Universidad ORT Uruguay

Obligatorio 1

Diseño de Aplicaciones 1

Agustín Ferrari 240503

Ivan Monjardín 239850

Francisco Rossi 219401

ORT-DA1/239850_219401_240503 (github.com)

2021

Índice:

1. Descripción General:

En este proyecto el equipo se propuso como objetivo, desarrollar código profesional, fácil de entender, consistente y extensible. Para llevar a cabo esto, se implementaron varias técnicas aprendidas durante el curso, como, Test Driven Development (TDD), recomendaciones de Clean Code, entre otras.

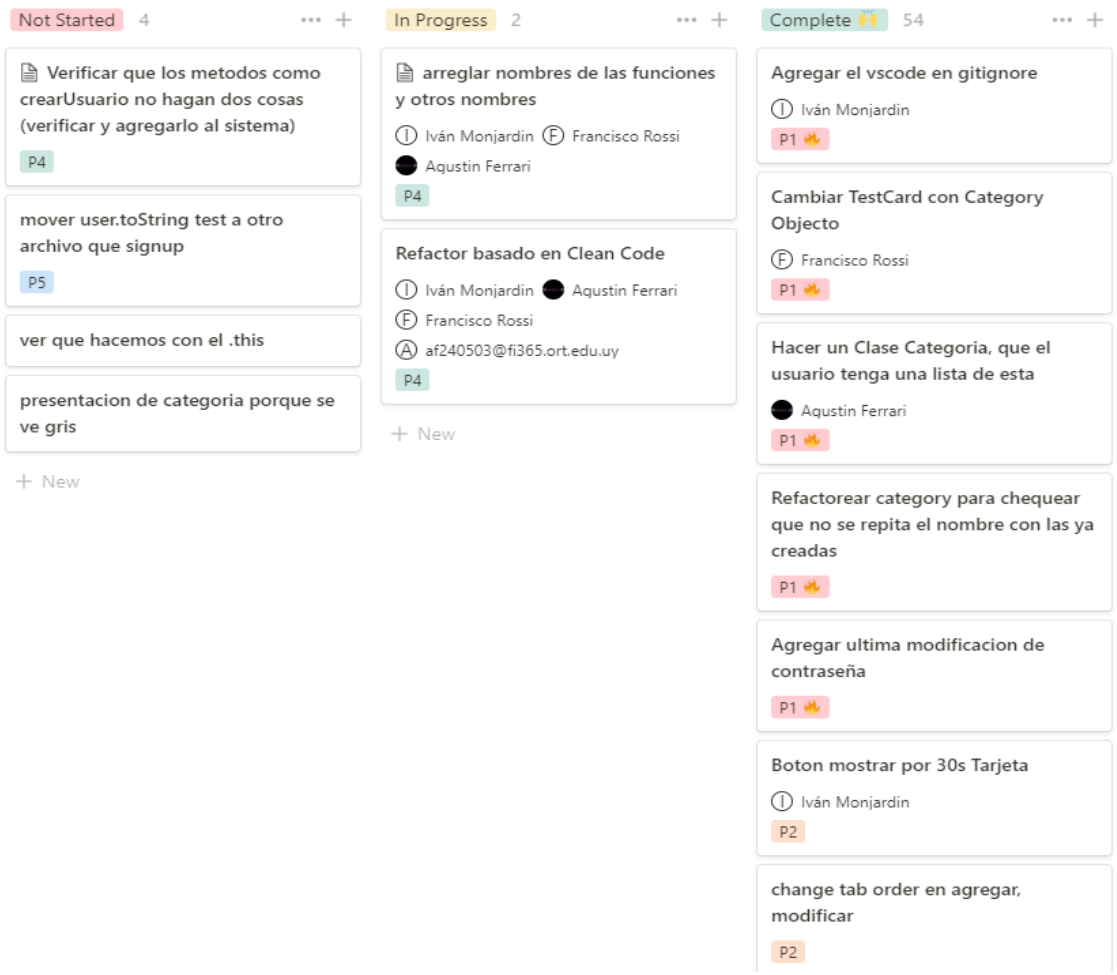
La idea fue seguir estrictamente TDD, plantear un test que no anduviera, escribir el código mínimo para que esta pasara, y luego realizar un refactor. Esto impedía generar código sin que exista una prueba para ello. Sin lugar a dudas generó un gran desafío debido a que ningún integrante del equipo, había desarrollado de esta forma antes, pero más sobre el final, pudimos apreciar ventajas de aplicar TDD.

El equipo cuenta con 3 integrantes, por lo que se decidió en conjunto, trabajar utilizando GitHub y Git Flow, creando ramas individuales para cada funcionalidad y/o refactors necesarios. Se planteó de esta manera para tener la posibilidad de avanzar sin tener que ser necesario la disponibilidad horaria de los otros integrantes todo el tiempo. Para que el código pueda ser consistente y todos estemos al tanto de los avances, se priorizó el uso de los pull request y reunirse al menos 1 o 2 veces al día para discutir y sacar dudas entre todos. Todos los pulls request fueron revisados por los otros dos integrantes y se hicieron comentarios con sugerencias o cambios necesarios, los cuales fueron arreglados antes de realizar los merge a develop.

Se establecieron algunos estándares para trabajar en el repositorio, como por ejemplo:

- El nombre de las ramas feature es "feature-NombreDeLaRama". Se utiliza camelCase y la primera letra del nombre va con mayúscula.
- Los commits se escriben en presente (Ej. “Adds modify button”).
- Para arreglar una funcionalidad se crea una rama “fix-NombreDeLaRama”
- Si se generó un problema al mergear a develop, se puede arreglar y commitear directo a develop o crear una rama Fix.

Utilizamos una herramienta llamada Notion, para organizarnos como equipo en la división de tareas. Cada vez que se nos presentaba una situación o se agregaba una modificación en el foro, se agregaba en la columna de Not Started y se le asignaba una prioridad del 1 al 5, siendo 1 lo más importante. Luego cada vez que algún integrante se ponía a desarrollar, elegía una tarea, la marcaba como In progress y creaba la rama para trabajar en ellas. Finalmente se pasaba a la columna de Complete y se podía avanzar con otra tarea.



Generamos un controlador principal (llamado PasswordManager), donde se encuentran las listas de Items y de Users. Es el experto y por lo tanto el encargado de manejar estas listas y distintas operaciones sobre estos objetos. Este controlador es el vínculo entre la interfaz y el dominio, con sus comportamientos especializados.

Pudimos implementar todas las funcionalidades requeridas e incluir algunas que consideramos importantes para mejorar la usabilidad del sistema. Las decisiones de cómo y cuáles funciones implementar fueron tomadas en conjunto por todo el equipo.
Al momento de crear el proyecto de interfaz gráfica, le pusimos el nombre de “Interfaz” el cual luego cambiamos a “Presentation” para seguir el estándar de escribir el código en inglés, sin embargo, sólo pudimos cambiar el nombre dentro del IDE pero no en el repositorio.

Tuvimos un problema a la hora de crear la interfaz de compartir contraseñas que consitía en que a veces al deshabilitar un botón nos tiraba una IndexOutOfRangeException. Luego de intentar encontrar donde estaba el error en nuestro código nos dimos cuenta que no era un problema nuestro sino más bien del set de la Property Enabled del botón, no podemos acceder a dicho código, por lo cual la solución fue hacer un try y catch esperando la exception, esto hace que no se caiga el programa pero a veces el botón queda semi-deshabilitado, es decir, en el proceso de deshabilitarlo el código interno falla y solo se cambia el color pero al hacer hover se sigue marcando el botón por ejemplo.

2. Descripción y justificación de diseño:

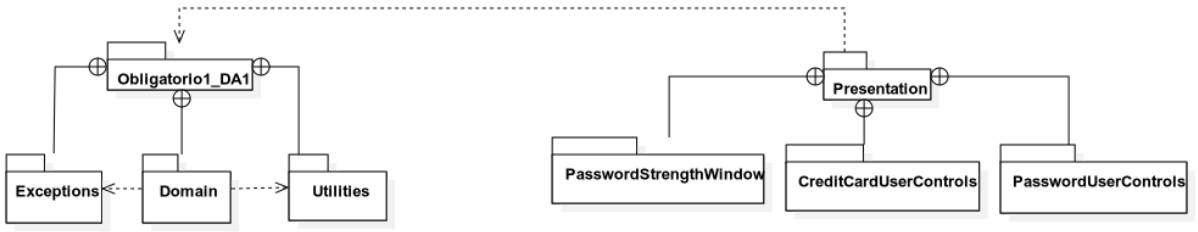
Todos los diagramas mostrados a continuación se encuentran en formato svg publicados en GitHub y adjuntos en el archivo .zip entregado en la carpeta Documentation/Diagrams.

Diagrama de Paquetes

Primero se planteó las clases fundamentales necesarias para poder cumplir los requerimientos. Entre ellas se creó PasswordManager, Password, CreditCard, User, Category, estos son los que conforman el dominio del proyecto y están representados en el UML. A medida que el código fue creciendo, en función de las pruebas y los principios de Clean Code, se fueron haciendo refactors para mejorar la solución. Por ejemplo, una clase abstracta Item, que también es parte del dominio y de la cual heredan Password y CreditCard. Esto se decidió implementarlo así, para aprovechar el polimorfismo y no repetir código.

Para el manejo de errores, decidimos crear excepciones personalizadas que fueron utilizadas en todas las clases del dominio. Todas las excepciones heredan de una padre que es la ValidationException y tienen un mensaje de error escrito en español para poder ser mostrado en la interfaz gráfica. Se planteó de esta manera para que a la hora de capturarlas fuera más sencillo y seguir las recomendaciones de Clean Code.

En el paquete de Utilities, se encuentran las clases que dan soporte a otras funcionalidades esenciales de otras clases. Inicialmente se creó después de un refactor donde nos dimos cuenta que más de una clase tenía que acceder a métodos de validación similares (Ej: largo de atributos).

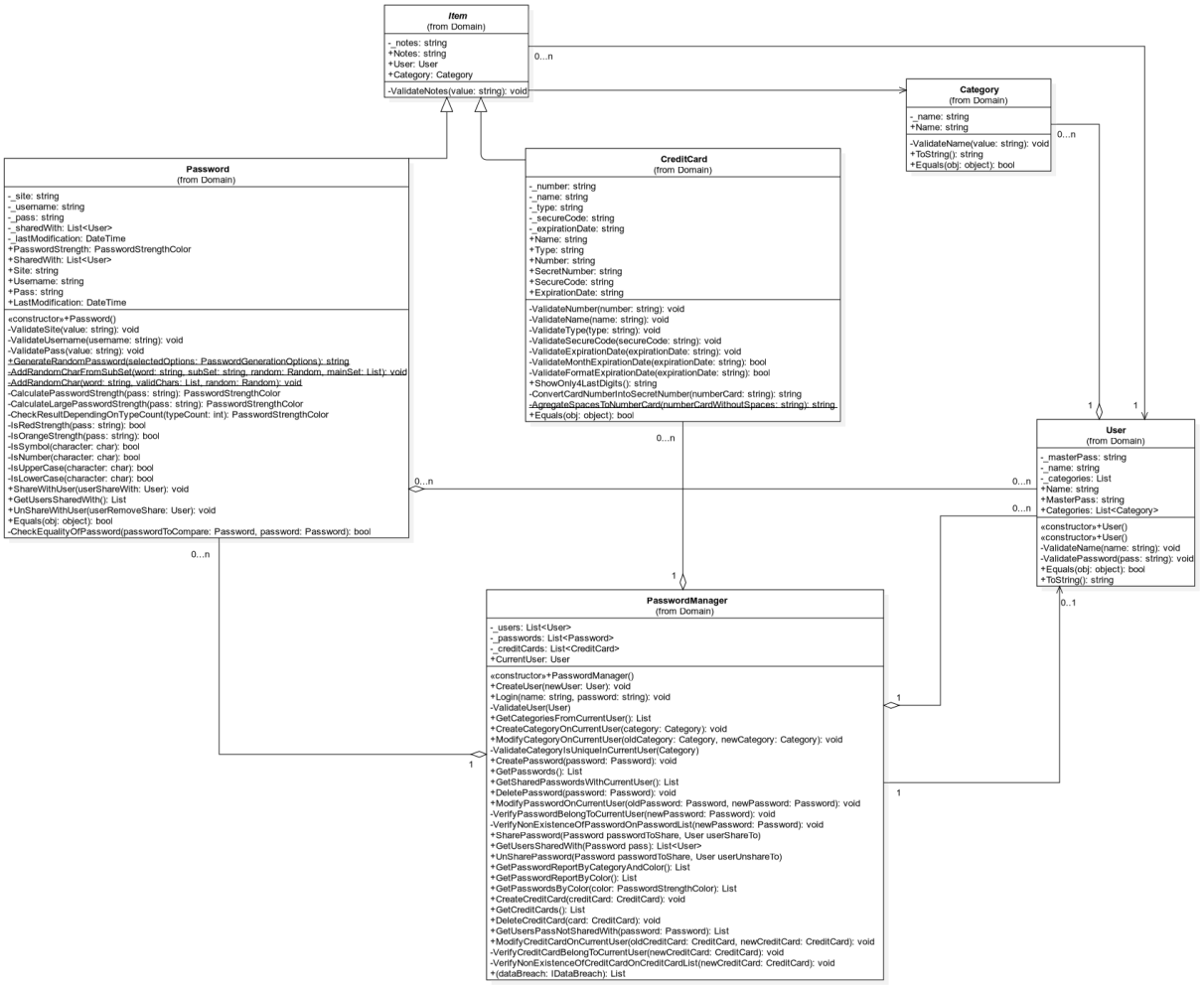


Como se puede apreciar en el Diagrama de paquetes, separamos la interfaz o la parte visible de la aplicación en un paquete Presentation, del resto. Se agrupó en distintos paquetes los user controls que representan un conjunto de funcionalidades similares. La principal idea fue tener un MainWindow, donde solo se fueran actualizando los user control sobre este, pero también se utilizó nuevos Forms en forma de Pop-Up para, por ejemplo, la creación/modificación de Items y confirmación de eliminación.

Diagramas de clases:

Dominio

Para entender más específicamente cómo diseñamos el paquete de dominio, podemos observar el Diagrama de Clases:



En cuanto a responsabilidades, nos enfocamos en tener un PasswordManager que sea el encargado y experto de manejar las listas de Items (Passwords y CreditCards) y de Users. Al tener estas listas, se decidió que fuera él, el encargado de agregarlos, removerlos, modificarlos, loguear usuarios, entre otras funcionalidades. Se diseñó de esta manera, considerando que en un futuro se podría querer conectar la aplicación con una base de datos para hacerla persistente, por lo tanto, se buscó una solución extensible, que minimice la cantidad de modificaciones, para realizar dicho cambio.

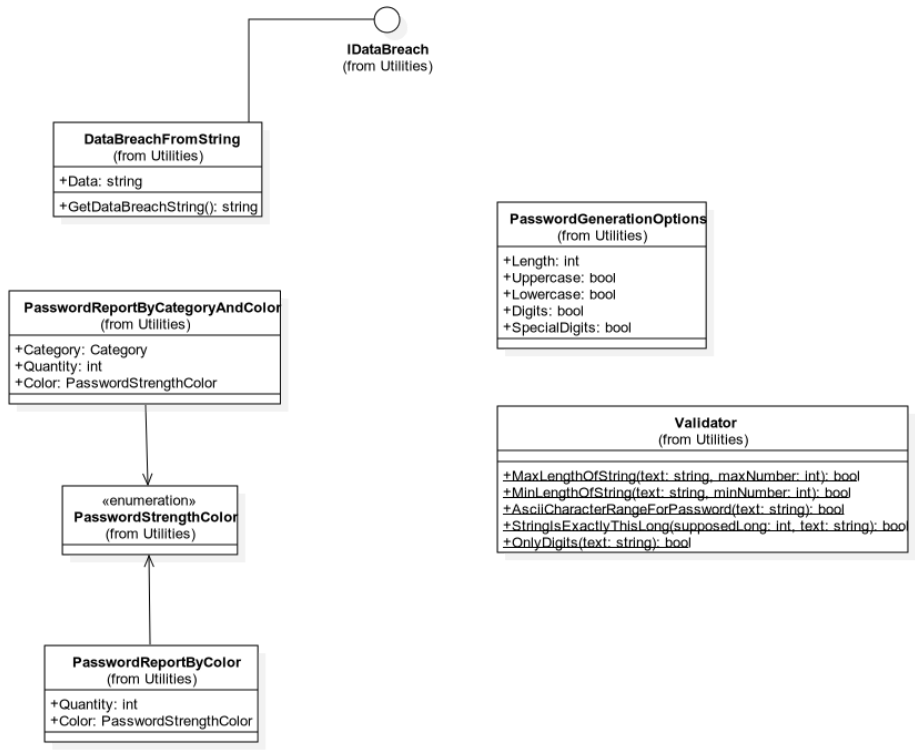
No solamente contamos con una lista de Users, sino que como se indica en el diagrama, existe una asociación con la clase User. Esto se debe a que decidimos utilizar un “currentUser”, donde se almacenará el usuario logueado actualmente y de esta manera el mismo solo podrá acceder a la información que le pertenezca y no podrá modificar la del resto de usuarios.

La Category se diseñó para que fuera User quien tenga una lista de las mismas y por consiguiente sea experto. Es por esto que aunque al agregar una Category se pase por PasswordManager éste le delega la creación y validación a User. Esto respeta el patrón de diseño de GRASP, Information/Expert.

La clase abstracta Item surgió a partir de reconocer que CreditCard y Password, comparten User, Notes, Category. Fue clave darse cuenta de esto para no repetir código y cumplir con las sugerencias de Clean Code.

Utilities

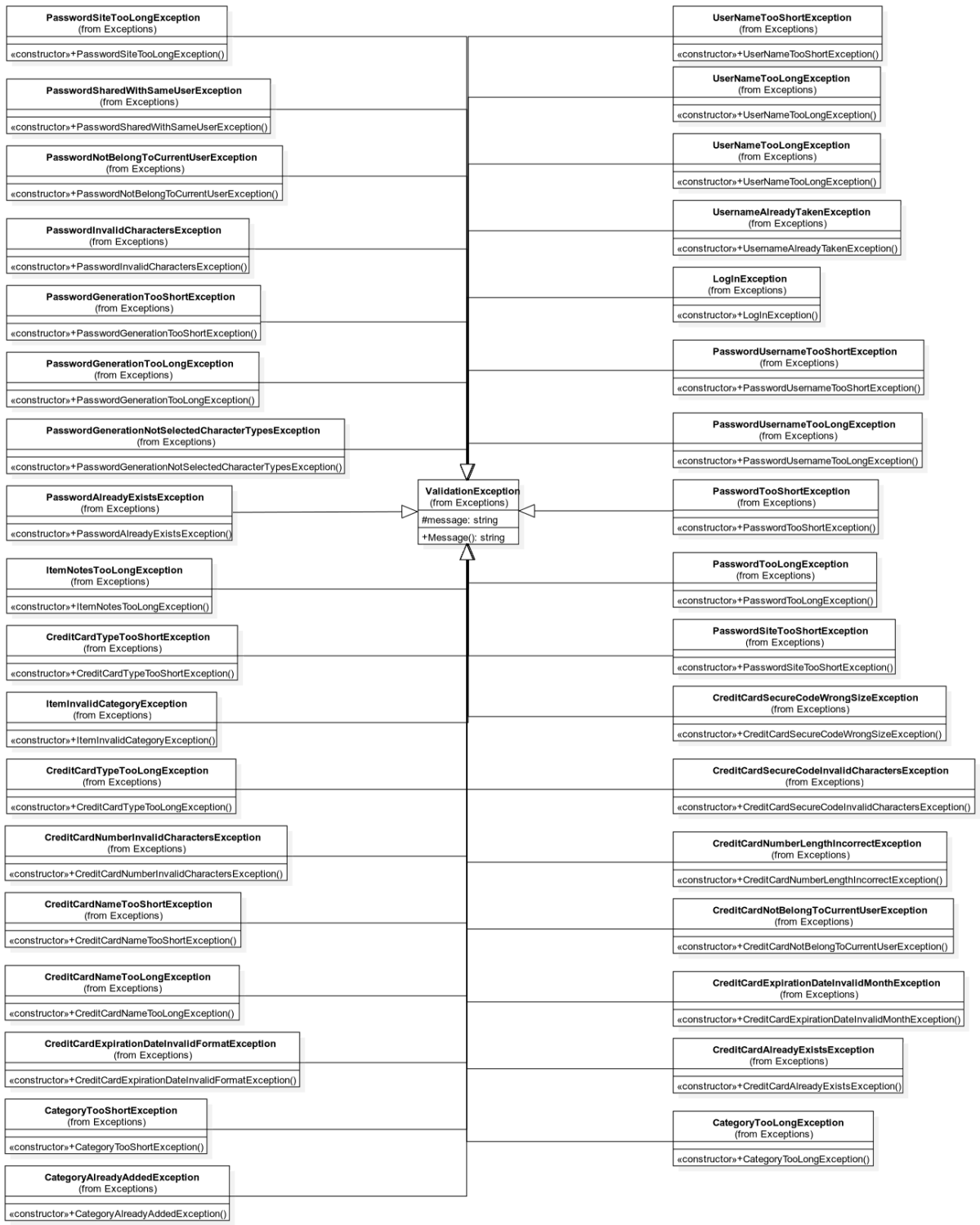
En Utilities, se añadieron clases que utilizamos como soporte para funcionalidades de otras clases. Como por ejemplo Validator que valida los largos generales para distintas properties, también se encuentra PasswordReportByColor, la cual es fundamental para el reporte de fortaleza de contraseñas. Otras clases que se encuentran aquí son : DataBreachFromString; IDataBreach; PasswordStrengthColor; PasswordReportByCategoryAndColor; PasswordGenerationOptions.



Para el Data Breach decidimos crear una interface, lo que nos permite que en caso de necesitarlo, podemos aceptar data breaches en formatos diferentes a el que tenemos actualmente (String a través de textbox). Simplemente se debería crear una clase concreta, que implemente la interface DataBreach y el método traduce el tipo del input a un String con cada dato breacheado en una línea individual.

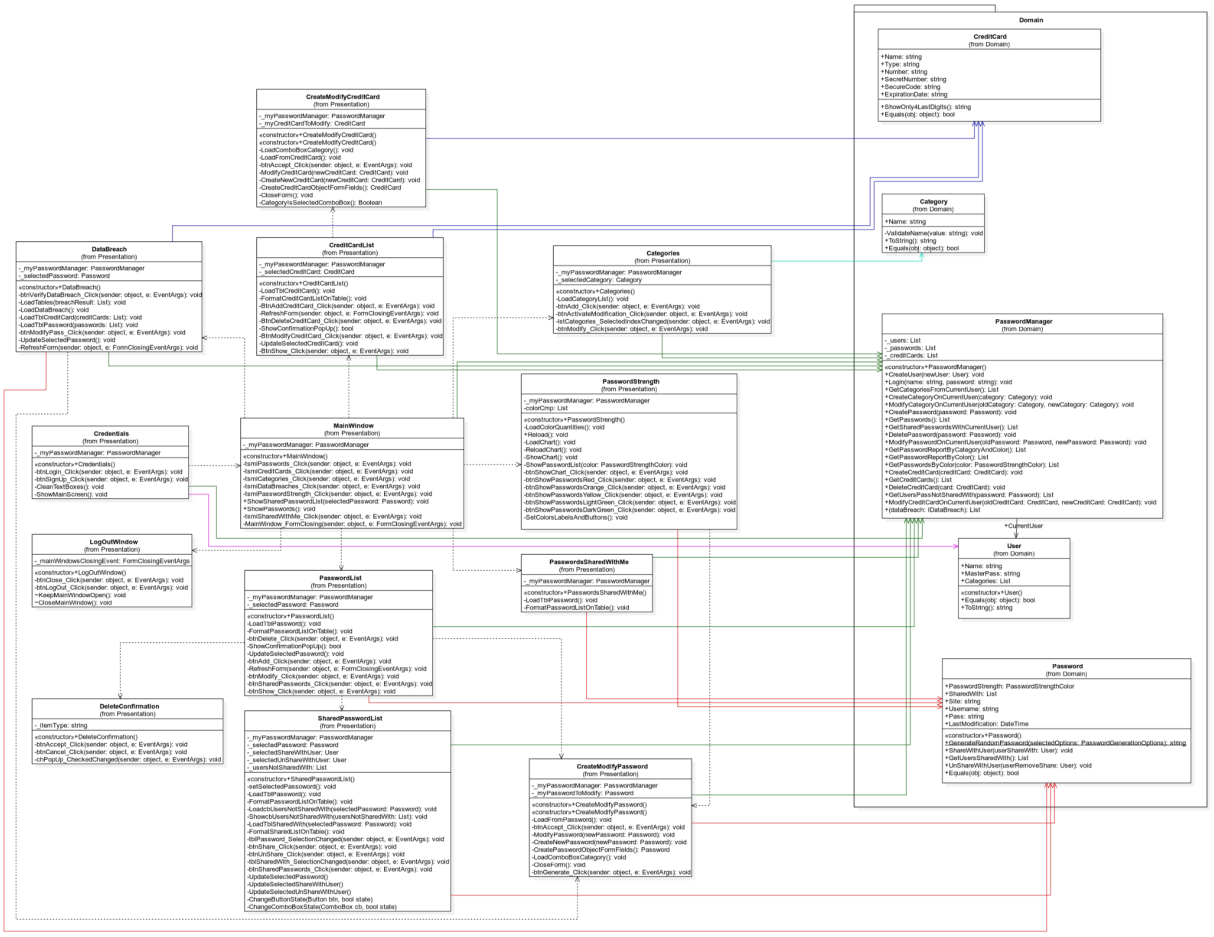
Excepciones

Como fue mencionado anteriormente creamos una clase ValidationException y para cada excepción posible de nuestro programa, se crea una excepción con un claro mensaje de error. La property del mensaje de error se inicializa en ValidationException y se especifica en cada una. El mensaje se escribe en español ya que puede ser mostrado al usuario.



Presentación

Para la presentación modelamos la interfaz gráfica usando una ventana principal donde se muestran los listados y ventanas principales. También usamos windows separadas para agregar y modificar contraseñas/tarjetas, esto nos permitió poder reutilizarlas en diferentes ventanas, donde era un requerimiento poder a acceder a la modificación directamente y le permite al usuario poder seguir viendo las listas, mientras agrega o modifica un item. Utilizamos también popups para acciones donde se necesita una confirmación, por ejemplo para eliminar un item o cerrar sesión/aplicación.



Para este diagrama se decidió agregar algunas clases del dominio simplificadas ya que nuestra intención era visualizar y comunicar de manera clara cómo se relacionan las clases de la presentación con las mismas. Además estas relaciones están marcadas con diferentes colores dependiendo de la clase para poder diferenciarlas de las de la presentación.

Diseño y Mecanismos generales

Algunas de las decisiones de diseño ya fueron comentadas anteriormente, sin embargo queríamos resaltar otros puntos relevantes.

Estándares de codificación

Al inicio del proyecto entendimos que era necesario establecer en grupo estándares de codificación para poder llegar a una solución clara y con buena calidad de código.
Algunos estándares establecidos fueron:

- El código y commits se escribe en inglés
- La interfaz gráfica se hace en español
- Cuando pasamos como parámetro el passwordManager, el nombre del atributo del parámetro es pPasswordManager
- El nombre del atributo privado de las clases es _nombreAtributo
- No utilizar el "this." para referirse a atributos al menos que sea:
    - Para evitar errores de compilación
    - Para ayudar a la comprensión del código
    - Para referirnos a una property adentro de una misma clase
- Los métodos con nombres autogenerados de la interfaz, no se cambia la primera letra a mayúscula.
- Todos los componentes del mismo tipo deben comenzar con el mismo prefijo. Ejemplo, los botones con btn, las etiquetas con lbl etc.

Password Strength Report

Para la funcionalidad del reporte de fortaleza de contraseña, la interfaz gráfica precisaba tener la cantidad de contraseña por color y la cantidad de contraseñas por color y categoría (para mostrar en la gráfica). Por lo tanto para cada caso se creó un struct que contenga la información necesaria. La idea detrás de esta decisión fue que la interfaz solo tenga que realizar un pedido de información al PasswordManager y no, por ejemplo, realizar 5 pedidos de cantidad de contraseñas (uno por cada color).

Almacenamiento de Datos

Para almacenar los datos se crearon listas en el PasswordManager, una para usuarios, contraseñas y tarjetas. Se decidió inicialmente dividir los ítems en dos ya que la mayoría de las consultas, por ejemplo en las ventanas creadas, se mostraban las contraseñas o las tarjetas, casi nunca ambas. Por lo tanto, nos pareció innecesario recorrer una lista que tenga todos los items cuando solo queríamos una parte.
De todos modos, se podía haber expandido el polimorfismo que ya utilizamos en Item (para conseguir las notas y el usuario) y haber agregado métodos que nos permitan almacenar juntas

las contraseñas y tarjetas (por ejemplo AgregarItem). Cuando agreguemos una base de datos, el problema de recorrer la lista seguramente deje de ser tan importante.
Donde nos beneficiamos del polimorfismo fue en la función que calcula los data breach, ya que devolvemos una lista de items donde están contenidas las contraseñas y tarjetas que fueron expuestas.

Confirmación de eliminación

Para la eliminación de Items, se implementó un pop-up de confirmación de la acción, en caso de que la acción no fuera intencional, teniendo en cuenta las heurísticas de diseño de interfaz. Dado que el usuario de la aplicación, puede encontrar estos pop-up molestos, se incorporó un checkbox que pregunta si no quiere que se muestre otra vez la ventana de confirmación. Para resolver esta situación se agregó dos variables de proyecto a Presentation de tipo bool, que tienen alcance (scope) a cada User y guardan la elección para eliminación de CreditCard y de Password por separado. Decidimos resetear estas variables cuando se cierre la aplicación.

Name	Type	Scope	Value
DontShowAgainPopUpPassword	bool	User	False
DontShowAgainPopUpCreditCard	bool	User	False

TestData

Creamos una clase, con el objetivo que se puedan probar las distintas funcionalidades de la interfaz gráfica. Para esto, contamos con varios usuario pre-cargados, que tiene tarjetas, contraseñas, categorías.
Algunos usuario para probar ingresar sesión con su par usuario, contraseña:

- Juana, Juana
- Laura, Laura
- Pedro, Pedro

3. Cobertura de pruebas unitarias con su debido análisis y justificaciones.

Test Coverage

Conseguimos una cobertura de pruebas unitarias del 100% en las clases del dominio. Las excepciones también cumplen un 100% de cobertura, esto es porque incorporamos excepciones personalizadas y para poder crearlas usando TDD, cada una tiene tests correspondientes.
Utilities o clases de soporte, también cumplió con el máximo de cobertura sobre el código.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
ivanm_IVAN 2021-05-13 16_34_44.coverage	137	5,99 %	2152	94,01 %
obligatorio1_da1.dll	0	0,00 %	846	100,00 %
Obligatorio1_DA1.Domain	0	0,00 %	735	100,00 %
Obligatorio1_DA1.Exceptions	0	0,00 %	69	100,00 %
Obligatorio1_DA1.Utilities	0	0,00 %	42	100,00 %

No solo realizamos pruebas para llegar al 100% pero también para casos bordes o pruebas que nos parecieron relevantes implementar. También utilizamos DataTestMethod para probar diferentes valores sin tener que escribir repetir código, como cada DataTestMethod solo se cuenta como un test, la cantidad total de test no necesariamente refleja el alcance o la cantidad de valores probados.

El 5,99% de bloques no cubiertos hace referencia al paquete de UnitTest no al dominio en el cual no hubo bloques sin probar.

La clase TestData no fue considerada para realizar pruebas unitarias ya que fue creada para probar únicamente la parte de presentación, no la parte de dominio.

Last Modified

Para cumplir el requerimiento de que que al crear o modificar las contraseñas, estas tengan una fecha de cuando fue esta última modificación, se usó System.DateTime. Este es el encargado de setear el atributo de password, con la hora configurada en la computadora que está ejecutando la aplicación. Para poder utilizar TDD y comprobar que efectivamente podía cambiar esta fecha de una día para otra, nos vimos forzados a dejar el set de la property public, y poner una fecha antigua para que la comparación fuera efectiva.

Testing funcional

Para el testing funcional, decidimos testear cada elemento de la presentación, probando todas las posibles combinaciones de botones, valores límite o vacíos en el caso de que haya que ingresarlos. También probamos que los mensajes de las excepciones realizados se mostrarán correctamente, ya que algunos eran muy largos por lo que tuvimos que acortarlos y manejar el tamaño de las ventanas para que se muestren de la forma esperada.

A continuación dejamos evidencia de los test que hicimos en alta, baja, modificar y listado de contraseñas, donde comenzamos testeando la parte del listado, viendo que ocurre en caso de utilizar las funcionalidades sin haber ingresado ninguna contraseña, luego agregando una y chequeando que no se puedan dejar campos vacíos. Seguimos con la modificación de la contraseña, donde modificamos uno de los atributos que la identifican, es decir el par (Site, Username), junto con otros y luego modificamos ambos atributos identificadores. Para terminar testeamos la parte de eliminar contraseña, sin haber seleccionado ninguna, luego seleccionandola y aceptando el popup, y por último utilizando la función no mostrar más del popup de confirmación.

A continuación dejamos el link al video. Cada parte está dividida por timestamps para permitir encontrarlas fácilmente.
https://www.youtube.com/watch?v=uiWTPmrKEJc