

**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](#_Toc71748414)

[Objetivo de la aplicación 3](#_Toc71748415)

[Errores conocidos 3](#_Toc71748416)

[Descripción y justificación de diseño 4](#_Toc71748417)

[Diagrama de paquetes 4](#_Toc71748418)

[Diagramas de clases 4](#_Toc71748419)

[Interacción de la Interfaz con el Dominio 5](#_Toc71748420)

[Almacenamiento de datos 5](#_Toc71748421)

[Diseño para chequear Data Breaches 6](#_Toc71748422)

[Diseño de la Clase Password 6](#_Toc71748423)

[Manejo de errores y excepciones 7](#_Toc71748424)

[Cobertura de pruebas unitarias 7](#_Toc71748425)

[Documentación de caos de prueba elaborados 8](#_Toc71748426)

# 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 usamos en otro sitio y la fortaleza de la contraseña (la cual esta clasificada en colores con los mismos criterios de la entrega anterior).

## Errores conocidos y/o mejoras para alguna siguiente versión:

* Se utiliza 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.
* USER PANEL CONTRASEÑA, ERROR DETECTADO PROFE
* 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.

Texto

Descripción generada automáticamente

* 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.

# 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:

Diagrama

Descripción generada automáticamente

En anexo “A” imagen 2.

## Diagramas de clases

A continuación, se muestra un diagrama de clases simplificado, donde se observan los atributos de las clases principales del dominio y la interacción entre las mismas. Se simplifica el diagrama quitándole las excepciones y los métodos a efectos de que sea legible, se adjunta como **Anexo\_1** el diagrama de clases completo.

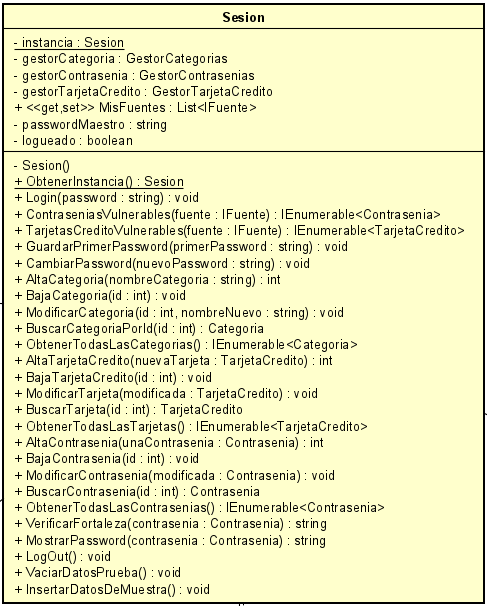
Diagrama

Descripción generada automáticamente

## Interacción de la Interfaz con el Dominio

En el sistema realizamos una separación entre la interfaz y la capa de negocio, su único vínculo es la clase Sesión que actúa de **“fachada”** ya que la misma no contiene lógica, si no que recibe los requerimientos de la interfaz y los deriva a la clase que corresponda. Esta decisión fue tomada al momento de hacer clean code y aplicar la ley de Demeter.

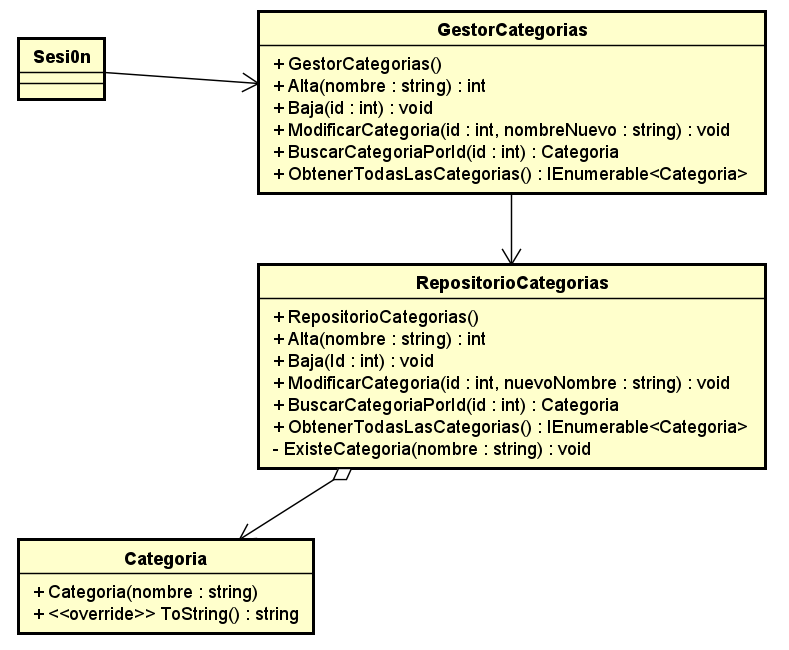
A continuación, se muestra la clase Sesión con sus atributos y métodos.



## Almacenamiento de datos

El almacenamiento de datos en esta etapa se hace mediante listas en memoria, cuando se termina la ejecución de la aplicación la información no persiste. A pesar de esto, se diseñó la aplicación pensando que esta situación iba a cambiar a futuro, entonces la lógica específica para la manipulación de listas se encapsuló en las clases repositorios de cada entidad (Categorías, Contraseñas y Tarjetas).

De esta forma los gestores desconocen la lógica de guardar la información, facilitando el cambio de la forma de persistencia en una segunda etapa del proyecto, a continuación, se muestra un ejemplo de la clase categoría.



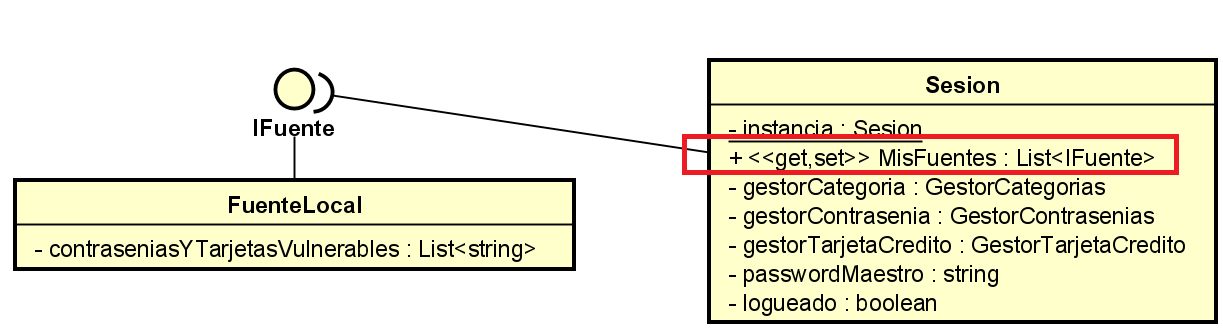
## Diseño para chequear Data Breaches

Siendo que por el momento se cuenta con una fuente local, y que en el futuro se puede contar con otro tipo de fuente, se decide implementarla a través de una interfaz **IFuente.**

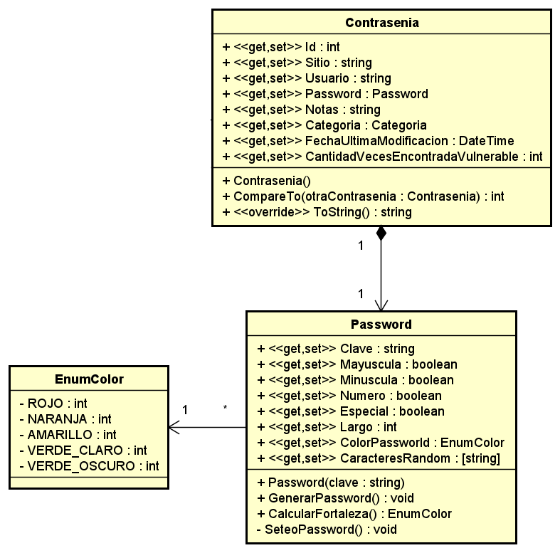
Esta interfaz posee la firma de los métodos necesarios para encontrar contraseñas y/o tarjetas de créditos vulnerables y retornarnos cuantas veces fueron halladas vulnerables.

Actualmente solo se cuenta con una fuente sola, la cual llamamos “**FuenteLocal”**, que consiste en una lista de strings. Esta clase **Implementa** la interfaz IFuente y pone lógica a los métodos, luego la clase **Sesión** **Usa** lainterfaz ya que cuenta con una lista de elementos IFuentes.

De esta forma a futuro si nos agregan otra fuente, no tenemos que modificar el código existente, solamente debemos agregamos una clase nueva que implemente la interfaz IFuente y gracias al polimorfismo podemos mandar el mismo mensaje (Ej buscar contraseña y/o tarjeta en fuente) a cualquier objeto de la List<IFuente>.



## Diseño de la Clase Password

Al finalizar las funcionalidades básicas, nos dimos cuenta de que la clase correspondiente al guardado de contraseñas contenía demasiado lógica, por lo que se decidió crear una clase Password, la cual es la encargada de la lógica correspondiente a la clave de la contraseña, auto creación del mismo, reporte de fortalezas, etc.

## Manejo de errores y excepciones

Las excepciones en la aplicación fueron creadas de forma genérica, por ejemplo, “**ExcepcionLargoTexto”**, “**ExcepcionElementoFaltante”**, etc.

Las mismas son utilizadas para todas las clases en las que se realizan validaciones, diferenciando en ellas un mensaje personalizado de acuerdo con el error especifico que la haya disparado. De esta forma evitamos que exista un paquete repleto de excepciones ocurridas por cada atributo controlado. Las excepciones son controladas por la interfaz, quien tiene control de estas, mostrándole al usuario de forma oportuna cuando alguna de ellas ocurre.

Se decide “trimar” los campos de texto, a efecto de que los caracteres espacio en blanco no ocupen mayor lugar del necesario, esto no se toma en cuenta para los “*password*”, debido a que se decidió en la letra del foro de que el espacio en blanco fuera tomado como carácter especial.

# Cobertura de pruebas unitarias

Los casos que no llegan al 90% de cobertura corresponden a las clases creadas para manejar nuestras propias Excepciones como se muestra en la siguiente foto:

# Documentación de casos de prueba elaborados