Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de Aplicaciones 1

Obligatorio 1

Klaus Gugelmeier Nro. Est. 247174

Facundo Sentena Nro. Est. 251661

Grupo M4A

Docente: Mario Souto

Estudiantes

Nro. Estudiante	247174
Nombre:	Klaus
Apellido:	Gugelmeier
Grupo / Turno:	M4A

Nro. Estudiante	251661
Nombre:	Facundo
Apellido:	Sentena
Grupo / Turno:	M4A

Contenido

II. Descripción general del trabajo

A. Descripción

Para este obligatorio se tuvo como objetivo crear un gestor de contraseñas, este es un programa cómputo que se utiliza para almacenar una gran cantidad de parejas usuario/contraseña. Los datos donde se guarda esta información están protegidos mediante una única clave (contraseña maestra; en inglés, master password), de forma que el usuario sólo tenga que memorizar una clave para acceder a todas las demás. Esto facilita la administración de contraseñas y fomenta que los usuarios escojan claves complejas sin miedo a que no podrán recordarlas posteriormente.

Las principales características pedidas fueron:

1 - Se debe permitir ingresar con una clave a la aplicación. Antes de hacerlo no se puede ver o modificar datos guardados.

2 - La aplicación debe mantener un registro de categorías de tarjetas de crédito y contraseñas.

3 - Se debe permitir agregar combinaciones usuario-contraseña para sitios o aplicaciones específicas.

4 - Se debe poder guardar datos de tarjetas de crédito.

5 - Se debe poder chequear que los datos guardados en la aplicación no hayan aparecido en algún data breach.

6 - Se debe ver un reporte que indique el nivel de seguridad de las contraseñas guardadas.

Enlace al repositorio de GitHub: https://github.com/ORT-DA1/251661-247174

B. Puntualizaciones

La única funcionalidad que presenta un detalle se encuentra en “editar Tarjeta de crédito”, dicho detalle consiste en que, si se desea editar una tarjeta, se deben rellenar todos los campos (aunque se usen datos antiguos de la tarjeta) para que el sistema permita reasignar dichos datos.

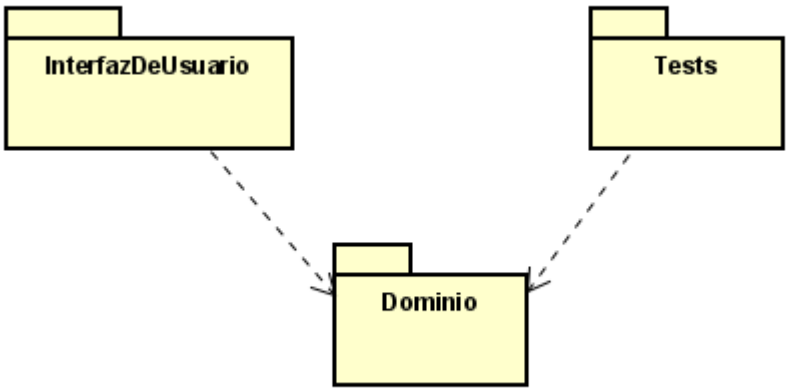
En cuanto al versionado y la aplicación de la metodología GitFlow: Cabe destacar que se siguió la metodología, respetando el no realizar commits directos en develop, creando ramas asociadas a features que son posteriormente mergeadas con develop utilizando el - - no – ff de git para no perder todos los commits de la rama mergeada. Las únicas puntualizaciones que se deben hacer sobre esta metodología son las siguientes:

1) Por un tema de desconocimiento al comienzo del obligatorio, algunas ramas creadas no parten desde develop, ya que se desconocía que al hacer git Branch “nombre de la nueva rama” desde una rama feature, la rama se crea desde dicha rama.

2) Las ramas “Feature/Tarjeta de crédito” y “Feature/Contrasena” fueron utilizadas para permitir a los integrantes desarrollar en paralelo. Sin embargo, al darnos cuenta de que sus funcionalidades se vinculaban casi directamente con la “Feature/Usuario, es decir no estaban autocontenidas, decidimos trabajar las funcionalidades de dichas ramas directamente en la Feature/Usuario, no mergeando las primeras ramas a develop, pero si dejándolas en el repositorio para evidenciar el uso de la metodología TDD. Estas dos puntualizaciones se realizan con tal de expresar que dichos errores cometidos no fueron con intenciones de violar la metodología Gitflow y desde el punto de vista de los docentes se nos recomendó que lo documentáramos.

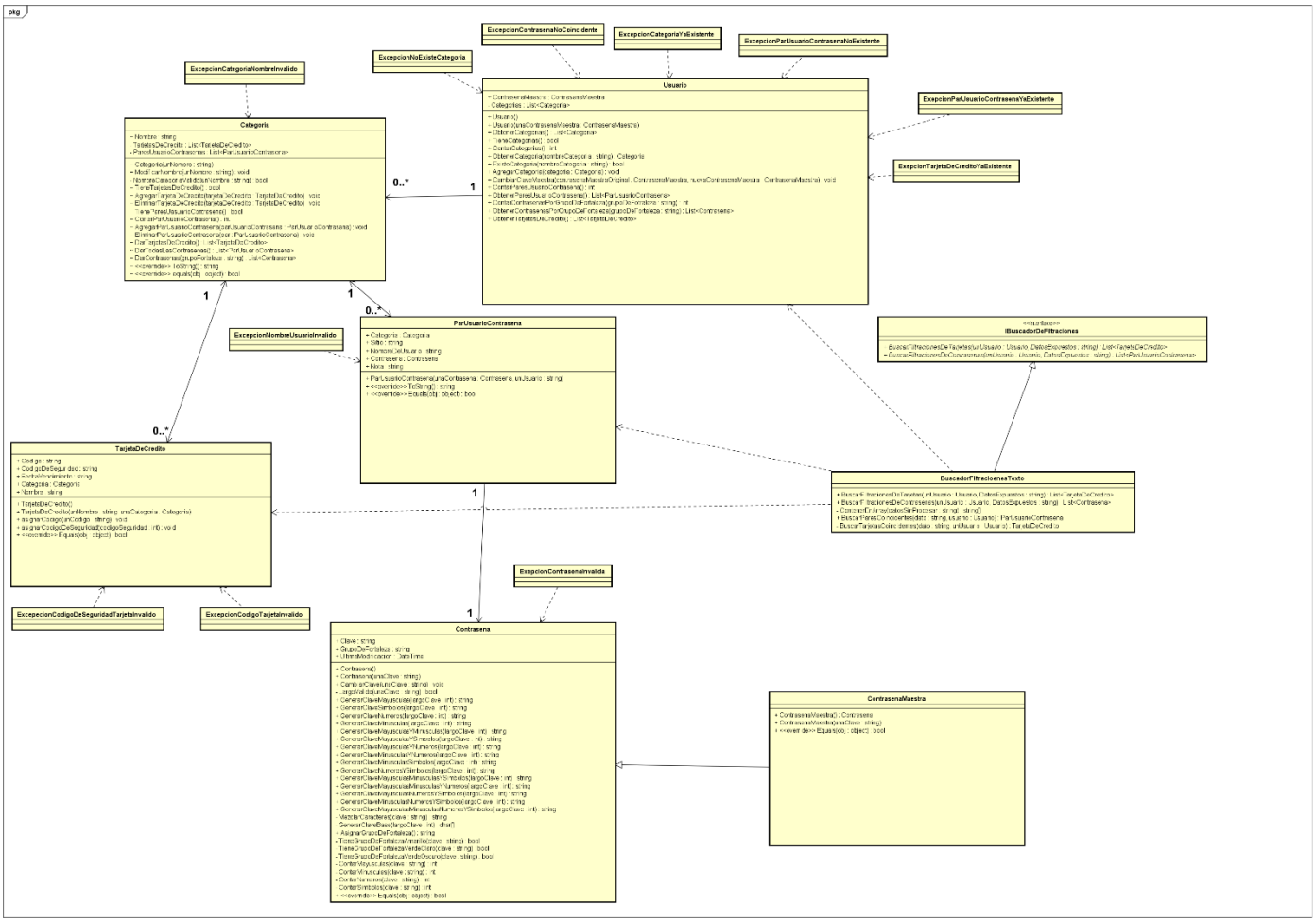
III. Descripción y justificación de diseño

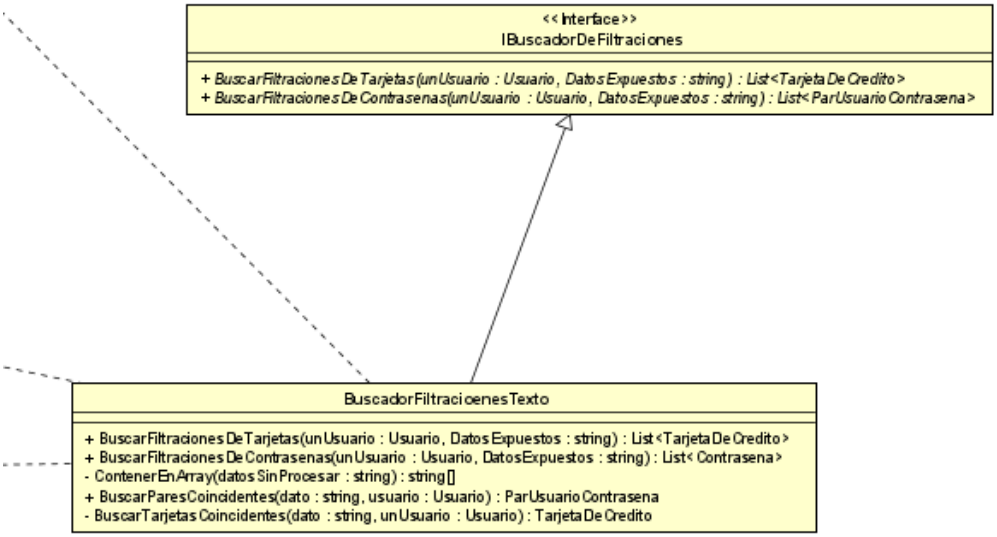
A. Diagramas de paquetes



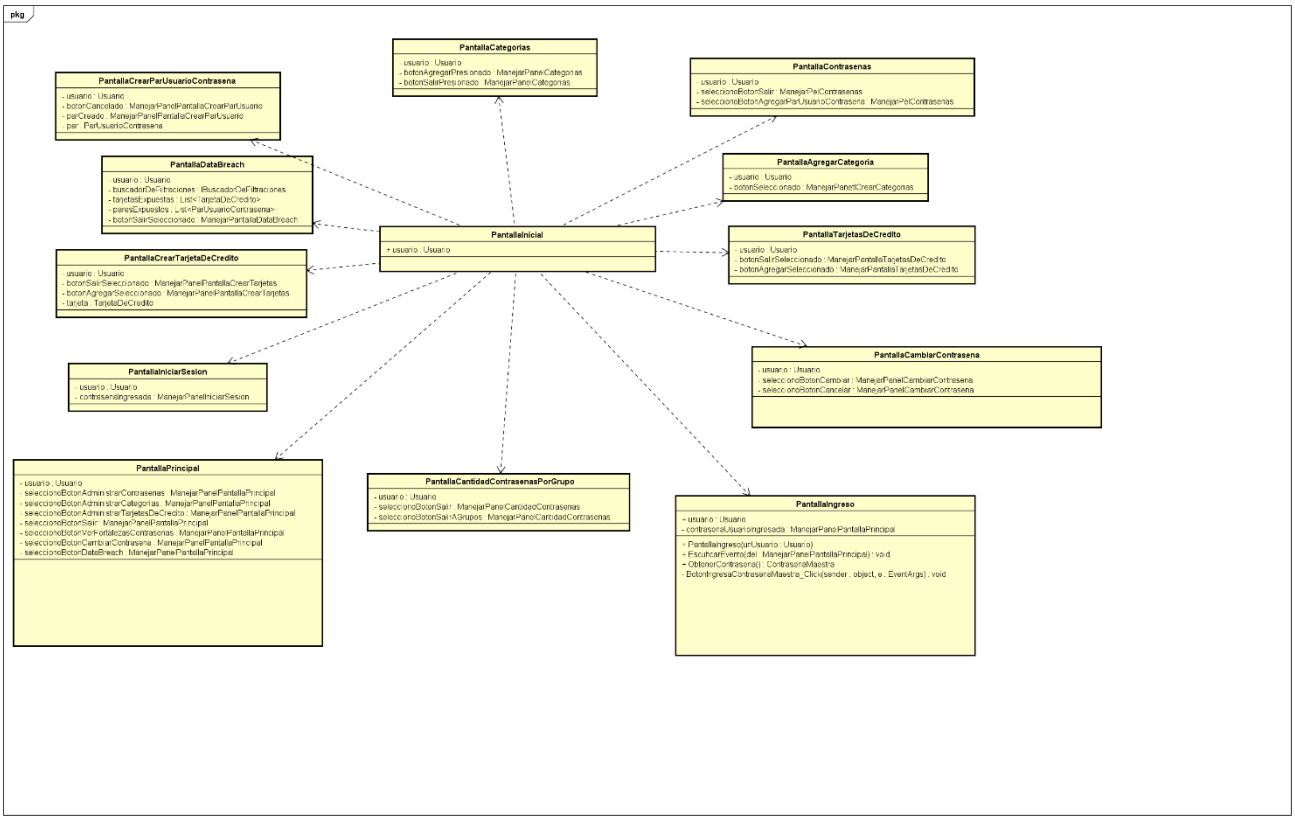
B. Diagrama de clases

Dominio:





Interfaz:



Como se muestra en el diagrama la clase, la clase PantallaInicial mantiene una dependencia con el resto de las ventanas ya que actúa como manejador de todas estas pantallas instanciándolas y pasándoles el usuario por parámetro.

C. descripción general del sistema

El sistema presenta una pantalla inicial de la cual se ingresa una contraseña maestra para acceder al menú, en este se encuentran distintas opciones:

- Administrar contraseñas (agregar, editar, eliminar y ver).
- Administrar categoría (agregar y modificar).
- Administrar tarjetas de crédito (agregar, editar, eliminar y ver).
- Verificar data breaches.
- Ver fortaleza de contraseñas.
- Cambiar contraseña maestra.
- Un apartado con la cantidad de categorías, contraseñas y tarjetas de crédito.
- Un botón salir, que al tocarlo habrá que reingresar la contraseña maestra para volver a entrar en dicho menú.

D. Explicación de los mecanismos generales

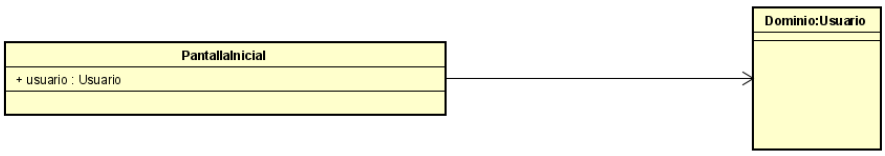
Optamos por la utilización de una herencia a modo de que “ContrasenaMaestra” heredara de la clase “Contrasena” para reutilizar sus métodos, pero principalmente porque semánticamente ambas clases se encuentran muy vinculadas.

Algo a detallar es que tomamos la decisión de que cada categoría contuviera en forma de lista tanto los pares asociados, como las tarjetas. Conforme avanzaba la implementación nos dimos cuenta de que tal vez esto no era la más eficiente a la hora de obtener datos de dichas colecciones, sin embargo, por temas de tiempo y de que el sistema ya estaba bastante avanzado decidimos continuar con dicha implementación.

En cuanto a la clase “BuscadorFiltracionesTexto” (Clase que contiene responsabilidades de la funcionalidad DataBreach) decidimos que dicha clase implementara una interfaz (en este caso IBuscadorFiltracionesDeTexto) para permitir que la funcionalidad no se acople a un tipo determinado de fuente de datos, permitiendo que en un futuro se pueda cambiar dicha fuente sin preocuparse de modificar el código, solucionándose únicamente creando una clase que también herede de dicha interfaz.

Con respecto a la interacción de la interfaz con el dominio, al manejar un único usuario debido a los requisitos del sistema para 2 estudiantes, es que optamos porque el “controlador” entre el dominio y la interfaz fuera la misma clase usuario, la cual se pasa por parámetro entre las diferentes ventanas.

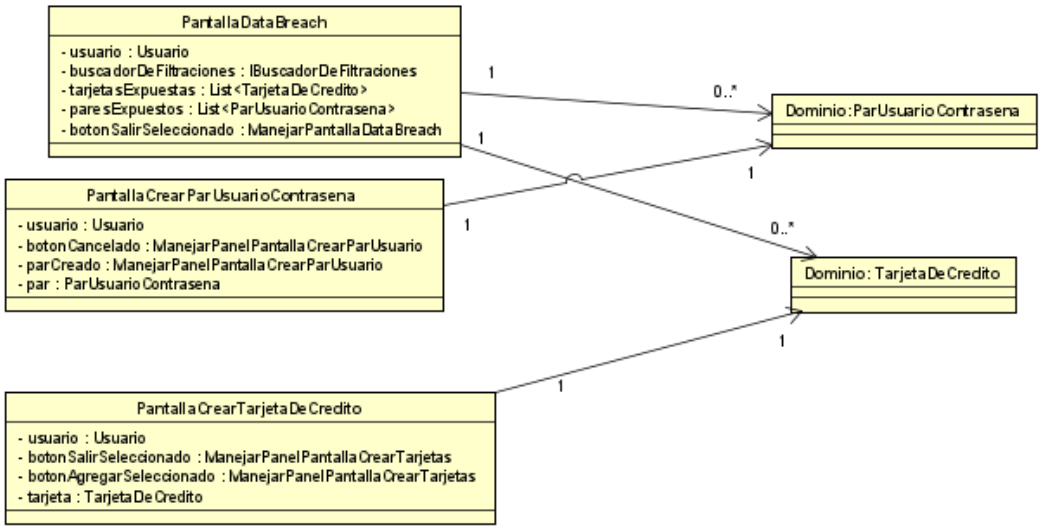
Por lo tanto, todas las ventanas se vinculan de con el usuario mediante una asociación de la siguiente manera:



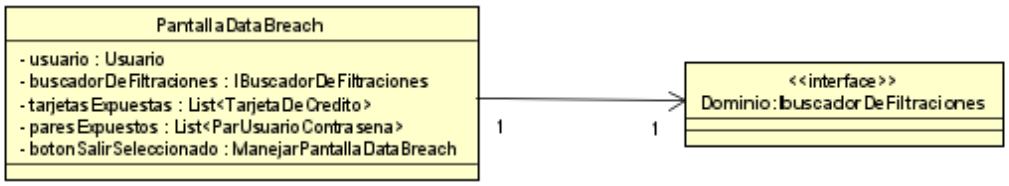
A modo de autocrítica, Esto genera un acoplamiento mayor que utilizando un propio controlador ya que además a través de los métodos de la clase usuario se generaron dependencias hacia clases como “Categoría” (ver en anexo).

También optamos por asociaciones en las ventanas que involucran la creación de una determinada clase, o el uso de esta para guardar instancias determinadas en forma de colección, como se ve en los siguientes ejemplos:

Ej1:



Ej2:

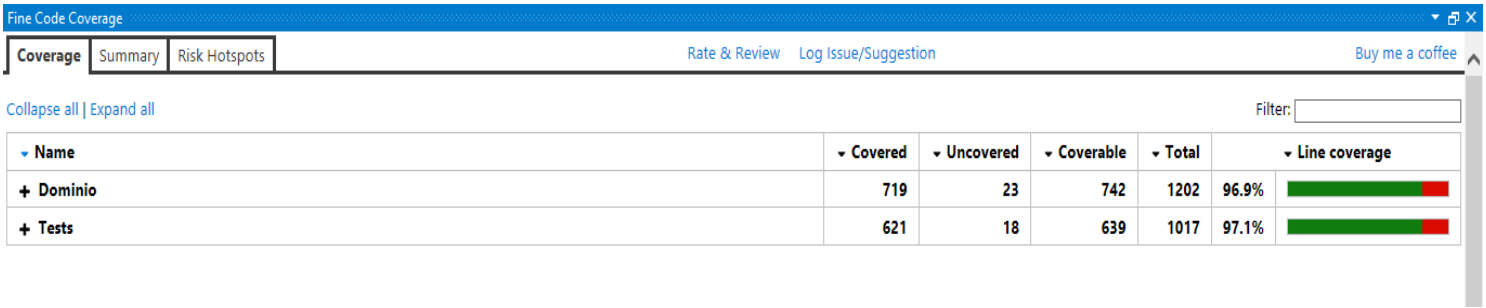


Cabe destacar, se crearon excepciones propias que eran lanzadas cuando se daban casos no permitidos dentro de la lógica del dominio (valores inválidos, excepciones de no unicidad de un elemento, etc.). Dichas excepciones son “atrapadas” dentro de la lógica de la interfaz de usuario, a modo de evitar que la aplicación lance errores durante la interacción con el usuario. Todas las excepciones implementadas se encuentran dentro del paquete dominio.

IV. Cobertura de pruebas unitarias

A. Cobertura de líneas de código

Este obligatorio fue hecho desde un principio a través de la metodología TDD (Test Driven Development) enseñado en clase, por este motivo, la cobertura de pruebas unitarias dio un buen resultado, superando el 90%.



Name	Covered	Uncovered	Coverable	Total	Line coverage
+ Dominio	719	23	742	1202	96.9%
+ Tests	621	18	639	1017	97.1%

B. Casos de prueba

Enlace a YouTube con respectivo video llevando a cabo los casos de prueba:
https://www.youtube.com/watch?v=laHuMoarX34

V. Anexo