



Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de Aplicaciones 1

Segundo Obligatorio



Cristian Palma 208443



Federico Alonso 182999

Gestor de Contraseñas

Docentes: Leonardo Cecilia, Bruno Balduccio.

Grupo N5A

Repositorio: https://github.com/ORT-DA1/208443_182999

Índice

Descripción general del trabajo y del sistema.	3
Objetivo de la aplicación.....	3
Errores conocidos y/o mejoras para alguna siguiente versión:.....	3
Descripción y justificación de diseño	4
Diagrama de paquetes	4
Diagramas de clases.....	5
Interacción de la Interfaz con el Dominio.....	6
Almacenamiento de datos	7
Diseño para chequear Data Breaches.....	8
Diseño de la Clase Historial.....	9
Decisiones de diseño relevantes.....	12
DIAGRAMAS DE INTERACCION.....	13
Verificar Fortaleza	13
Modificar Tarjeta de Crédito.....	14
Consultar Vulnerabilidades	15
DIAGRAMA DE LAS TABLAS DE LA BASE DE DATOS	16
Proceso de instalación	17

Descripción general del trabajo y del sistema.

Objetivo de la aplicación

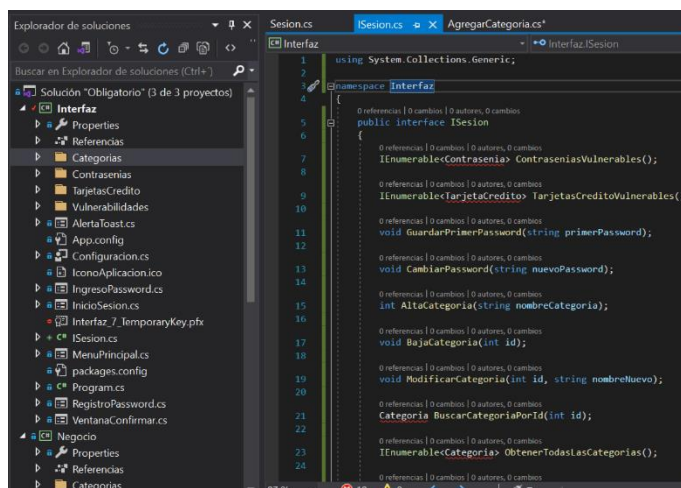
El objetivo de esta aplicación es permitir al usuario registrar sus contraseñas e información de tarjetas de crédito de forma segura mediante un gestor de contraseñas, para que el usuario sólo tenga que memorizar una clave para acceder a todas las demás.

A las funcionalidades presentadas en la entrega anterior, se le adicionaron los siguientes requerimientos:

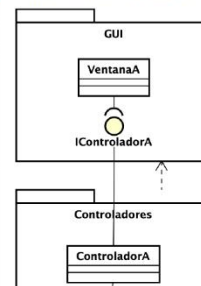
- Persistencia de todas las entidades en una base de datos.
- Histórico de resultados de los chequeos de contraseñas y tarjetas de crédito que hayan sido fugados (comparando nuestras tarjetas y contraseñas guardadas con fuentes conocidas).
- Se agrega otra fuente de Data Breaches, ahora se pueden cargar como fuentes archivos de texto.
- Sugerencias de mejora de contraseña, al momento de crear o actualizar una contraseña, el sistema nos hace sugerencias y nos informa si la misma apareció en un data breach conocido, en el caso de la contraseña, si previamente la utilizamos en otro sitio y la fortaleza de la misma (la cual está clasificada en colores con los mismos criterios de la entrega anterior).

Errores conocidos y/o mejoras para alguna siguiente versión:

- Se utiliza “DataAnnotations” para indicarle a Entity Framework las características y propiedades de columnas y tablas, lo cual hace que las entidades tengan un acoplamiento con la persistencia. Comenzamos desde el principio utilizando esta técnica, y al final cuando aprendimos que sería mejor utilizar “Fluent API” ya no contábamos con mucho tiempo para hacer todos los cambios y no queríamos perder el historial de las Migrations.
- En la interfaz de usuario, contamos con código repetido en los formularios de registro y modificar contraseña. Esto se debe a que en uno se utiliza UserControl y en otro WindowsForm, en sí cambia el dato de la contraseña que modifica los datos al cambiar un combobox con la contraseña de referencia. No se contó con tiempo para reformular el diseño de interfaz a efecto de solucionar dicho problema.
- No se pudo aplicar el Principio de inversión de Dependencias de los principios SOLID. Esto implicaba muchos cambios para intentar evitar el error de referencia circular y desconocíamos las técnicas utilizadas para resolver este inconveniente.



Ejemplo que cumple DIP:



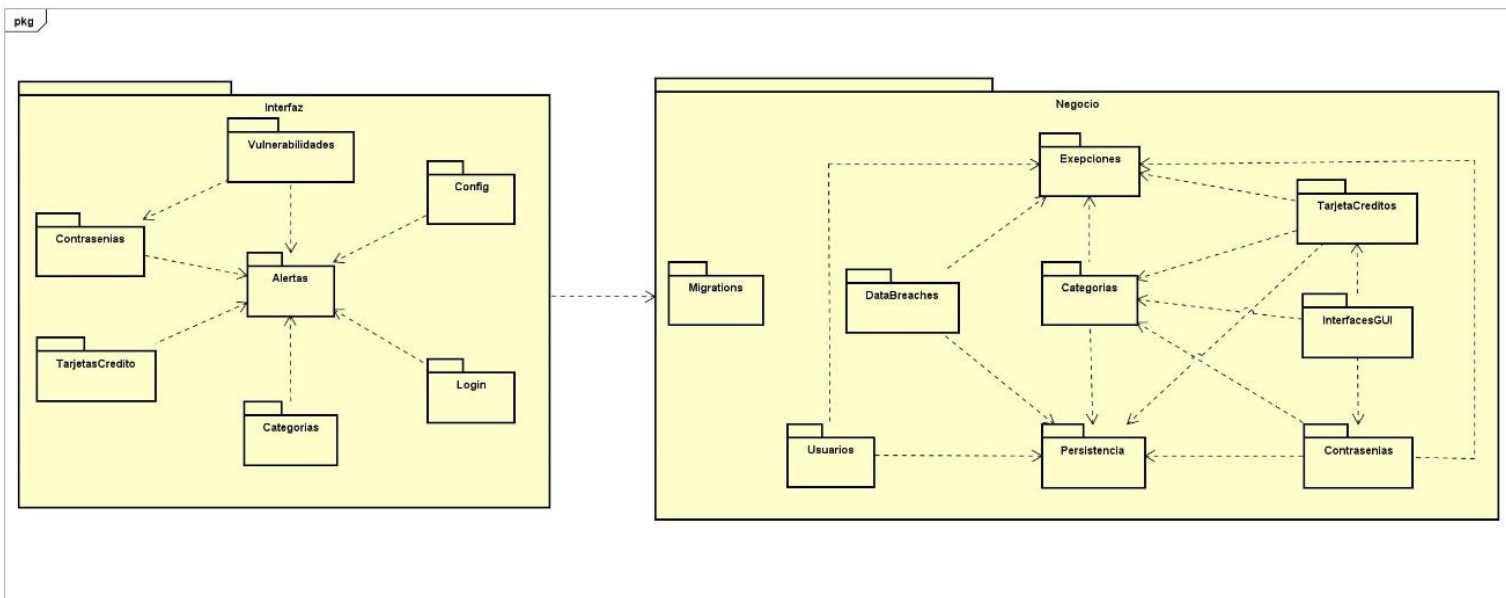
- Similar al punto anterior, no se pudo manejar la persistencia en un proyecto separado, la lógica de la misma se encuentra en el proyecto junto con la lógica de negocio, en un paquete especial llamado “persistencia”.
- El control sobre las excepciones que puede lanzar errores en la base de datos no fue muy profundo, si bien en la capa de la aplicación hay controles exhaustivos para evitar las mismas, entendemos que se debe hacer el mismo énfasis en controlar lo mismo a nivel de base de datos.
- Por razones de tiempo, no se implementa la opción de almacenar en memoria los historiales de data breaches, por lo que dicha opción no funciona si se vuelve atrás a utilizar listas en memoria, solo funciona en base de datos ya que es una funcionalidad implementada para esta entrega.

Descripción y justificación de diseño

Diagrama de paquetes

El diagrama de paquetes corresponde a la forma en la que están distribuidas las clases del sistema.

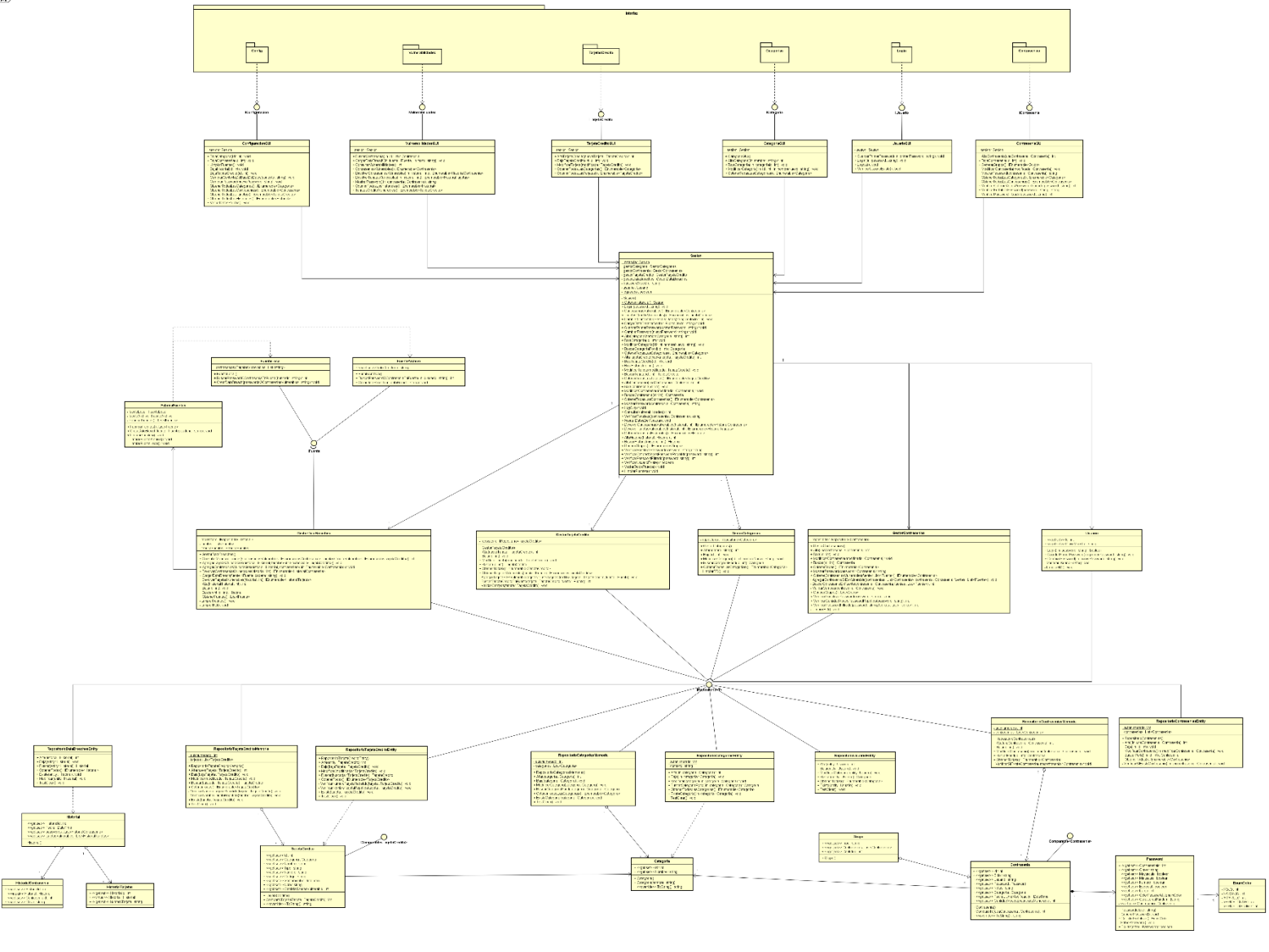
Las mismas se encuentran separadas en dos grandes proyectos (Interfaz y Negocio), y dentro de ellos se vuelve a subdividir como se muestra a en la siguiente imagen:



199

A continuación, se muestra un nuevo diagrama de clases simplificado para mostrar un primer acercamiento al diseño.

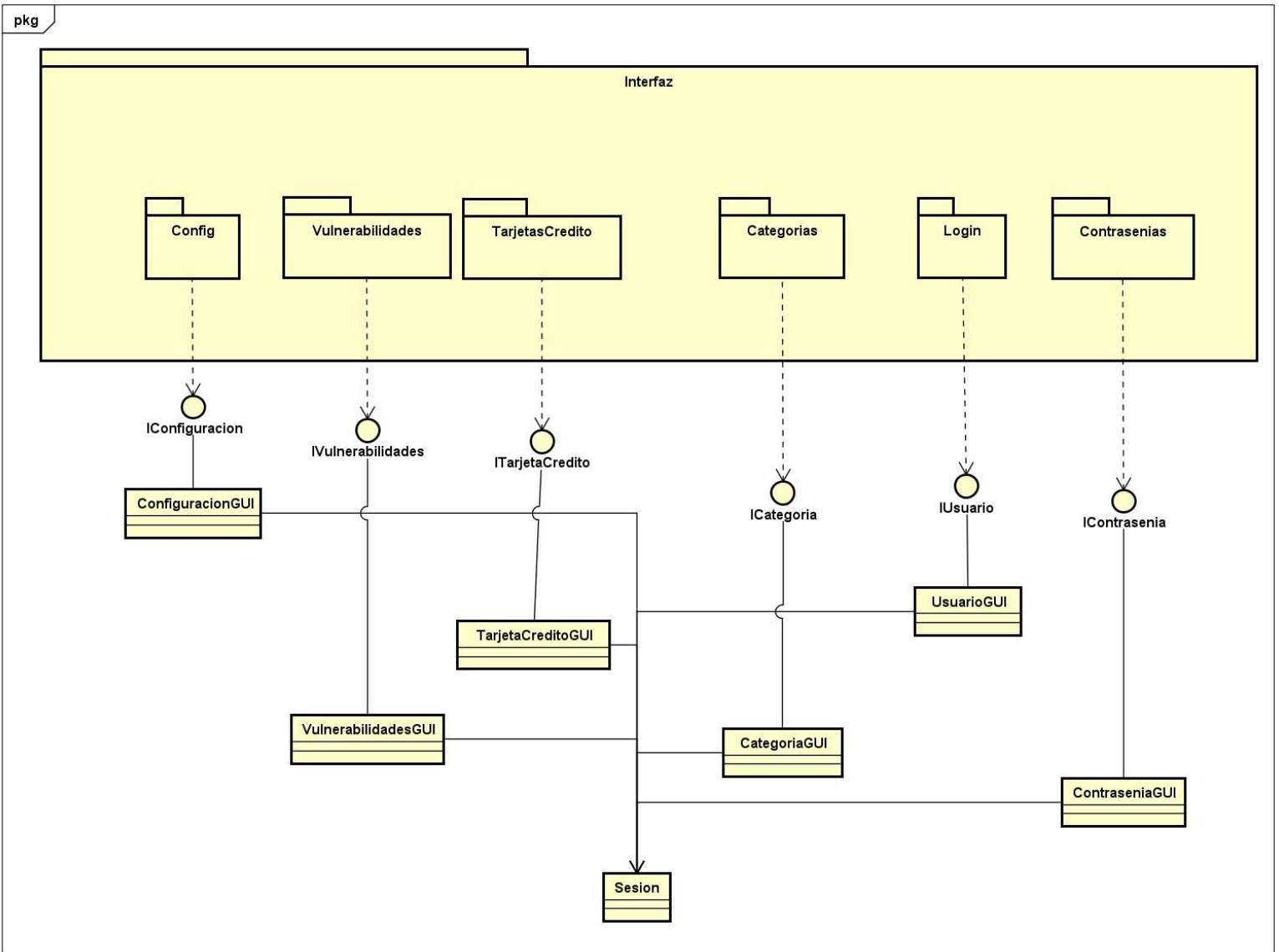
Se simplifica el diagrama quitándole las excepciones y algunas relaciones de uso a efectos de que sea legible, se adjunta como anexo el diagrama de clases completo.



Interacción de la Interfaz con el Dominio

En esta entrega limitamos a las ventanas a que conozcan toda la lógica de la fachada, aplicando el principio SOLID: **ISP - Interface Segregation Principle**

A continuación, se muestra el diseño:



En cierto momento teníamos que toda la interfaz podía realizar todas las acciones del sistema, debido a que poseemos un controlador fachada que deriva las solicitudes al resto de los controladores, teníamos que la interfaz por tener una instancia de sesión tenía acceso a todas las funcionalidades de la misma. Para evitar lo mismo, implementamos una interfaz para cada paquete del proyecto interfaz. Se pensó implementar para cada formulario, pero conllevaba muchas clases nuevas que dificultaba su entendimiento, por lo que se decidió acoplar por paquetes.

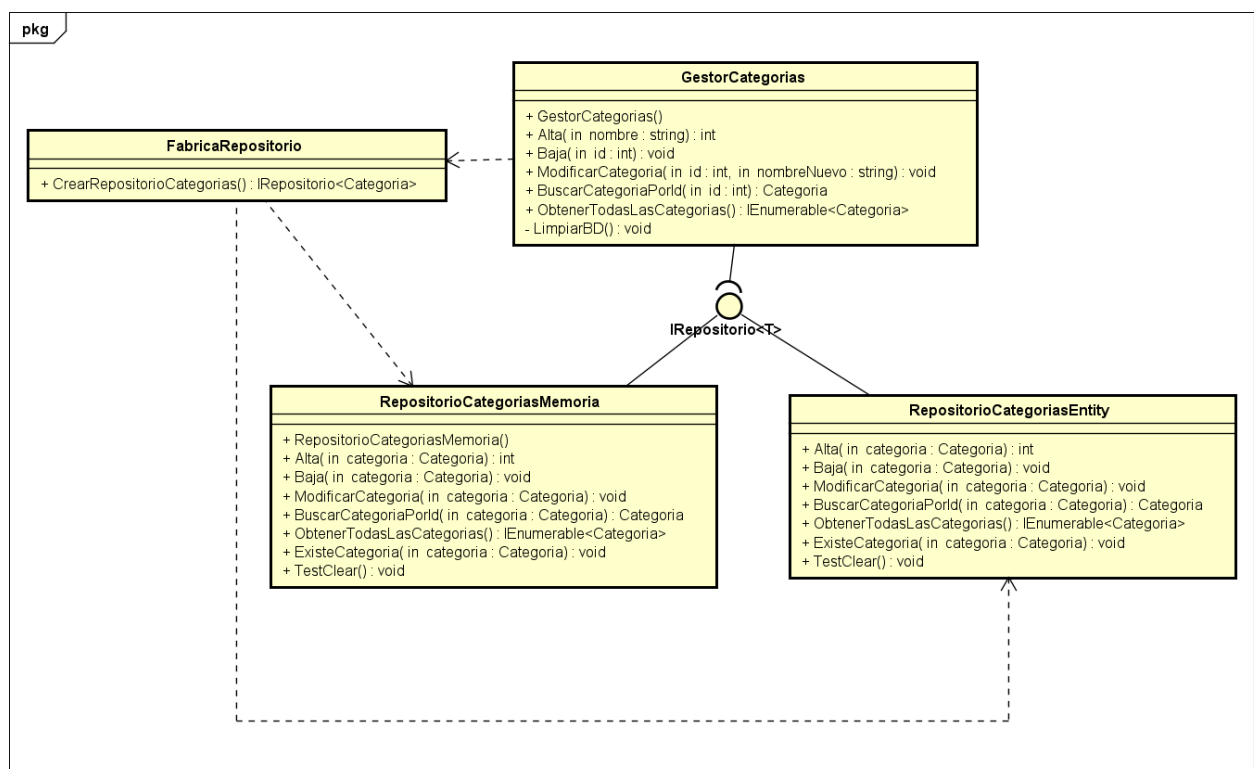
Almacenamiento de datos

Al almacenamiento de datos, a la opción de almacenar mediante listas en memoria, se le agrega la opción de realizarlo en base de datos usando **Entity Framework Code First**. Para el funcionamiento de esta se implementa que una clase concreta (Repositorio) se encargue del almacenamiento de datos, y ésta implementa una interfaz llamada **IRepositorio**. Dependiendo si queremos trabajar en memoria o en base de datos.

Debido al diseño utilizado en la primera entrega, no fue muy costoso adaptar el código a que funcione con la nueva modalidad. Conllevó crear el **IRepositorio** y adaptar los nombres de los métodos a la firma dispuesta.

A efecto de que sea más fácil implementar un nuevo cambio, se construye una “fábrica” de controladores, la misma es utilizada por el gestor correspondiente y crea los repositorios dependiendo del método de almacenamiento que se desee utilizar. Dicho método es almacenado en el archivo **App.config**, en la constante “**ENTORNO_PERSISTENCIA**”. El cambio de dicha constante entre “**Entity**” y “**Memoria**” establece qué repositorio debe utilizar la aplicación.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <configuration>
3   <configSections>
4     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral,
6   </configSections>
7   <entityFramework>
8     <providers>
9       <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
10    </providers>
11  </entityFramework>
12  <connectionStrings>
13    <add name="ContextoProd" connectionString="Server=.\SQLEXPRESS;Database=ObligatorioDA1Prod;Integrated Security=True" providerName="System.Data.SqlClient" />
14    <add name="ContextoTest" connectionString="Server=.\SQLEXPRESS;Database=ObligatorioDA1Test;Integrated Security=True" providerName="System.Data.SqlClient" />
15    <add name="ContextoUnitTest" connectionString="Server=.\SQLEXPRESS;Database=ObligatorioDA1UnitTest;Integrated Security=True" providerName="System.Data.SqlClient" />
16  </connectionStrings>
17  <appSettings>
18    <add key="DATABASE_CONTEXT" value="ContextoTest"/>
19    <add key="ENTORNO_PERSISTENCIA" value="Entity"/>
20    <!--<add key="ENTORNO_PERSISTENCIA" value="Memoria"/>-->
21  </appSettings>
22 </configuration>
```



Además de lo anterior, y a efecto educativo, tenemos la posibilidad de cambio entre bases de datos en tiempo de ejecución. El mismo archivo cuenta con otra constante llamada “DATABASE_CONTEXT” que me altera entra un “ContextoTest”, “ContextoProd” y “ContextoUnitTest”, de esta forma se puede utilizar una base de datos para las pruebas unitarias, una para producción y otra para los datos de prueba. Lo que le brinda al usuario de la aplicación una base de datos con todo precargado para que pueda experimentar la totalidad de la experiencia de usuario sin comprometer sus datos.

```

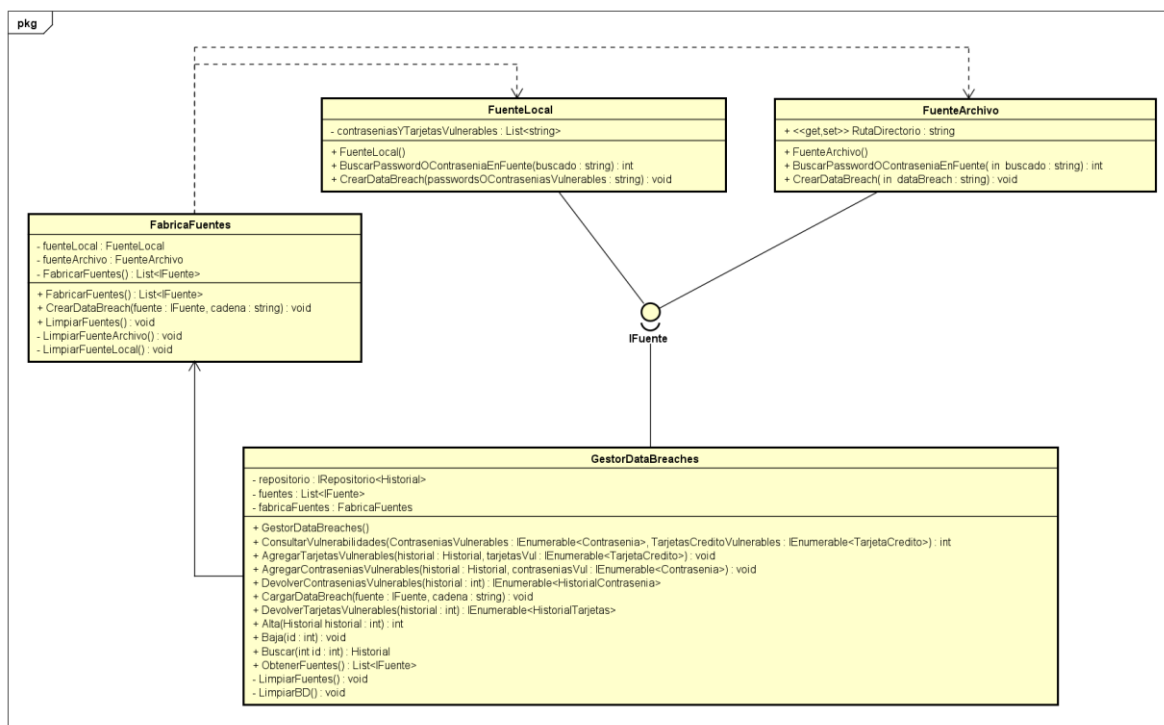
1 <?xml version="1.0" encoding="utf-8"?>
2 <configuration>
3   <configSections>
4     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral,
6   </configSections>
7   <entityFramework>
8     <providers>
9       <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
10    </providers>
11  </entityFramework>
12  <connectionStrings>
13    <add name="ContextoProd" connectionString="Server=.\\SQLEXPRESS;Database=ObligatorioDA1Prod;Integrated Security=True" providerName="System.Data.SqlClient" />
14    <add name="ContextoTest" connectionString="Server=.\\SQLEXPRESS;Database=ObligatorioDA1Test;Integrated Security=True" providerName="System.Data.SqlClient" />
15    <add name="ContextoUnitTest" connectionString="Server=.\\SQLEXPRESS;Database=ObligatorioDA1UnitTest;Integrated Security=True" providerName="System.Data.SqlClient" />
16  </connectionStrings>
17  <appSettings>
18    <add key="DATABASE_CONTEXT" value="ContextoTest"/>
19    <add key="ENTORNO_PERSISTENCIA" value="Entity"/>
20    <!--<add key="ENTORNO_PERSISTENCIA" value="Memoria"/>-->
21  </appSettings>
22 </configuration>

```

Diseño para chequear Data Breaches

En el obligatorio anterior, previendo una nueva implementación de fuente, se creó una IFuente que reglamentaba la firma que necesitaba implementar las fuentes de data breaches. Al implementar la nueva funcionalidad de guardar con archivos de texto, mantuvimos casi en su totalidad el primer código utilizado, hubo ciertas adaptaciones que se debieron a confusiones del estilo de “¿qué es un databreach?”. Para mantener ambas funcionalidades, y debido a que la lógica de las data breaches creció con este nuevo obligatorio, se crea el Gestor de Data Breaches.

El gestor cuenta con una lista de IFuentes, las que son creadas por una “fábrica” de IFuentes. De esta forma al implementar una nueva fuente de data breaches, se debe únicamente modificar el código de esa sección.



En el obligatorio se implementa la funcionalidad de cargar databreaches desde un archivo, la forma de implementarlo es copiar dicho archivo a una carpeta creada en el directorio raíz de la aplicación. De esta forma mantenemos la fuente local sólo funcionando en memoria y la de archivos sólo con archivos. Al momento de chequear vulnerabilidades va hacia el directorio donde se almacena dicha información y chequea contra todas las fuentes almacenadas en el mismo.

Diseño de la Clase Historial

Por la cantidad de lógica necesaria para almacenar las búsquedas de vulnerabilidades, se crea la clase historial. La misma se crea al momento de verificar vulnerabilidades y almacena una lista de búsquedas de tarjetas vulnerables y otra de contraseñas vulnerables. De las tarjetas lo único que nos interesa es el número, y de las contraseñas, además del identificador de esta (a efecto de poder buscarla), se cuenta con el password que poseía en el momento de ser encontrada vulnerable. De esta manera se puede verificar si el password fue cambiado.

Como ya se explicó anteriormente, el historial no cuenta con una implementación en memoria, por lo que las pruebas fallan al intentar correrlo con almacenamiento en memoria.

Cobertura de pruebas unitarias

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
crist_DESKTOP-4A14HMC 2021-06-17 14.33...	948	19,13 %	4007	80,87 %
negocio.dll	808	27,78 %	2101	72,22 %
Negocio	0	0,00 %	275	100,00 %
Negocio.Categorias	0	0,00 %	41	100,00 %
Negocio.Contrasenias	0	0,00 %	346	100,00 %
Negocio.DataBreaches	0	0,00 %	226	100,00 %
Negocio.Excepciones	26	61,90 %	16	38,10 %
Negocio.InterfacesGUI	0	0,00 %	132	100,00 %
Negocio.Migrations	525	99,24 %	4	0,76 %
Negocio.Persistencia	10	8,93 %	102	91,07 %
Negocio.Persistencia.EntityFramework	6	0,72 %	824	99,28 %
Negocio.Persistencia.Memoria	241	100,00 %	0	0,00 %
Negocio.TarjetaCreditos	0	0,00 %	99	100,00 %
Negocio.Usuarios	0	0,00 %	36	100,00 %
pruebasunitarias.dll	140	6,84 %	1906	93,16 %

Las pruebas unitarias cubren todo el código de lógica utilizado para la aplicación. Se verifica la pérdida de cobertura de ciertas clases de la siguiente forma:

- 1- Fábrica de Repositorios: Debido a que las pruebas se corren con una sola instancia de persistencia (memoria o base de datos), la fábrica sólo para por una de las implementaciones, no cubriendo la otra.

```
private string entorno_persistencia;
5 referencias | Federico Alonso, Hace 1 día | 1 autor, 1 cambio
public FabricaRepositorio()
{
    entorno_persistencia = ConfigurationManager.AppSettings["ENTORNO_PERSISTENCIA"];
}

1 referencia | Federico Alonso, Hace 1 día | 1 autor, 1 cambio
internal IRepository<Usuario> CrearRepositorioUsuario()
{
    IRepository<Usuario> repositorio;
    if (entorno_persistencia.Equals("Entity"))
    {
        repositorio = new RepositorioUsuarioEntity();
    }
    else
    {
        repositorio = new RepositorioUsuarioMemoria();
    }
    return repositorio;
}
```

- 2- Usuario: Se implementa una clase para persistir al usuario dueño de la aplicación en la base de datos. Debido a que al ser un solo usuario no es necesario toda la implementación de la interfaz IRepository, hay parte del código que no queda cubierto.

```
1 referencia | Federico Alonso, Hace 15 horas | 1 autor, 3 cambios
public class RepositorioUsuarioEntity : IRepository<Usuario>
{
    string contexto = "name=" + ConfigurationManager.AppSettings["DATABASE_CONTEXT"];
    15 referencias | 1/1 pasando | Federico Alonso, Hace 1 día | 1 autor, 1 cambio
    public int Alta(Usuario entity)
    {
        using (Contexto context = new Contexto(contexto))
        {
            context.Usuarios.Add(entity);
            context.SaveChanges();
            return entity.Id;
        }
    }

    13 referencias | Federico Alonso, Hace 1 día | 1 autor, 1 cambio
    public void Baja(Usuario entity)
    {
        throw new NotImplementedException();
    }
}
```

- 3- Las excepciones no son cubiertas al 100% por la misma razón que en el obligatorio anterior, no pasa el código por las inner exceptions, pero se decide dejarlas para quedar con el código listo para nuevas implementaciones.

```
namespace Negocio.Excepciones
{
    27 referencias | Naokirlz, Hace 4 días | 1 autor, 1 cambio
    public class ExcepcionElementoYaExiste : Exception
    {
        0 referencias | Naokirlz, Hace 4 días | 1 autor, 1 cambio
        public ExcepcionElementoYaExiste() : base() { }

        8 referencias | Naokirlz, Hace 4 días | 1 autor, 1 cambio
        public ExcepcionElementoYaExiste(string message) : base(message) { }

        0 referencias | Naokirlz, Hace 4 días | 1 autor, 1 cambio
        public ExcepcionElementoYaExiste(string message, Exception innerException) : base(message, innerException) { }
    }
}
```

Al realizar el cambio de almacenamiento y pasar a memoria en lugar de base de datos, se tiene que las pruebas relacionadas con el historial no funcionan, esto se debe a que no está implementado el repositorio en memoria para los historiales.

Prueba	Duración
✖ PruebasUnitarias (211)	12,1 s
✖ PruebasUnitarias (211)	12,1 s
✔ PruebasCategoriaGUI (7)	10 s
✔ PruebasConfiguracionGUI (10)	751 ms
✔ PruebasContrasena (17)	7 ms
✔ PruebasContraseniaGUI (10)	89 ms
✔ PruebasFuenteArchivo (4)	74 ms
✔ PruebasGestorCategoria (15)	21 ms
✔ PruebasGestorContrasenia (50)	93 ms
✔ PruebasGestorTarjetaCredito (25)	33 ms
✖ PruebasHistorial (19)	572 ms
✔ PruebasSesion (31)	305 ms
✔ PruebasTarjetasGUI (4)	31 ms
✔ PruebasUsuarioGUI (7)	43 ms
✖ PruebasVulnerabilidadesGUI (12)	77 ms
✔ AciertaDosVecesLasContraseniasVuln...	12 ms
✖ ElHistorialDevuelveContraseniasVulner...	13 ms
✖ ElHistorialDevuelveTarjetasVulnerables	10 ms
✔ Encuentra2VecesTarjetaVulnerableEn...	3 ms
✔ Encuentra2VecesUnaContraseniaVuln...	3 ms
✔ EncuentraContraseniaVulnerableEnFu...	6 ms
✔ EncuentraTarjetaVulnerableEnFuente	3 ms
✖ SeDevulvenHistoriales	5 ms
✔ SePuedeEjecutarBuscarContraseniaEs...	5 ms
✔ SePuedeEjecutarMostrarPasswordEst...	3 ms
✖ SePuedeGenerarUnHistorial	11 ms
✔ SiApareceMuchasVecesVulnerableDe...	3 ms

Decisiones de diseño relevantes

Se utiliza un patrón controlador, las interacciones del dominio con la interfaz se canaliza a través de un único punto de acceso que es la clase Sesion. La misma no tiene lógica, sino que deriva a cada controlador las peticiones de la interfaz correspondiente. Para no violar el principio ISP, se crean interfaces sobre las que interactúa cada paquete de la interfaz con las funcionalidades que debe conocer y no con todas. Con esta misma decisión estamos aplicando también el patrón fachada y singleton. Se guarda una instancia de sesión que se crea al momento de iniciar el sistema y se obtiene durante la implementación de las funciones en las diferentes ventanas de la aplicación. Es aplicada desde el obligatorio anterior, si bien no es tan necesaria en esta nueva versión (sólo aplica para la fuente local), al cambiar de entorno (de memoria a base de datos) se caen una gran cantidad de pruebas, por lo que se decidió mantener el singleton hasta una nueva restructuración de código donde sólo se utilizaría en los lugares necesarios.

Se decide utilizar el patrón experto, las ventanas en la interfaz son las que poseen toda la información para crear los objetos y los pasan hasta el repositorio para que los almacene. Además se utiliza el patrón experto para elegir el método Calcular fortaleza en la clase password.

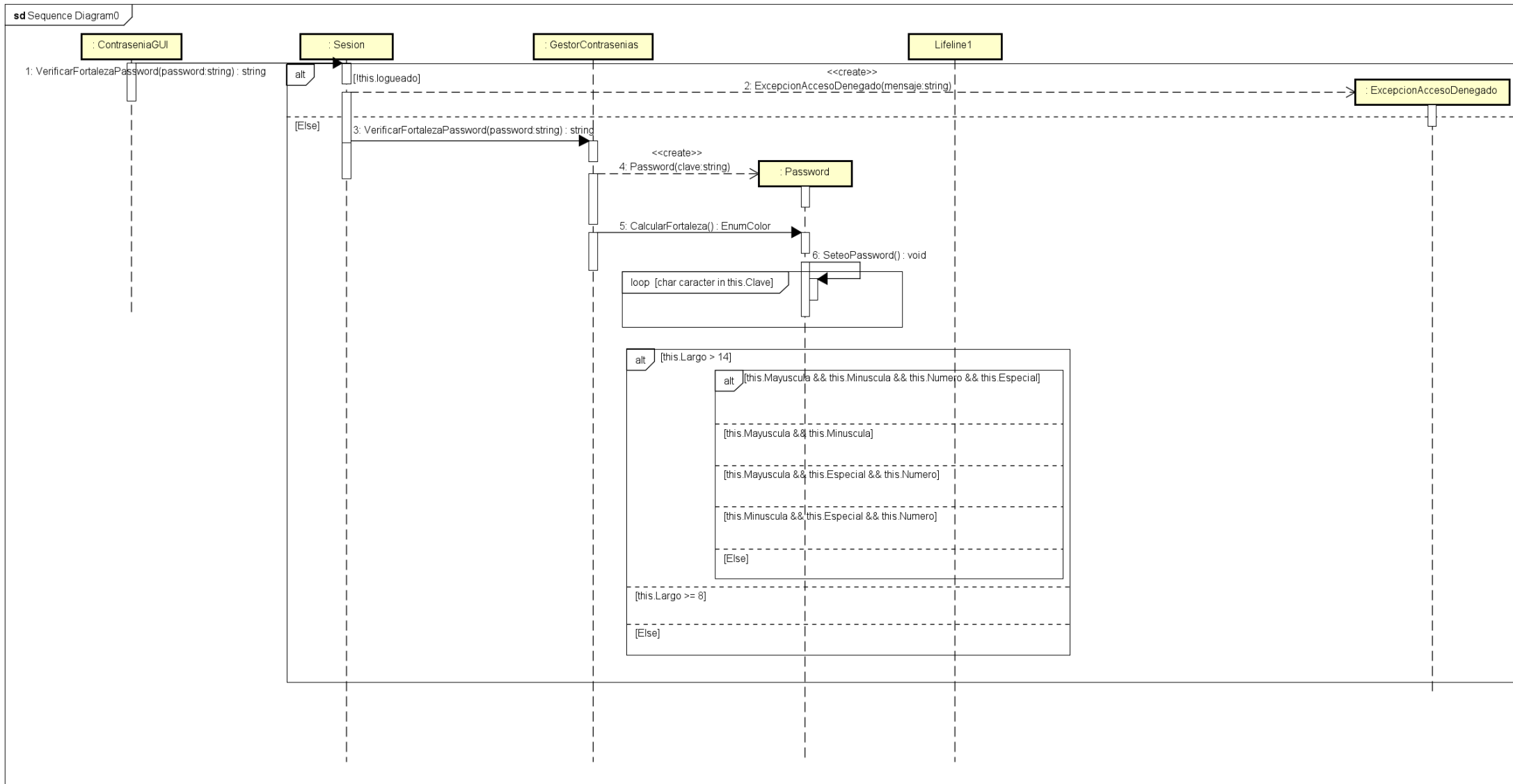
Al implementar el patrón controlador, también se favoreció el bajo acoplamiento y alta cohesión al separar la aplicación en diferentes controladores en base a su funcionalidad, cumpliendo con el principio SRP, manteniendo a cada clase con una única razón por la cual podrá ser modificada.

Se usa el principio OCP en la creación de repositorios y la creación de nuevas fuentes de data breaches. Debido a que estos son los supuestos puntos de variación de la aplicación, se crea una fábrica para cada uno de ellos, a efectos de que con la implementación de un nuevo sistema de almacenamiento (en la nube por ejemplo) o nuevas fuentes de databreaches (por ejemplo una API), no sea necesario modificar el código de los controladores correspondientes, sino que sólo en las fábricas.

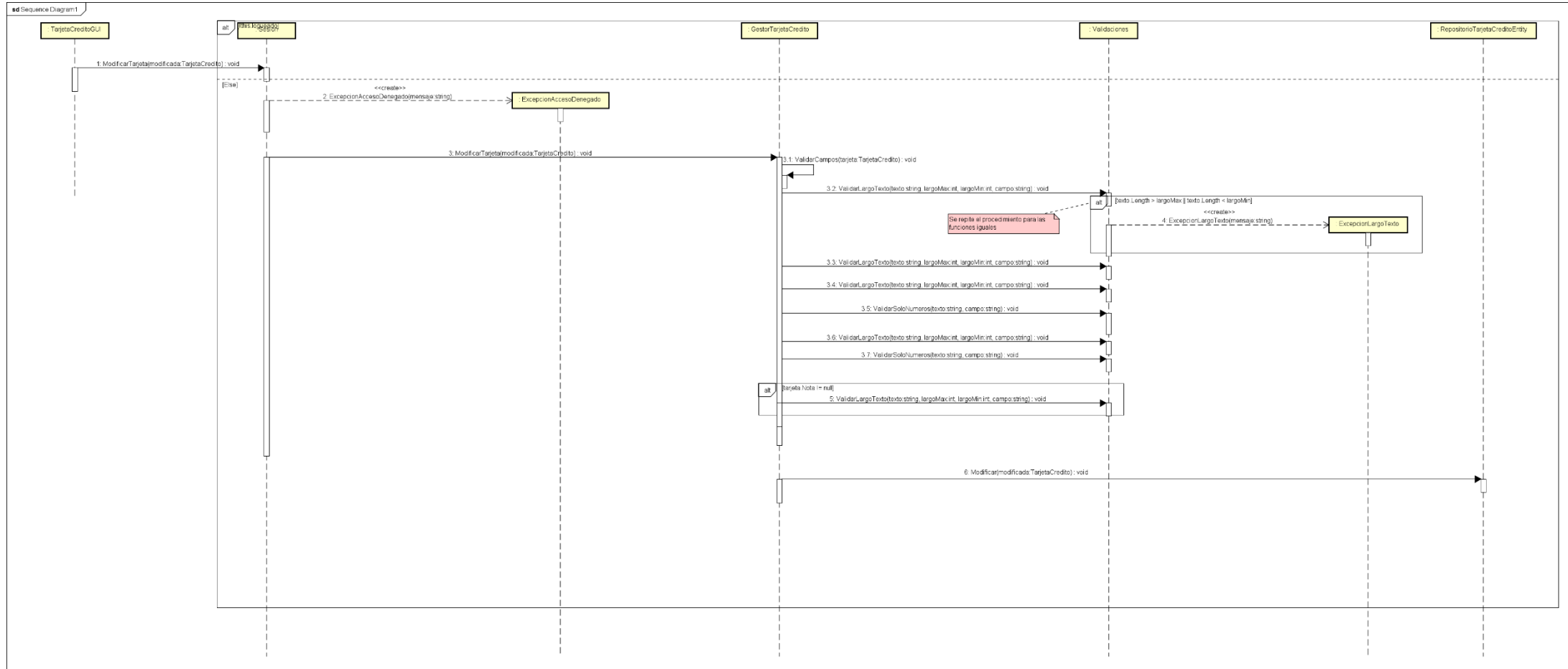
Se aplica el patrón strategy en la utilización de las fuentes, al ser contemplado su comportamiento a través de la implementación de una interfaz única para su funcionalidad. De esta forma se puede cambiar en tiempo de ejecución de fuente por ejemplo, y el sistema sigue funcionando como se espera

DIAGRAMAS DE INTERACCION

Verificar Fortaleza



Modificar Tarjeta de Crédito



Consultar Vulnerabilidades

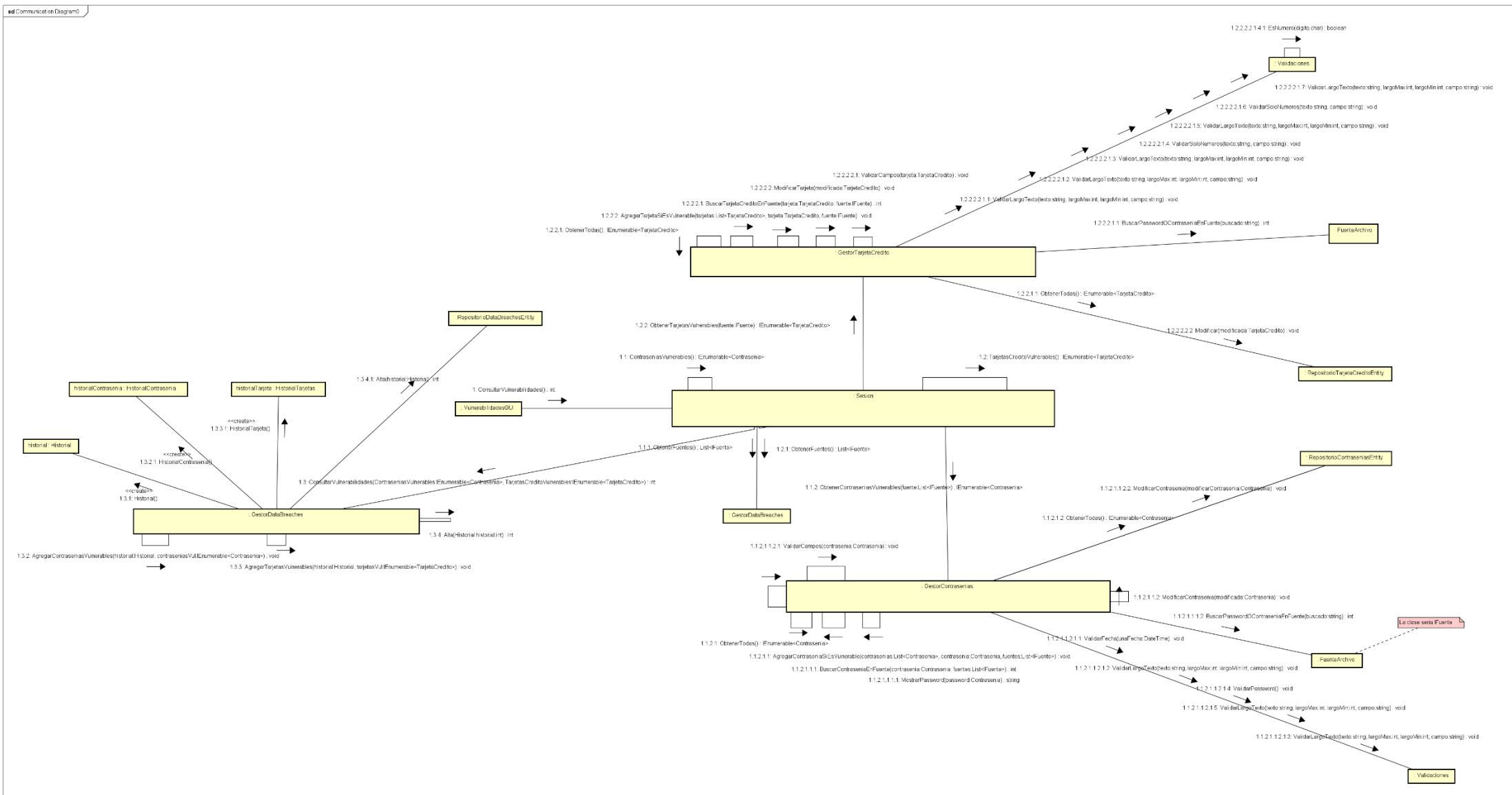
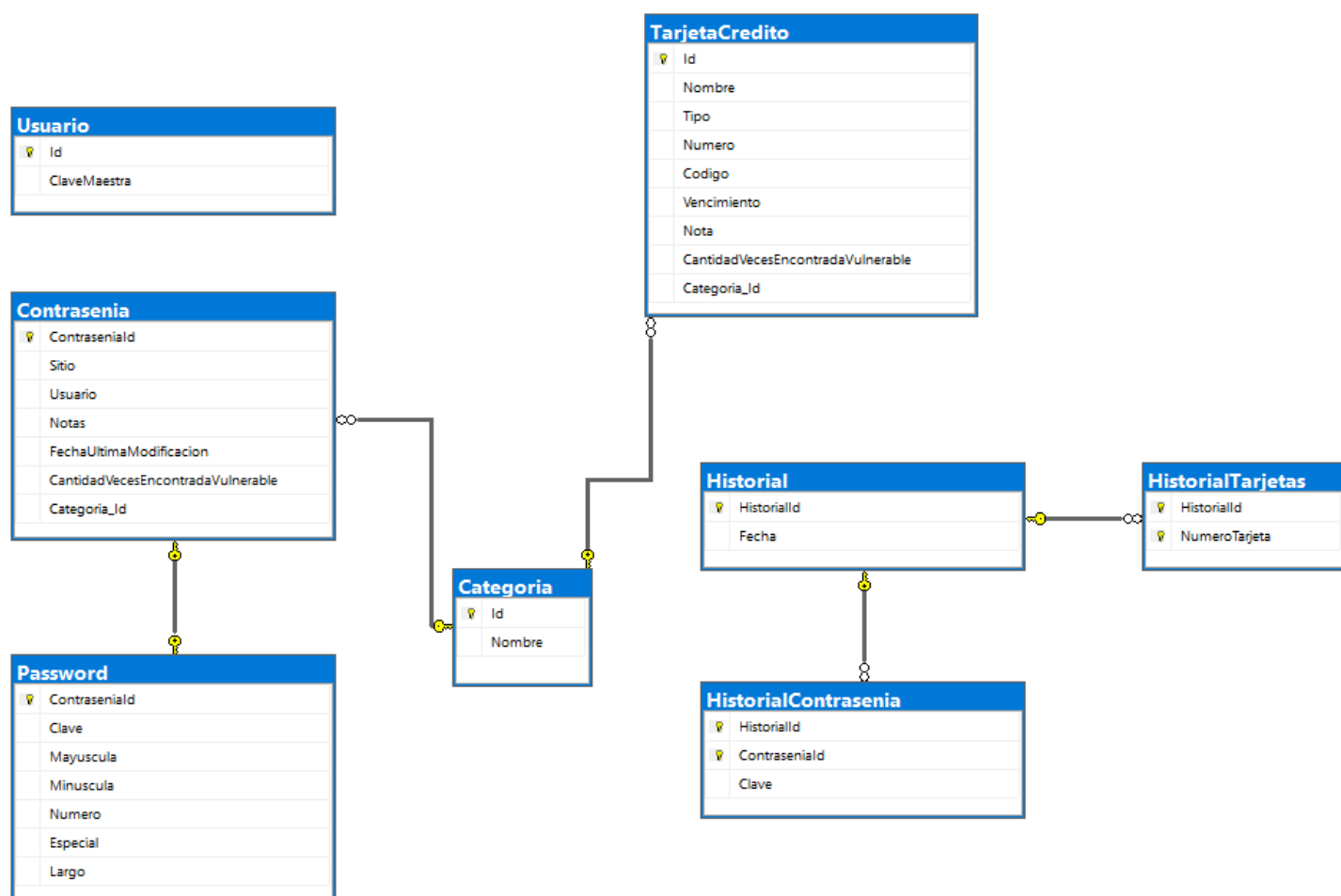
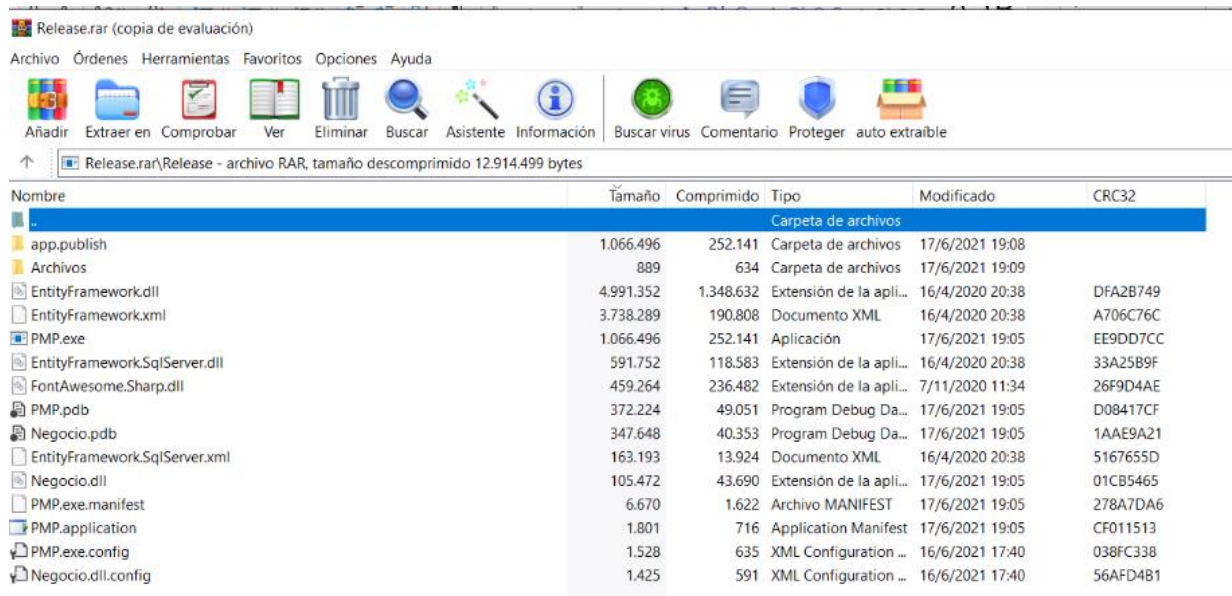


DIAGRAMA DE LAS TABLAS DE LA BASE DE DATOS



Proceso de instalación

1) Descomprimir la carpeta llama Release.rar.



Release.rar (copia de evaluación)

Archivo Órdenes Herramientas Favoritos Opciones Ayuda

Añadir Extraer en Comprobar Ver Eliminar Buscar Asistente Información Buscar virus Comentario Proteger auto extraíble

Release.rar\Release - archivo RAR, tamaño descomprimido 12.914.499 bytes

Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
Carpeta de archivos					
app.publish	1.066.496	252.141	Carpeta de archivos	17/6/2021 19:08	
Archivos	889	634	Carpeta de archivos	17/6/2021 19:09	
EntityFramework.dll	4.991.352	1.348.632	Extensión de la apli...	16/4/2020 20:38	DFA2B749
EntityFramework.xml	3.738.289	190.808	Documento XML	16/4/2020 20:38	A706C76C
PMP.exe	1.066.496	252.141	Aplicación	17/6/2021 19:05	EE9DD7CC
EntityFramework.SqlServer.dll	591.752	118.583	Extensión de la apli...	16/4/2020 20:38	33A25B9F
FontAwesome.Sharp.dll	459.264	236.482	Extensión de la apli...	7/11/2020 11:34	26F9D4AE
PMP.pdb	372.224	49.051	Program Debug Da...	17/6/2021 19:05	D08A17CF
Negocio.pdb	347.648	40.353	Program Debug Da...	17/6/2021 19:05	1AAE9A21
EntityFramework.SqlServer.xml	163.193	13.924	Documento XML	16/4/2020 20:38	5167655D
Negocio.dll	105.472	43.690	Extensión de la apli...	17/6/2021 19:05	01CB5465
PMP.exe.manifest	6.670	1.622	Archivo MANIFEST	17/6/2021 19:05	278A7DA6
PMP.application	1.801	716	Application Manifest	17/6/2021 19:05	CF011513
PMP.exe.config	1.528	635	XML Configuration ...	16/6/2021 17:40	038FC338
Negocio.dll.config	1.425	591	XML Configuration ...	16/6/2021 17:40	56AFD4B1

2) Ejecutar PMP.exe.

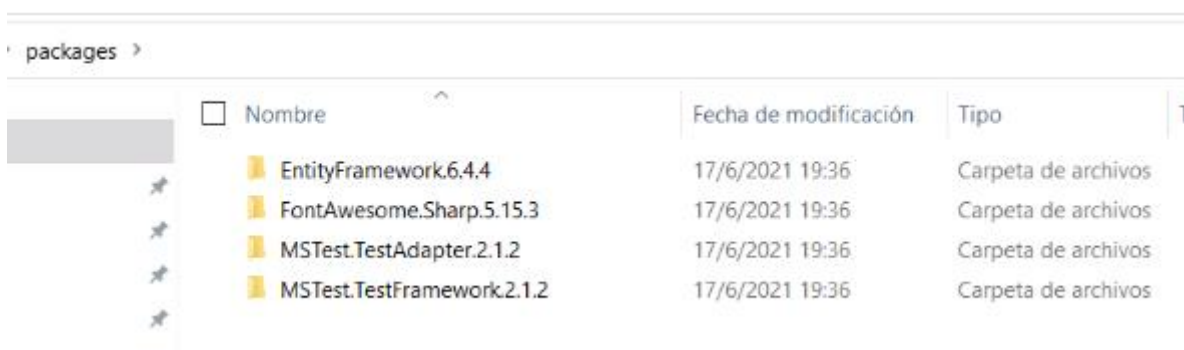
Se debe tener en cuenta que se debe estar corriendo el motor de bases de datos SQL SERVER. Además, la carpeta cuenta con todo lo necesario para funcionar, por lo que no debe desarmarse la misma.

Se cuenta con una carpeta llamada Archivos que es donde se encuentran las fuentes de databreachs de archivos, por lo que sólo debe haber archivos *.txt.

En caso de importar la base de datos que tiene datos en ella, la clave de ingreso es "cosas".

En la carpeta que contiene la entrega del obligatorio, en el directorio **Documentacion\2do obligatorio\Anexo A**, se encuentra el respaldo de la base de datos con datos de prueba, así como los scripts para generarlas.

Estos packages deben ser instalados nuevamente, al momento de ejecutar la aplicación en Visual Studio. (fueron eliminados porque pesan más de 50 MB).



packages >

Nombre	Fecha de modificación	Tipo
EntityFramework.6.4.4	17/6/2021 19:36	Carpeta de archivos
FontAwesome.Sharp.5.15.3	17/6/2021 19:36	Carpeta de archivos
MSTest.TestAdapter.2.1.2	17/6/2021 19:36	Carpeta de archivos
MSTest.TestFramework.2.1.2	17/6/2021 19:36	Carpeta de archivos