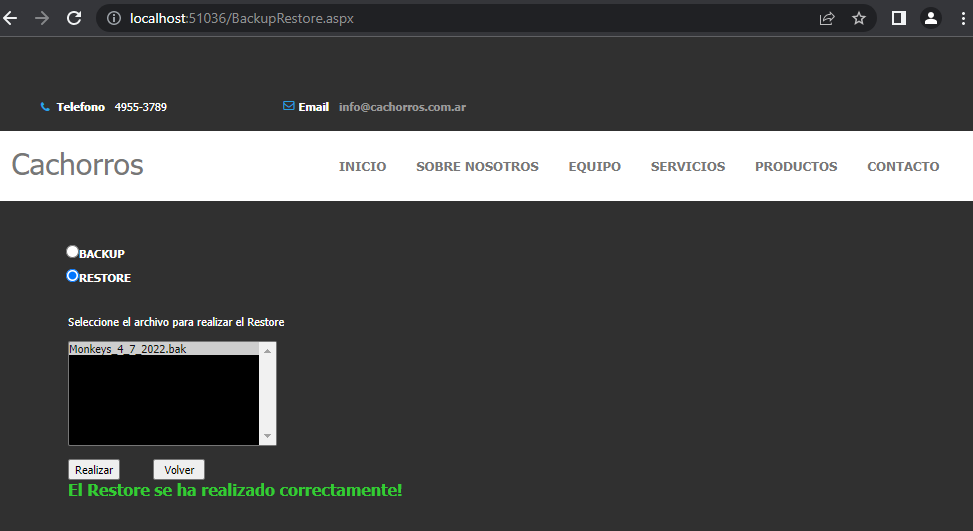
**Usuario\_BLL.cs**

****

**Usuario\_DAL.cs**

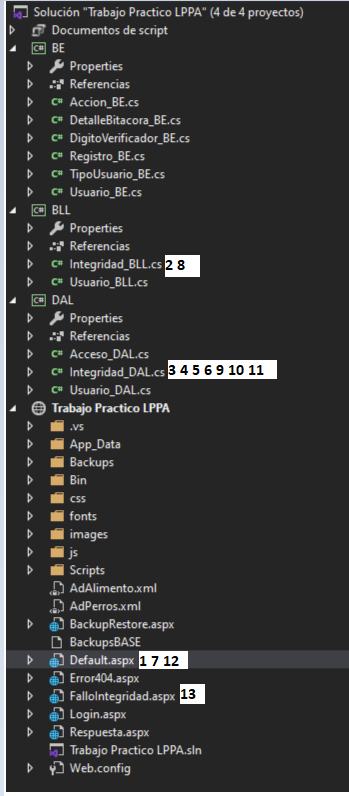
****

6. Una vez realizado correctamente el Restore, se informa por pantalla y se devuelve el flujo a la pantalla de **BackupRestore.aspx** (UI).

****

# **3- Dígito Verificador**

### **3.1 Ciclo de vida**



### **Secuencia de navegación:**

1. Inicia en el método Page\_Load comprobando la integridad digito verificador horizontal y luego hace el del vertical. Si hay tablas con errores se guarda registro de la misma tanto para horizontal como vertical, estos datos luego se mostrarán por pantalla. (UI)
2. Se obtienen las tablas con errores de verificación DVH y las DVV. (**Integridad\_BLL.cs**).
3. Se recalculan los valores de los DVH (paso 6) de cada registro (paso 5) de tablas que tienen dígitos verificadores (paso 4). En caso de encontrarse un registro corrompido, se agrega a una lista de registros para luego mostrar en pantalla. (**Integridad\_DAL.cs**)
4. Se obtienen desde la base de datos todas las tablas que tienen dígitos verificadores. Se guardan en objetos del tipo DigitoVerificador\_BE tanto el nombre de la tabla como el DVV (se usa en la verificación del DVV, no del DVH). (**Integridad\_DAL.cs**)
5. Se leen todos los registros de la tabla enviada como parámetro y se guardan como una lista de objetos Registro\_BE. Cada Registro\_BE guarda el id del registro (para luego mostrar por pantalla si el registro está comprometido), un string tiene concatenados todos los campos del registro con el DVH separado por el carácter ‘;’, y el nombre de la tabla. (**Integridad\_DAL.cs**)
6. Se calcula el valor del DVH del registro a partir de la codificación ASCII de todos los campos del registro concatenados, multiplicando este valor por el peso que tienen en la cadena. Esta función se utiliza tanto para guardar nuevos DVH como para el recálculo en la verificación de integridad. (**Integridad\_DAL.cs**)
7. Se realiza la verificación de integridad de la base de datos respecto al dígito verificador vertical. (UI)
8. 8.Se obtienen las tablas con errores de digito verificador vertical. (**Integridad\_BLL.cs**)
9. Se recalculan los valores de los DVV (paso 11) de cada tabla que tiene dígito verificador (paso 10). En caso de encontrarse una tabla corrompida, se agrega a una lista de registros (cuyo id es DVV) para luego mostrar en pantalla. (**Integridad\_DAL.cs**)
10. Se obtienen desde la base de datos todas las tablas que tienen dígitos verificadores. Se guardan en objetos del tipo DigitoVerificador\_BE tanto el nombre de la tabla como el DVV (se usa en la verificación del DVV, no del DVH). (**Integridad\_DAL.cs**)
11. Se calcula el DVV para la tabla como la suma total de todos los DVH de cada registro de la tabla. (**Integridad\_DAL.cs**)
12. En caso de encontrarse errores con errores de DVH, DVV o ambos, se redirige a la página de error de integridad (UI).
13. Finalmente se imprime por pantalla una lista con las tablas comprometidas y los id de los registros que fallaron en la comprobación de DVH. En caso de error de DVV, se imprime ‘DVV’ en lugar del ID del registro (UI)

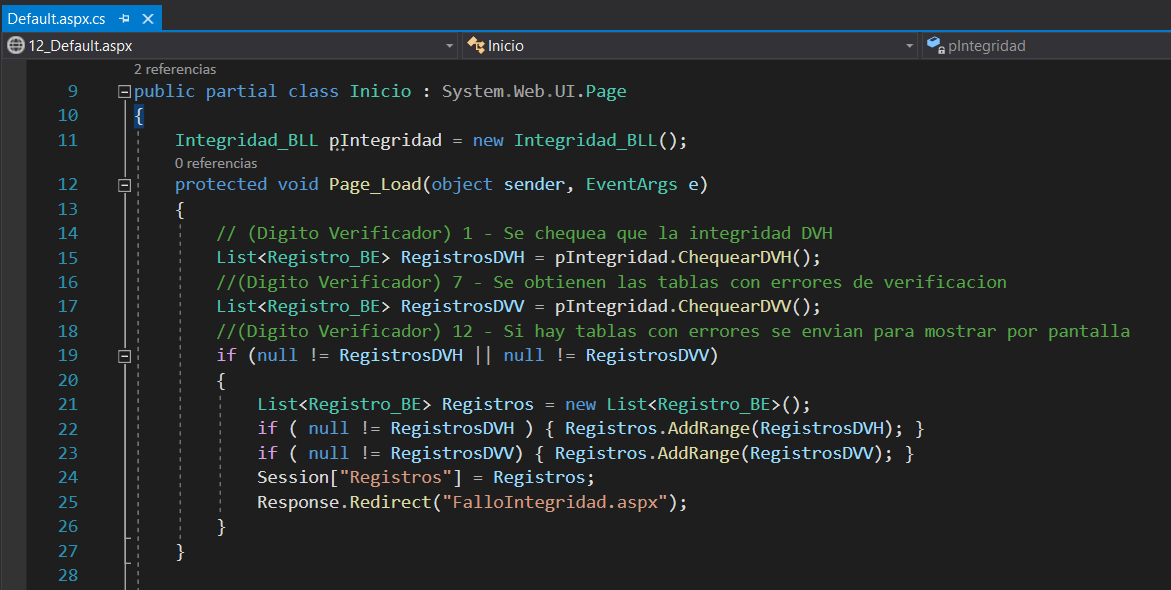
### **3.2 Código**

**Default.aspx.cs**

1.Se realiza la verificacion de integridad de la base de datos respecto al digito verificador horizontal (línea 14)

7.Se realiza la verificacion de integridad de la base de datos respecto al dígito verificador vertical (línea 16)

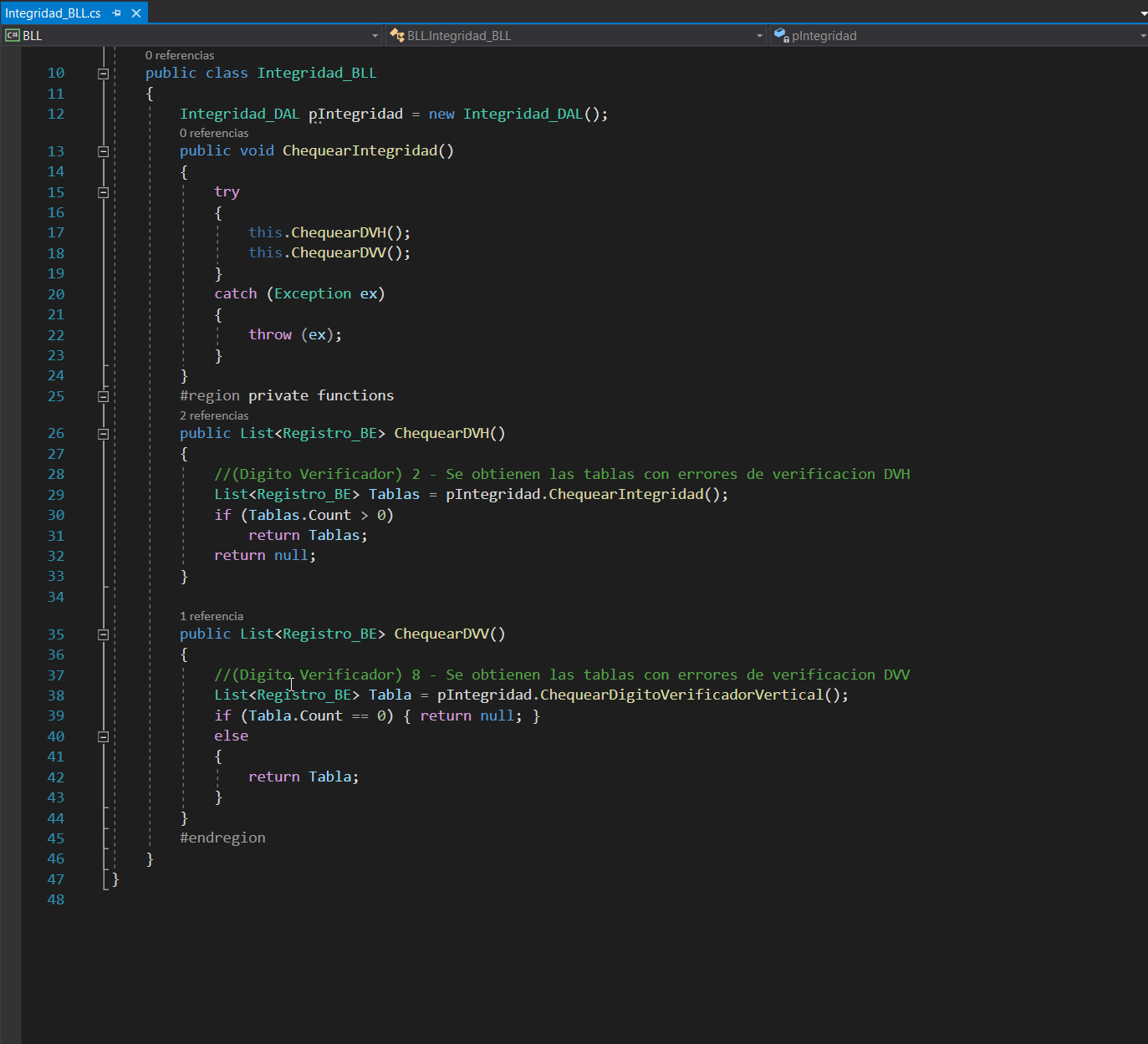
12.En caso de encontrarse errores con errores de DVH, DVV o ambos, se redirige a la página de error de integridad (línea 25)



**Integridad\_BLL.cs**

2.Se obtienen las tablas con errores de digito verificador horizontal (linea 29)

8.Se obtienen las tablas con errores de digito verificador vertical (linea 37)



**Integridad\_DAL.cs**

3.Se recalculan los valores de los DVH (paso 6) de cada registro (paso 5) de tablas que tienen dígitos verificadores (paso 4) (línea 73). En caso de encontrarse un registro corrompido, se agrega a una lista de registros para luego mostrar en pantalla.

4.Se obtienen desde la base de datos todas las tablas que tienen dígitos verificadores (línea 166). Se guardan en objetos del tipo DigitoVerificador\_BE tanto el nombre de la tabla como el DVV (se usa en la verificación del DVV, no del DVH)

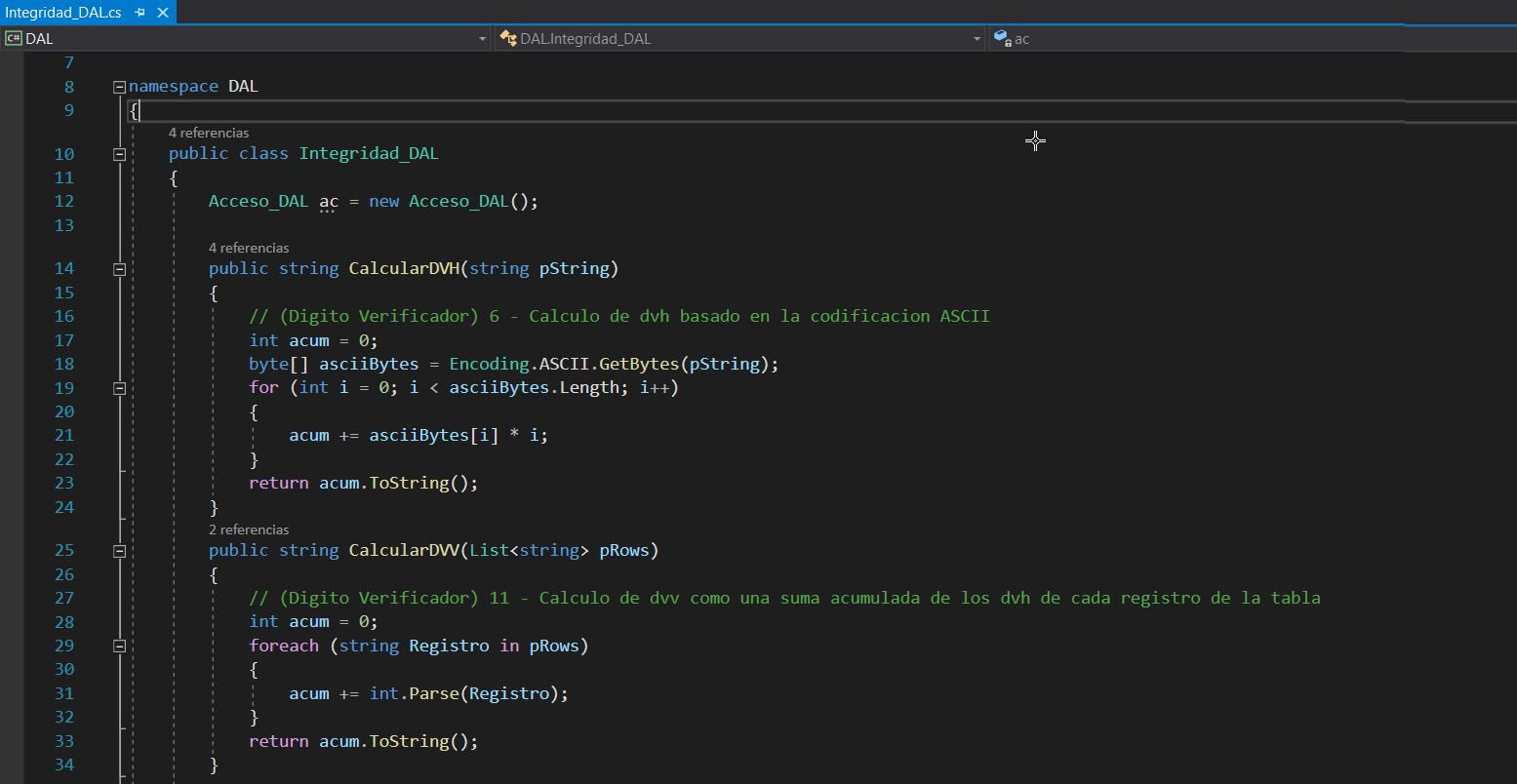
5.Se leen todos los registros de la tabla enviada como parámetro y se guardan como una lista de objetos Registro\_BE (línea 185). Cada Registro\_BE guarda el id del registro (para luego mostrar por pantalla si el registro está comprometido), un string tiene concatenados todos los campos del registro con el DVH separado por el carácter ‘;’, y el nombre de la tabla.

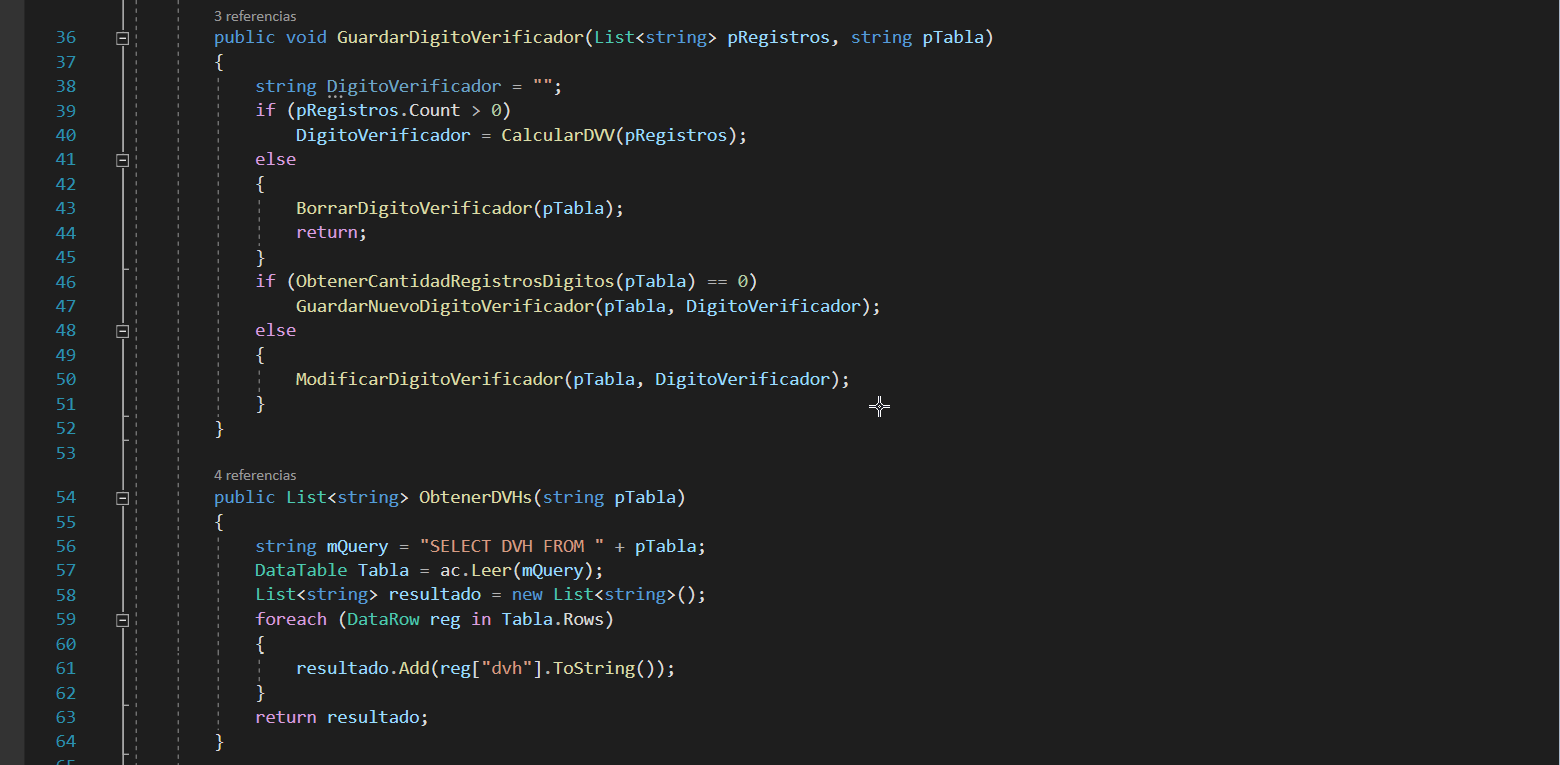
6.Se calcula el valor del DVH del registro a partir de la codificación ASCII de todos los campos del registro concatenados, multiplicando este valor por el peso que tienen en la cadena (línea 14). Esta función se utiliza tanto para guardar nuevos DVH como para el recálculo en la verificación de integridad.

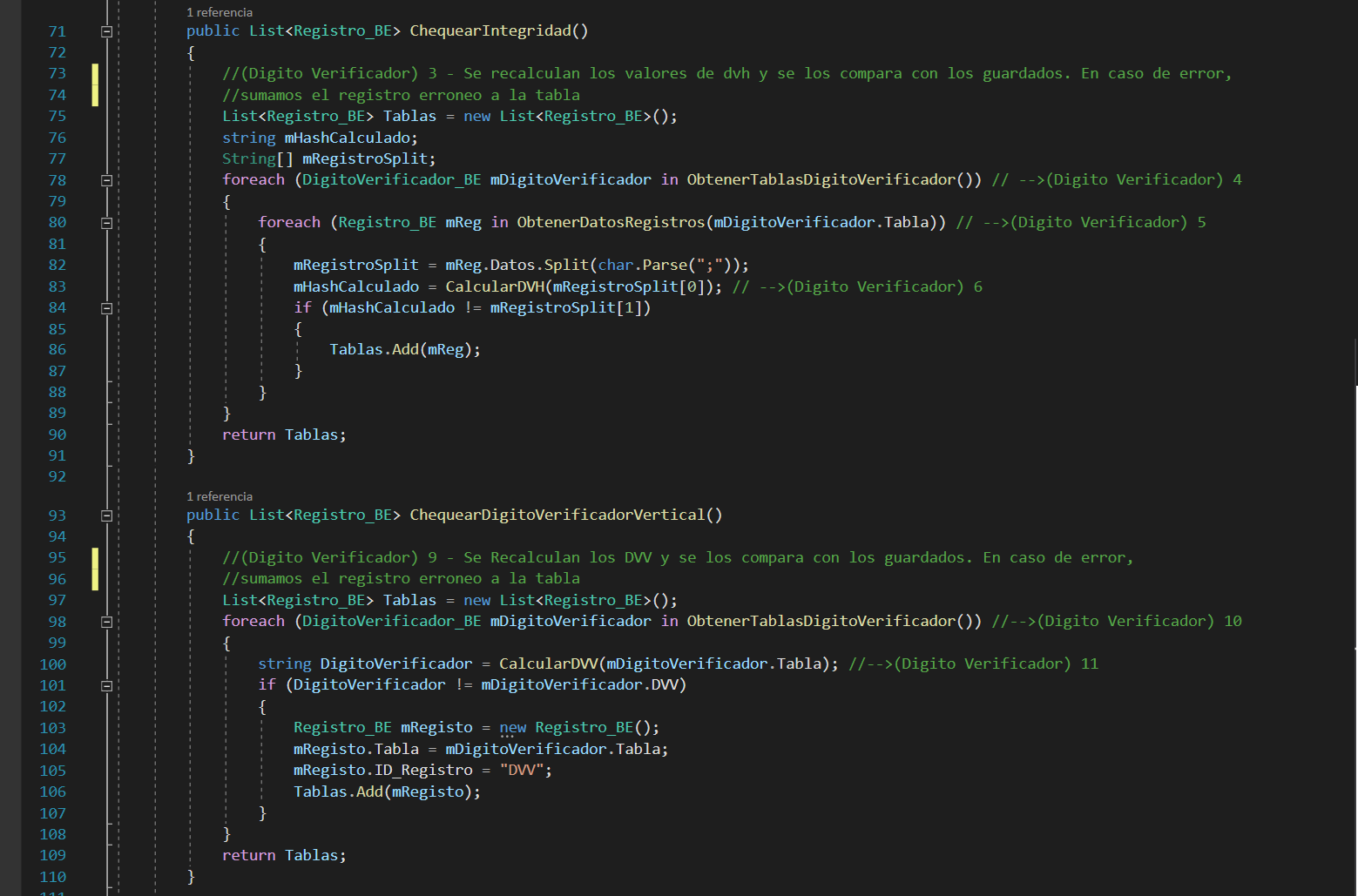
9.Se recalculan los valores de los DVV (paso 11) de cada tabla que tiene dígito verificador (paso 10) (línea 93). En caso de encontrarse una tabla corrompida, se agrega a una lista de registros (cuyo id es DVV) para luego mostrar en pantalla.

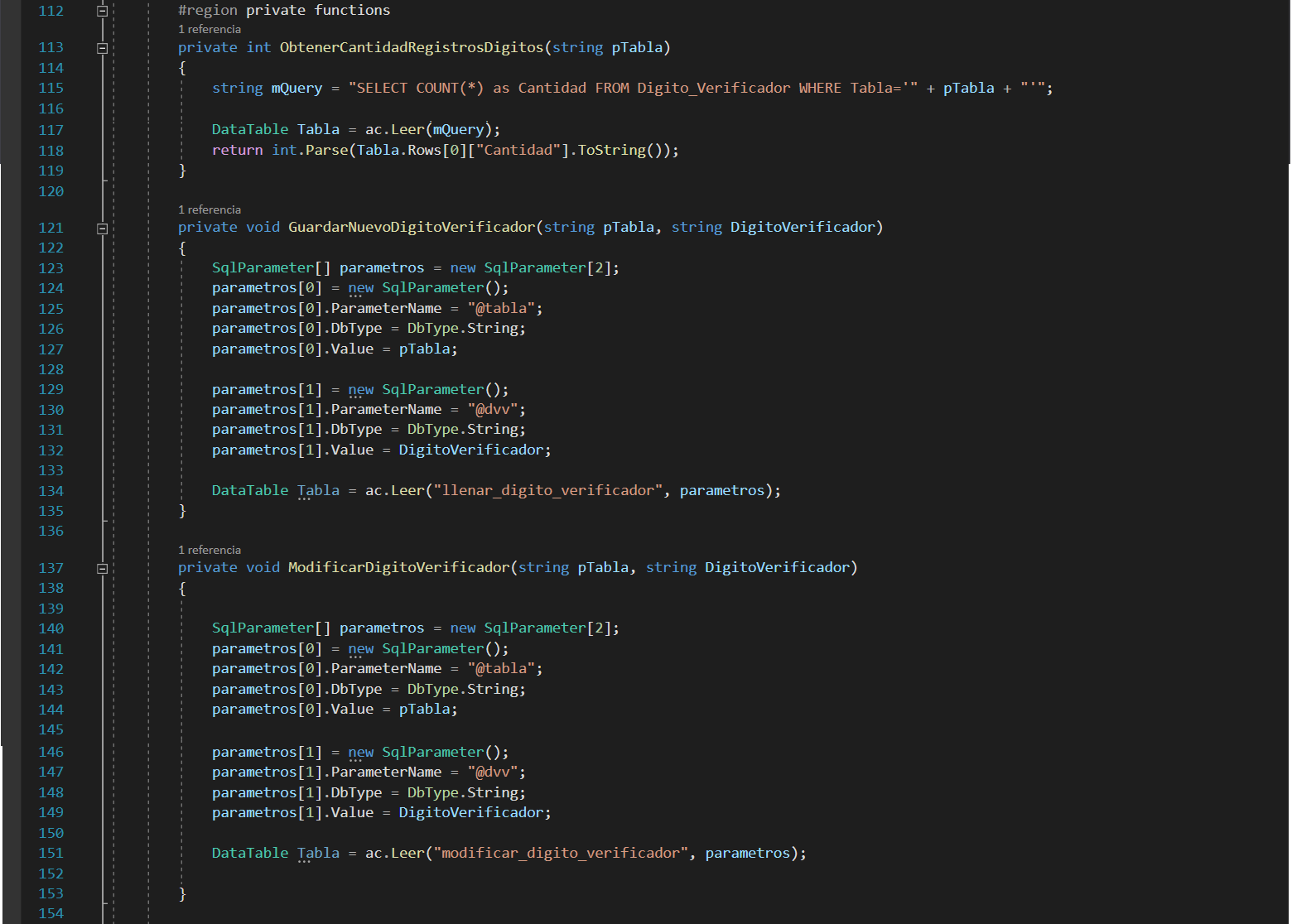
10.Se obtienen desde la base de datos todas las tablas que tienen dígitos verificadores (línea 166). Se guardan en objetos del tipo DigitoVerificador\_BE tanto el nombre de la tabla como el DVV (se usa en la verificación del DVV, no del DVH)

11.Se calcula el DVV para la tabla como la suma total de todos los DVH de cada registro de la tabla (línea 25)

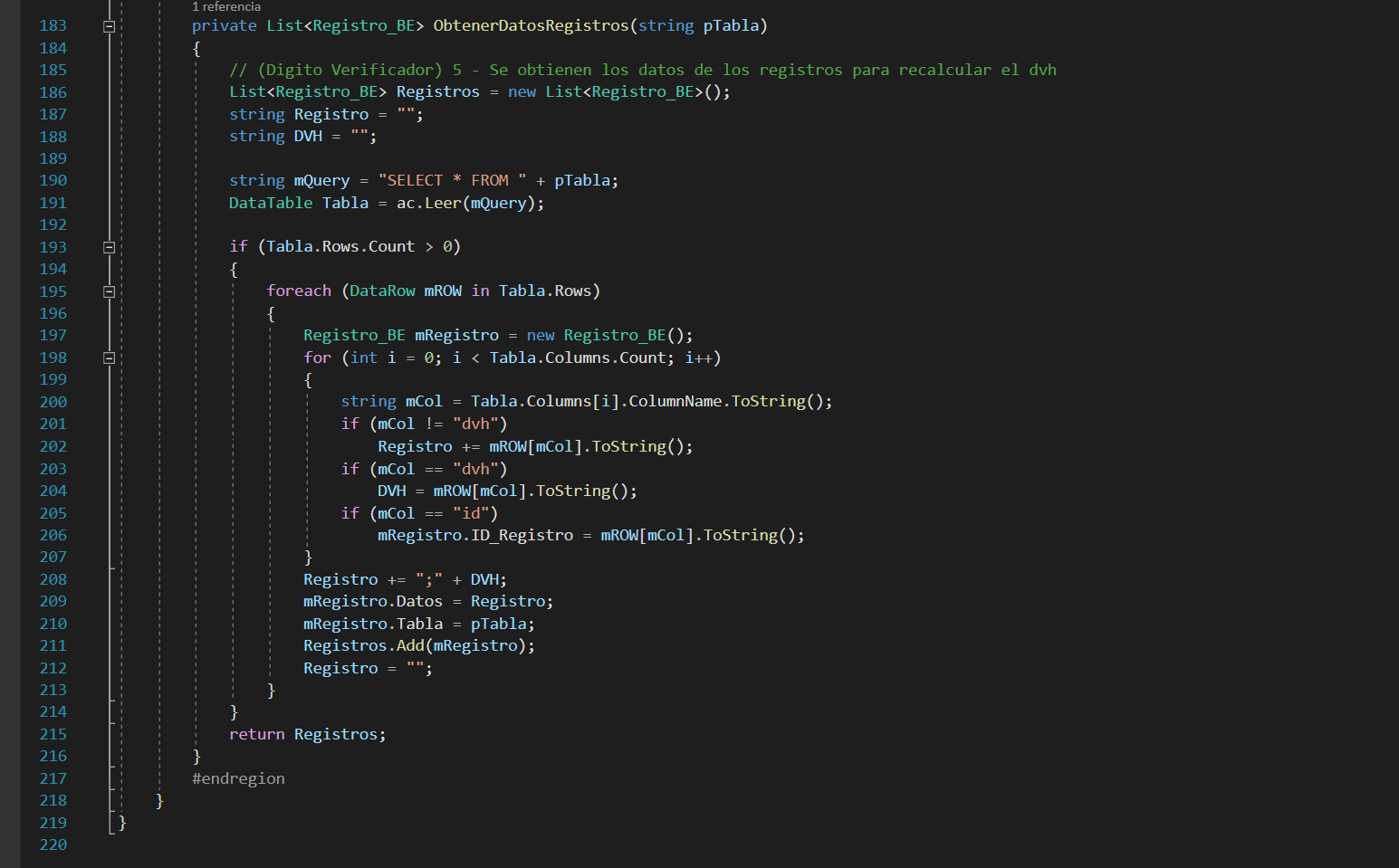




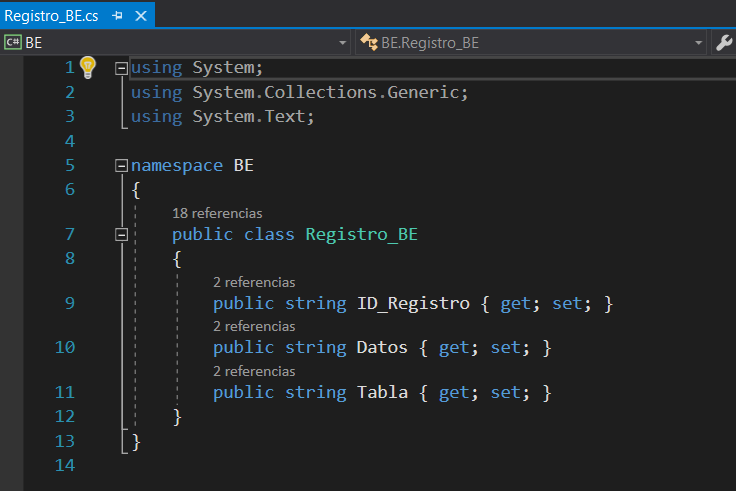




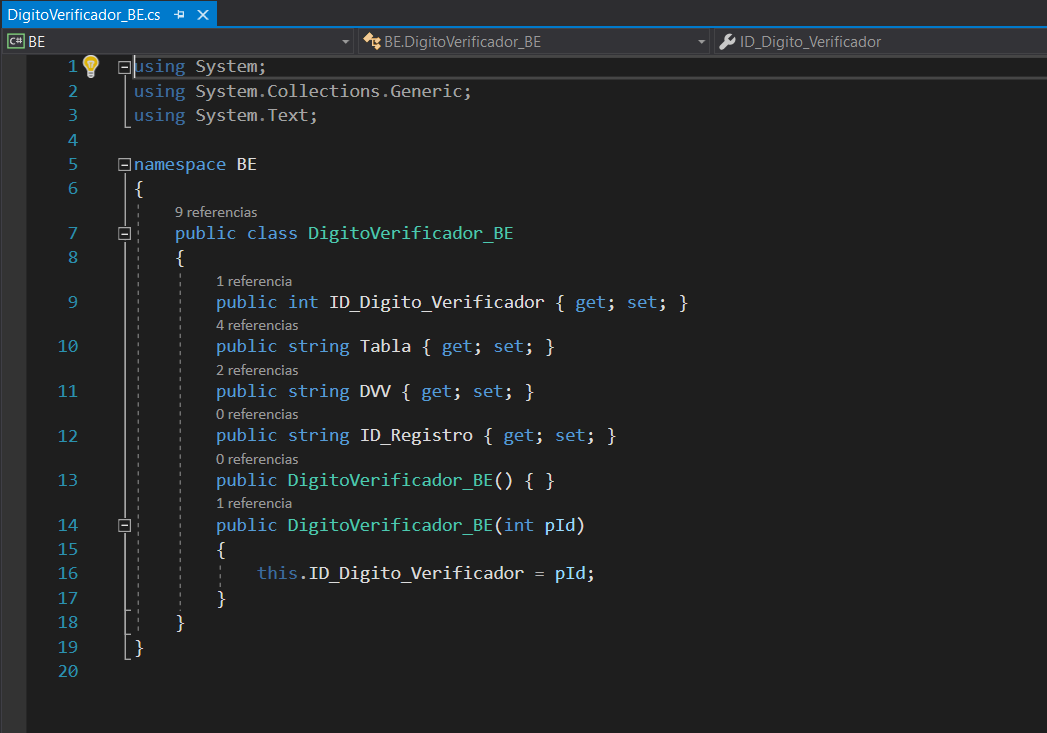


****

**Registro\_BE.cs**

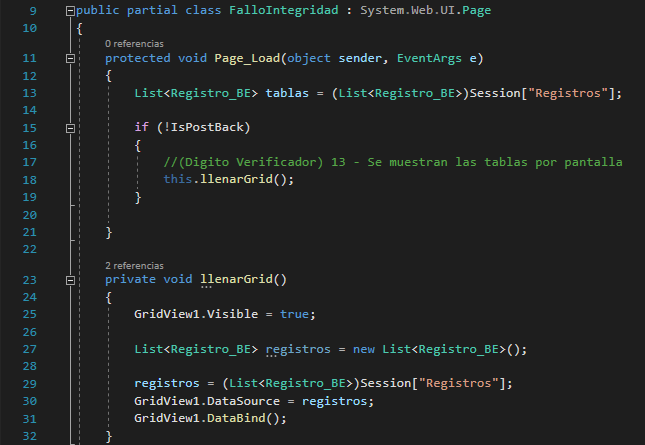


**DigitoVerificador\_BE.cs**



**FalloIntegridad.aspx.cs**

13.Finalmente se imprime por pantalla una lista con las tablas comprometidas y los id de los registros que fallaron en la comprobación de DVH. En caso de error de DVV, se imprime ‘DVV’ en lugar del ID del registro.

****

**4.1 Código de Store Procedures**

**SP blanquear\_password**

CREATE procedure [dbo].[blanquear\_password]

@usu varchar(100)

as

begin

UPDATE dbo.Usuario SET bloqueado = 0 where usuario = @usu

end

select u.id as id, u.usuario as usuario, u.contraseña as contraseña, u.nombre as nombre, u.bloqueado as bloqueado, u.id\_tipo\_usuario as tipo\_usuario From Usuario u where usuario = @usu

GO

**SP bloquear\_usuario**

CREATE procedure [dbo].[bloquear\_usuario]

@usu varchar(100)

as

begin

UPDATE dbo.Usuario SET bloqueado = bloqueado + 1 where usuario = @usu

end

select u.id as id, u.usuario as usuario, u.contraseña as contraseña, u.nombre as nombre, u.bloqueado as bloqueado, u.id\_tipo\_usuario as tipo\_usuario From Usuario u where usuario = @usu

GO

**SP borrar\_digito\_verificador**

CREATE procedure [dbo].[borrar\_digito\_verificador]

@tabla varchar(100), @dvv varchar(500)

as

begin

declare @id int

set @id = isnull((Select max(ID\_Digito\_Verificador) from Digito\_Verificador),0 ) +1

DELETE [dbo].[Digito\_Verificador] WHERE Tabla = @tabla

end

GO

**SP desbloquear\_usuario**

CREATE procedure [dbo].[desbloquear\_usuario]

@usu varchar(100)

as

begin

UPDATE Usuario

SET bloqueado = 0

WHERE usuario = @usu;

end

select u.id as id, u.usuario as usuario, u.contraseña as contraseña, u.nombre as nombre, u.bloqueado as bloqueado, u.id\_tipo\_usuario as tipo\_usuario From Usuario u where usuario = @usu

GO

**SP listar\_acciones**

Create procedure [dbo].[listar\_acciones]

@id\_tipo\_usu varchar(50)

as

begin

select b.id, b.detalle

from TipoUsuario\_Accion a

inner join Accion b

on a.id\_accion= b.id

where a.id\_tipousuario = @id\_tipo\_usu

end

GO

**SP listar\_bitacora**

CREATE procedure [dbo].[listar\_bitacora]

as

begin

select a.\*, b.nombre, b.usuario

from Bitacora a

inner join Usuario b

on a.id\_usuario = b.id

ORDER BY a.fecha DESC

end

GO

**SP listar\_usuariosBloqueados**

Create procedure [dbo].[listar\_usuariosBloqueados]

as

begin

select \*

from Usuario

where bloqueado = 3

end

GO

**SP llenar\_bitacora**

CREATE procedure [dbo].[llenar\_bitacora]

@id\_usu varchar(100), @detalle varchar(500)

as

begin

declare @id int

declare @fecha datetime

set @id = isnull((Select max(id) from Bitacora),0 ) +1

INSERT INTO [dbo].[Bitacora]

([id]

,[detalle]

,[fecha]

,[id\_usuario])

VALUES

(@id,

@detalle,

FORMAT(SYSDATETIME(), 'dd/MM/yyyy hh:mm:ss', 'en-gb'),

@id\_usu)

end

select @id as id, @detalle as detalle, SYSDATETIME() as fecha, @id\_usu as id\_usu

GO

**SP llenar\_digito\_verificador**

CREATE procedure [dbo].[llenar\_digito\_verificador]

@tabla varchar(100), @dvv varchar(500)

as

begin

declare @id int

set @id = isnull((Select max(ID\_Digito\_Verificador) from Digito\_Verificador),0 ) +1

INSERT INTO [dbo].[Digito\_Verificador]

([ID\_Digito\_Verificador]

,[Tabla]

,[DVV])

VALUES

(@id,

@tabla,

@dvv)

end

GO

**SP modificar\_digito\_verificador**

CREATE procedure [dbo].[modificar\_digito\_verificador]

@tabla varchar(100), @dvv varchar(500)

as

begin

declare @id int

set @id = isnull((Select max(ID\_Digito\_Verificador) from Digito\_Verificador),0 ) +1

UPDATE [dbo].[Digito\_Verificador] SET DVV = @dvv WHERE Tabla = @tabla

end

GO

**SP update\_bitacora\_dvh**

CREATE procedure [dbo].[update\_bitacora\_dvh]

@id varchar(100), @dvh varchar(500)

as

begin

UPDATE [dbo].[Bitacora] SET dvh = @dvh WHERE id = @id

end

GO

**SP update\_usuario\_dvh**

CREATE procedure [dbo].[update\_usuario\_dvh]

@id varchar(100), @dvh varchar(500)

as

begin

UPDATE [dbo].[Usuario] SET dvh = @dvh WHERE id = @id

end

GO

**SP verificar\_usuario**

CREATE procedure [dbo].[verificar\_usuario]

@usu varchar(100), @pass varchar(100)

as

begin

select a.id, a.nombre, a.usuario, a.contraseña, a.id\_tipo\_usuario, b.tipo\_usuario

from Usuario a

inner join Tipo\_Usuario b

on a.id\_tipo\_usuario = b.id

where a.usuario = @usu and a.contraseña = @pass

end

GO

**SP verificar\_usuario\_sinpassword**

CREATE procedure [dbo].[verificar\_usuario\_sinpassword]

@usu varchar(100)

as

begin

select a.id, a.nombre, a.usuario, a.contraseña, a.id\_tipo\_usuario,a.bloqueado, b.tipo\_usuario

from Usuario a

inner join Tipo\_Usuario b

on a.id\_tipo\_usuario = b.id

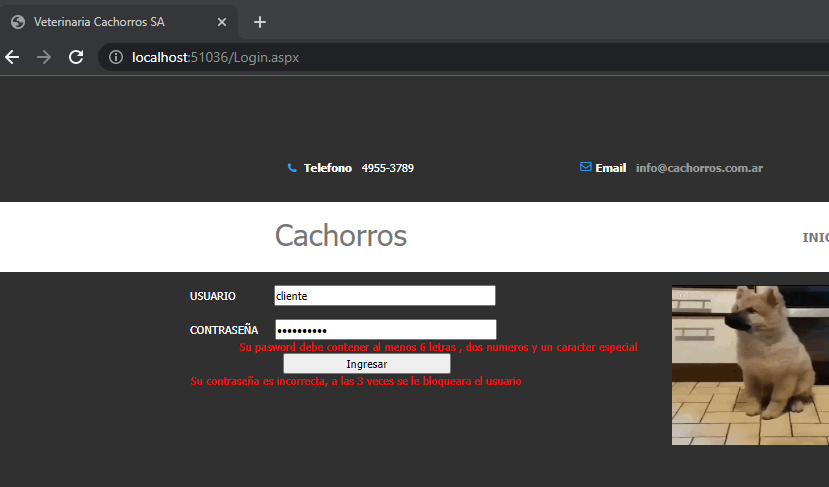
where a.usuario = @usu

end

GO

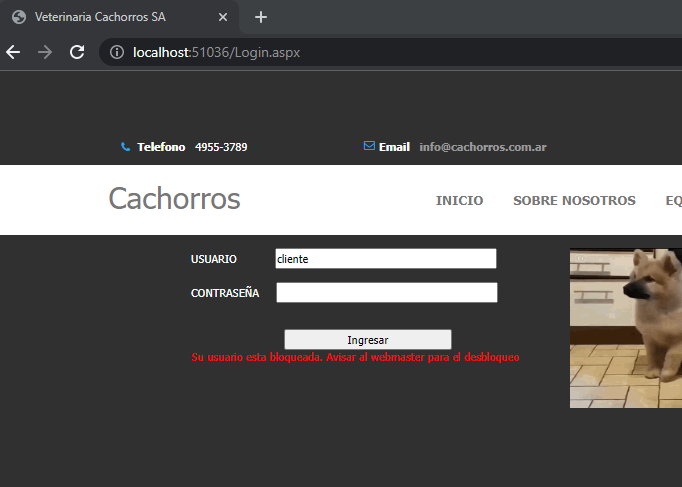
# **5.1-Validaciones de control**

Se realizan validaciones sobre los controles “Usuario” y “Contraseña” donde se solicitan que ambos estén completos para poder acceder a los métodos.<RequiredFieldValidator>. Como también, si se ingresan valores que no corresponden. Se le agrega una “regexp”, [a-zA-Z]{6}\w\*\d{2}\W{1}, para validar 6 letras, 2 dígitos y 1 carácter especial, al campo de contraseña. Se muestra un mensaje genérico “Usuario o contraseña Incorrectos” en caso de no cumplir.



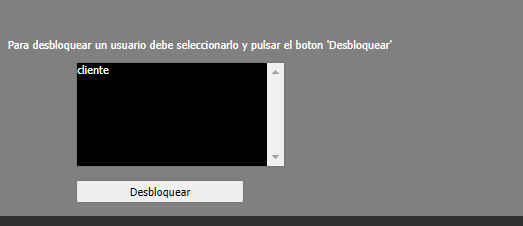
A continuación se valida la pass ingresada, esto es mediante una función de encriptación, que utiliza el algoritmo Message-Digest Algorithm 5 (MD5). Para luego comparar su hash con el que se encuentra almacenado en la base. Utilizamos la librería System.Security.Cryptography.MD5

En el caso que el usuario ingrese incorrectamente la contraseña 3 veces, la misma será bloqueada

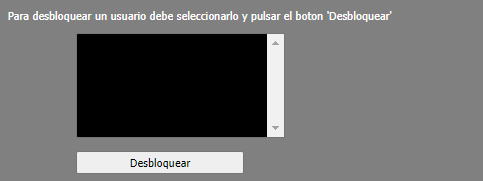


El desbloqueo le será solicitado al usuario ADMIN quien tendrá la funcionalidad para realizar la operación:

**Respuesta.aspx**



En esta pantalla el usuario ADMIN únicamente, podrá desbloquear a los usuarios que aparezcan en la lista de bloqueados. Deberá seleccionarlos y presionar el botón desbloquear. Cuando la lista esté vacía significa que no hay usuarios bloqueados.



## **6.1.2 - CONTROL DE INTEGRIDAD**

Se realiza un control de integridad para validar que la base de datos no haya sido corrompida, y en caso de serlo se informa por pantalla al iniciar el aplicativo y no dejará operar el sistema.

**5.1.3 ESTRATEGIAS DE PROGRAMACIÓN**

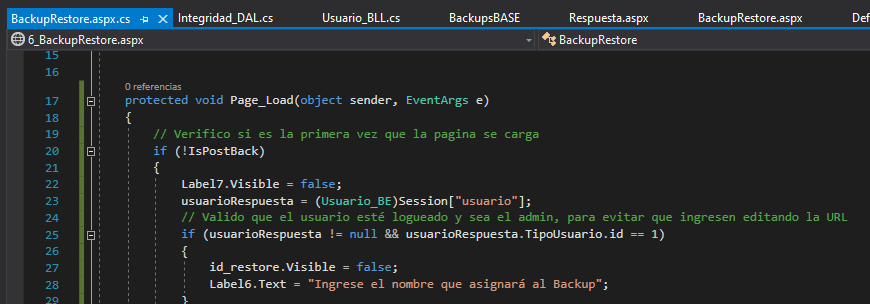
El diseño se realizó en 4 capas:

* UI: Capa de interfaz de usuario
* BLL: Capa de entidad de lógica de negocio
* DAL Capa de acceso a datos
* BE: Capa de entidades del negocio

La conexión realizada mediante la DAL a la base de datos es ADO.desconectado.

**5.1.4 CONTROL DE ACCESO POR URL**

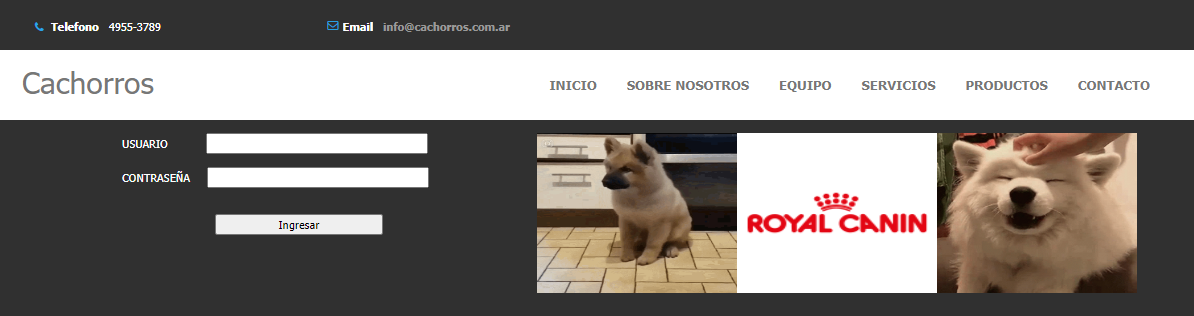
Se realiza un control por el cual no se puede acceder de forma manual ingresando la direcciones de las url en el navegador. Esto es mediante validación de permisos necesarios, es decir que quien no tenga permisos no podrá acceder a los recursos que no le corresponda, por ejemplo, un usuario no logueado o del tipo cliente no podrá acceder a la página del BackUp and Restore, “http://localhost:51036/BackupRestore.aspx“.



Mediante el uso el de variables de Session, se verifica si el usuario está logueado, además si tiene permisos para ver el recurso.

**5.1.5 AdRotators:**

Se incorporó una adrotator en la sección de login, donde se agrega publicidad de productos que vendemos:



Además de incorporar una galería de marcas que trabajan con nosotros.

