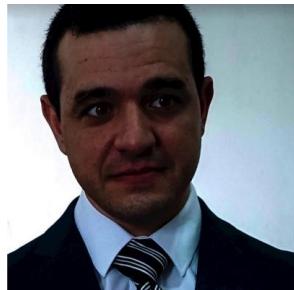




Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de Aplicaciones 2

Segundo Obligatorio



Cristian Palma 208443



Federico Alonso 182999

Descripción del diseño

Grupo N6A

Repositorio: https://github.com/ORT-DA2/182999_208443

Índice

1.	Descripción del diseño	3
1.1	Descripción general del trabajo	3
1.2	Errores conocidos	4
1.3	Diagrama general de paquetes.....	5
1.3.1	Dominio.....	6
1.3.2	Lógica	7
1.3.3	LogicaFabrica.....	8
1.3.4	LogicalInterfaz.....	8
1.3.5	Datos	9
1.3.6	DatosInterfaz.....	10
1.3.7	Datos Fabrica.....	11
1.3.8	Excepciones.....	11
1.3.9	Dtos	12
1.4	Modelo de tablas de la estructura de la base de datos.....	12
1.5	Justificación del diseño	13
1.5.1	Uso de principios y patrones de diseño	13
1.5.2	Descripción del mecanismo de acceso a datos utilizado	16
1.5.3	Descripción del manejo de excepciones.....	17
1.5.4	Decisiones de diseño propias.....	17
1.5.5	Descripción de extensibilidad de importaciones de bugs.....	17
1.5.6	Implementación de DTO	19
1.5.7	Análisis de la calidad del código.....	19
1.6	Resumen de mejoras al diseño respecto a la primer entrega	24
	ANEXO I Diagrama de componentes	25
	ANEXO II Diagrama de interacción relevantes.....	26
	ANEXO III Cambios en Web API	30
	ANEXO VI Contrato con desarrolladores de terceros	31
	ANEXO V Informe de cobertura	33
	ANEXO VI Casos de Prueba	35

1. Descripción del diseño

1.1 Descripción general del trabajo

El sistema tiene como funcionalidad principal la gestión de Incidentes asociados a un proyecto, contemplando roles como administrador, desarrollador y tester.

Para resolver el problema se definió la siguiente arquitectura de alto nivel:

- Web Frontend utilizando una aplicación Angular (SPA) que permite a los usuarios registrarse e interactuar con la aplicación.
- Backend implementando una API REST que resuelve toda interacción con el repositorio de datos.
- Base de datos relacional de SQL SERVER que almacena los datos de la aplicación.

Funcionalidades implementadas para cada rol:

Administradores:

- Realizar altas de **usuarios** registrando su nombre, apellido, nombre de usuario, contraseña, dirección de correo electrónico y su rol. Si esta creando un desarrollador o un tester se le habilita un atributo extra que corresponde al valor hora en \$.
- Visualizar la lista de usuarios registrados.
- Crear **proyectos** los cuales tienen nombre del proyecto.
- Agregar o eliminar desarrolladores y/o testers a un proyecto seleccionado.
- Visualizar el listado de incidentes de un proyecto seleccionado.
- Visualizar el listado de tareas de un proyecto seleccionado.
- Crear una tarea registrando el nombre, el proyecto al cual pertenece, costo por hora en \$ y una duración en horas.
- Ver el listado de todas las tareas de todos los proyectos.
- Obtener un reporte de cantidad de bugs por proyecto.
- Obtener un reporte de cantidad de bugs resueltos para un desarrollador seleccionado.
- Consultar costo y duración de un proyecto.
- Módulo que permite importar incidentes en formato XML ,archivo de texto y JSON, pero que es extensible para que en tiempo de ejecución pueda utilizar código de terceros que implemente otros formatos para la importación sin necesidad de recompilar la aplicación.

Desarrolladores:

- Visualizar los proyectos a los cuales pertenece, sus incidentes y sus tareas.
- Visualizar todos los incidentes de todos los proyectos de los cuales pertenece pudiendo filtrar por id, proyecto, nombre y estado.
- Visualizar todas las tareas de todos los proyectos de los cuales pertenece.
- Ver la información detallada de un bug de un proyecto al cual pertenece.
- Modificar el estado de un bug (activo/ resuelto).
- Modificar la duración en horas que llevó solucionarlo.

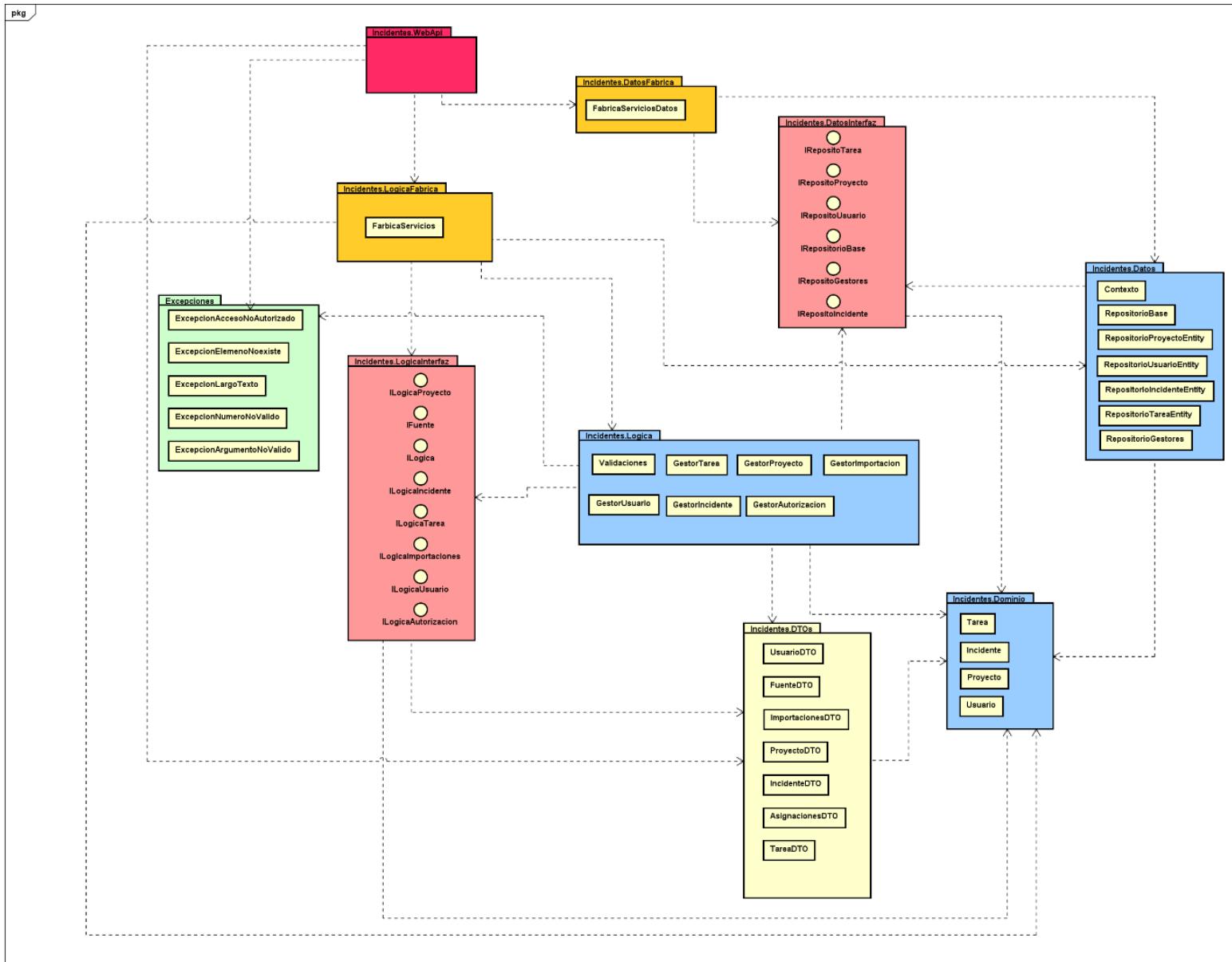
Desarrolladores:

- Visualizar los proyectos a los cuales pertenece, sus incidentes y sus tareas.
- Crear un bug para un proyecto al cual pertenece, brindando nombre del incidente, descripción, versión y duración.
- Editar y eliminar bugs de proyectos de los cuales pertenece.
- Visualizar todos los incidentes de todos los proyectos de los cuales pertenece pudiendo filtrar por id, proyecto, nombre y estado.
- Visualizar todas las tareas de todos los proyectos de los cuales pertenece.
- Ver la información detallada de un bug de un proyecto al cual pertenece.
- Modificar el estado de un bug (activo/ resuelto).
- Modificar la duración en horas que llevó solucionarlo.

1.2 Errores conocidos

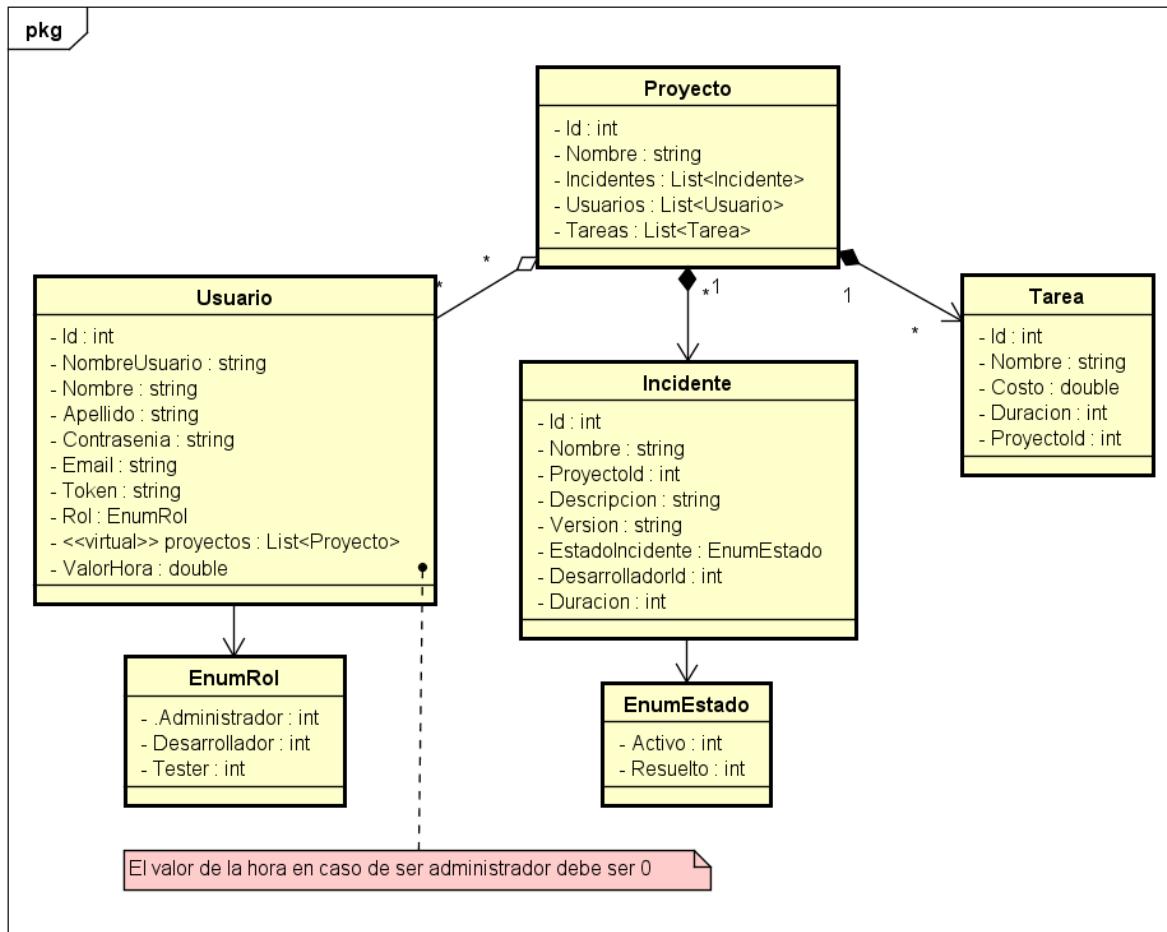
- Algunos mensajes de error del backend, de librerías de terceros o de base de datos se muestran en inglés:
- La WebAPI controla mediante las **TrapExceptions** los errores provenientes de las otras capas de la aplicación, por lo que, si no fueron errores de lógica y fueron por ejemplo de la base de datos, dichos mensajes se muestran en el frontend en inglés, el resto sí se encuentran en español.
- La extracción de los DTO de la WebAPI provocó que la capa de lógica asumiera más responsabilidades:
- En el afán por mejorar las métricas y desacoplar al dominio de paquetes, se decide que la WebAPI sólo funcionará con DTOs, por lo que la capa de lógica se pasa a encargar de convertirlos para la capa de datos, luego para quitarle dicha responsabilidad se implementa que el paquete de DTO se convierta a sí mismo, decisión que llevó a que, si bien esta mejor que antes, una vez estudiado el tema verificamos que no fue la mejor opción. Dicha situación se estudia en detalle en posteriores secciones.
- Luego de la realización del estudio de métricas, se verifica que los paquetes poseen poca cohesión relacional:
- Se manejó la realización de paquetes para separar las clases en las capas correspondientes y planeando extensibilidad, pero se vieron las mediciones de las métricas una vez finalizada la aplicación, por lo que no se pudo utilizar como guía de mejora durante el proceso de creación. Una vez estudiado, se nota que dentro de algún paquete se podía haber extraído ciertos métodos comunes a todos en otras clases, por ejemplo, o unir clases dentro de algún paquete para generar mayor cohesión. Se analiza en detalle en la sección de métricas.

1.3 Diagrama general de paquetes



1.3.1 Dominio

Contiene las entidades más básicas del proyecto que se identificaron del problema a resolver.



Cambios respecto a la primera entrega:

- Se agrega la clase **Tarea**
- En la clase **Incidente** se agrega el atributo: Duración
- En la clase **Usuario** se agrega el atributo: ValorHora
- En la clase **Proyecto** se agrega listado de tareas.

Decidimos mantener los usuarios en la misma clase y no aplicar herencia dado que el único atributo que podría diferir era el ValorHora, que simplificamos restringiendo a que el administrador tenga valor 0.

1.3.2 Lógica

En este paquete se encuentran los gestores de cada repositorio, la implementación de las interfaces de la capa `ILogicalInterfaz`.

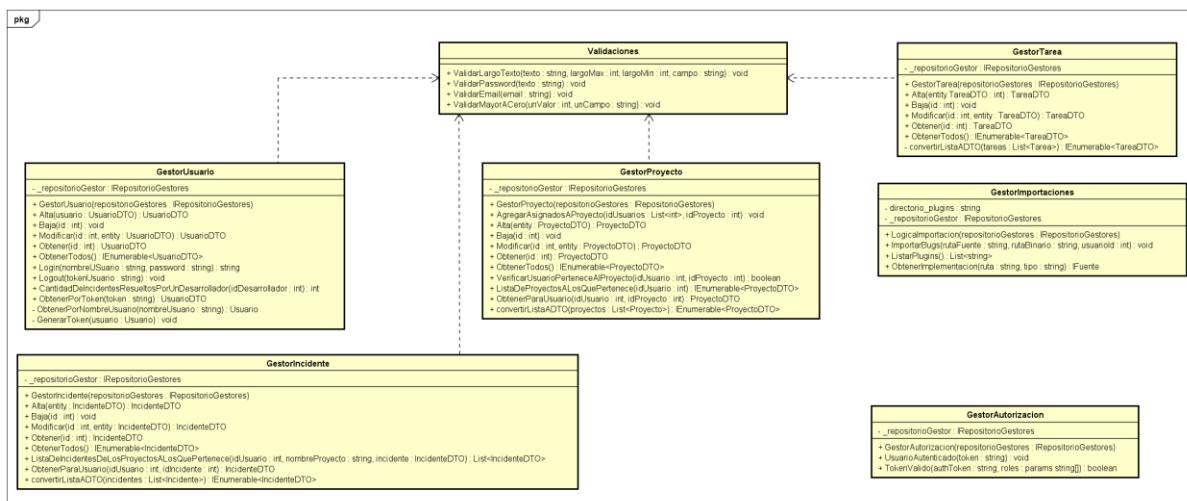
Con el objetivo de reutilizar código, creamos la clase **Validaciones**, la cual es utilizada por distintos Gestores para hacer comprobaciones, entre ellas :

- Largo del texto
 - Validar número mayor a cero
 - Validar password
 - Validar email

Para cada entidad del dominio se creó un gestor, que tiene básicamente la responsabilidad implementar el CRUD, interactuando con el repositorio correspondiente.

El GestorImportaciones tiene la responsabilidad de gestionar las implementaciones de terceros sobre las importaciones. Dentro de ello se encarga de listar los plugins que se encuentren dentro de una carpeta determinada en el servidor e implementen la interfaz **IFuente**. Además es el encargado de realizar reflexión tomando una librería determinada y utilizando los métodos de la misma.

El **GestorAutorizacion** sirve para comprobar si un usuario está autenticado, verificando si el token de este es válido. También tiene una funcionalidad que comprueba si un usuario está autorizado según su rol a utilizar una funcionalidad.



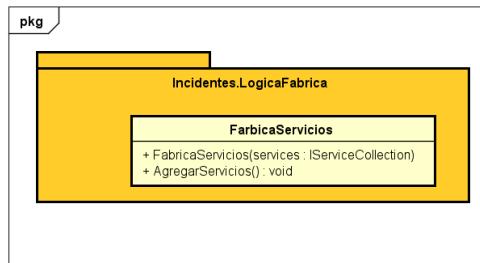
Cambios respecto a la primera entrega:

Con el objetivo de aplicar los principios a nivel de paquetes se achican el siguiente paquete apuntando al futuro mantenimiento y al reuso (REP y CRP) se extrajo:

- El paquete de excepciones a otro paquete independiente.
 - El paquete de fuentes para permitir desarrollo de terceros hacer sus propias implementaciones para las importaciones implementando reflexión que luego será detallado en la documentación.
 - El paquete DTO a otro paquete independiente.

1.3.3 LogicaFabrica

El paquete LogicaFabrica tiene como responsabilidad hacer la inyección de dependencia entre la capa de lógica y la WEB API así de esta manera logramos aplicar el principio de inversión de dependencias haciendo que un módulo de alto nivel NO dependa de un módulo de más bajo nivel.



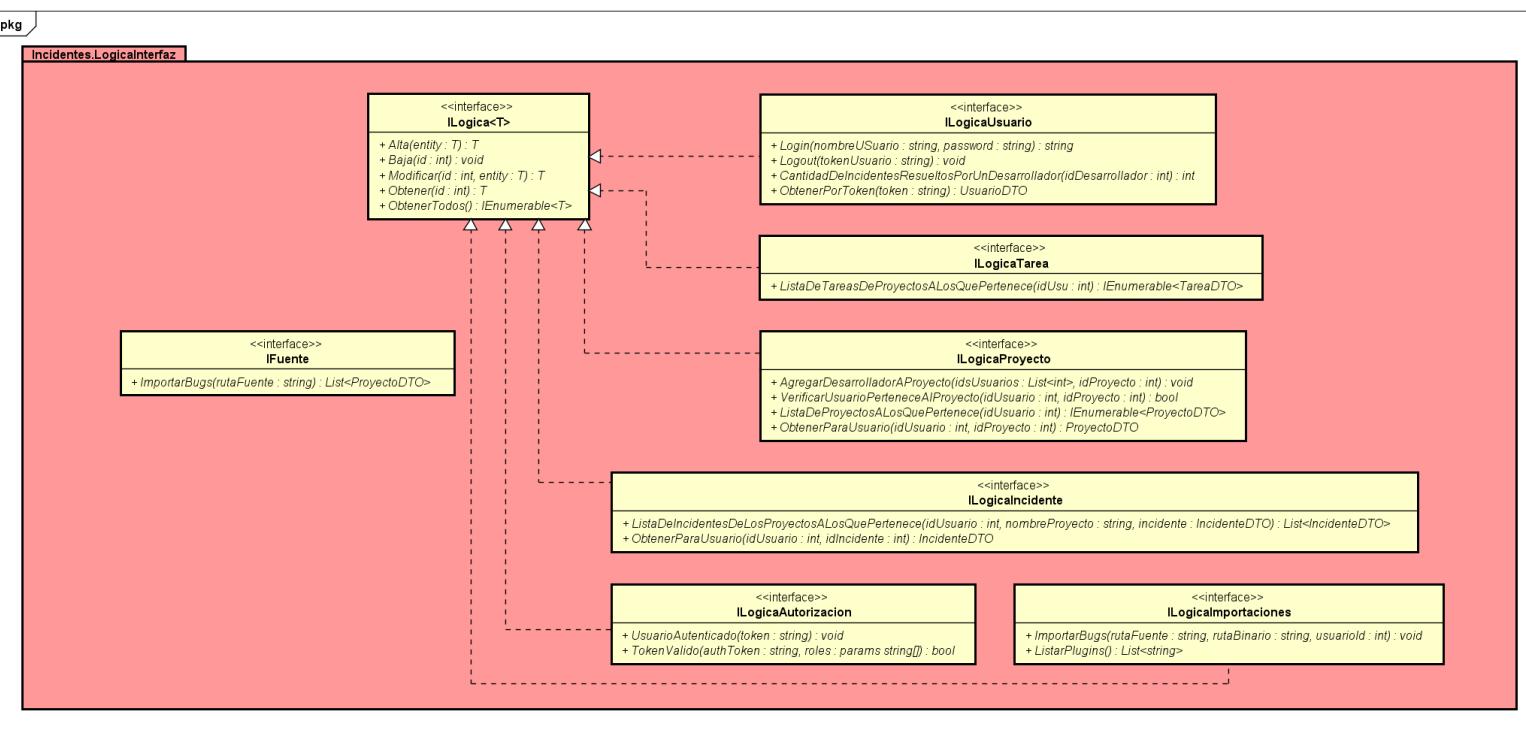
1.3.4 LogicalInterfaz

El paquete lógica interfaz puntualmente provee un conjunto de interfaces que serán implementadas por clases del paquete de lógica, y utilizadas a más alto nivel con el objetivo de evitar acoplamiento e invertir dependencias.

La clase **IFuente** ofrece una interfaz para que desarrolladores externos implementen sus propios códigos para crear importaciones de bugs en sus propios formatos sin tener que compilar nuestro código.

Para reutilizar código en Altas, Bajas, Modificar, Obtener y Obtener todos, se implementa una clase **ILogica<T>** Generics y Herencia, estableciendo una firma de método sin importar de qué tipo sea la entidad.

Esta estrategia de implementaciones de interfaz nos permite evitar que módulos de alto nivel dependan de módulos de bajo nivel.

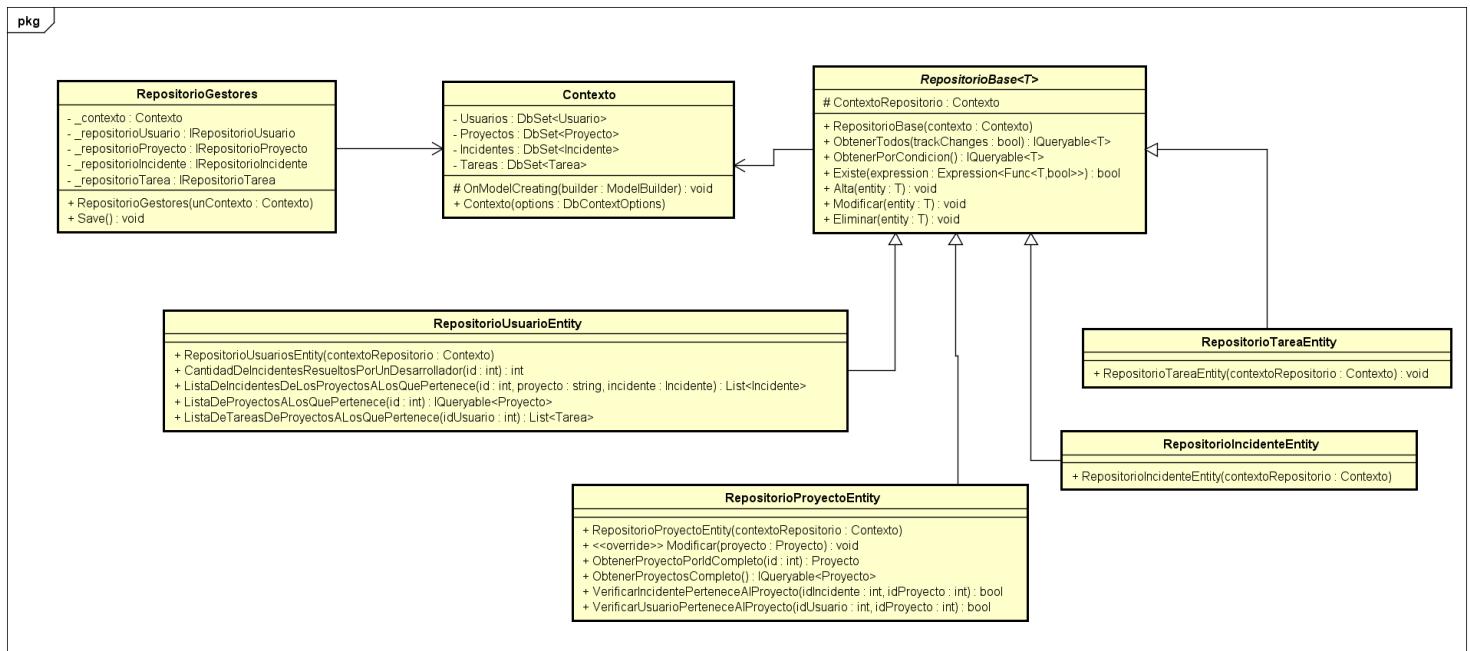


Cambios respecto a la primera entrega:

- Se agrega la interfaz ILogicaTarea
- Se agrega la interfaz ILogicalImportaciones que antes se encontraba como única clase en un paquete.

1.3.5 Datos

Este paquete tiene la responsabilidad de implementar las funcionalidades para interactuar con la BD, acoplándose con EntityFrameworkCore. Como podemos ver en el siguiente diagrama, tenemos los repositorios necesarios para trabajar con la base de datos:



Se utilizó Herencia en conjunto con la implementación de Generics <T> con el objetivo de reutilizar código creando un **RepositorioBase<T>** abstracto que implementa funciones comunes a todos los repositorios que son: Alta, Modificar, Eliminar, ObtenerPorCondicion, ObtenerTodos y de esta manera ahorraremos repetir el mismo código en cada repositorio.

Cada repositorio desarrollará únicamente las funciones particulares que requiera implementar fuera de las anteriores mencionadas.

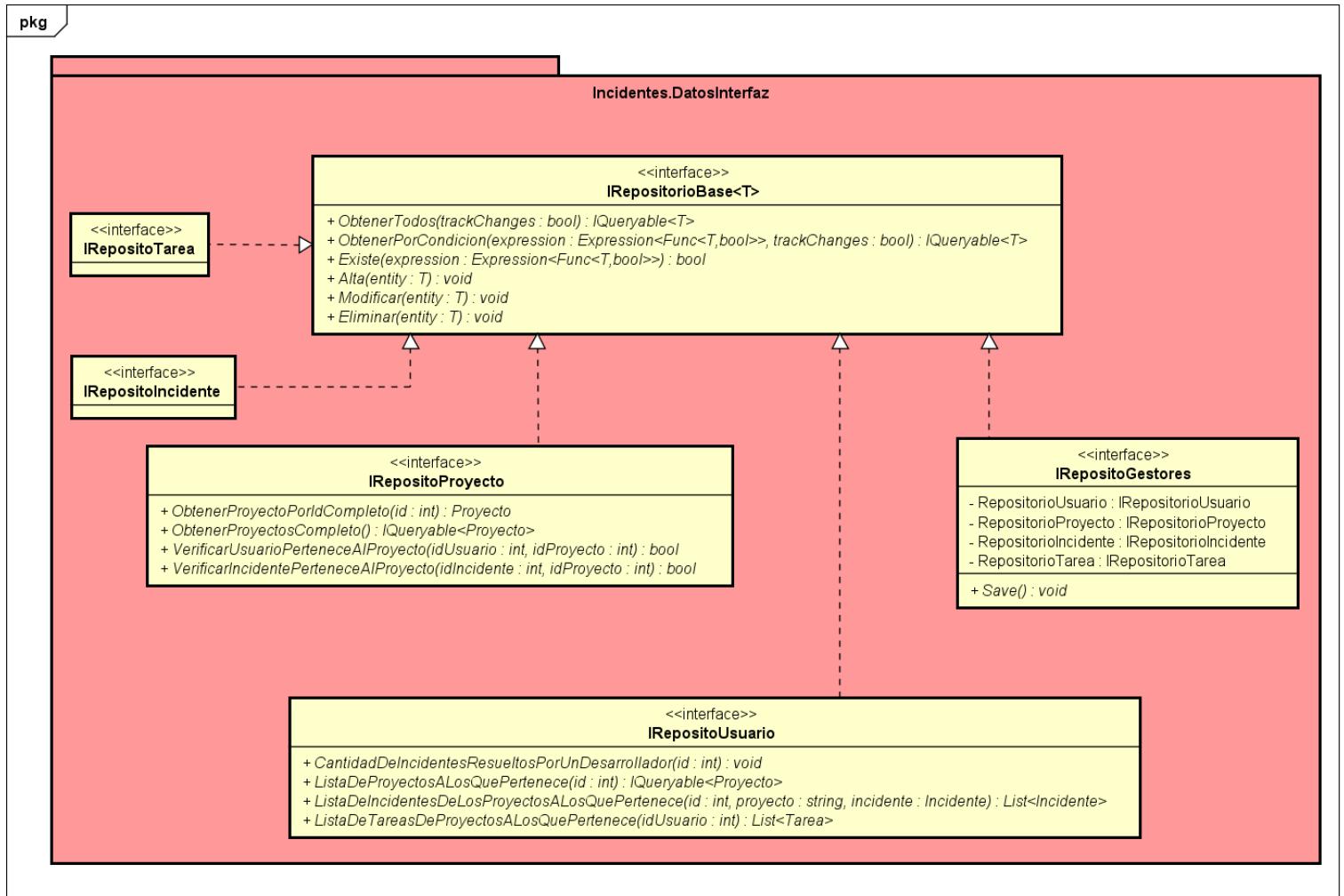
Este diseño permite no acoplarnos a un tipo de repositorio puntual cumpliendo con el principio OCP, ya que, si a futuro deseamos implementar otro tipo de repositorio, no debemos hacer cambios en las clases existentes, si no crear nuevas implementaciones que implementen la interfaz **IRepositorioBase<T>** que se detallará en el siguiente paquete.

Cambios respecto a la primera entrega:

- Se agrega la clase **RepositorioTareasEntity**.
- Se agrega el DbSet Tareas en la clase Contexto.
- Se agrega el atributo _repositorioTareas

1.3.6 DatosInterfaz

El paquete DatosInterfaz provee una serie de interfaces de repositorios, las cuales serán implementadas en el paquete de Datos que vimos anteriormente.



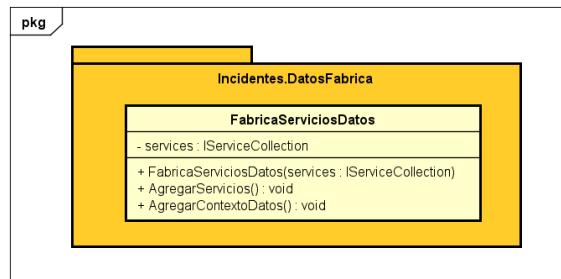
También contamos con aplicar `Generic<T>` para la interfaz `IRepositoryBase<T>` y luego cada interfaz particular además de sus firmas particulares, debe implementar `IRepositoryBase<T>` como se muestra en el diagrama y así evitar repetir los nombres de las firmas de aquellos métodos que vamos a necesitar usar en todos los repositorios (CRUD).

Cambios respecto a la primera entrega:

- Se agrega la interfaz `IRepositoryTarea`
- La interfaz `IRepositoryGestores` agrega el atributo `RepositoryTarea`

1.3.7 Datos Fabrica

El paquete DatosFabrica tiene una clase llamada FabricaServiciosDatos, que tiene como responsabilidad hacer la inyección de dependencias antes mencionada, y además tener un método para obtener la cadena de conexión sql a la base de datos.



Su responsabilidad es hacer la inyección de dependencia entre la web API y la capa de acceso a datos logrando que el módulo de la web API no tenga dependencias con la capa de acceso a datos ya que es aquí donde hacemos agregamos el contexto y la cadena de conexión, quitando esta responsabilidad del Startup de la web API y así pudiendo ser extensible a cualquier otra plataforma.

1.3.8 Excepciones

Este paquete contiene las clases con las excepciones creadas por nuestro equipo que son las siguientes:

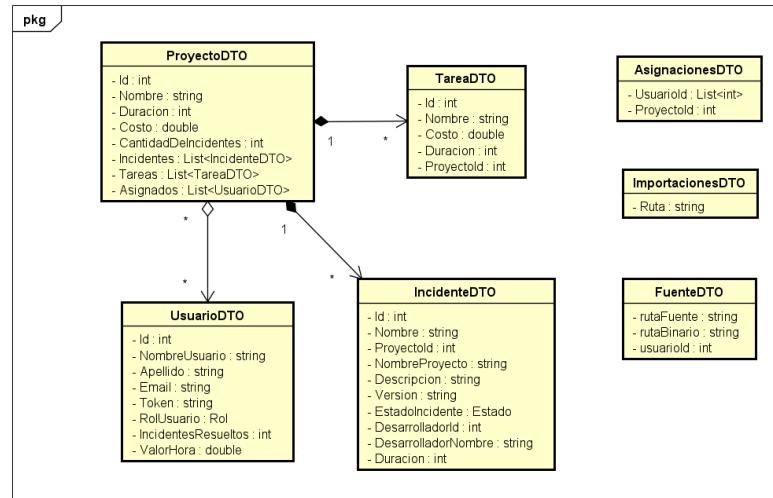
- **ExpcionAccesoNoAutorizado:** es lanzada cuando un usuario autenticado pretende utilizar una funcionalidad que no corresponde con su rol o también cuando el usuario intenta acceder o modificar datos de un proyecto el cual no pertenece.
- **ExpcionArgumentoNoValido:** es lanzada cuando se ingresa en un campo de texto algún carácter no permitido, o el formato no coincide con lo solicitado como por ejemplo al querer ingresar un email que no cumpla con las condiciones de un mail valido.
- **ExpcionElementoNoExiste:** se produce cuando se pretende acceder a un elemento inexistente, como por ejemplo agregar un usuario a un proyecto que no exista, o eliminar un objeto que ya fue borrado.
- **ExpcionLargoTexto:** se produce cuando se pretende ingresar un dato con la cantidad de caracteres menor o mayor al establecido por las reglas del negocio.
- **ExpcionNumeroNoValido:** se produce cuando en un campo que esperamos un valor numérico se ingresa un texto.

Decidimos agruparlas en un paquete para aplicar los principios a nivel de paquetes, colocando en uno mismo clases que tengan el mismo sentido, especialmente el Principio de reuso común (CRP) que apunta al reuso.

1.3.9 Dtos

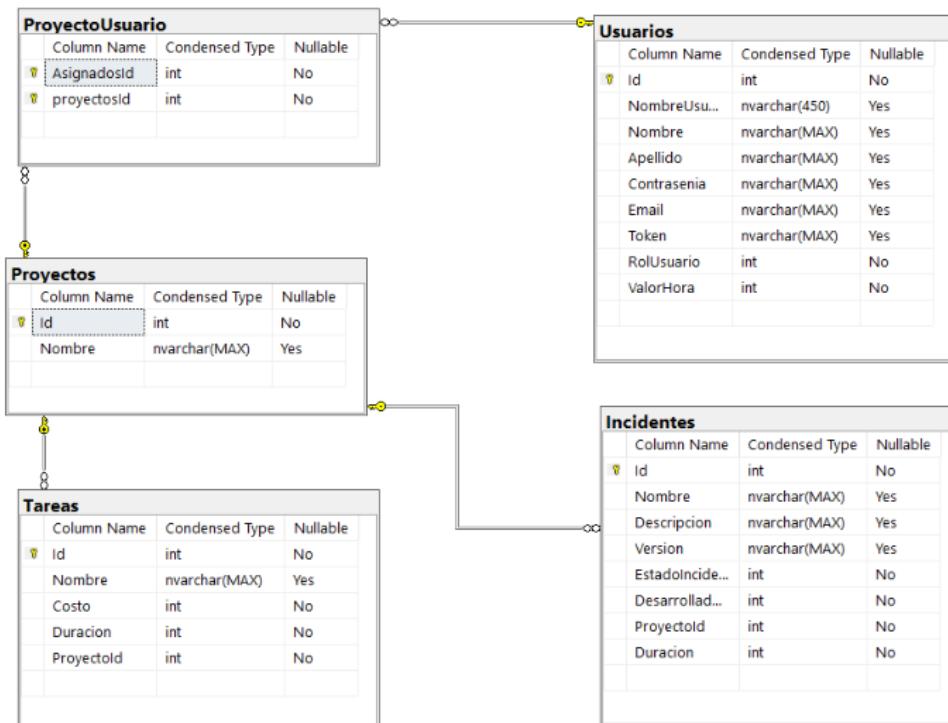
Siguiendo las sugerencias en la devolución de la primera entrega, agrupamos los DTOs en un solo paquete con el objetivo de uniformizar, ya que antes los teníamos en dos paquetes distintos objetos DTO (en WEBAPI y Lógica).

Los DTOs tienen la responsabilidad de transmitir información desde y hacia la API, con el objetivo de no exponer directamente nuestros objetos de dominio, sus atributos y su lógica.



1.4 Modelo de tablas de la estructura de la base de datos.

La base de datos representa los objetos del dominio del sistema, la relación N a N entre usuarios y proyectos y la relación 1 a N entre incidentes y proyectos.



Cambios respecto a la primera entrega:

- Se agrega la tabla **tareas**
- En la tabla usuarios se agrega el campo **ValorHora**
- En la tabla incidentes se agrega el campo **Duración**

1.5 Justificación del diseño

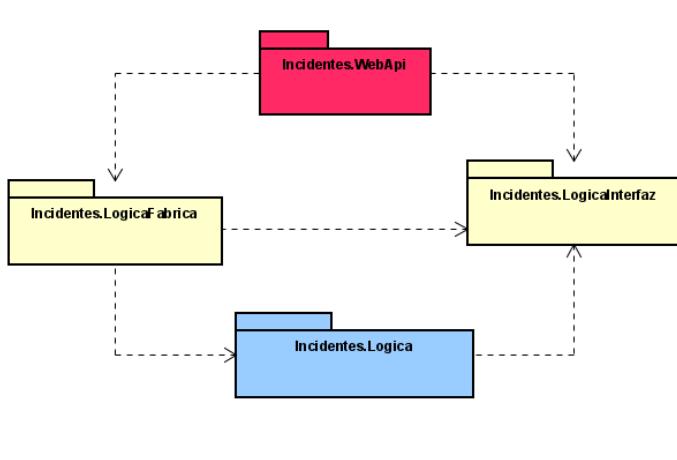
1.5.1 Uso de principios y patrones de diseño

En los puntos anteriores se mencionaron las inyecciones de dependencia que fueron utilizadas entre los paquetes WebApi, Lógica y la capa de Datos. Además, se crean las fábricas correspondientes para que haya total independencia entre estos módulos principales.

De esta manera logramos que un componente reciba sus dependencias en lugar de instanciarlas para evitar violar SRP, el acoplamiento se da entre interfaces evitando violar OCP y al no revertir las dependencias evitamos violar DIP.

Este punto sigue igual que la entrega anterior, pero se agrega la nueva gestión de tareas.

Inyección de dependencia y fábricas para desacoplar de lógica y WebAPI

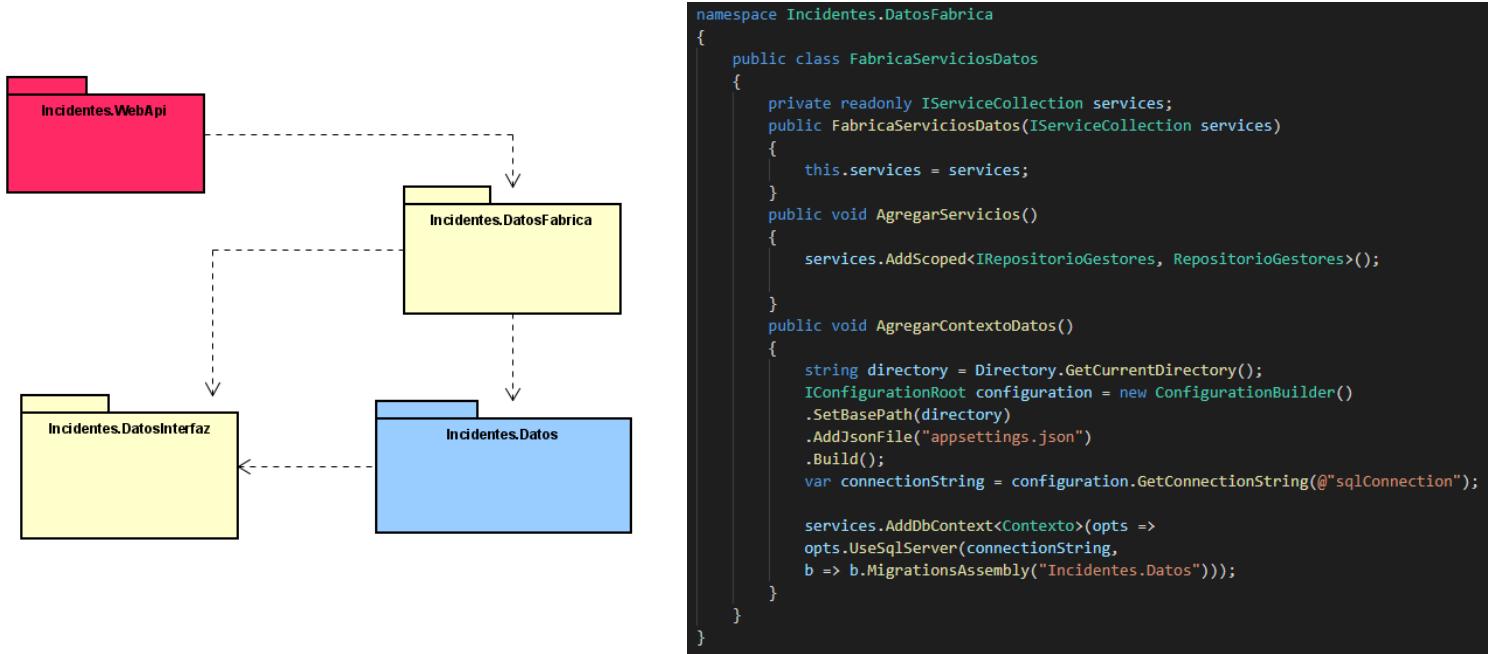


```
public class FabricaServicios
{
    private readonly IServiceCollection services;
    public FabricaServicios(IServiceCollection services)
    {
        this.services = services;
    }
    public void AgregarServicios()
    {
        services.AddScoped<ILogicaUsuario, GestorUsuario>();
        services.AddScoped<ILogicaProyecto, GestorProyecto>();
        services.AddScoped<ILogicaIncidente, GestorIncidente>();
        services.AddScoped<ILogicaAutorizacion, GestorAutorizacion>();
        services.AddScoped<ILogicaTarea, GestorTarea>();
        services.AddScoped<ILogicalImportaciones, GestorImportacion>();
    }
}
```

Cambios respecto a la primera entrega:

- Se agrega la inyección de dependencia <ILogicaTarea, GestorTarea>
- Se agrega la inyección de dependencia <ILogicalImportaciones, GestorImportacion>

Inyección de dependencia y fábricas para desacoplar de datos y WebAPI



Se utilizó SINGLETON para la creación de los repositorios:

- RepositorioUsuario
- RepositorioProyecto
- RepositorioIncidentes
- RepositorioTarea

```

namespace Incidentes.Datos
{
    public class RepositorioGestores : IRepositoryGestores
    {
        private Contexto _contexto;
        private IRepositoryUsuario _repositorioUsuario;
        private IRepositoryProyecto _repositorioProyecto;
        private IRepositoryIncidente _repositorioIncidente;

        public RepositorioGestores(Contexto unContexto)
        {
            _contexto = unContexto;
        }

        public IRepositoryUsuario RepositorioUsuario
        {
            get
            {
                if (_repositorioUsuario == null)
                    _repositorioUsuario = new RepositoryUsuariosEntity(_contexto);

                return _repositorioUsuario;
            }
        }
    }
}

```

Aplicación del patrón fabricación pura

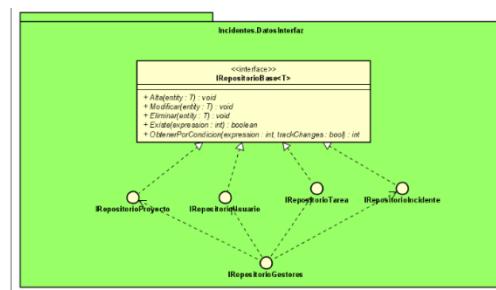
En el paquete de lógica para que la clase GestorIncidentes no se encargue de las importaciones de terceros, se crea una clase “inventada” (no surge del dominio) que es el GestorImportaciones, de esta forma ambas clases tienen un único motivo por el cual cambiar por lo a su vez cumplen con el principio de responsabilidad única SRP manteniendo una alta cohesión y bajo acoplamiento.

También se aplica el mismo patrón al crear las fábricas antes mencionadas:



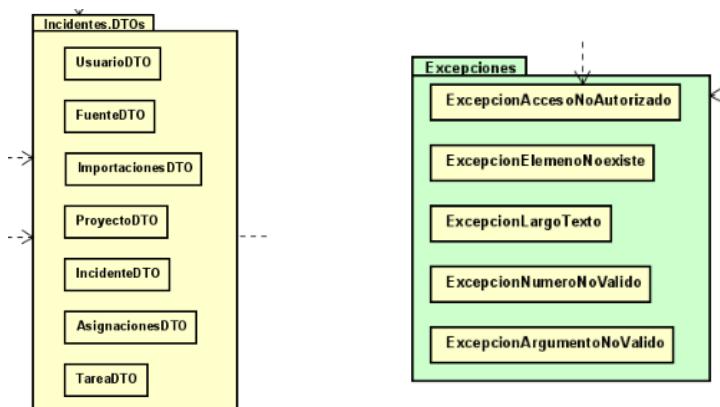
Realization, Generics <T>, punto de entrada al paquete “fachada”

Para interactuar con el paquete repositorio, expusimos una interfaz que oficia de punto de entrada para interactuar con cualquier otro repositorio, intentando que **IRepository** actúe de una especie de “fachada”. Además se reutiliza código heredando de **IRepositoryBase<T>**



Principio de clausura común

Agrupamos las clases DTOs en un mismo paquete, al igual que las excepciones:



1.5.2 Descripción del mecanismo de acceso a datos utilizado

La persistencia de datos fue realizada utilizando Entity Framework code FIRST, de esta manera definimos las clases mediante código y luego EF se encargó de generar la base de datos y todo lo necesario para mapear nuestros objetos a las tablas creadas.

Las clases que implementan la lógica de guardado de datos para entity están agrupadas en el siguiente paquete:



En este paquete si hicieron migraciones granulares, las cuales fueron impactando nuestra base de datos, teniendo cada migración un nombre representativo.

En la clase contexto se encuentran las propiedades `DbSet` para cada entidad que será almacenada en la base de datos.

```
0 references
public DbSet<Usuario> Usuarios { get; set; }
0 references
public DbSet<Proyecto> Proyectos { get; set; }
0 references
public DbSet<Incidente> Incidentes { get; set; }
0 references
public DbSet<Tarea> Tareas { get; set; }
```

1.5.3 Descripción del manejo de excepciones

Al contrario del obligatorio anterior, el manejo de errores se extrajo de la capa de lógica y se trató en una clase aparte, logrando así que lógica no violara el principio SRP y se logran independizar los paquetes que la utilizaban de ella. Innecesariamente los paquetes como el WebAPI al utilizar las excepciones tenían dependencia del paquete de lógica, lo que generaba un acoplamiento incorrecto.

En el nuevo paquete de excepciones, mantuvimos las clases de excepciones creadas en el obligatorio anterior. En el paquete de lógica como vimos en el diagrama en el punto 1.3.2 capturamos las diversas excepciones que se pueden generar durante la ejecución del programa, y luego lanzamos las mismas al paquete de la API donde son controladas y mostradas de manera clara para que el usuario comprenda que ocurrió con la información necesaria.

1.5.4 Decisiones de diseño propias

Utilizando como base la letra del obligatorio y las consultas del foro se tomaron las siguientes decisiones:

- Para los usuarios se creó una clase Usuario, y cada uno de los roles existentes (administrador, desarrollador, tester) se guarda en el atributo Rol. Al principio del proyecto lo habíamos separados en clases que heredaban de Usuario, pero a medida que fuimos avanzando nos dimos cuenta de que las distintas clases concretas no tenían atributos tan específicos a cada clase, sino que la diferencia es según el rol a que funcionalidades pueden acceder por lo que no justificaba hacer la herencia.
- El usuario administrador se cargó directamente en la base de datos ya que necesitamos un usuario previamente registrado para poder ejecutar las funcionalidades.
- Los incidentes pueden tener dos estados: activos o resueltos.
- Desarrolladores y Tester pueden resolver incidentes, tomamos este supuesto en base a una consulta en el foro
- Cuando un desarrollador cambia el estado de un incidente a resuelto, automáticamente se la asigna a dicho desarrollador el atributo Desarrollador Id.
- Para utilizar la funcionalidad de importar incidentes decidimos que sea visible solo para usuarios con rol administrador.

1.5.5 Descripción de extensibilidad de importaciones de bugs

En esta segunda entrega se solicita implementar una extensibilidad de los importadores, para que diferentes entregas puedan realizar sus propias implementaciones de los reportes de bugs hacia nuestra aplicación. Se solicitaba claramente que dicha implementación pudiera ser realizada en tiempo de ejecución, por lo que se tuvo que aplicar reflexión para implementar la misma.

Desde el obligatorio anterior tuvimos el reto de que fuera lo más extensible posible la importación de los bugs, por lo que lo habíamos implementado una interfaz **IFuente** que tenía un solo método que importaba los bugs, y una clase lógica que la utilizaba, llamando a dicha clase (**ImportacionesTXT**, **ImportacionesXML**) para que ella misma analizara la entrada y realizara una importación de sus bugs.

Para realizar el cambio solicitado por la letra del obligatorio, descubrimos que la solución del obligatorio 1 no poseía la extensibilidad necesaria. Dicho método de implementación poseía lógica que conocía los repositorios de la aplicación, lo que generaba dependencia de ellos.

Este fue el momento en el que reflexionamos la importancia de mantener las clases que serán extensibles lo más desacopladas posibles.

Para solventar lo anterior extraemos dicha lógica a un gestor como el resto de las entidades trabajadas, y provocamos que el método de la interfaz nos devuelva una lista de DTOs de Proyecto.

La clase anterior es un ejemplo de lo mismo, es la implementación por parte de una “empresa” por medio de archivos JSON. Como vemos tiene dependencia de la **IFuente** (interfaz que debe implementar) y de los DTO de proyecto e incidente.

Realizando dicho cambio hacemos pública solamente información sin relevancia y no información de implementación propia de la solución.

Actualmente no se implementó una importación del assembly particular hacia la WebAPI debido a que no se solicitaba en la letra y mantuvimos el foco en otras solicitudes, por lo que se supone que se encuentran físicamente en una dirección específica del servidor, por lo que la aplicación en tiempo de ejecución lee de dicha carpeta todas las librerías que implementan la interfaz **IFuente** y las devuelve en una respuesta HTTP. Dicha funcionalidad la tiene el GestorImportaciones. También se encarga de recibir una ruta de una implementación y aplicársela a un archivo (ambos recibidos por parámetro). De esta manera sólo basta con agregar librerías a dicha carpeta como para que en tiempo de ejecución puedan ser tomadas e implementadas por la aplicación.

```
namespace Incidentes.LogicaInterfaz
{
    public interface ILogicaImportaciones
    {
        public void ImportarBugs(string rutaFuente, string rutaBinario, int usuarioId);
        public List<string> ListarPlugins();
    }
}
```

Se detalla en documento anexo contrato al cual se debe apegar la empresa.

1.5.6 Implementación de DTO

En la anterior entrega utilizamos los DTO de forma incorrecta, se presentaba un grupo de DTO en la lógica y otro grupo en la WebAPI. En la presente se corrigió dicha implementación realizando un único paquete de DTOs, el mismo se crea en parte para poder desacoplar las implementaciones de importaciones de la lógica de negocio.

Asimismo, descubrimos, primero con la modificación de los objetos de dominio y luego con la implementación de métricas, la importancia de la existencia de los DTO. Al implementar las nuevas clases de dominio solicitadas nos dimos cuenta del impacto que realiza sobre la solución entera que un paquete de bajo nivel sufra un cambio. Posteriormente, al ir implementando el frontend, se nos presentaron situaciones donde era conveniente que el objeto volviera con algún dato extra, por ejemplo, un incidente con el nombre del proyecto al que pertenece, por lo que empezamos a utilizar los DTO con mayor frecuencia. La importancia de tenerlos en un solo paquete radicó en no tener que realizar cambios en varios paquetes al necesitar brindar una nueva propiedad.

También se notó en el empleo de las métricas, el paquete de dominio se encontraba en una zona crítica y posteriormente a la implementación de los DTO se fue moviendo hacia una zona de mayor solvencia.

En el afán de sacar al dominio de la zona crítica y permitir que la aplicación en un futuro pueda ser implementada sin la WebAPI (ejemplo aplicación de escritorio), nos propusimos a desacoplar ciertas clases de este, por lo que desacoplamos completamente el paquete WebAPI del dominio, y pasamos a que la lógica recibiera DTOs y devolviera DTOs.

Esto provocó que se mueva un poco el dominio hacia la zona segura, pero le asignó responsabilidades a lógica que no le pertenecían (la conversión de objetos de dominio a DTO y viceversa). Dicha situación quedó en evidencia al momento de realizar la documentación y los diagramas correspondientes, por lo que no hubo tiempo para solventarlo y se presenta como un error conocido. Sabemos que no es correcto que un paquete como la lógica se vea modificado por cambios en los DTO (aunque es en menor medida ya que los DTO se convierten a sí mismos).

Estudiando el problema encontramos que una solución mejor hubiera sido mantener los DTO en la WebAPI y aplicar el mapeo de objetos mediante otro paquete encargado del mismo y que conozca tanto a los DTO como los objetos de dominio. Buscar una solución de inyección de dependencia para utilizar los repositorios de forma similar a la que lo hace la WebAPI, y sería dicho paquete el único que se modifica si desde el frontend existe la necesidad de visualizar un objeto de forma distinta. Dicha solución no tuvimos oportunidad de implementarla en el obligatorio por razones de tiempo.

1.5.7 Análisis de la calidad del código

Para la toma de métricas se utiliza la herramienta NDepend. La misma se realiza en varias instancias para que se evidencie el cambio de estas si es que ocurre.

En cuanto al análisis de los datos tomados, nos vamos a basar en las siguientes medidas: H (cohesión relacional), I (inestabilidad), A (Abstracción) y D (distancia de la secuencia principal).

Además de las medidas tomadas por la aplicación, se procede a replicarlo con las fórmulas vistas en clase, a efecto de no tomar en cuenta los paquetes de terceros que sí toma la aplicación.

Cuadro de métricas

Assemblies	# lines of code	# IL Instruction	# Types	# Abstract Types	# lines of comment	% Comment	% Coverage	Afferent Coupling	Efferent Coupling	Relational Cohesion	Instability	Abstractness	Distance
Incidentes.Dominio v1.0.0.0	62	227	6	0	0	0	-	17	9	1.17	0.35	0	0.46
Incidentes.DatosInterfaz v1.0.0.0	0	0	6	6	-	-	-	13	11	1.5	0.46	1	0.32
Incidentes.Excepciones v1.0.0.0	15	90	5	0	0	0	-	13	3	0.2	0.19	0	0.57
Incidentes.DTOs v1.0.0.0	189	1003	9	0	0	0	-	22	22	0.67	0.5	0	0.35
Incidentes.LogicalInterfaz v1.0.0.0	0	0	8	8	-	-	-	20	12	0.62	0.38	1	0.27
Incidentes.Logica v1.0.0.0	344	2854	7	0	28	7.53	-	1	75	0.71	0.99	0	0.01
Incidentes.Datos v1.0.0.0	925	12708	31	0	13	1.39	-	1	91	0.94	0.99	0.03	0.02
Incidentes.LogicaFabrica v1.0.0.0	8	34	1	0	0	0	-	1	16	1	0.94	0	0.04
Incidentes.DatosFabrica v1.0.0.0	9	65	1	0	0	0	-	1	25	1	0.96	0	0.03
Incidentes.WebApi v1.0.0.0	226	1381	13	0	0	0	-	0	91	1.31	1	0	0

Paquete	Ce	Ca	I	R	Na	Nc	H	A	D
WebAPI	27	0	1,00	15	0	11	1,45	0	0,00
Logica	30	6	0,83	5	0	7	0,86	0	0,12
Dominio	0	21	0,00	3	0	4	1,00	0	0,71
DatosInterfaz	2	13	0,13	0	6	6	0,17	1	0,09
Excepciones	0	8	0,00	0	5	5	0,20	1	0,00
DTOs	4	21	0,16	3	0	7	0,57	0	0,59
LogicalInterfaz	5	24	0,17	6	8	8	0,88	1	0,12
Datos	11	2	0,85	6	0	7	1,00	0	0,11
LogicaFabrica	2	1	0,67	0	0	1	1,00	0	0,24
DatosFabrica	2	1	0,67	0	0	1	1,00	0	0,24

Abstracción

La abstracción es la medida que nos indica cuan abstracto es un paquete, a mayor abstracción, más clases abstractas e interfaces tiene el mismo.

Podemos ver en el cuadro que las dos clases con mayor abstracción son DatosInterfaz y LogicalInterfaz, claramente porque ambas cuentan con las firmas que deben implementar las clases de lógica y de datos.

El resto de los paquetes cuentan con una abstracción de cero, esto se debe a que sólo contienen clases concretas, que implementan dichas interfaces.

Inestabilidad

La inestabilidad es mide la relación entre las conexiones salientes y el total de conexiones salientes y entrantes, es la representación de que tan inestable es un paquete, siendo 0 muy estable y 1 muy inestable.

Se obtiene que la WebAPI es completamente inestable, ningún paquete depende de él, lo cual es coherente, por otro lado, el dominio es completamente estable, ya que no depende de ningún paquete externo, en igual situación se encuentran las excepciones, lo cual es coherente ya que no dependen de ningún paquete externo.

En el punto medio se encuentran el resto de las clases

Cohesión Relacional

Mide la cohesión entre las clases de un paquete, indica que tan relacionadas están entre ellas. Se toma como correcto que se encuentre entre 1.5 y 4.

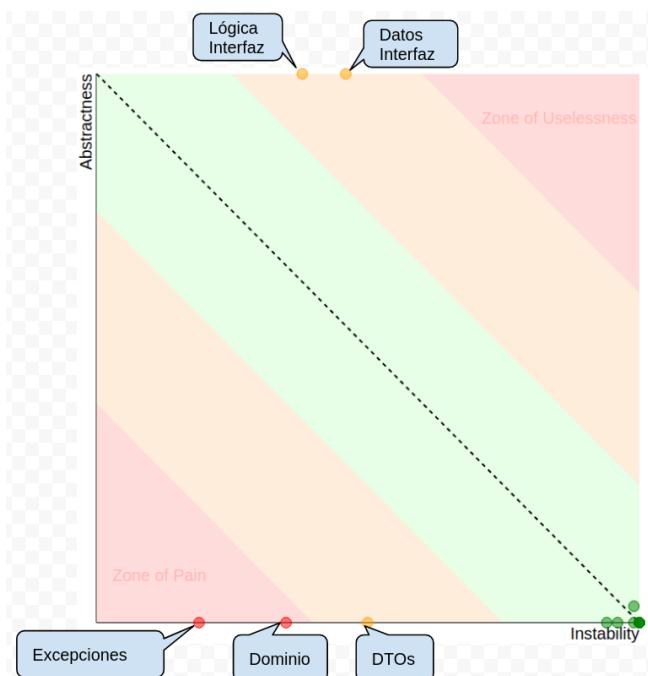
En la tabla vemos que el único que se aproxima a la cohesión solicitada es la clase WebAPI. Dicha situación se debe a que se intentó separar en paquetes distintos las distintas clases por tipo de clase, por ejemplo, todos los gestores en un paquete, todas las interfaces en otro, etc. Y una vez aprendidas las métricas nos encontramos que puede ser que existan más paquetes que los necesarios para la implementación solicitada.

Se establece esta situación como un error conocido, debido a que no se contó con el tiempo para solucionarlo.

Dentro de las posibles soluciones se encuentra:

- Se parar funcionalidades comunes en las clases de un paquete a una clase aparte, lo que generaría más conexiones entre ellas.
- Se puede reducir la cantidad de paquetes, separando las clases de los paquetes de interfaz por ejemplo para generar acoplamiento entre las clases, cuidando no violar el DIP.

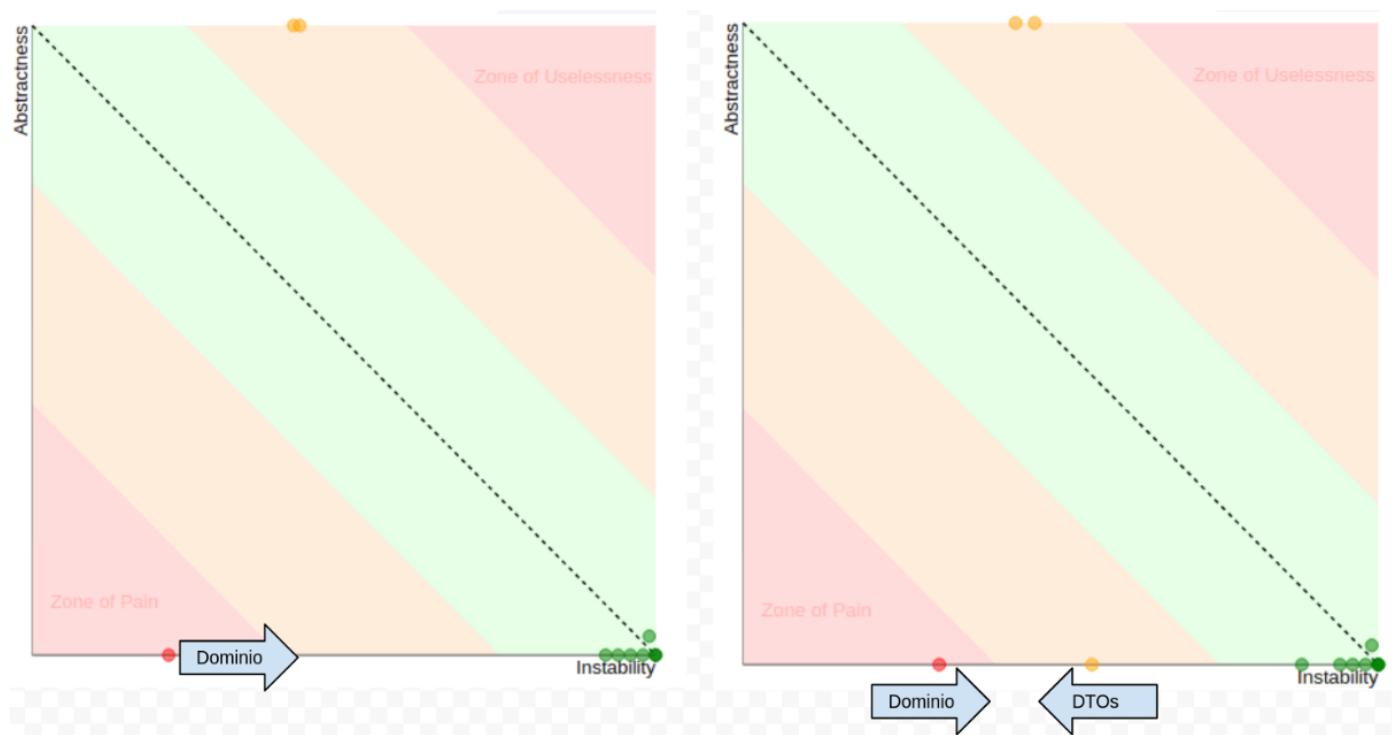
Abstracción vs Inestabilidad



En la gráfica se muestra que la mayoría de los paquetes se encuentran dentro de la zona correcta, se marcan los paquetes que no lo realizan, que son excepciones, dominio, DTOs, lógica interfaz y datos interfaz.

En la gráfica se pueden ver que ciertos paquetes se encuentran en la zona de pánico, esto se dio porque paquetes de clases concretas resultaron ser utilizados por otras clases, por ejemplo, el paquete de dominio, que posee únicamente clases concretas y es utilizado por gran parte de la aplicación. De aquí que se notó un gran impacto al implementar los cambios solicitados para esta segunda entrega. Nos deja claro como informáticos que los cambios que más afectan son los que provocan cambios en el dominio.

A fin de solventar un poco el impacto, se implementó el paquete de DTOs, el cual se muestra en la figura de abajo, cómo el implementarlo conllevó a que el dominio se desplazara a la derecha y los DTOs a la izquierda.

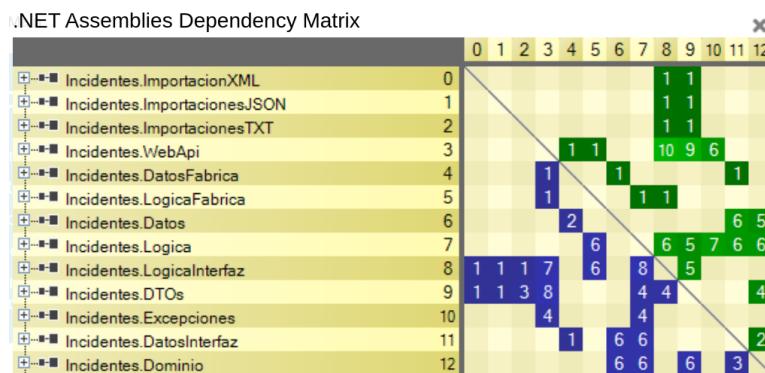


Principio de Dependencias Estables

Para verificar dicho principio, se ordena la tabla en sentido de la inestabilidad, y con la matriz de abajo se verifica que se cumpla dicho principio.

El principio de dependencias estables establece que un paquete debe depender de paquetes que sean más estables que él. Y no en el caso contrario.

Paquete	Ce (dep. salientes)	Ca (dep. entrantes)	I inestabilidad (Ce/(Ce + Ca))	Depende de más estable
Dominio	0	21	0,00	Cumple
Excepciones	0	8	0,00	Cumple
DatosInterfaz	2	13	0,13	Cumple
DTOs	4	21	0,16	Cumple
LogicalInterfaz	5	24	0,17	Cumple
LogicaFabrica	2	1	0,67	No, usa lógica que es más inestable
DatosFabrica	2	1	0,67	No, usa datos que es mas inestable
Logica	30	6	0,83	No, depende de él lógica fábrica que es más estable
Datos	11	2	0,85	No, depende de él datos fábrica que es más estable
WebAPI	27	0	1,00	Cumple



Luego de ver los resultados se encuentran dos violaciones a dicho principio, ambas son las fábricas, tanto de datos como de lógica. El eliminar dichas fábricas provoca que la WebAPI conozca directamente las clases de lógica y de datos, si bien cambiar la forma de almacenado es poco común, puede ser común el cambiar alguna clase de lógica, por ejemplo, una nueva implementación de ILogicaProyecto que sea más performante, por lo que dicho cambio también provocaría el recompilado de la WebAPI, en cambio al existir la fábrica esto no sucedería, se harían los cambios en la fábrica correspondiente. En nuestro caso decidimos desde el inicio del obligatorio utilizar esta segunda opción.

1.6 Resumen de mejoras al diseño respecto a la primer entrega

El implementar los cambios en esta segunda entrega no tuvo un mayor impacto, si bien se afectaron todos los paquetes, se continuó con el mismo diseño que se tenía del obligatorio anterior.

El mayor impacto fue en la implementación de Reflexion para las importaciones, esto se debió a que no teníamos tan desacopladas las clases que implementaban dichas importaciones, ambas tenían un poco de lógica dentro, por lo que extrajimos dicha lógica y las desacoplamos lo más posible del resto de la solución. Se explica dicho cambio en la sección correspondiente.

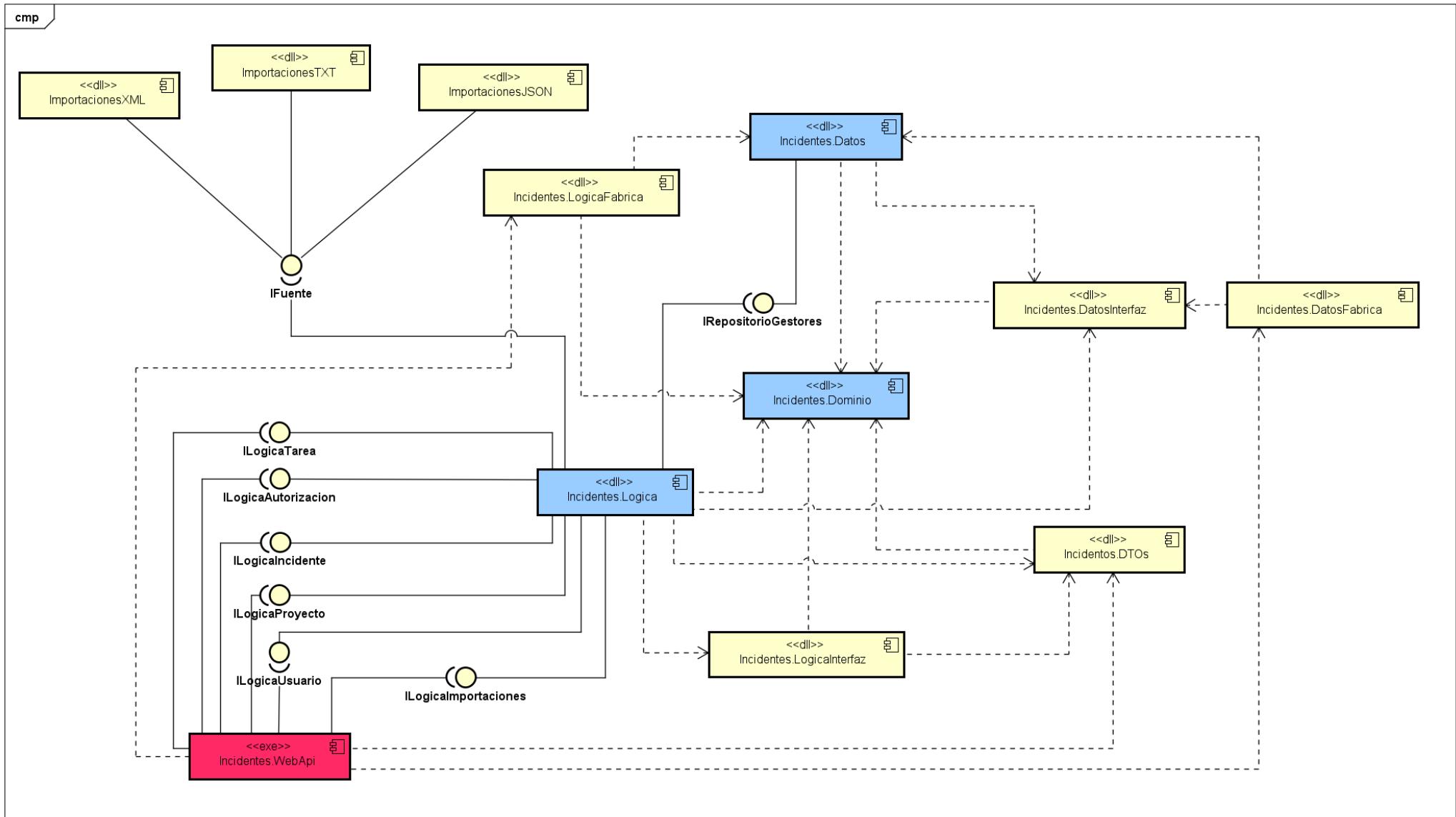
Lo que sí tuvo impacto en nuestro diseño fue el intento de mejora de las métricas, aplicar lo aprendido en la segunda sección del curso y mejorar las correcciones brindadas del obligatorio anterior.

En cuanto a las correcciones realizamos lo siguiente:

- Uniformizar DTOs: Realizamos un paquete de DTOs, a los DTO que se encontraban en la WebAPI y la lógica los quitamos hacia este nuevo paquete. Dicho cambio tuvo un impacto significativo analizado en detalle en la sección correspondiente. El uso de los DTO fue esencial para facilitar la entrega de diversos datos al frontend, situación que se manifestó al implementarlo.
- Controllers exponiendo objetos al dominio: Con la implementación de los DTO se subsanó este problema.
- Login y logout en REST: Cambiamos dicho recurso por AutenticacionesController, el mismo, en la ruta /api/Autenticacion mediante el método POST recibe un loginDTO que se compone de un nombre de usuario y una contraseña, verifica los datos y devuelve el token correspondiente con los datos necesario para que el frontend funcione correctamente. También mediante un método DELETE “elimina” dicha autenticación. Se dice “elimina” debido a que el LoginDTO no es una entidad en sí, desde el primer obligatorio guardamos el token como propiedad del usuario, por lo que en realidad vacía dicho token.
- AsociacionesController: Revisamos el método y extrajimos la mayor parte de las funciones. Esto fue posible debido a que pudimos implementar cierta autenticación mediante el token brindado en los otros controllers y verificar que el usuario que lo enviaba correspondiera al proyecto en el que intentaba realizar los cambios. Del AsociacionesController sólo queda el método para asociar usuarios a los proyectos, método que se explica en un diagrama de interacción y que en la interfaz gráfica implementada se evidencia la idea que teníamos en un principio y quizás mediante en la entrega anterior no supimos explicar.
- Analizar ReportesController: Se modifican las rutas del mismo para que sean intuitivas, anteriormente si se ingresaba a /reportes y se obtenían los proyectos con los incidentes de cada uno, ahora se ingresa mediante reportes/incidentes/proyectos, a su vez, para acceder a los incidentes resueltos por un desarrollador, se accede a reportes/:idDesarrollador/incidentes

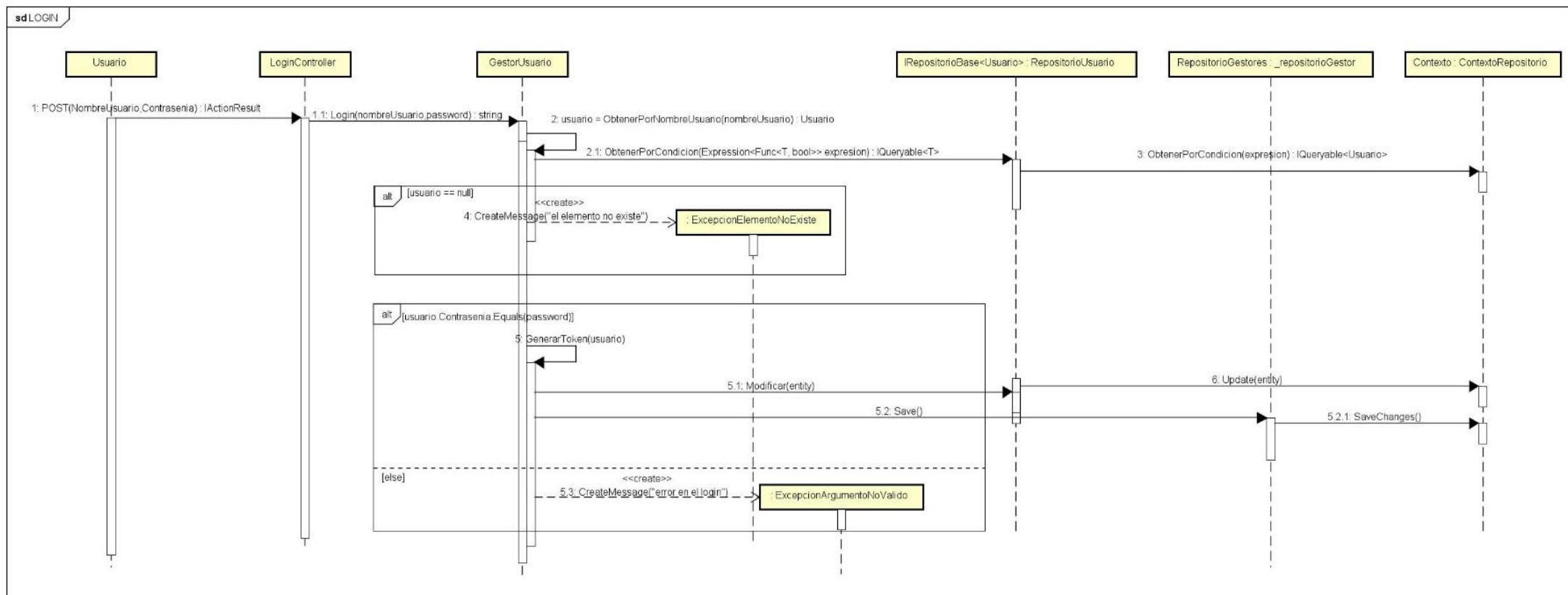
Con respecto al intento de mejora de las métricas se verificó que la clase dominio se encontraba en la zona de pánico, por lo que se intenta utilizar los DTO para moverlo a la derecha. Se ogra el cometido en cierta medida, pero resultó en extraer el dominio de la WebAPI, situación que se analiza en detalle en la sección correspondiente.

ANEXO I Diagrama de componentes

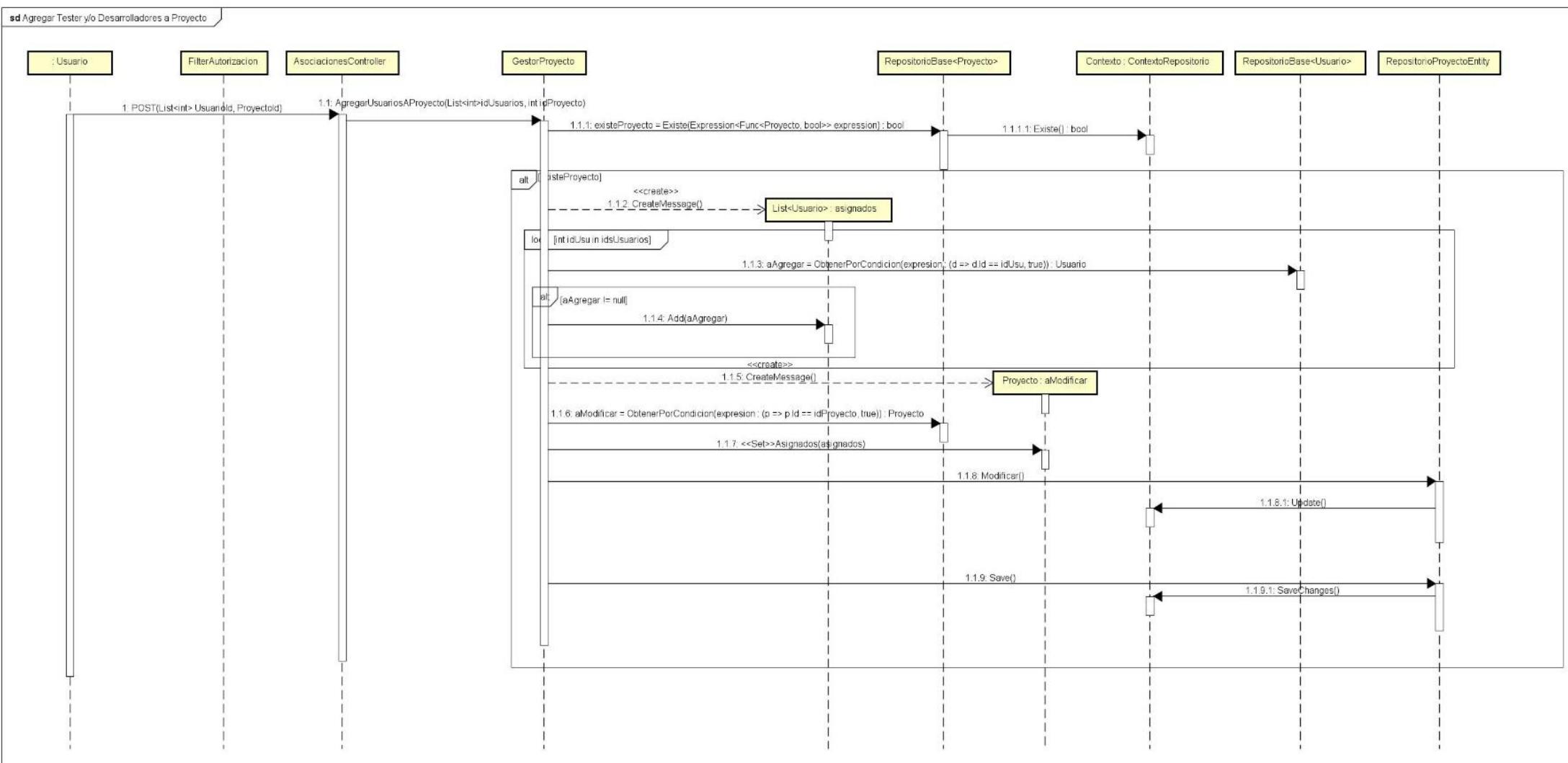


ANEXO II Diagrama de interacción relevantes

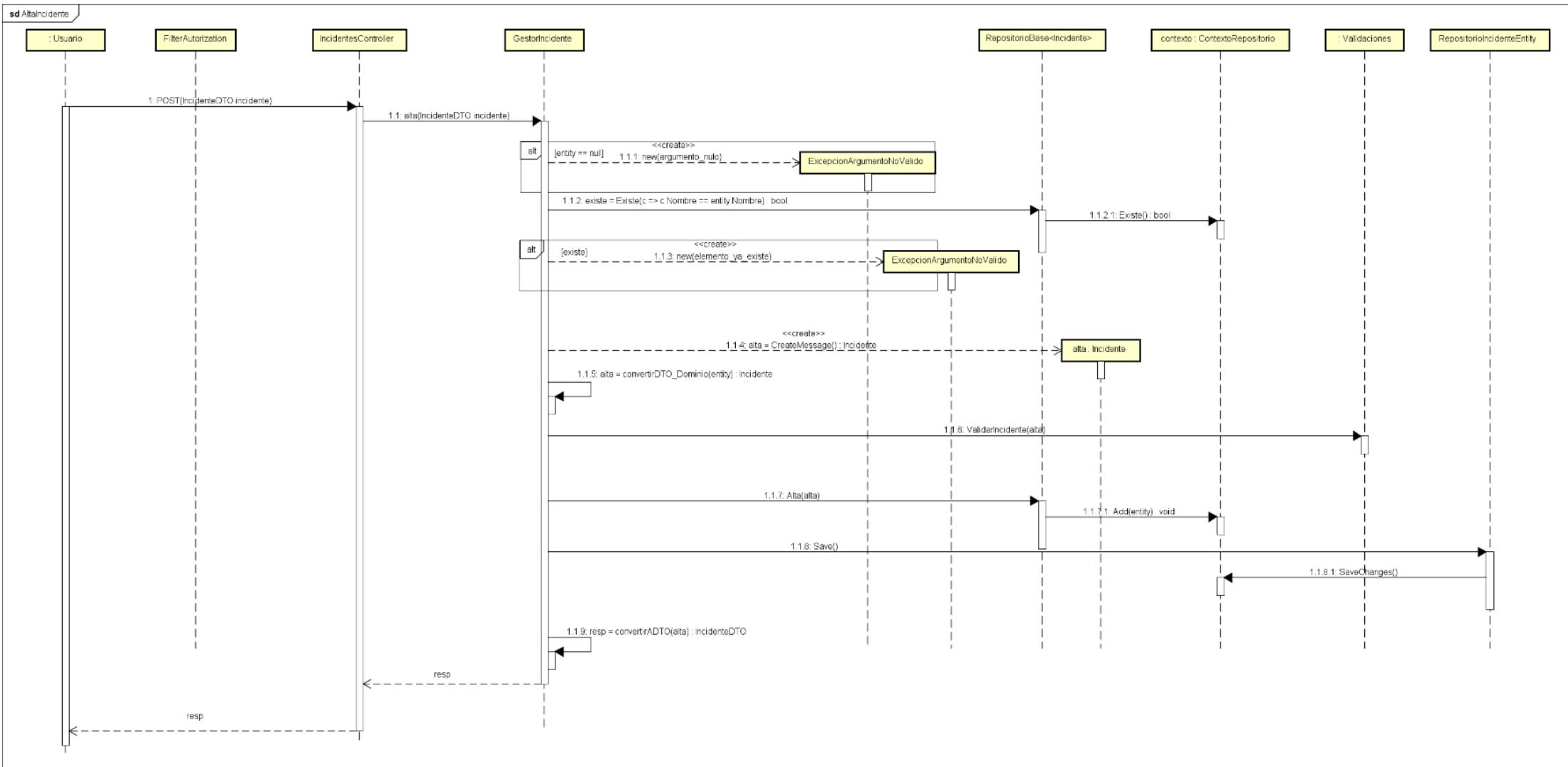
LOGIN



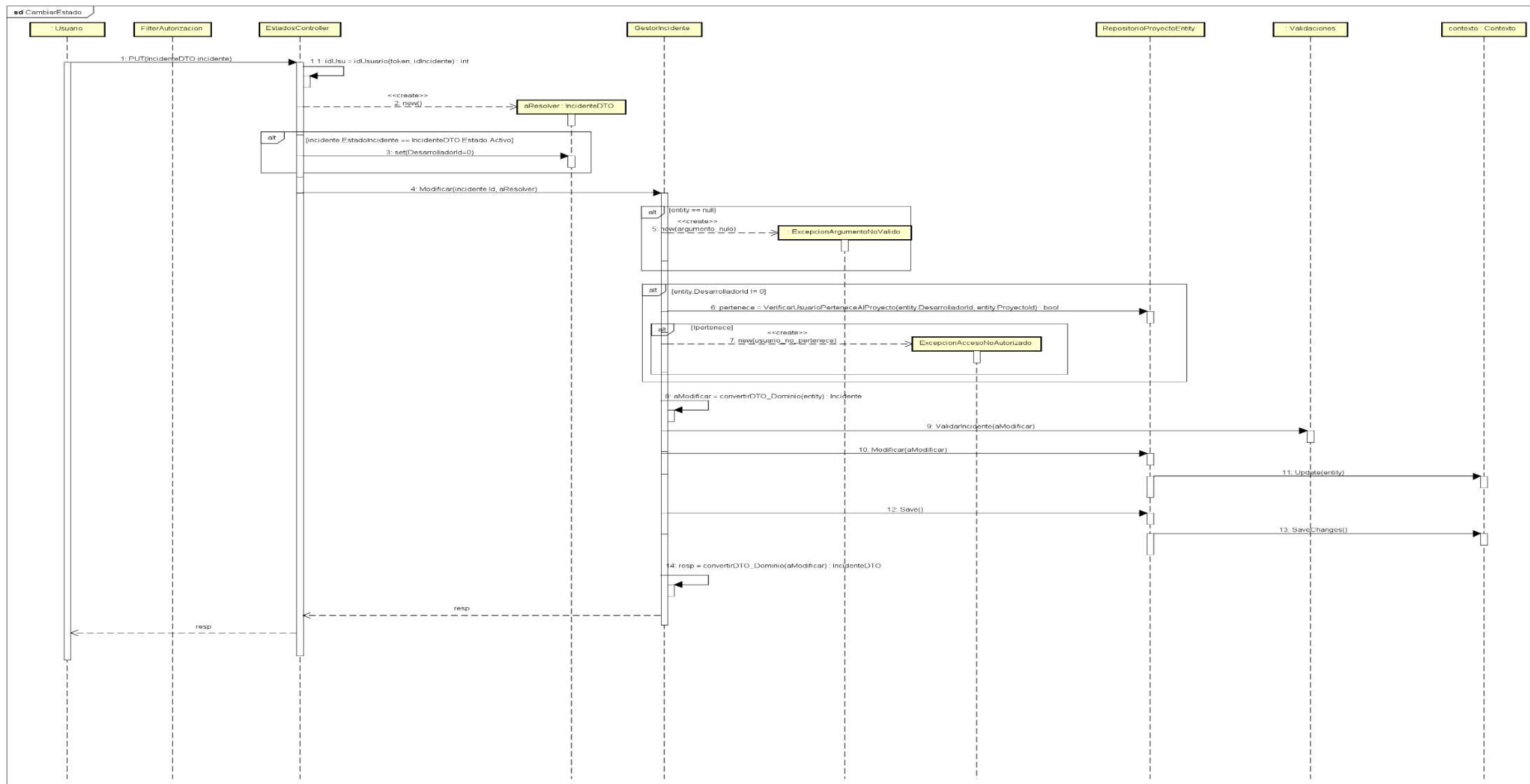
AGREGAR DESARROLLADORES Y TESTER A PROYECTO



ALTA INCIDENTE



CAMBIAR ESTADO A INCIDENTE (ACTIVO/RESUELTO)



ANEXO III Cambios en Web API

1- Se agrega el endpoint TareasController

- EndPoint: ~/api/Tareas
- Verbos HTTP: GET
- Descripción : Obtener todas las tareas.
- Parámetros { }
- Responses {200, 401, 500 }
- Headers: { **Key:** autorización./ **Value:** token }

- EndPoint: ~/api/Tareas/id
- Verbos HTTP: GET
- Descripción : Obtener la tarea con que tenga el {id}.
- Parámetros { }
- Responses {200, 401, 500 }
- Headers: { **Key:** autorización./ **Value:** token }

- EndPoint: ~/api/Tareas/id
- Verbos HTTP: DELETE
- Descripción : Como administrador eliminar la tarea {id}
- Parámetros { }
- Responses {204, 401, 500 }
- Headers: { **Key:** autorización./ **Value:** token }

- EndPoint: ~/api/Tareas/id
- Verbos HTTP: POST
- Descripción : Como administrador dar de alta una tarea.
- Parámetros { tarea}
- Responses {201, 401, 422, 500 }
- Headers: { **Key:** autorización./ **Value:** token }

- EndPoint: ~/api/Tareas/id
- Verbos HTTP: PUT
- Descripción : Como administrador modificar la tarea {id}
- Parámetros { tarea}
- Responses {204, 401, 500 }
- Headers: { **Key:** autorización./ **Value:** token }

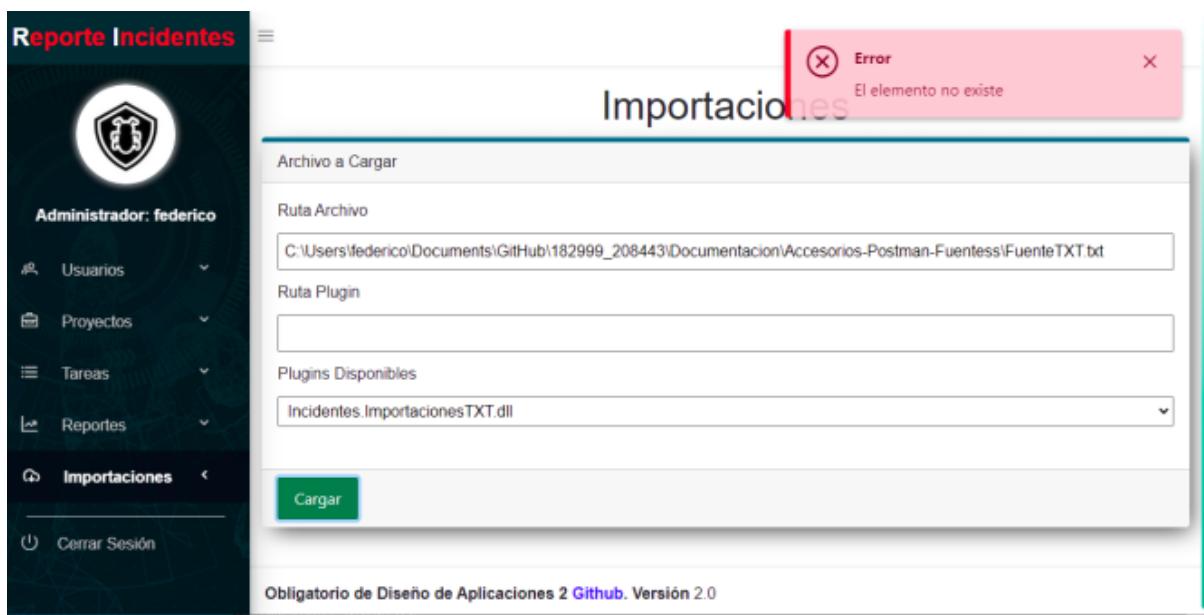
- 2- Se cambia el nombre del endpoint login por autenticaciones para no usar verbos.
- 3- Se cambia la llamada de los endpoint en reporte en base a la sugerencia de la primera devolución.
- 4- Los recursos del endpoint asociaciones controller se distribuyen en los otros endpoints ya que no se entendía bien el motivo
- 5- Se manejan objetos DTOs (ya fue explicado en la documentación).
- 6- Se modifican TODOS los verbos PUT y DELETE ya que para un item puntual en la primer entrega no enviábamos en la ruta el id del objeto, lo enviábamos en el body lo cual no cumplía con los criterios REST.

ANEXO VI Contrato con desarrolladores de terceros

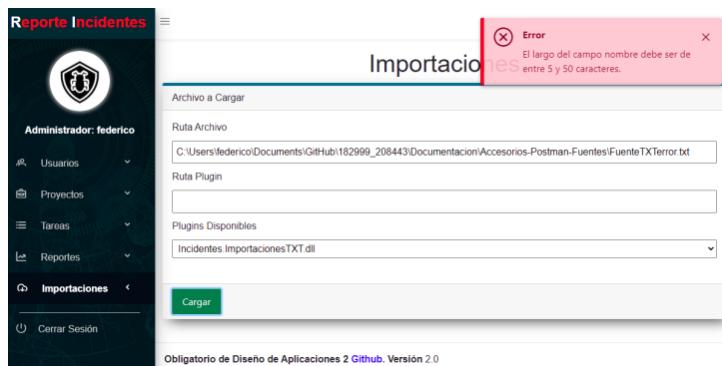
- Se debe utilizar dos tipos de objetos, los ProyectoDTO y los IncidenteDTO.
- **ProyectoDTO** tiene los siguientes atributos:
 - public string Nombre
 - public List<IncidenteDTO> Incidentes
- El **ProyectoDTO** es el proyecto en el que se encuentra trabajando su empresa, se forma por un nombre que es un string de largo entre 5 y 25 caracteres, el cual deben solicitar a la administración para que concuerde con el que están trabajando. Luego también posee una lista de IncidenteDTO.
- **IncidenteDTO** tiene los siguientes atributos:
 - public string Nombre
 - public string Descripcion
 - public string Version
- **IncidenteDTO** es el bug en sí, contiene un nombre que posee entre 5 y 50 caracteres, una descripción de este también string de entre 5 y 500 caracteres y una versión del bug que es un string de entre 5 y 25 caracteres.
- La librería debe tener un único método público que cumpla con la siguiente firma:
 - public List<ProyectoDTO> ImportarBugs(string rutaFuente)

Posibles errores:

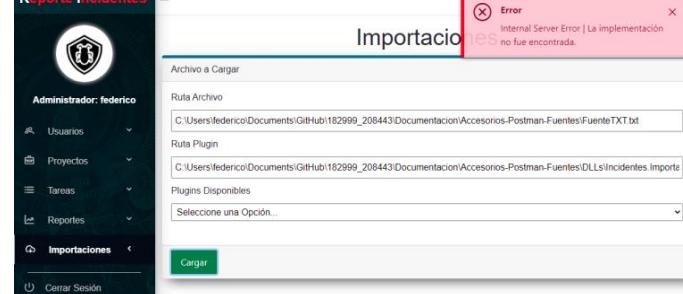
- En caso de no existir el proyecto previamente se muestra un error de **ExcepcionElementoNoExiste**.



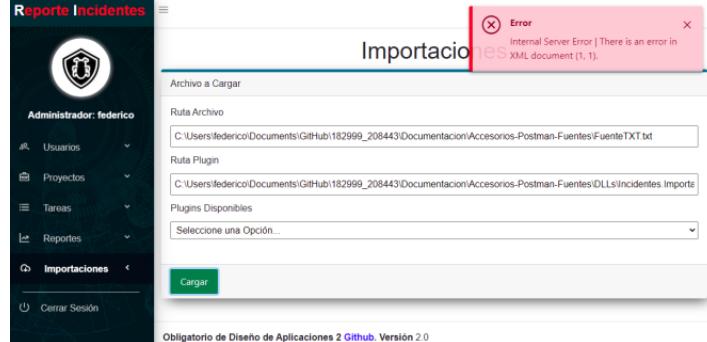
- En caso de no cumplir con los requerimientos establecidos anteriormente se mostrará un error de **ExcepcionArgumentoNoValido**.



- En caso de no encontrar la implementación se mostrará un error de **DllNotFoundException**.



- En caso de utilizar una implementación incorrecta se mostrará un error procedente de la librería utilizada.



- Para ejemplificarlo se puede mostrar la implementación de JSON y el siguiente dato para que verifiquen el formato de un tipo implementado:

```
{
  "Nombre": "Nuevo Proyecto 3",
  "Incidentes": [
    {
      "Nombre": "Error en el envío de correo 1",
      "Descripcion": "Descripción del incidente 1",
      "Version": "1.0"
    },
    {
      "Nombre": "Error en el envío de correo 2",
      "Descripcion": "El error se produce cuando el usuario no tiene un correo asignado 2",
      "Version": "1.0"
    }
  ]
}
```

ANEXO V Informe de cobertura

En esta segunda instancia se continúa con el empleo de TDD. El mismo se verifica en las siguientes imágenes:

Paquete de datos

En el paquete de datos queda sin cubrir el método update de proyecto debido a que el mismo al implementar las pruebas deja de funcionar en la aplicación.

incidentes.datos.dll	2547	86.46%	399	13.54%
{ } Incidentes.Datos	79	16.53%	399	83.47%
▷ Contexto	6	8.11%	68	91.89%
▷ RepositorioBase<T>	0	0.00%	29	100.00%
▷ RepositorioGestores	0	0.00%	24	100.00%
▷ RepositorioIncidenteE...	0	0.00%	2	100.00%
▷ RepositorioProyectoE...	72	46.15%	84	53.85%
▷ RepositorioTareaEntity	0	0.00%	2	100.00%
▷ RepositorioUsuariosE...	1	0.52%	190	99.48%

Paquete de dominio

En el paquete de dominio queda implementar los métodos de usuario que no fueron solicitados para la aplicación, el actualizar y el eliminar usuario.

incidentes.dominio.dll	2	2.94%	66	97.06%
{ } Incidentes.Dominio	2	2.94%	66	97.06%
▷ Incidente	0	0.00%	18	100.00%
▷ Proyecto	0	0.00%	18	100.00%
▷ Tarea	0	0.00%	12	100.00%
▷ Usuario	2	10.00%	18	90.00%

Paquete de DTOs

El paquete se cubre en un 100%.

incidentes.dtos.dll	0	0.00%	343	100.00%
{ } Incidentes.DTOs	0	0.00%	343	100.00%
▷ AsignacionesDTO	0	0.00%	4	100.00%
▷ FuenteDTO	0	0.00%	6	100.00%
▷ ImportacionesDTO	0	0.00%	2	100.00%
▷ IncidenteDTO	0	0.00%	75	100.00%
▷ ProyectoDTO	0	0.00%	137	100.00%
▷ ProyectoDTO,<>c_Di...	0	0.00%	3	100.00%
▷ TareaDTO	0	0.00%	37	100.00%
▷ UsuarioDTO	0	0.00%	79	100.00%

Paquete de lógica

En los paquetes de lógica quedan sin pasar por ciertos métodos que suceden cuando la aplicación llama a las librerías externas de importación.

incidentes.logica.dll	21	2.09%	985	97.91%
{} Incidentes.Logica	21	2.09%	985	97.91%
▷ GestorAutorizacion	2	4.17%	46	95.83%
▷ GestorImportacion	3	2.91%	100	97.09%
▷ GestorImportacion.<>c_DisplayClass5_0	1	16.67%	5	83.33%
▷ GestorImportacion.<>c_DisplayClass6_0	1	16.67%	5	83.33%
▷ GestorIncidente	8	3.27%	237	96.73%
▷ GestorProyecto	0	0.00%	190	100.00%
▷ GestorProyecto.<>c_DisplayClass16_0	0	0.00%	2	100.00%
▷ GestorTarea	0	0.00%	114	100.00%
▷ GestorUsuario	6	2.82%	207	97.18%
▷ GestorUsuario.<>c_DisplayClass16_0	0	0.00%	4	100.00%
▷ Validaciones	0	0.00%	75	100.00%

Paquete de WebAPI

En el paquete WebAPI se cubre el 100% del código realizado.

incidentes.webapi.dll	103	21.73%	371	78.27%
{} Incidentes.WebApi	46	100.00%	0	0.00%
{} Incidentes.WebApi.Controllers	0	0.00%	371	100.00%
▷ AsociacionesController	0	0.00%	8	100.00%
▷ AutenticacionesController	0	0.00%	25	100.00%
▷ EstadosController	0	0.00%	40	100.00%
▷ ImportacionesController	0	0.00%	24	100.00%
▷ IncidentesController	0	0.00%	137	100.00%
▷ ProyectosController	0	0.00%	54	100.00%
▷ ReportesController	0	0.00%	14	100.00%
▷ TareasController	0	0.00%	54	100.00%
▷ UsuariosController	0	0.00%	15	100.00%

Resumen

Se contó con un total de 219 pruebas unitarias.

Test Explorer		
Test	Duration	Traits
▷ ✓ Incidentes.DatosTest (43)	1.7 sec	
▷ ✓ Incidentes.Dominio.Test (26)	41 ms	
▷ ✓ Incidentes.DTOsTest (12)	111 ms	
▷ ✓ Incidentes.Logica.Test (89)	810 ms	
▷ ✓ Incidentes.WebApITest (49)	488 ms	

ANEXO VI Casos de Prueba

Administrador

- Un administrador puede ver los usuarios registrados en la aplicación

The screenshot shows the application's interface with a dark theme. On the left is a sidebar with a shield logo and navigation links: 'Reporte Incidentes', 'Administrador: federico', 'Usuarios', 'Proyectos', 'Tareas', 'Reportes', 'Importaciones', and 'Cerrar Sesión'. The main area has a breadcrumb path: 'Usuarios > Usuarios'. The title 'Administración de Usuarios' is centered above a table. The table has columns: id, Nombre, Apellido, Email, Rol, and Valor. The data is as follows:

id	Nombre	Apellido	Email	Rol	Valor
1	Federico	Alonso	federico@gmail.com	Administrador	0
2	Federico	Alonso	f.nicolas.alonso@gmail.com	Desarrollador	0
3	Federico	Alonso	f.nicolas.lionso@gmail.com	Tester	0
4	Federico	Alonso	f.nicolas.alonso@gmail.com	Desarrollador	500
5	Federico	Alonso	fnicolas.alonso@gmail.com	Desarrollador	500
6	Micaela	Olivera	email1@valido.com	Tester	526
7	Federico	Alonso	fnicolasalonso@gmail.com	Tester	600
8	Tester	Tester	tester@tester.com	Tester	300

- Un administrador puede dar de alta usuarios

The screenshot shows the application's interface with a dark theme. On the left is a sidebar with a shield logo and navigation links: 'Reporte Incidentes', 'Administrador: federico', 'Usuarios', 'Proyectos', 'Tareas', 'Reportes', 'Importaciones', and 'Cerrar Sesión'. The main area has a breadcrumb path: 'Usuarios > Alta'. The title 'Administración de Usuarios' is centered above a form titled 'Alta de Usuario'. The form fields are: Nombre (text input), Apellido (text input), Contraseña (text input), Repita la Contraseña (text input), Rol (dropdown menu set to 'Desarrollador'), Valor por Hora (text input), Email (text input), and Nombre de Usuario (text input). A blue 'Alta' button is at the bottom.

- El administrador puede dar de alta nuevos proyectos

Reporte Incidentes

Administrador: federico

Usuarios Proyectos Tareas Reportes Importaciones Cerrar Sesión

Proyectos

Nombre del Proyecto

Alta

Obligatorio de Diseño de Aplicaciones 2 [Github](#), Versión 2.0

- Puede ver todos los proyectos

Reporte Incidentes

Administrador: federico

Usuarios Proyectos Tareas Reportes Importaciones Cerrar Sesión

Proyectos

ID	Nombre	Detalles	Acciones
16	Proyecto 6	1 incidentes 0 tareas 0 asignados 0 Información	
18	Proyecto 3	1 incidentes 0 tareas 0 asignados 0 Información	
19	Proyecto 4	1 incidentes 0 tareas 0 asignados 0 Información	
22	Nuevo Proyecto 3	0 incidentes 0 tareas 0 asignados 0 Información	
25	Proyecto Xx	0 incidentes 0 tareas 0 asignados 0 Información	
26	Proyecto Nuevo 5	0 incidentes 0 tareas 0 asignados 0 Información	

Obligatorio de Diseño de Aplicaciones 2 [Github](#), Versión 2.0

- Puede eliminar un proyecto

Reporte Incidentes

Administrador: federico

Usuarios Proyectos Tareas Reportes Importaciones Cerrar Sesión

Proyectos

ID	Nombre	Detalles	Acciones
16	Proyecto 6	1 incidentes 0 tareas 0 asignados 0 Información	
18	Proyecto 3	1 incidentes 0 tareas 0 asignados 0 Información	
19	Proyecto 4	1 incidentes 0 tareas 0 asignados 0 Información	
22	Nuevo Proyecto 3	0 incidentes 0 tareas 0 asignados 0 Información	
25	Proyecto Xx	0 incidentes 0 tareas 0 asignados 0 Información	
26	Proyecto Nuevo 5	0 incidentes 0 tareas 0 asignados 0 Información	

Está seguro?
Realmente desea el Proyecto

Si No

Obligatorio de Diseño de Aplicaciones 2 [Github](#), Versión 2.0

- Se pueden ver los incidentes de un proyecto.

Reporte Incidentes

Administrador: federico

Volver

Incidentes del Proyecto

ID	Proyecto	Nombre	Descripción	Versión	Estado	Resuelto Por	Duración
1	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	Desarrollador	23
2	Nuevo Proyecto 3	Error En El Envío De Correo2 105	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Activo	Desarrollador	0
4	Nuevo Proyecto 3	Incidente 44	descripción larga	2.3	Activo	Desarrollador	0
5	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	2.0	Activo	Desarrollador	60
6	Nuevo Proyecto 3	Error En El Envío De Correo2 10	El error se produce cuando el usuario no tiene un correo asignado 2.	1.0	Resuelto	Tester	70
7	Nuevo Proyecto 3	Error En El Envío De Correo3 11	El error se produce cuando el usuario no tiene un correo asignado 2.	1.0	Resuelto	Desarrollador	0
8	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	Desarrollador	0

- Se pueden ver las tareas de un proyecto.

Reporte Incidentes

Administrador: federico

Volver

Tareas del Proyecto

ID	Nombre	Costo/Hora (\$)	Duración (hs)	Total (\$)	Acciones
12	Nueva Tar	660	15	9,900	
13	Otra Tarea	451	28	12,628	

Cantidad de Horas: 43 hs.

- Se pueden actualizar los desarrolladores y testers de un proyecto.

Reporte Incidentes

Administrador: federico

Volver

Administración de Proyectos

Usuarios Agregados

ID	Nombre	Apellido	Email	Rol	Acción
8	Tester	Tester	tester@tester.com	Tester	
9	Desarrollador	Desarrollador	desarrollador@desarr.com	Desarrollador	
11	Desarrollador	Desarrollador	desarrollador@desr.com	Desarrollador	
12	Tester	Tester	tester@test.com	Tester	

Usuarios No Agregados Aún

ID	Nombre	Apellido	Email	Rol	Acción
2	Federico	Alonso	f.nicolas.alonso@gmail.com	Desarrollador	
3	Federico	Alonso	f.nicolas.lionso@gmail.com	Tester	

- Puede ver el costo y la duración de un proyecto en particular.

ID	Nombre	Duración (hs)	Costo (\$)
16	Proyecto 6	72	36,000

- Se pueden ver todas las tareas con los datos relevantes de las mismas.

ID	Nombre	Proyecto	Costo/Hora (\$)	Duración (hs)	Total (\$)	Acciones
5	Tarea 2	Proyecto 4	900	9	8,100	
6	Tarea 25	Proyecto 3	200	5	1,000	
7	Tarea 256	Proyecto 4	233	9	2,097	
9	Tarea X	Proyecto 4	655	12	7,860	
11	Nueva Tarea	Proyecto XX	600	24	14,400	
12	Nueva Tar	Nuevo Proyecto 3	660	15	9,900	

- Se puede modificar una tarea y crear una tarea

Nombre de la Tarea:

Seleccione el proyecto:

Costo por Hora:

Duración en horas:

Obligatorio de Diseño de Aplicaciones 2 [Github](#). Versión 2.0

- Se puede eliminar una tarea

ID	Nombre	Proyecto	Costo/Hora (\$)	Duracion (hs)	Total (\$)	Acciones
5	Tarea 2	Proyecto 4	900	9	8,100	
6	Tarea 3	Proyecto 3	200	5	1,000	
7	Tarea 4	Proyecto 4	233	9	2,097	
9	Tarea 5	Proyecto 4	655	12	7,860	
11	Nueva Tarea Modificada	Proyecto XX	600	24	14,400	
12	Nueva Tar	Nuevo Proyecto 3	660	15	9,900	
13	Otra Tarea	Nuevo Proyecto 3	451	28	12,628	
14	Tarea Prueba	Proyecto 6	500	72	36,000	

- Se pueden ver los incidentes por proyecto

Proyecto	Cantidad Incidentes
Proyecto 6	0
Proyecto 3	0
Proyecto 4	0
Nuevo Proyecto 3	23
Proyecto Xx	0

- Se pueden ver los incidentes resueltos por un desarrollador en particular.

ID	Nombre	Apellido	Email	Cantidad de Incidentes Resueltos
5	Federico	Alonso	fnicolas.alonso@gmail.com	0

- Se pueden importar incidentes desde un JSON, TXT y XML

Desarrollador

- El desarrollador ve los proyectos de los cuales es integrante

id	Nombre	Detalles
22	Nuevo Proyecto 3	

- Puede ver los incidentes de un proyecto al cual pertenece

id	Proyecto	Nombre	Descripción	Versión	Estado	Resuelto Por	Duración
1	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	Desarrollador Desarrollador	23
2	Nuevo Proyecto 3	Error En El Envío De Correo2 105	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Activo		0
4	Nuevo Proyecto 3	Incidente 44	descripción larga	2.3	Activo		0
5	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	2.0	Activo		60
6	Nuevo Proyecto 3	Error En El Envío De Correo2 10	El error se produce cuando el usuario no tiene un correo asignado 2.	1.0	Resuelto	Tester Tester	70
7	Nuevo Proyecto 3	Error En El Envío De Correo3 11	El error se produce cuando el usuario no tiene un correo asignado 2.	1.0	Resuelto	Desarrollador Desarrollador	0

- Puede ver las tareas de un proyecto al cual pertenece.

Id	Nombre	Costo/Hora (\$)	Duracion (hs)	Total (\$)
12	Nueva Tar	660	15	9,900
13	Otra Tarea	451	28	12,628
18	Nueva Tarea	500.2	21	10,504.2
19	Otra Tarea	25.4	22	558.8

Cantidad de Horas: 86 hs.

- Puede ver de forma ordenada los incidentes de los proyectos a los que pertenece, así como filtrar por id, nombre de proyecto, nombre de incidente y estado del incidente

Id	Proyecto	Nombre	Descripción	Versión	Estado	Acciones
0	Nue	Err			Indiferente	
1	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	
2	Nuevo Proyecto 3	Error En El Envío De Correo2 105	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Activo	
5	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	2.0	Activo	
6	Nuevo Proyecto 3	Error En El Envío De Correo2 10	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	

- Puede resolver el mismo e indicar solo las horas que le llevó solucionarlo

Detalles del Incidente

Detalles del Incidente

Nombre del Incidente
Error en el envío de correo 9

Descripción
El error se produce cuando el usuario no tiene un correo asignado.

Versión
1.0

Estado del Incidente

 Resuelto - Resuelto por: desarrollador desarrollador

Duración
23

- Puede ver todas las tareas de los proyectos a los que pertenece.

ID	Nombre	Proyecto	Costo/Hora (\$)	Duración (hs)
12	Nueva Tar	Nuevo Proyecto 3	660	15
13	Otra Tarea	Nuevo Proyecto 3	451	28
18	Nueva Tarea	Nuevo Proyecto 3	500.2	21
19	Otra Tarea	Nuevo Proyecto 3	25.4	22

- Tester

- Puede ver los incidentes a los que pertenece al igual que el desarrollador.
- Puede ver los incidentes de un proyecto en particular.
- Puede ver las tareas de un proyecto en particular.
- Puede ver los incidentes de los proyectos a los que pertenece y filtrarlos al igual que un desarrollador.
- Puede ver las tareas de los proyectos a los que pertenece al igual que el desarrollador.
- Puede crear nuevos incidentes en los proyectos que pertenece

- Puede modificar los incidentes, opción que el desarrollador no podía

ID	Proyecto	Nombre	Descripción	Versión	Estado	Acciones
0					Indiferente	
1	Nuevo Proyecto 3	Error En El Envío De Correo 9	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Resuelto	
2	Nuevo Proyecto 3	Error En El Envío De Correo 105	El error se produce cuando el usuario no tiene un correo asignado.	1.0	Activo	
4	Nuevo Proyecto 3	Incidente 44	descripción larga	2.3	Activo	