

Obligatorio 2 - Diseño de aplicaciones

Gastón Landeira - 238473
Matías González - 219329
Iñaki Etchegaray - 241072

Profesor: Darío Alejandro Macchi Heins
de Tecnologías: Balduccio - Anduano

Universidad ORT Uruguay

ÍNDICE

Vista General	2
Clean Code	2
GitHub y TDD	4
Persistencia de Datos	4
Diseño	6
Soporte Multiusuario	8
Alta, Modificación y Listado de Categorías	9
Alta, Baja, Modificación y Listado de Contraseñas	10
Alta, Baja, Modificación y Listado de Tarjetas de Crédito	12
Detección de Data Breaches	14
Reporte de Fortaleza de Contraseñas	15
Encriptación	17
Sugerencia de contraseñas	18
Nuevo formato de Data Breaches	18
Historial de Data Breaches	19
Manejo de Excepciones	20
Testing	20
ANEXO	21

Vista General

La aplicación es un gestor de contraseñas y tarjetas de crédito con funcionalidades que permiten a varios usuarios entrar y poder compartir contraseñas, ver la seguridad de las mismas e incluso generarlas si lo desea. La idea es realizar una aplicación con código profesional por detrás, aplicando técnicas de código que aseguren su escalabilidad en el futuro.

En esta segunda instancia se le agregó la persistencia de datos a través de una base de datos, la encriptación de las contraseñas así como las sugerencias de estas. Un historial de data breaches y una nueva forma de subir estos mismos.

La solución se compone de cinco sub-proyectos:

- Domain, donde yace la lógica del proyecto.
- Domain Tests, donde se realizan los tests de la lógica del proyecto.
- Repository, donde yacen los accesos a la base de datos
- RepositoryTests, donde se realizan los tests de los accesos a la base de datos
- User Interface, donde encontramos los componentes visuales de la aplicación.

Cada uno de estos proyectos tienen su estructura interna que será mencionada más detalladamente en el documento.

Antes de comenzar, una **lista de los bugs** conocidos del proyecto:

- La sucesión: Breachear contraseña, Abrir historial, Ver ese breach, Modificar esa contraseña en particular, Volver a breachear la vieja. Tira error de manera inconsistente.
- Una vez, nos tiró error de base 64. No sabemos reproducir
- La librería Efortless no es perfecta y en ocasiones descripta de manera incorrecta y deja información corrupta (En DataBreach history al modificar una Password)

Notas generales del proyecto:

- Nuestro proyecto de clases de negocio posee un archivo Program.cs, que constituye la entrada del proyecto Domain. Quisimos quitarlo ya que la entrada verdadera es por el Program de User Interface, pero hacer esto no permitía que corriera el programa.

Clean Code

Aplicamos técnicas de Clean Code a nuestro código. Para asegurarnos la consistencia nos definimos una lista de Standards de Codificación:

Nombres

- Para atributos públicos usamos auto-properties o properties y comienzan con mayúsculas.
- Para atributos privados comienzan con un “_”.
- Para variables locales y parámetros comienzan con minúscula.
- Si la variable posee más de una palabra de nombre, usar camelCase.
- Uso abierto de constantes const o readonly.
- Las clases comienzan con mayúsculas.

- Para Interfaces comienzan con I.
- Utilización de Paquetes para funcionalidades que se encapsulan solas.
- No tenerle miedo a los nombres de métodos muy largos.
- Ser lo más descriptivos con los nombres de tests que se permita.

Formato:

- El largo de una línea no puede ser tan larga que no se pueda leer toda la función en una pantalla.
- Aceptar a lo sumo 3 parámetros, no más.
- Evitar los comentarios lo más posible, al menos que hayan cosas que aclarar.
- Atributos los más arriba de la clase posible.
- Funciones abajo en orden de más abstracta a menos abstracta.
- Auto-properties en una línea.
- Separación de un enter entre funciones.
- Indentar con 4 espacios o un tab entre brackets abiertos.
- El primer y último bracket deben tener sus propias líneas.
- Separar operadores con un solo espacio.
- El código debe de estar en Inglés.

Sufijos para los tipos de ventanas:

- NombreWindow para ventanas grandes.
- NombreModal para los modals.
- NombreController para los UserController.

Prefijos para los elementos:

- Boton: btnNombre
- Textbox: txtbxNombre
- Label: lblNombre
- Panel: pnlNombre
- DataGridView: grdvwNombre
- ComboBox: cmbbxNombre
- ListView: lstvwNombre
- Number: numNombre
- Checkbox: chkbxNombre
- FlowLayoutPanel: fwlytNombre
- Chart: chrtNombre

En el caso de la interfaz, no pudimos cumplir que los métodos comiencen con mayúscula ya que son auto-generados por Windows Forms. Para esta segunda instancia los estándares de codificación no cambiaron en lo absoluto.

GitHub y TDD

Para el versionado del proyecto utilizamos GitHub y nos basamos en el framework de trabajo Git Flow, ya que se ajusta de manera perfecta a la técnica de diseño TDD.

Nuestro repositorio tiene una rama principal main, la cual está destinada a versiones release del proyecto. De la rama main se creó una rama develop, en donde expandimos a distintos branches para realizar las features propuestas. Además, nos propusimos nombrar a las branches con nombres en secuencia dependiendo de donde surgían. Por ejemplo, si alguien trabajaba en la interfaz de usuario para las contraseñas, esa branch tendría el nombre: “develop-UserInterface-Passwords”, esto hace que sea más fácil seguir de donde surge la branch y a donde tendría que mergeear.

Git Flow también nos ayudó a separar las responsabilidades de cada Usuario a distintos branches, para que podamos trabajar tranquilos sin preocuparnos por conflictos constantes.

Utilizamos la técnica de diseño TDD para desarrollar el dominio y el repositorio. Es decir, nos ajustamos a realizar las clases de negocio y los data access creando primero sus pruebas unitarias, luego implementarlas en la lógica, y finalmente realizar el refactoring de código.

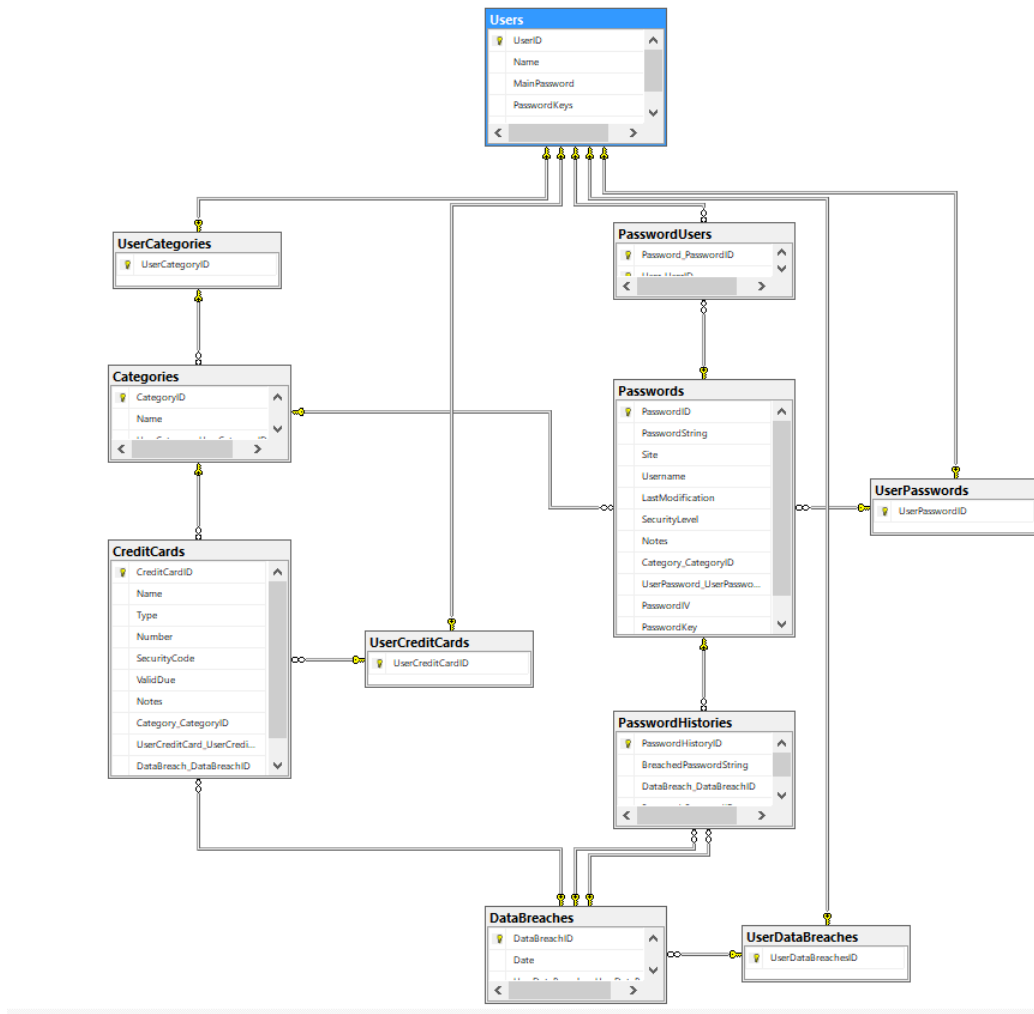
Aprendimos de nuestro error del Obligatorio pasado, y aplicamos más commits por stage de TDD donde fuera aplicable. Aun así, admitimos que hubieron varios commits que fueron más grandes de lo que deberían ser. Estos se dieron en situaciones donde el TDD no era claro de aplicar o en las que había que realizar muchas tareas para cumplir una implementación.

Link al repositorio del proyecto:

https://github.com/ORT-DA1/238473_219329_241072/tree/main

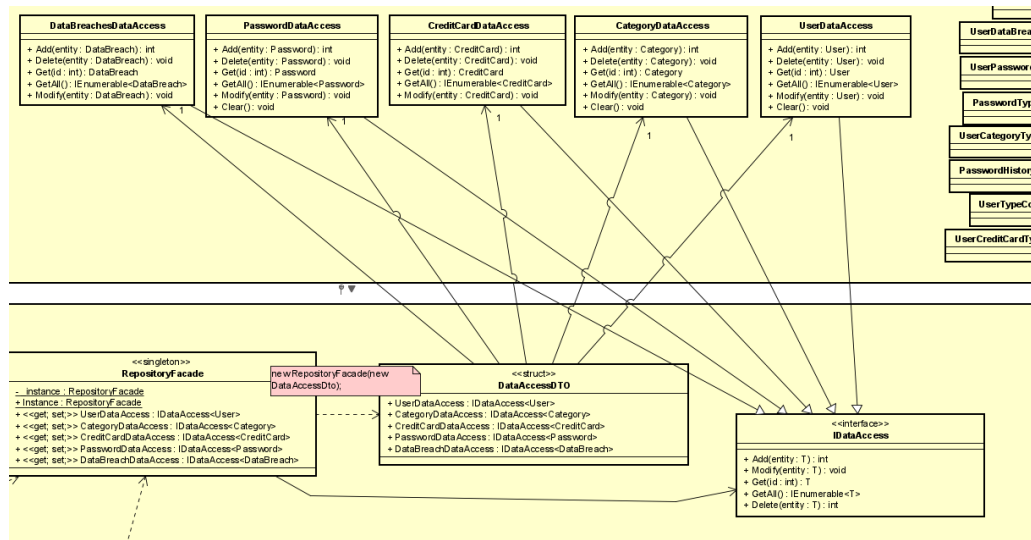
Persistencia de Datos

Para la persistencia de datos utilizamos un enfoque code first lo cual significa primero hacer el dominio de la aplicación creando las clases adecuadas para el mismo y luego una vez el dominio está terminado hacer la base de datos. Para hacer esto posible utilizamos Entity Framework 6.0 que permite crear una base de datos a partir de las clases del dominio sin la necesidad de tener que realizar grandes cambios en las mismas. Igualmente algunos cambios tuvieron que ser realizados para simplificar el acceso a las distintas tablas y la navegabilidad entre ellas. Además, se agregó un historial de data breaches lo cual no solo necesitaba la persistencia de los mismos sino pequeños cambios a su lógica y funcionamiento. Todos los cambios necesarios para la adaptabilidad de cada clase a Entity Framework serán documentados debajo en más detalle. La base de datos resultó en la siguiente tabla de entidades:



Para facilitar la interacción entre el dominio y el repositorio se implementó una interfaz `IDataAccess<T>`, donde `T` es la clase a la que se accede en el dominio, con los métodos básicos de interacción como son `Add`, `Modify`, `Get`, `GetAll` y `Remove` de la cual luego se extendieron los `DataContext` de cada clase con estos métodos adaptados a las mismas. Por último, se implementó un Singleton que funciona como Fachada del repositorio en el dominio, la cual permite utilizar los data access y así persistir los datos en la base.

Es importante notar la decisión de diseño de incluir el `IDataAccess` en Domain y no en el paquete del repositorio. Esto se debe a que se aplicó el patrón DIP (Dependency Inversion Principle) de los patrones SOLID. Quisimos que Domain dependiera de una abstracción de los `DataContext` para interactuar con Repository, y no en las concreciones de los mismos.



IDataAccess es, además, una Inyección de Dependencia. Es decir, si no fuera por esta interfaz, Domain no podría acceder a los DataAccess, ya que eso implicaría una dependencia circular entre Repository y Domain, razón por la cual la interfaz se encuentra en Domain y no en Repository.

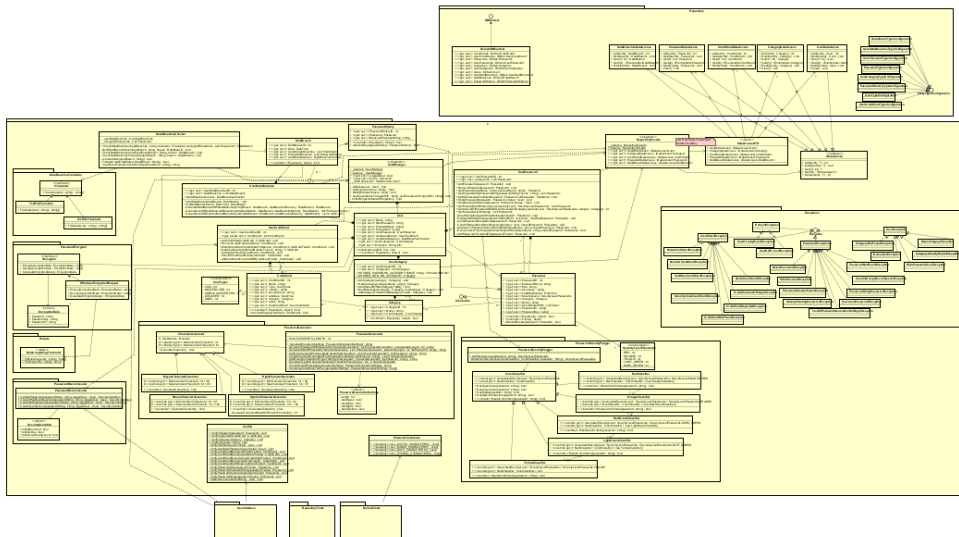
Domain no se preocupa por lo que suceda en Repository, ni cómo interactúe con la base de datos, solo le interesa que le traiga o modifique la información pertinente. Esto hace que el día de mañana, si cambia la base de datos o si Repository tiene que ser modificado extensamente, no haya que tocar mucho código de Domain.

La inyección fue, además, muy útil para el testeo de Domain ya que se pudo crear una clase MockDataAccess la cual extiende de IDataAccess. Luego en MockDataAccess simulamos una base de datos local muy simple, de esta manera se prueba en Domain solamente la funcionalidad de Domain.

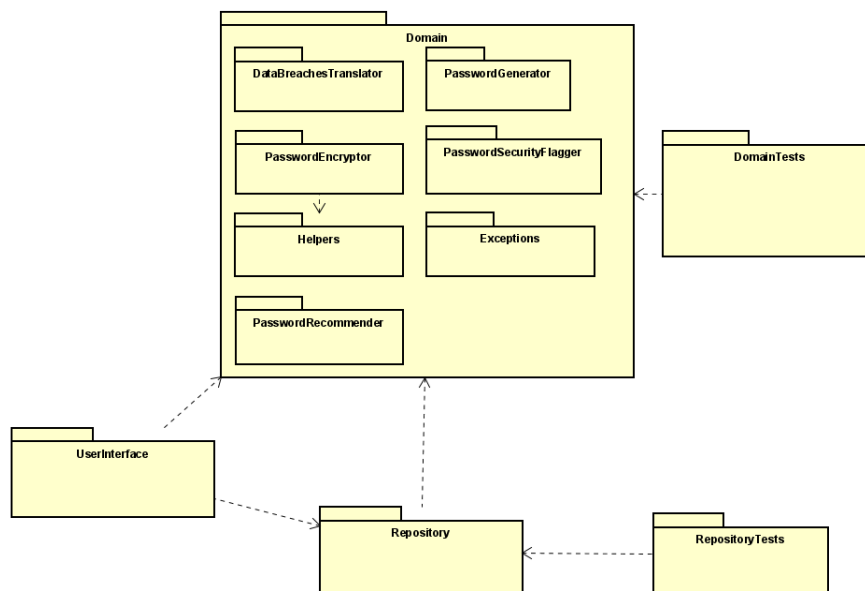
Diseño

Para lograr comunicar bien el proyecto y mejorar la experiencia de desarrollo del mismo, lo primero que hicimos fue crear un diagrama UML de las clases de negocio. Por supuesto que este diagrama fue cambiando a lo largo del camino pero, nos definió las ideas fundamentales de cómo el proyecto iba a ser organizado por dentro.

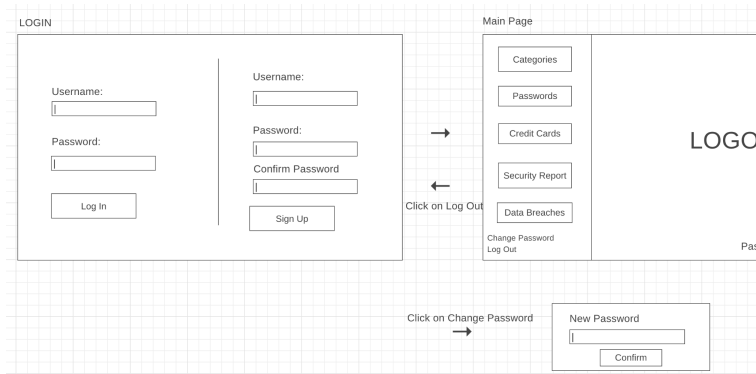
Este diagrama UML cubre todas las clases de negocio con sus interacciones, y además las clases de Interfaz y su interacción con las clases de negocio.



(El diagrama UML se encuentra en el proyecto junto a este PDF)
Además se realizó un diagrama de paquetes también incluido dentro del proyecto:



Además, decidimos realizar un Wireframe de la interfaz del usuario para poder previamente decidir cómo sería la composición general de la aplicación visualmente antes de empezar a hacerla. Esto generó discusión entre los integrantes del grupo que, en caso de tener que deshacer algo, utilizando esta herramienta es tan fácil como borrar los componentes y reorganizarlos. Mucho más barato que hacerlo en el camino con código ya escrito.



Notar que el Wireframe puede tener algunas diferencias con el resultado final, ya que era un rough de la interfaz para asistir en el desarrollo. Pero las ideas principales se mantienen.

Link al wireframe completo:

<https://wireframe.cc/pro/pp/336489a28439365>

Soporte Multiusuario

Al tener que diseñar el soporte multiusuario, identificamos que:

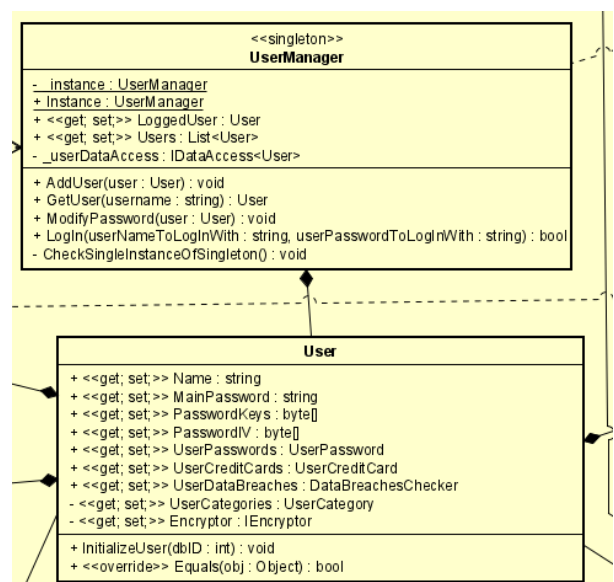
- La aplicación debe de tener una lista de usuarios registrados en memoria identificados por su nombre.
- Debe haber una noción de sesión, lo cual implica un log in y un log out.
- Implica la existencia de un perfil básico, con un nombre identificador y una contraseña. Esto implica un sign up y una edición de contraseña.

A base de estas necesidades definimos la clase UserManager, cuya responsabilidad es contener a todos los usuarios existentes con su información, todo lo que implique acceso a los usuarios y, la clase User, la cual representa al usuario en la aplicación. Esta clase es el centro de toda la aplicación y el acceso a la mayoría de la funcionalidad.

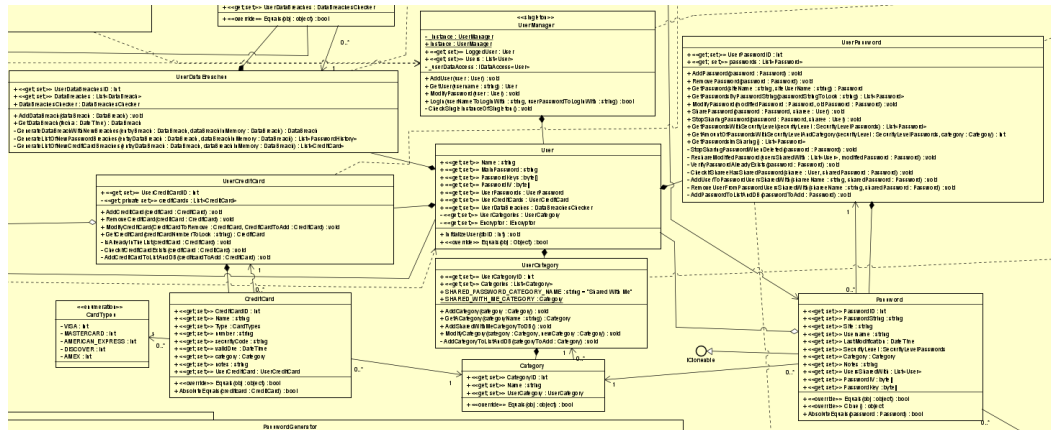
El siguiente paso fue adaptar esto a entity framework para poder conseguir su persistencia en la base de datos. Para esto primero se lo transformó en un singleton el cual almacena, además del usuario activo actualmente y una lista local con todos los usuarios, también el data access de User que le permite interactuar con la tabla de Usuarios. Este data access origina de la fachada mencionada en la parte Persistencia de Datos. El resto implicaba agregar las sentencias de agregado y modificado en los métodos correspondientes.

Esto en el UML se manifiesta de la siguiente manera:

Como se ve en el UML, la presencia de sesión se da en el atributo LoggedUser de UserManager. Los métodos de UserManager están relacionados con obtener o modificar usuarios, el login y operaciones que requieren de una visión global de los usuarios registrados.



El User se encarga del registro de su categorías y el guardado de las mismas, también se va a encargar de todo lo relacionado a las contraseñas y tarjetas de crédito. Esto causó que la clase User fuera una clase “monolítica”. Es decir, era una clase con muchas responsabilidades y además muy larga para que sea fácil de leer. Para solucionar esto, dividimos las responsabilidades de la clase User en otras clases como se muestra en el siguiente diagrama: (UserPassword, UserCreditCard, UserCategory y UserDataBreaches)



Lo que hacen estas clases y sus responsabilidades estará cubierto más adelante en el documento.

La interfaz interactúa con UserManager de la siguiente forma:

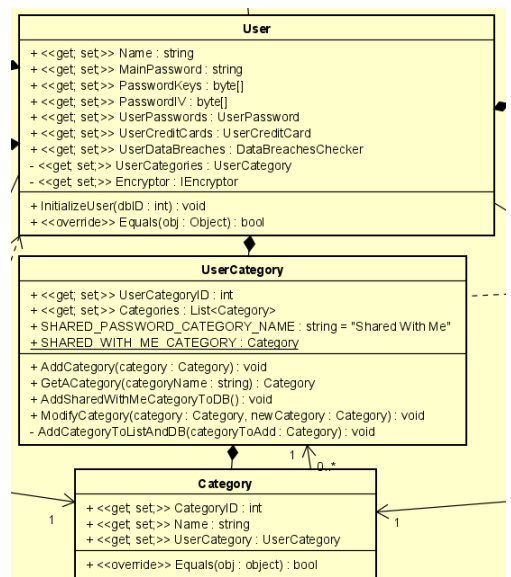
- Al darle al botón de Log In, corre la función Log In.
- Al hacer un sign up, accede a la función de agregar un usuario.
- Al tocar Change Password, accede a la función de cambio de contraseña de un usuario.

Alta, Modificación y Listado de Categorías

En práctica, las Categorías son simplemente un string, ya que se compone solamente de un nombre.

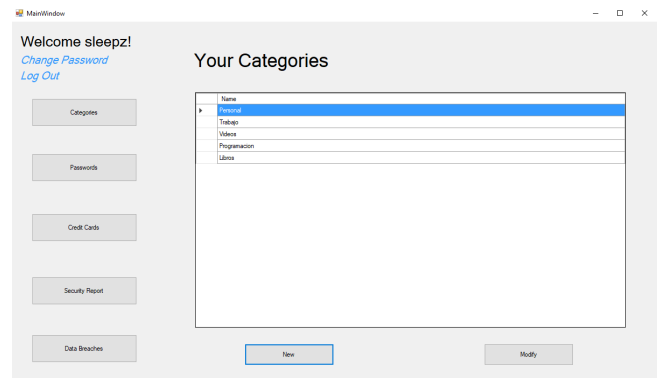
Sin embargo, nos parece que es un participante importante en el sistema y decidimos crear una clase `Categoría` que posee un nombre. Esto permite que la funcionalidad de `Categoría` sea mucho más escalable, ya que si llega a haber funcionalidad nueva ligada a `Categoría`, podemos agregarla en su clase.

Como se mencionó anteriormente, si bien el User es el encargado de manejar sus categorías, contraseñas, data breaches y tarjetas de crédito, mantener todo eso en una sola clase la volvería muy monolítica por lo cual lo dividimos en bloques. En este caso el usuario tiene una clase `UserCategory` la cual tiene la lista de categorías de ese usuario y se encarga de las lógicas sobre las mismas:



La persistencia de las categorías no dio problema, simplemente se tuvo que adaptar el código para utilizar la Fachada de repositorio y así actualizar la base de datos conforme se crean/modifican categorías.

Listamos las categorías con un DataGridView en Winforms que accede a la lista del usuario logueado. Los botones de Add y Modify interactúan directamente con las funciones de User respectivas.



Alta, Baja, Modificación y Listado de Contraseñas

Para el manejo de sus contraseñas cada usuario tiene un atributo UserPasswords que guarda una lista con todas sus contraseñas. A su vez utilizamos una clase Password que almacena toda la información correspondiente a una contraseña(username, site, password, etc), entre ellos almacenamos una lista de Users con los usuarios a los que la misma está siendo compartida en ese momento y un ENUM securityLevelPassword donde se guarda el color correspondiente al nivel de seguridad de esa contraseña. Además la misma tiene dos atributos IV y Key los cuales son utilizados en la encriptación de las mismas dentro de la base de datos brindando mayor seguridad al sistema y un atributo UserPassword simplemente para facilitar la navegabilidad al utilizar Entity Framework.

Para su persistencia, al igual que con las categorías, se creó un Password Data access que se encarga de interactuar con la base de datos. A este data access accedemos a través de la fachada de repositorio permitiendo así realizar todas las acciones necesarias con respecto a el guardado, obtención y modificación de contraseñas.

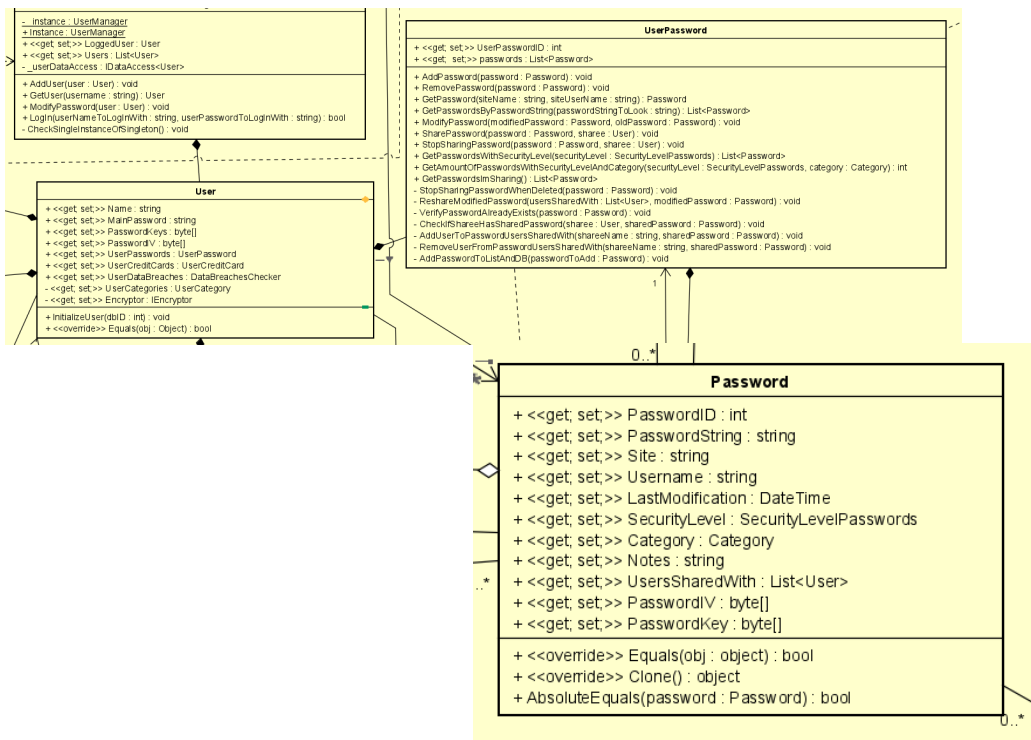
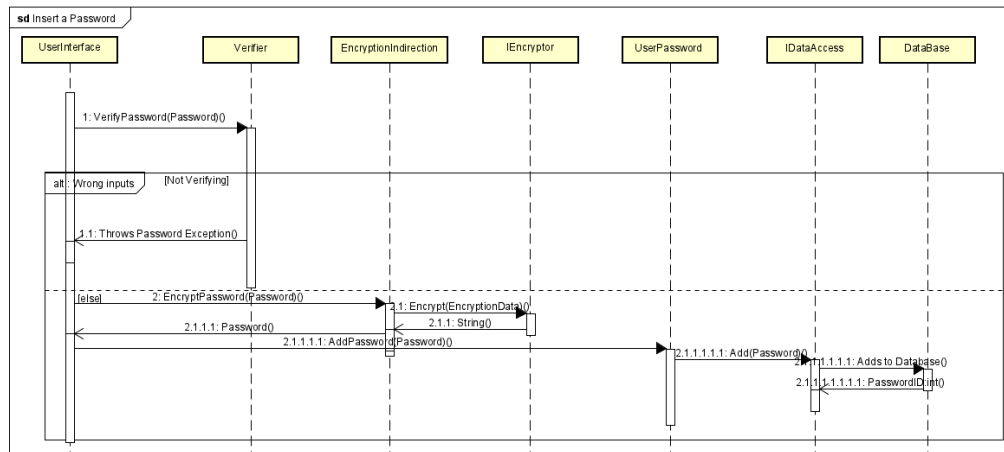
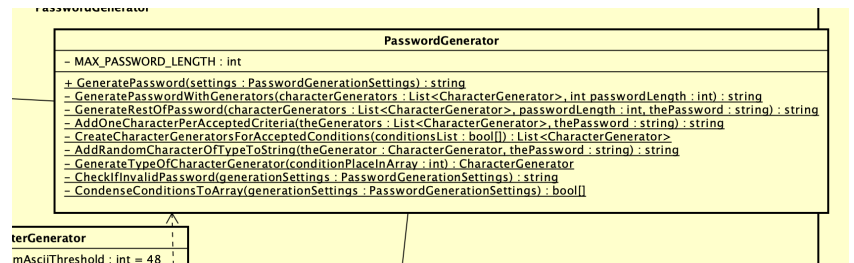


Diagrama de Secuencia: Agregar una contraseña



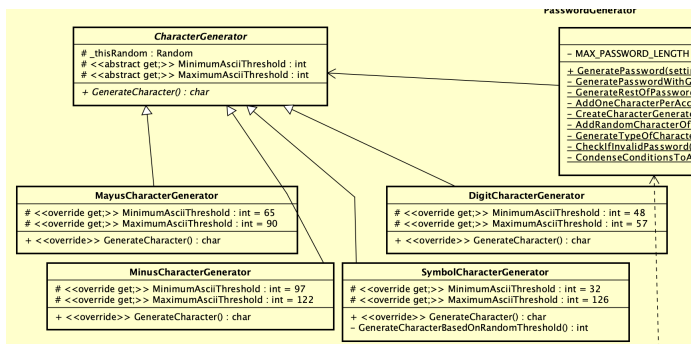
Para la generación de contraseña identificamos que tenía sentido tener una clase separada llamada PasswordGenerator que posee un único método público GeneratePassword, luego la UI podría utilizarla para generar la contraseña al apretar el botón de generar.



El generador nos trajo muchos desafíos de TDD y Clean Code. Los primeros problemas yacían en Clean Code. La cantidad de parámetros en un mismo método no respetaba la regla y, esto afectaba directamente a nuestro método GeneratePassword. Por esto, creamos un struct llamado PasswordGenerationSettings el cual se encarga de encapsular los settings posibles de generación de contraseñas, luego pasamos esto al método y logramos mantener Clean Code.

Seguido se nos presentó una posibilidad de polimorfismo, ya que teníamos presencia de switch statements y code smells. Para cada instancia de generar un char teníamos que generar alguno aleatorio de algún tipo (Mayus, Minus, Digit o Symbol), para

eso se nos ocurrió iterar el "largo del string" cantidad de veces sobre un array generando un número aleatorio (que representa un lugar en el array) para decidir qué tipo de carácter generamos. Para ello, este array podría ser un array de CharacterGenerators, una clase de la cual heredan cuatro clases que implementan el método GenerateCharacter que nos devuelve un caracter del tipo necesitado:



Luego tuvimos un problema con TDD: ¿Cómo probamos funciones con comportamiento randomico? Esto presenta un gran problema, ya que no podemos asegurarnos los mismos resultados dos veces para un test, algo que rompe con el concepto

de una prueba Unitaria adecuada. Necesitábamos una manera de asegurarnos que un test no pueda dar bien en un caso, y mal en otro.

Se nos ocurrieron dos maneras de resolver esto:

- Manipular el Random() en los tests.
- Que los Asserts se hagan sobre un conjunto exhaustivo de posibilidades.

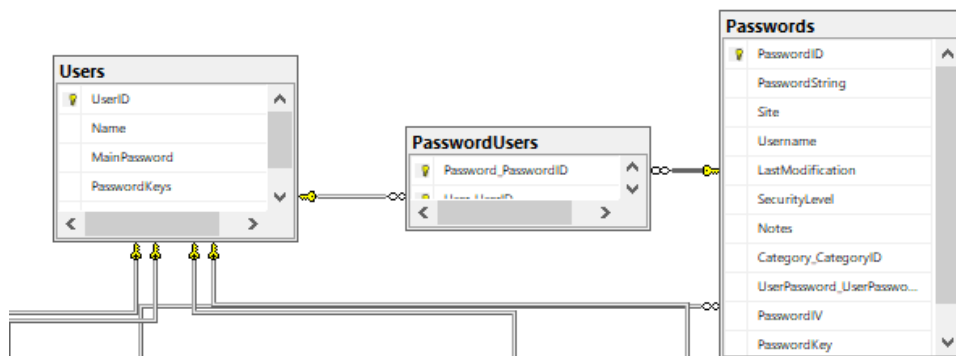
Decidimos ir por la segunda por un tema de preferencia. Lo realizamos de la siguiente manera: creamos arrays readonly de chars que poseen todas las posibilidades para el tipo de carácter (const de strings habría funcionado perfectamente también).

```
CharacterConstants
+ <<readonly>> ALL_MAYUS_CHARACTERS() : char[]
+ <<readonly>> ALL_MINUS_CHARACTERS() : char[]
+ <<readonly>> ALL_DIGIT_CHARACTERS() : char[]
+ <<readonly>> ALL_SYMBOL_CHARACTERS() : char[]
```

Luego, en los Assert, nos fijamos que este char se encuentre en el array. Debido a que el array posee todos los caracteres posibles exhaustivamente (a responsabilidad del tester), si encontramos un error es, indudablemente, debido a la funcionalidad testada y no al carácter randomico de la situación.

Finalmente, revisamos la funcionalidad de compartir contraseñas. Este tuvo algunos cambios de la primera entrega. El más grande es que ahora Password tiene una lista de Users en vez de una lista de String. Estos representan los Users a los cuales está compartida la contraseña. Esto se debió a que teníamos que representar en la base de datos este comportamiento, y para Entity Framework lo mejor era cambiar a la lista de usuarios.

En la base de datos quedó una relacion N-N entre Users y Passwords:



Denominada PasswordUsers, es la relación de que una contraseña puede ser compartida a varios usuarios. Luego Entity Framework se encarga de traer para la contraseña los Users pertinentes.

Alta, Baja, Modificación y Listado de Tarjetas de Crédito

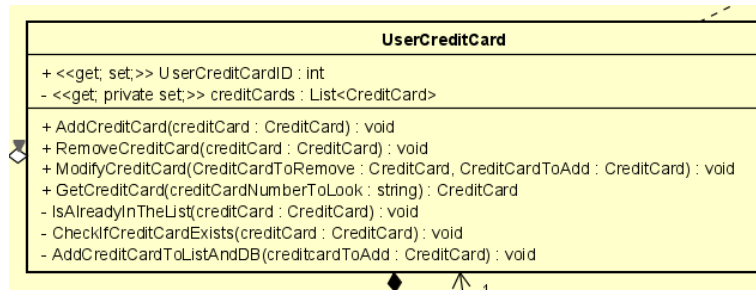
Utilizamos la clase UserCreditCard para almacenar la lista de tarjetas de crédito(CreditCard) y sus métodos.

Agregamos un atributo UserCreditCard para que, al igual que en las contraseñas, se facilite la

```
Credit Card
+ <<get; set;>> Name : string
+ <<get; set;>> Type : Card Types
+ <<get; set;>> number : string
+ <<get; set;>> securityCode : string
+ <<get; set;>> validDue : DateTime
+ <<get; set;>> category : Category
+ <<get; set;>> notes : string
+ <<override>> Equals(obj : object) : bool
```

navegación a la hora de utilizar Entity Framework. Un dato no menor es que al igual que con las demás clases se utiliza el RepositoryFacade para gestionar la interacción con la base de datos.

Se explicará en conjunto a UserCreditCard y sus llamados desde la interfaz desde el punto de vista del usuario en el gestor. UserCreditCardController es la clase que se encarga de esta última. Al clicar en el botón “CreditCards” se despliega la lista en un dataGridView(una lista) que obtiene sus datos del get público de la lista de UserCreditCard. Solo se deja visible una parte de los datos almacenados de la tarjeta, se mencionara la funcionalidad de acceso a todos estos datos más adelante.



Para crear una nueva tarjeta se puede clicar en “New” que crea un modal llamado “NewOrModifyCreditCardModal” que despliega los input a llenar. Con respecto a los input, tanto Type como Category de la tarjeta se muestran con un combobox. El primero accede a la lista de “categories” del usuario mientras que el segundo al enum “CardTypes”.

Además, se crearon eventos y métodos para que el input de “numberCreditCard” sea interactivo al usuario mientras inserta los números, facilitando su uso. Lo último mencionable es que la fecha de caducidad de la tarjeta se recoge de un dateTimePicker, el cual su menú de selección fue removido y, está limitado al mes y año actual para prevenir confusiones del usuario y facilitar el uso. A la hora del formateo de datos usar componente permite guardarlos directamente en DateTime, formato que utilizamos.

Number 2113-2233-21

Al presionar el botón “Confirm” se llama al método de UserCreditCard “AddCreditCard” que se encarga de añadirla a la base de datos y luego a la lista con su ID correspondiente. Se cierra el modal y se refrescan los datos del datagridview para que estén a la par del dominio. Aquí recién se habilitan los botones de borrar “Delete” y modificar “Modify” (si se elimina el elemento y se vuelve a tener una lista vacía, vuelven a deshabilitarse).

El botón delete se encarga de llamar al método “RemoveCreditCard” de UserCreditCard después de una confirmación del usuario (imagen). En caso afirmativo elimina la tarjeta de crédito del sistema y refresca la lista.

Confirmation

Are you sure you want to delete Nombre de la tarjeta Credit Card?

Yes No

El botón modify importa los datos de la tarjeta de crédito seleccionada y se muestran en el mismo modal mencionado en “New”. De esta manera el usuario puede modificar cualquier dato. Al presionar confirmar, se llama al método “ModifyCreditCard”, que elimina la tarjeta anterior e inserta su modificación. También se cierra el modal y se refresca la lista.

Es importante destacar que ambos métodos de añadir como modificar se hacen cargo de chequear si el número de la tarjeta es repetido, lo que no es admitido en el sistema.

La última funcionalidad de este segmento es la de obtener todos los datos de una tarjeta de crédito. Para esto el usuario solo debe hacer doble click en una celda del elemento deseado. De esta manera se crea el modal “CreditCardMoreInfoModal”. Este es

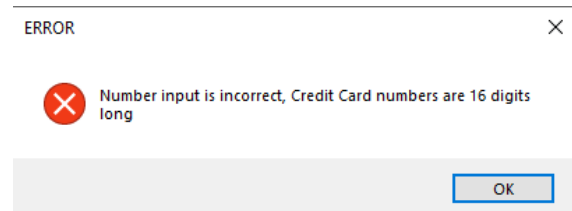
muy similar al modal anterior con la diferencia que no se pueden modificar los datos (si se pueden copiar), además de que se le añade un timer a la ventana que permite su lectura durante treinta segundos. Al terminar este tiempo o al cerrar la ventana se efectúa lo segundo.

Se decidió en el equipo separar estas funcionalidades en dos modals para conseguir un código prolijo y no tener que enviar parámetros extras para poder distinguir su funcionalidad.

Los siguientes dos párrafos se aplican a todo el proyecto pero lo mencionaremos aquí.

Antes verificábamos el Dominio en el Dominio. Decidimos cambiar a que se verifique en la interfaz gráfica. Esto es porque en algunos casos (como la encriptación) impedía que pudiéramos verificar en el Domain. Además, esto hacía que no hubiera un buen uso de las excepciones creadas por la UI, sino que las utilizaba el Domain y luego la UI las interceptaban.

Todos los modals y controllers muestran las excepciones del dominio, con su texto de error correspondiente, que pueden saltar al insertar o modificar. El Verifier corre antes de agregar en el Domain y corre los mensajes si sale la excepción en el mismo.

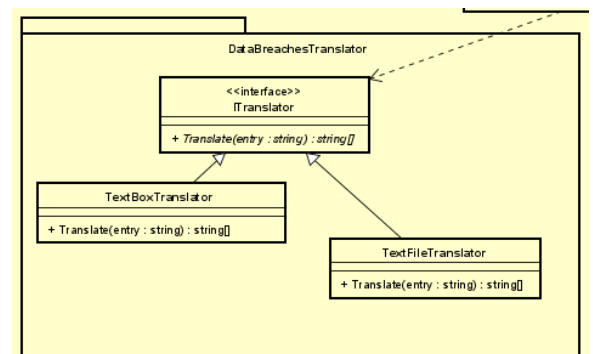


Detección de Data Breaches

Al pensar el diseño de los data breaches nos surgió un gran problema. Si el día de mañana llegase a cambiar el formato de entrada de los data breaches, el código presente debería de seguir funcionando sin tener que modificarlo.

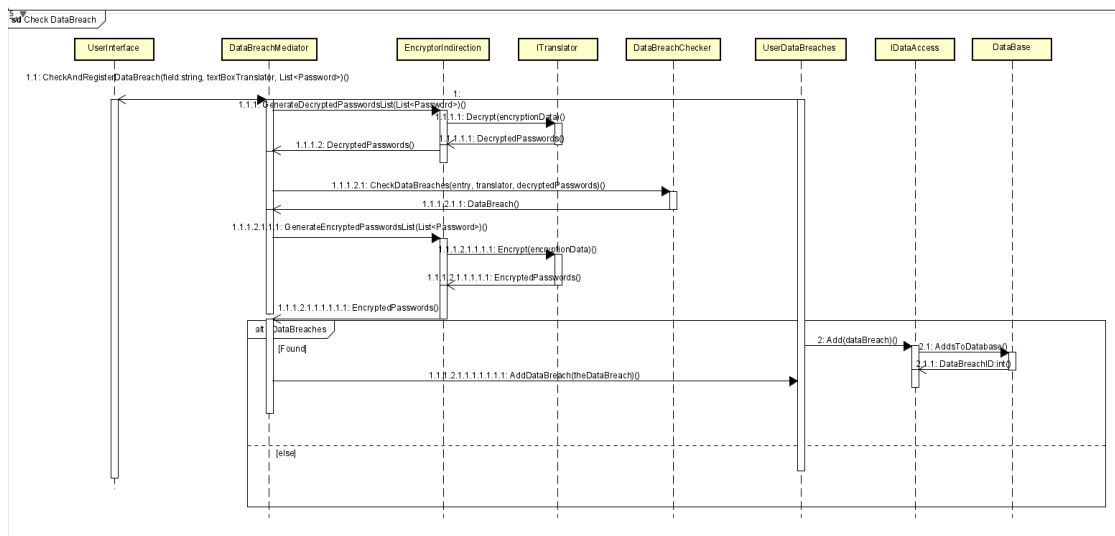
Para ello, se nos ocurrió que la función que se encarga de detectar los data breaches (el que compara) esté separada de User en un UserDataBreaches que funciona similar a los UserPassword y UserCreditCards mencionados previamente. Este UserDataBreaches posee una función que toma un array de strings, el formato más genérico posible para esta funcionalidad.

Luego habría una clase intermediaria que se encarga de tomar el formato entrante del texto y, de acuerdo a reglas establecidas, transforme esa entrada en una entrada aceptable para la función data breaches, es decir, un array de strings. Es como una especie de traductor para nuestro detector de datos breaches.



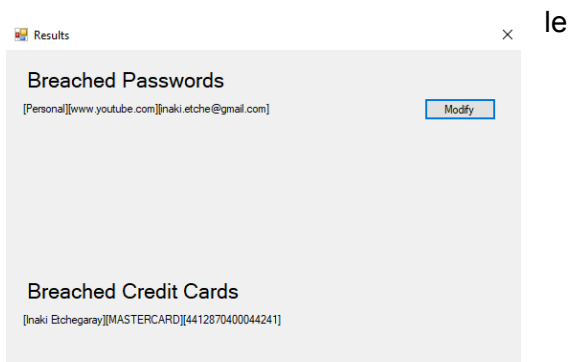
El día de mañana, bastaría con agregar una clase nueva que implemente las particularidades de traducción del formato nuevo y no habrá que cambiar nada de la clase encargada de los data breaches.

Diagrama de secuencia: Check de data breach asumiendo que hay contraseñas agregadas en el sistema



En este caso en concreto nosotros permitimos dos formatos, el primero es una gran textbox en donde el usuario puede escribir contraseñas y tarjetas de crédito y el segundo un documento txt. Las limitaciones son que las tarjetas de crédito tienen que ser escritas con los espacios cada cuatro números y los elementos separados por un enter. Al darle check nos muestra las coincidencias.

Luego la interfaz accede al traductor y pasa este texto traducido al detector de data breaches para luego mostrarlo:



Durante la implementación de esta función se dio la oportunidad de implementar en el paquete de la interfaz un mediador que nombramos DataBreachMediator. Este se creó con el objetivo de que no se encadenen funciones entre la UserInterface, UserDataBreaches y el DataBreachesChecker (UserInterface -> UserDataBreaches -> DataBreachesChecker). Este mediador se encarga de tener en un mismo lugar métodos que cumplen la misma función y llamarlos cuando sea necesario, permitiendo a estas dos clases y a la interfaz ser independientes unas de otras.

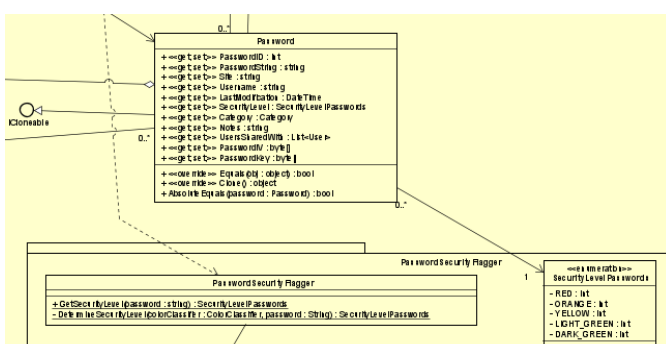
Reporte de Fortaleza de Contraseñas

Para el reporte de fortaleza de contraseñas se nos presentaron dos opciones de implementación:

- Calcular el nivel de seguridad cada vez que sea necesario mostrarlo.
- Guardar el nivel de seguridad con la contraseña.

Optamos por la segunda, ya que nos parece un desperdicio de computo calcular el nivel de seguridad cada vez que se necesite. Además, nos pareció que es la opción más escalable para incorporar a la base de datos ya que podemos guardar esta clasificación.

Para representar los niveles de seguridad, utilizamos un enum llamado

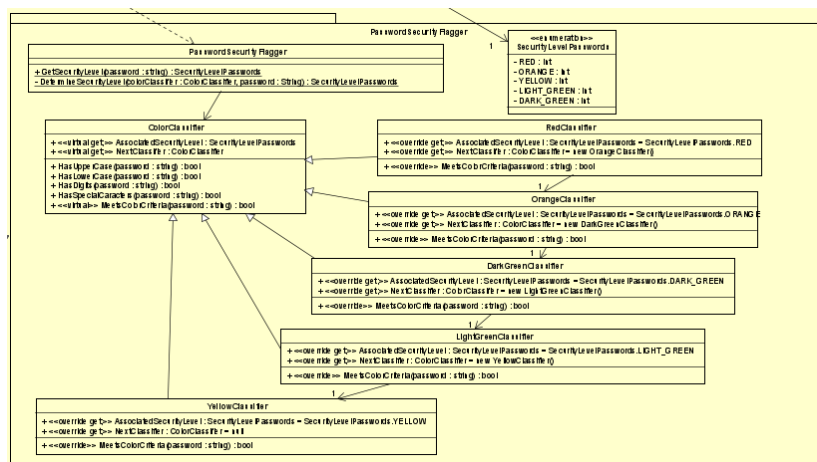


SecurityLevelPasswords que encapsula a cada uno de los colores.

Luego tuvimos que pensar cómo vamos a realizar la clasificación de los strings. Para ello, identificamos que la funcionalidad podría abstraerse en un método estático que se encarga de todo. Pensamos en incluir esta funcionalidad en UserPassword o en Password, pero nos parece que le agregaba demasiada responsabilidad a las mismas, por lo tanto terminamos creando una clase PasswordSecurityFlagger que encapsula la funcionalidad principal al respecto.

Al implementar la funcionalidad de esta clase se nos presentó el problema de que teníamos muchas condiciones para los distintos casos. Muchas de las funciones que aprobaban estas condiciones podrán ser reutilizadas en los distintos casos de los colores. Realizando el código, quedaban muchas funciones parecidas repetidas y switch statements que rápidamente notamos que podrían ser reemplazados por un polimorfismo.

El polimorfismo implementado se da con una herencia de una clase ColorClassifier, la misma tiene las funciones de aprobación de condiciones las cuales sus hijos pueden utilizar para aprobar sus criterios. Luego cada hijo hace un override de la función MeetsColorCriteria en donde se define qué es lo necesario para que una contraseña se considere de su color:



Antes, en la primera entrega, hacíamos un switch en el Flagger por el tipo de ColorClassifier que venía. Cambiamos esto, ya que en realidad no estábamos haciendo uso del polimorfismo (dependíamos del tipo de ColorClassifier, no aplica LSP de SOLID), y agregamos una propiedad overridable la cual era el próximo ColorClassifier. Este próximo ColorClassifier se usa en caso de que la contraseña no fuera del color que intentábamos clasificar con ese Classifier. De esta manera nos deshacemos del Switch y logramos aplicar el Polimorfismo.

La UI nunca accede a las funcionalidades de PasswordSecurityFlagger ya que basta con recorrer la lista de contraseñas en busca de aquellas con el nivel de seguridad requerido. Esto se hace mediante UserPasswords que posee las funciones necesarias. El uso del flagger en sí se da a la hora de modificar la contraseña o de agregarla. Accediendo a las funciones de GetPasswordsWithSecurityLevel y GetAmountOfPasswordsWithSecurityLevelAndCategory podemos armar la información necesaria para la UI y la gráfica.

Encriptación

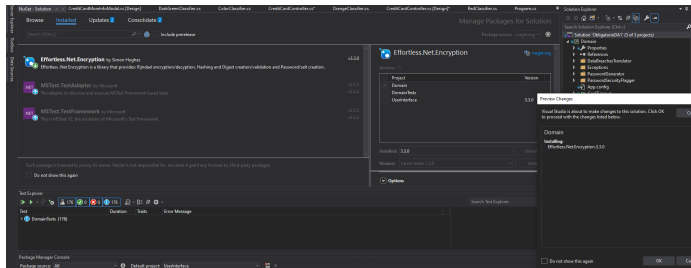
Otra de las funcionalidades nuevas requeridas por cliente es la de encriptación de contraseñas. Es requerido, por un tema de seguridad, que las contraseñas registradas y la del usuario sean insertadas en la base de datos encriptadas.

En vez de aprender a encriptar nosotros mismos las contraseñas, decidimos buscar una librería que nos ayudara con ello. Encontramos una librería llamada Effortless Encryption, una librería escrita en C# que permite encriptar y desencriptar contraseñas con una llamada de método.

GITHUB DE EFFORTLESS: <https://www.nuget.org/packages/Effortless.Net.Encryption/#>

Para instalar Effortless:

- Ir al Package Manager
- Correr: Install-Package Effortless.Net.Encryption -Version 3.3.0
- Ir a la solución
- Click derecho y seleccionar "Manage NuGet Packages for Solution".
- Seleccionar Effortless.Net.Encryption y bindearlo al paquete.

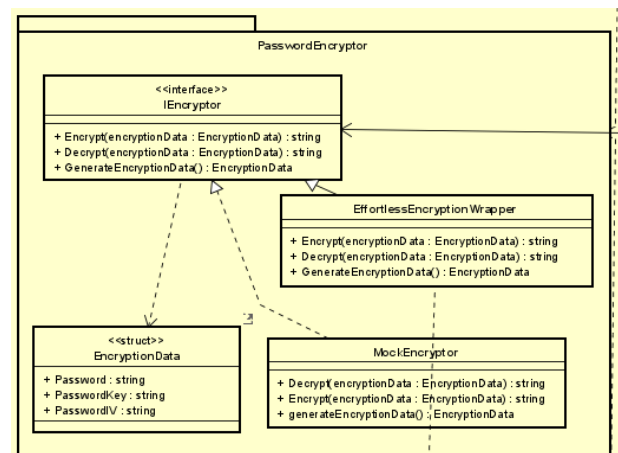


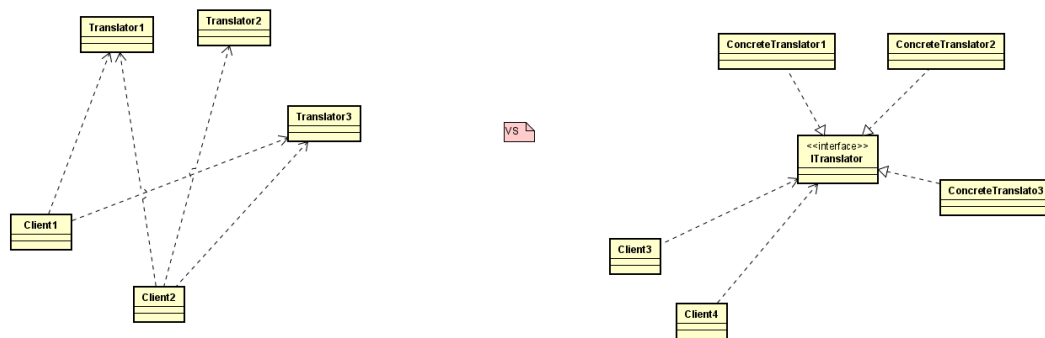
Al ser una dependencia nueva, decidimos pensar bien el diseño de las clases alrededor de las llamadas a la librería. Esto es para mantener la reusabilidad del código si se cambia la librería por otra. Para ello, era necesario no acoplar mucho nuestro código al ya existente.

Utilizamos el principio de bajo acoplamiento de GRASP creando una interfaz llamada IEncryptor. La misma tiene dos métodos muy simples: Encrypt y Decrypt. A las clases que utilizan IEncryptor no le interesan como hace el trabajo, siempre y cuando traiga de nuevo un string encriptado y su key para luego desencriptarlo. Además, de esta manera también aplicamos DIP (como en el caso del IDataAccess), ya que dependemos de una abstracción y no de una concreción.

Luego creamos la clase EffortlessEncryptor que implementa la interfaz nueva, y concreta el trabajo. También creamos un MockEncryptor que también implementa la interfaz para las pruebas en Domain, purificando las pruebas del nuevo funcionamiento.

Creemos que la implementación reduce el acoplamiento enormemente, ya que filtra el acoplamiento por la interfaz, en vez de repartir las dependencias entre concreciones. La siguiente imagen muestra muy bien cómo se acopla menos y sube también la cohesión:





También creemos que cumple bien con el principio OCP de SOLID, ya que si el día de mañana queremos usar una nueva dependencia para encriptar, basta con solo crear una clase nueva que implemente la interfaz y concrete la especificación, sin tener que cambiar mucho código en los otros paquetes.

Sugerencia de contraseñas

Una nueva funcionalidad que fue implementada dentro de la interfaz es la de un asesor de contraseñas. Esto no quiere decir que el sistema te genere una nueva contraseña recomendada particular, sino que conforme uno va escribiendo la nueva contraseña a agregar/modificar el sistema le indica si la misma fue breacheada anteriormente, si ya existe una contraseña antigua que use ese mismo string y si cumple o no con los estándares definidos la pasada entrega.

Para realizar esto se implementó una nueva clase PasswordRecommender que mediante un método estático el cual recibe un string compara si el mismo apareció previamente en un data breach o si existe actualmente alguna contraseña en el sistema que lo utilice. Por último se fija el nivel de seguridad de la misma tomando por válidos los niveles Verde claro y oscuro. Además, para poder utilizar distintos labels que brinden información sobre cada una de estas condiciones de forma separada creamos un struct SecurityCondition el cual guarda un booleano por cada una de las condiciones mencionadas previamente y se manda a la UI para así definir el texto mostrado en la ventana.

Nuevo formato de Data Breaches

El sistema permite revisar la existencia de nuevos data breaches mediante formato .txt. por un botón en la interfaz en la sección DataBreaches. Al utilizar un nuevo formato de input, se decidió tener una interfaz ITranslator que tenga las pautas para que las dos formas de traducir los inputs cumplan un patrón, además de hacer el sistema fácilmente escalable. Estas dos formas son TextBoxTranslator y TextFileTranslator, el primero se encarga de traducir directamente del textbox de la aplicación, mientras que el segundo de la nueva funcionalidad.

Con respecto a la lógica, esta se localiza en DataBreachChecker, la cual utiliza como input en su método más importante a la interfaz mencionada, otra vez resaltando los beneficios de escalabilidad. Es importante destacar que también se creó otra clase lógica llamada UserDataBreaches que tiene la clase Checker, además de la lista de DataBreach. Se separó la lógica de estas dos clases para mantener alta cohesión, delegando el uso de la interfaz a la clase Checker.

Por último, se utiliza la clase DataBreach como el objeto para almacenar los datos, esto se explicará más adelante en historial de Databreaches.

Con respecto a la implementación original, creemos que estuvimos cerca de realizar una buena implementación. Solo que no era utilizado acordemente. Teníamos las llamadas en la UI y no pasabamos el encryptor por parámetro. Esto hacía que no cumpliera con el principio LSP y que no aplicáramos el polimorfismo realmente. Pasamos la llamada del translator al checker de databreaches:

```
10 references | 0/9 passing | Inaki Etxegaray, 21 hours ago | 1 author, 1 change  
public DataBreach CheckDataBreaches(string inputBreaches, ITranslator translator, List<Password> decryptedPasswords)  
{
```

Esto hace que realmente dependamos de la abstracción y no de la concreción del translator.

Luego solo basto con instanciar la clase TextFileTranslator donde realizamos los checks de DataBreaches que venían de los text files, llamar al método y pronto. Creemos que si hubiéramos hecho bien la implementación, casi no hubiéramos cambiado nada.

Historial de Data Breaches

Antes de hablar directamente del historial de data breaches es importante mencionar que para su funcionamiento era necesario lograr la persistencia de los mismos en memoria. Esto llevó a varios cambios dentro de la lógica de data breaches. Nos dimos cuenta que al poder ser modificadas las contraseñas breacheadas y que en la base de datos se guardan por referencia entre objetos, llegamos a la conclusión de que era necesario también guardar el string con el que ocurrió el breach así eso queda en memoria y no se perdía en caso de ser modificadas todas las contraseñas con ese string específico.

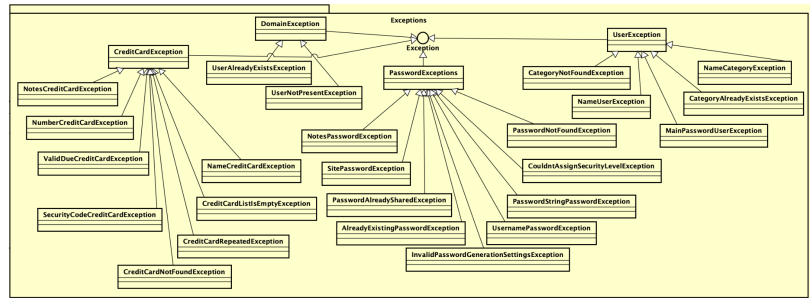
El historial de DataBreaches se encuentra en la sección DataBreaches de la interfaz. Al hacer click en el botón (History DataBreaches) se despliega una lista de los DataBreaches que dieron coincidencias en el chequeo, fuese cual fuese su método de input. Se destaca de esta lista la agrupación de chequeos en una misma hora, por lo que se necesitó de una completa reformulación del método add de UserDataBreaches, que ahora provoca que si un DataBreach se encuentra en la misma hora que otro ya creado, este lo debe incorporar al anterior.

La lista mencionada en el párrafo anterior tiene la función extra de poder visualizar dentro del DataBreach seleccionado. Para esto se abre un nuevo modal con la información del DataBreach (DataBreachHistoryResultModal). Es importante destacar que se tuvieron que modificar los contenidos de un DataBreach, que ya no guarda la lista de Password sino que guarda una nueva lista de objetos llamados PasswordHistory.

PasswordHistory se encarga de almacenar la Password y un string con una copia de la Password en ese momento. Almacenar este string por separado permite que al modificar una Password breacheada, quede guardado el string de la Password vieja. Por lo tanto se pueden visualizar tanto las Password que fueron breacheadas con sus respectivos strings al lado de cuando ocurrió el chequeo como la Password actual. Además, si estos dos coinciden se le da la oportunidad al usuario de modificar su Password. Aquí reusamos nuevamente AddOrModifyPasswordModal y nos ahorramos un mayor trabajo.

Manejo de Excepciones

Nuestra aplicación implementa una gran cantidad de excepciones personalizadas, todas heredan de Exception:



Además, las tenemos divididas en excepciones por funcionalidad. Por ejemplo, CreditCardException son excepciones lanzadas en las funcionalidades relacionadas a tarjetas de crédito.

Luego la interfaz es la que atrapa las excepciones lanzadas por el domain y muestra el mensaje pertinente al usuario:

```
try
{
    DialogResult dialogResultDeleteCreditCard = MessageBox.Show("Are you sure you want to delete " +
        _selectedCreditCard.Name + " Credit Card?", "Confirmation",
        MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
    if (dialogResultDeleteCreditCard == DialogResult.Yes)
    {
        _currentUser.UserCreditCards.RemoveCreditCard(_selectedCreditCard);
        CreditCardLoad();
    }
}
catch (CreditCardException creditCardException)
{
    MessageBox.Show(creditCardException.Message, "ERROR",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Testing

Como fue mencionado al principio del documento, la funcionalidad de la aplicación fue realizada en su totalidad con la técnica de desarrollo TDD. En la misma aplicamos los conceptos de las tres fases de TDD y desarrollamos los tests antes de hacer la funcionalidad.

El testing resultó en una totalidad de 250 tests distintos, divididos en las distintas funcionalidades de la aplicación. Todos los tests dan verdes y poseen la siguiente cobertura de código:

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Inaki_DESKTOP-OQK0190 2021-0...	535	8,60 %	5684	91,40 %
obligatorioda1.exe	180	11,10 %	1442	88,90 %
{ } Domain	45	3,92 %	1103	96,08 %
{ } Domain.DataBreachesTra...	0	0,00 %	7	100,00 %
{ } Domain.Exceptions	125	71,43 %	50	28,57 %
{ } Domain.Helpers	0	0,00 %	15	100,00 %
{ } Domain.PasswordEncrypt...	0	0,00 %	12	100,00 %
{ } Domain.PasswordGenera...	5	4,46 %	107	95,54 %
{ } Domain.PasswordRecom...	0	0,00 %	51	100,00 %
{ } Domain.PasswordSecurit...	5	4,90 %	97	95,10 %
obligatorioda1tests.dll	122	5,78 %	1987	94,22 %
repository.dll	220	12,69 %	1514	87,31 %
{ } Repository	51	5,97 %	803	94,03 %
{ } Repository.Migrations	169	19,20 %	711	80,80 %
repositorytests.dll	13	1,72 %	741	98,28 %

La cobertura del ejecutable da en 91.40% debido a que una gran parte del código no testeado es de excepciones y migrations, como se puede ver en la imagen (28.57% de cobertura para Domain.Exceptions) todos los paquetes relevantes se encuentran con una cobertura por encima del 90%.

ANEXO

Casos de prueba

Usuarios	
Nombre	Contraseña
MatixitaM	bAmBu27
GLandeira	1AceitunaS1
Sleepz	Mili7474
Tat1ana02	Tati5672890-9
exec1509	ta55adar





Categorías	
Nombre	Usuario Relacionado
Universidad	MatixitaM
Ocio	MatixitaM
Para Aprender	MatixitaM
Streaming	MatixitaM
Otros	MatixitaM
Streaming	GLandeira
Diversión	GLandeira
Universidad	GLandeira
Importantes	GLandeira
Otros	GLandeira
Juegos	Sleepz
Trabajo	Sleepz
Universidad	Sleepz
Servicios	Sleepz
Aprender	Sleepz
S. de Comida	Tat1ana02
Vacaciones	Tat1ana02
Trabajo	Tat1ana02
Universidad	Tat1ana02
Para Comprar	Tat1ana02
Trabajo	exec1509
Expert.com	exec1509
Desarrollo	exec1509
Compras	exec1509
Otros	exec1509

Tarjetas de Credito					
Nombre	Numero	CVV	Tipo	Usuario	Categoria
Matias Gonzalez	4564-2310-0030-2190	785	VISA	MatixitaM	Otros
Maria Jose Gonzalez	3331-2411-0233-0003	255	VISA	MatixitaM	Otros
Mario Gonzalez	1010-2003-2224-8091	306	MASTERCARD	MatixitaM	Universidad
Eugenia Gonzalez	9888-1056-2389-0392	928	AMEX	MatixitaM	Ocio
Gaston Landeira	5667-2448-9002-2847	100	AMERICAN	GLandeira	Importantes
Rosana Garrone	4897-0005-2367-2003	1	VISA	GLandeira	Universidad
Gonzalo Landeira	4999-1025-7852-9002	270	MASTERCARD	GLandeira	Importantes
Noemi Misa	2337-1990-1990-2003	998	MASTERCARD	GLandeira	Streaming
Iñaki Etchegaray	2444-2223-0985-0678	401	AMERICAN	Sleepz	Trabajo
Carlos Etchegaray	9033-8960-1001-2003	569	VISA	Sleepz	Trabajo
Pilar Sorhuet	4000-2331-6766-2032	903	VISA	Sleepz	Universidad
Mateo Etchegaray	7049-1749-5860-1205	786	MASTERCARD	Sleepz	Universidad
Tatiana Martinez	4566-4666-2313-4555	231	VISA	Tat1ana02	Trabajo
Federico Martinez	8910-4256-1096-2589	222	AMEX	Tat1ana02	S. de Comida
Juan Acosta	1123-4456-7778-9900	400	AMEX	Tat1ana02	Universidad
Silvia Perez	4231-5125-6232-1256	102	VISA	Tat1ana02	Para Comprar
Fabian Xenos	9999-1204-5260-4023	206	AMERICAN	exec1509	Trabajo
Tomas Sabia	6574-1251-4256-2131	324	MASTERCARD	exec1509	Trabajo
Aldo Radia	6910-1236-5437-7903	444	DISCOVER	exec1509	Desarrollo
Zacarias Atulo	6819-6347-3762-7473	106	DSICOVER	exec1509	Compras

Contraseñas				
Contraseña	Sitio	Username	Usuario	Categoría
mati23	www.ort.edu.uy	MatixitaM27	MatixitaM	Universidad
matXitam3	gestion.ort.edu.uy	MatixitaM	MatixitaM	Universidad
mat1az\$55-3aa	ucla.com	MatixitaM	MatixitaM	Universidad
Jj3qhQ6H28viVxwnod	camscanner.com	MatixitaM	MatixitaM	Universidad
mat1ass*10-3Aa	www.pdfs.com	MatixitaM	MatixitaM	Universidad
ocio1	minecraft.com	MatixitaM27	MatixitaM	Ocio
t787560v98t4nq5	bukkit.com	MatixitaM27	MatixitaM	Ocio
ocio3	valve.com	matias_27@hotmail.com	MatixitaM	Ocio
ociocuatro	blizzard.com	MatixitaM	MatixitaM	Ocio
Jj3>r655no3^=R_NOk	www.kioss.com	MatixitaM	MatixitaM	Ocio
quieroAprenderSI	www.lokz.com	MatixitaM	MatixitaM	Para Aprender
matiasAprende25	www.learn.com	MatixitaM	MatixitaM	Para Aprender
mati-asAprenda	diy.org	matias27@gmail.com	MatixitaM	Para Aprender
Bb0#GX	passcrypt.org.com	MatixitaM27	MatixitaM	Para Aprender
brilliantlydotcom	brilliantly.com	MatixitaM27	MatixitaM	Para Aprender
stream1	twitch.tv	MatixitaM27	MatixitaM	Streaming
streaming3	youtube.com	MatixitaM	MatixitaM	Streaming
matixiTam_22	streaming.com	MatixitaM	MatixitaM	Streaming
streamPass-*321	steam.com	MatixitaM	MatixitaM	Streaming
Ss6`8{#WXB w x~=bd8D	sttuts.org	MatixitaM	MatixitaM	Streaming
wlvqovqeyjftlkaxogczsexv	thot.com	matias27@gmail.com	MatixitaM	Otros
papasFritasConQueso	wikipedia.com	matias27@gmail.com	MatixitaM	Otros
mat1ass*10-3Aa	bukforums.com	MatixitaM27	MatixitaM	Otros
4>6~*{7:>	gmail.com	matias27@gmail.com	MatixitaM	Otros
Nn5]3jly9`}3jsJY7J<z9y9S	outlook.com	matias_27@hotmail.com	MatixitaM	Otros
gaston3	twitch.tv	GLandeira	GLandeira	Streaming
Gast0n32	youtube.com	GLandeira	GLandeira	Streaming
2A*&3a	streaming.com	GLandeira	GLandeira	Streaming
glAndeira23	stream.com	GLandeira	GLandeira	Streaming
Gaston_32A&*	sttuts.org	GLandeira	GLandeira	Streaming
Tt7[0B]8V.bvo=e5@x>	leagueoflegends.com	GLandeira	GLandeira	Diversión
bardo1920	steam.com	GLandeira	GLandeira	Diversión
Bardddo1920	gamesdonequick.com	GLandeira	GLandeira	Diversión

Bardddo1920	gamesdonequick.com	GLandeira	GLandeira	Diversión
y9}{_@r6<8]1v67	agar.io	GLandeira	GLandeira	Diversión
b@Rd0_231	tff.org.com	GLandeira	GLandeira	Diversión
Zz9{}F{	www.ort.edu.uy	gastonlandeira@gmail.co	GLandeira	Universidad
Pp6z9Zcq1o6CsJ	gestion.ort.edu.uy	gastonlandeira@gmail.co	GLandeira	Universidad
c1.x~/ z&?8w98)a0z9xl'8	ucla.com	gastonlandeira@gmail.co	GLandeira	Universidad
aceitunas27	camscanner.com	gastonlandeira@gmail.co	GLandeira	Universidad
aceitunas27	www.pdfs.com	gastonlandeira@gmail.co	GLandeira	Universidad
li3@x-X/D\$VdEYlv ~91C	www.santander.com	gastonlandeira@gmail.co	GLandeira	Importantes
aceitunas27	www.pedidosya.com.uy	gastonlandeira@gmail.co	GLandeira	Importantes
@ceit_nas27	www.mercadolibre.com	gastonlandeira@gmail.co	GLandeira	Importantes
S60B5P1D7S093CZKNP	www.italink.com	gastonlandeira@gmail.co	GLandeira	Importantes
{antalia23}	www.tirados.com.uy	gastonlandeira@gmail.co	GLandeira	Importantes
aclbo*(&252	www.mybreaches.com	GLandeira02	GLandeira	Otros
albondig@s	www.comidaahora.com	GLandeira02	GLandeira	Otros
lqozokuxfoegyuczcqnc	www.psnw.com	GLandeira	GLandeira	Otros
pantufas900	www.myblog.com	GLandeira02	GLandeira	Otros
pantufas9102	www.blogger.com	GLandeira02	GLandeira	Otros
fini7474	www.blizzard.com	SleepyWolfie	Sleepz	Juegos
f1n17474	www.valve.com	SleepyWolfie	Sleepz	Juegos
Jj3;dD t1knTKN7]4Kk5k4	www.steam.com	SleepyWolfie	Sleepz	Juegos
EeGWglwUCiucGg	www.discord.com	SleepyWolfie	Sleepz	Juegos
kqkirm	www.diablo2.com	SleepyWolfie74	Sleepz	Juegos
mili7474	www.unity.com	inaki.2012@hotmail.com	Sleepz	Trabajo
Mili7474	www.stackoverflow.com	inaki.2012@hotmail.com	Sleepz	Trabajo
M1i1747402	www.unityforums.com	inaki.2012@hotmail.com	Sleepz	Trabajo
F1n17443152000	www.potonunitynetwork.com	inaki.2012@hotmail.com	Sleepz	Trabajo
fini&^7474152000	www.net.com	inaki.2012@hotmail.com	Sleepz	Trabajo
Tt7\F\$f22FG2;320Jf(gj){G	www.ort.edu.uy	inaki.2012@hotmail.com	Sleepz	Universidad
49_86_5inaki	gestion.ort.edu.uy	inaki.2012@hotmail.com	Sleepz	Universidad
inaki7474Mili152000	ucla.com	inaki.2012@hotmail.com	Sleepz	Universidad
Mili3232	camscanner.com	inaki.2012@hotmail.com	Sleepz	Universidad
Mili7474	www.pdfs.com	inaki.2012@hotmail.com	Sleepz	Universidad
Jj3:aA~?v0VVv#8lsdL@S	www.generateservices.com	inaki.2008@hotmail.com	Sleepz	Servicios
f1n17474	www.dicethrow.com	inaki.2008@hotmail.com	Sleepz	Servicios
Mili7474	drive.google.com	inaki.2008@hotmail.com	Sleepz	Servicios

Mili7474	drive.google.com	inaki.2008@hotmail.com	Sleepz	Servicios
Mili7474	www.google.com	inaki.2008@hotmail.com	Sleepz	Servicios
Mili7474	www.refactoring.guru.com	inaki.2008@hotmail.com	Sleepz	Servicios
m4fr26qj63478k88t	www.lokz.com	inaki.2008@hotmail.com	Sleepz	Aprender
f1n17474	www.learn.com	inaki.etcche@gmail.com	Sleepz	Aprender
inaki7474Mili	diy.org	inaki.etcche@gmail.com	Sleepz	Aprender
inaki7474Mili	passcrypt.org.com	inaki.etcche@gmail.com	Sleepz	Aprender
inaki7474Mili	brilliantly.com	inaki.etcche@gmail.com	Sleepz	Aprender
g2fs277888tv830975x	www.pedidosya.com.uy	tat1ana	Tat1ana02	S. de Comida
tat13332	www.comidaahora.com	tat1ana	Tat1ana02	S. de Comida
tat_ana_3456A	www.ubereats.com	tat1ana	Tat1ana02	S. de Comida
pip0_A_&*&^	www.tiendaingles.com	tat1ana	Tat1ana02	S. de Comida
pip0_A_&*&^	www.disco.com	tat1ana	Tat1ana02	S. de Comida
Yy9j@6r.R:hHxd28X1<^9	www.despegar.com	tat1ana	Tat1ana02	Vacaciones
Rr6t7q687T917Qv8uVz	www.hotels.com	tat1ana	Tat1ana02	Vacaciones
Gg2@z_96r6rl47s	www.trivago.com	tat1ana	Tat1ana02	Vacaciones
li3EHOeh1oj2JtT5N	www.discounttravel.com	tat1ana	Tat1ana02	Vacaciones
tat13332	www.devacaciones.com	tat1ana	Tat1ana02	Vacaciones
papasfritasContra	www.net.com	tat1ana.co@gmail.com	Tat1ana02	Trabajo
tat13332	drive.google.com	tat1ana.co@gmail.com	Tat1ana02	Trabajo
tat13332	www.wondershare.com	tat1ana.co@gmail.com	Tat1ana02	Trabajo
Zz9~^9*Y<S7y	www.stackoverflow.com	tat1ana.co@gmail.com	Tat1ana02	Trabajo
pip0_A_&*&^	www.kioss.com	tat1ana.co@gmail.com	Tat1ana02	Trabajo
tat13332	www.ort.edu.uy	tat1ana.co@gmail.com	Tat1ana02	Universidad
pip0_A_&*&^	gestion.ort.edu.uy	tat1ana.co@gmail.com	Tat1ana02	Universidad
Ww8jA0aRKrA 6A4k:MD	ucla.com	tat1ana.co@gmail.com	Tat1ana02	Universidad
tat13332	camscanner.com	tat1ana.co@gmail.com	Tat1ana02	Universidad
pip0_A_&*&^	www.pdfs.com	tat1ana.co@gmail.com	Tat1ana02	Universidad
Jj=nr[Nem<RE jx+M^	www.mercadolibre.com	tat1ana.co@gmail.com	Tat1ana02	Para Comprar
Bb0DR16Ydr9ycCILPpWC	www.ebay.com	tat1ana.co@gmail.com	Tat1ana02	Para Comprar
t7[e~s /l1b#7	www.amazon.com	tat1ana.co@gmail.com	Tat1ana02	Para Comprar
tat13332	www.comprasmuchas.com	tat1ana.co@gmail.com	Tat1ana02	Para Comprar
tat13332	www.alibaba.com	tat1ana.co@gmail.com	Tat1ana02	Para Comprar
Aldari523	www.tarsonis.com	executor@gmail.com	exec1509	Trabajo
Vv871SDsGKdAFNCI2HC	www.sc.forums.com	executor@gmail.com	exec1509	Trabajo

Aldari523	www.tarsonis.com	executor@gmail.com	exec1509	Trabajo
Vv871SDsGKdAFNCI2HC	www.sc.forums.com	executor@gmail.com	exec1509	Trabajo
t~<-whi(fl-wf?n)~\{[]t{	www.bwsc.forums.com	executor@gmail.com	exec1509	Trabajo
Cc0.X;xJ9j]fnF	afreeca.tv	executor@gmail.com	exec1509	Trabajo
FeniX_&%@723	www.dikfg.com	executor@gmail.com	exec1509	Trabajo
Ta55adarR0cks	www.starcraft.com	executor@gmail.com	exec1509	Expert.com
Aldari523	www.teamliquid.com	executor@gmail.com	exec1509	Expert.com
Bb0 AP0`X5+A8R{[0WCa	www.artosistv.com	executor@gmail.com	exec1509	Expert.com
Aldari523	www.twitch.tv	executor@gmail.com	exec1509	Expert.com
Ta55adarR0cks	www.blizzard.com	executor@gmail.com	exec1509	Expert.com
Yy90A8V0a8A39WvaJwZ	www.microsoft.com	artan1s	exec1509	Desarrollo
FeniX_&%@723	www.apple.com	artan1s	exec1509	Desarrollo
FeniX_&%@723	www.devforums.com	artan1s	exec1509	Desarrollo
Aldari523	www.netcore.com	artan1s	exec1509	Desarrollo
hkjyskghfbxgrzqpqjsgneb	 Netcore netcore.com	  	xec1509	Desarrollo
r6_7:4u kc?09y32=&ig7u			xec1509	Compras
r6]49]4m,~\$zlg2t			xec1509	Compras
W84K77TT42	www.amazon.com	artan1s	exec1509	Compras
Cc1#GEELgEee21"l1e4Jt	www.comprasmuchas.com	artan1s	exec1509	Compras
Ssozvzl	www.alibaba.com	artan1s	exec1509	Compras
jtokoHzohl	www.paginas.com	artan1s	exec1509	Otros
gavron	www.appledev.com	artan1s	exec1509	Otros
Yy~;;uUjJDdl'Lfb_IFqubB	www.myforum.com	artan1s	exec1509	Otros
Jj3t78Twh2nWjH5Nxd3J	www.exec.org.uy	artan1s	exec1509	Otros
Ll4>s766_q[7]4}r9t	www.pipo.com	artan1s	exec1509	Otros