

**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](#_Toc88085945)

[1.1 Descripción general del trabajo 3](#_Toc88085946)

[1.2 Errores conocidos 4](#_Toc88085947)

[1.3 Diagrama general de paquetes 5](#_Toc88085948)

[1.3.1 Dominio 6](#_Toc88085949)

[1.3.2 Lógica 7](#_Toc88085950)

[1.3.3 LogicaFabrica 8](#_Toc88085951)

[1.3.4 LogicaInterfaz 8](#_Toc88085952)

[1.3.5 Datos 9](#_Toc88085953)

[1.3.6 DatosInterfaz 10](#_Toc88085954)

[1.3.7 Datos Fabrica y 11](#_Toc88085955)

[1.3.8 Excepciones 11](#_Toc88085956)

[1.3.9 Dtos 12](#_Toc88085957)

[1.4 Modelo de tablas de la estructura de la base de datos. 13](#_Toc88085958)

[1.5 Justificación del diseño 14](#_Toc88085959)

[1.5.1 Mecanismos de inyección de dependencias, fábricas, patrones y principios de diseño. 14](#_Toc88085960)

[1.5.2 Descripción del mecanismo de acceso a datos utilizado 16](#_Toc88085961)

[1.5.3 Descripción del manejo de excepciones 17](#_Toc88085962)

[1.5.4 Decisiones de diseño propias 17](#_Toc88085963)

[ANEXO I 18](#_Toc88085964)

[ANEXO II Diagrama de interacción relevantes 18](#_Toc88085965)

# Descripción del diseño

## 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, brindado 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.

## Errores conocidos

## Diagrama general de paquetes

Diagrama

Descripción generada automáticamente

### Dominio

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

Diagrama

Descripción generada automáticamente

**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.

### Lógica

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

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 ….

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.

Diagrama

Descripción generada automáticamente

**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 reúso (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.

### 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.

Escala de tiempo

Descripción generada automáticamente con confianza media

### LogicaInterfaz

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 que tipo sea la entidad.

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

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

**Cambios respecto a la primera entrega:**

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

### 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:

Diagrama

Descripción generada automáticamente

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 ahorramos repetir el mismo código en cada repositorio.

Cada repositorio desarrollará únicamente las funciones particulares que requiera implementar fuera de las antes 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

### DatosInterfaz

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

Imagen que contiene Diagrama

Descripción generada automáticamente

También contamos con aplicar Generic<T> para la interfaz **IRepositorioBase<T>** y luego cada interfaz particular además de sus firmas particulares, debe implementar IRepositorioBase<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 **IRepositorioTarea**
* La interfaz **IRepositorioGestores** agrega el atributo **RepositorioTarea**

### Datos Fabrica y

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.

Texto

Descripción generada automáticamente con confianza baja

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.

### Excepciones

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

* **ExcepcionAccesoNoAutorizado**: 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.
* **ExcepcionArgumentoNoValido**: es lanzada cuando se ingresa en un campo de texto algún carácter no permitido, o el formato no coincida con lo solicitado como por ejemplo al querer ingresar un email que no cumpla con las condiciones de un mail valido.
* **ExcepcionElementoNoExiste**: 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.
* **ExcepcionLargoTexto**: se produce cuando se pretende ingresar un dato con la cantidad de caracteres menor o mayor al establecido por las reglas del negocio.
* **ExcepcionNumeroNoValido**: 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 reúso común (CRP) que apunta al reúso.

### 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.

Diagrama

Descripción generada automáticamente

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

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

Diagrama, Tabla

Descripción generada automáticamente

**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**

## Justificación del diseño

### Mecanismos de inyección de dependencias, fábricas, patrones y principios de diseño.

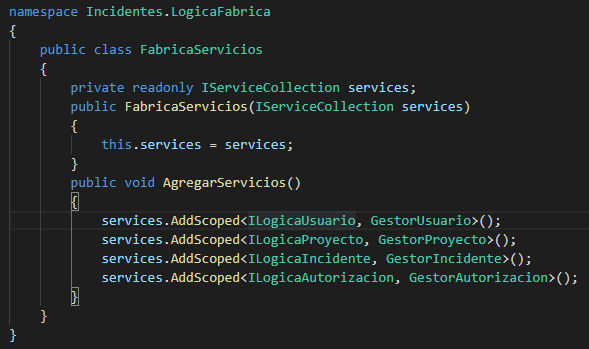
En los puntos 1.3.3 y 1.3.5 se mencionaron las inyecciones de dependencia que fueron utilizadas entre los paquetes WebApi, Lógica y la capa de Datos. Además, se crean las fabricas correspondientes para que haya total independencia entre estos módulos principales.

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.

Inyección de dependencia y fabricas para desacoplar de lógica y WebAPI

Diagrama

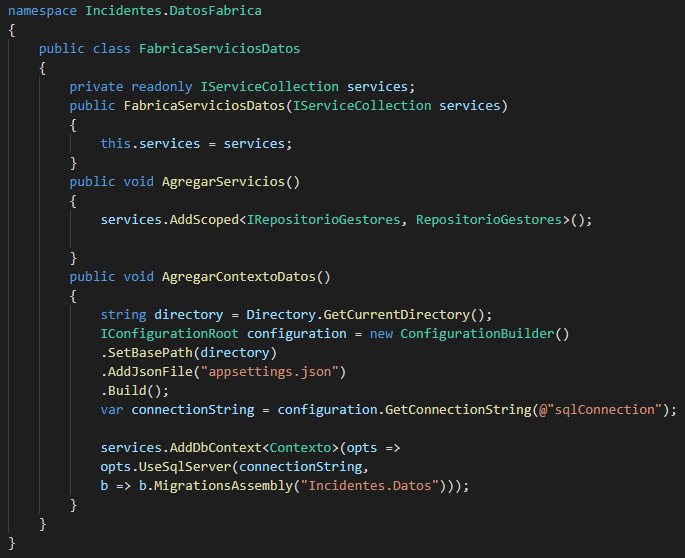
Descripción generada automáticamente



Inyección de dependencia y fabricas para desacoplar de datos y WebAPI

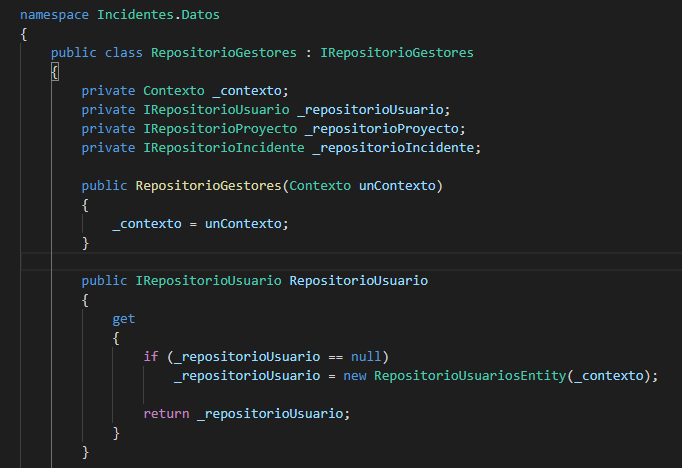
Diagrama

Descripción generada automáticamente



Se utilizó SINGLETON para la creación de los repositorios:

* RepositorioUsuario
* RepositorioProyecto
* RepositorioIncidentes



### 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.

### Descripción del manejo de excepciones

Para el manejo de errores creamos diferentes clases de excepciones. 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.

### Decisiones de diseño propias

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

1. 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.
2. El usuario administrador se cargó directamente en la base de datos ya que necesitamos un usuario previamente registrado para poder ejecutar las funcionalidades.
3. Para actualizar los desarrolladores y/o testers correspondientes a un proyecto, se debe enviar todo el listado en una sola acción, es decir por ejemplo que, si tenemos 5 desarrolladores asociados al proyecto y deseamos agregar 1 desarrollador más, debemos incluir los que ya se encontraban antes asociados. Se piensa que a futuro cuando se tenga un front end, en la funcionalidad de update, los que ya se encuentren asociados al proyecto van a estar en la vista del usuario ya seleccionados.
4. Los incidentes pueden tener dos estados: activos o resueltos.
5. Se creó una clase **Validaciones** con métodos estáticos, que sirve para validar atributos de diferentes clases con las siguientes validaciones:

* Validar largos de textos.
* Validar passwords.
* Validar emails.

Si bien se tiene en cuenta que no es la mejor opción tener una clase estática, se provee eliminar en la futura entrega.

1. Cuando un desarrollador cambia el estado de un incidente a resuelto, automáticamente se la asigna a dicho desarrollador el atributo Desarrollador Id**.**
2. Para utilizar la funcionalidad de importar incidentes como no se aclaró que rol tiene permitido usar este recurso se decide que puede ser público para el caso que lo quiera importar una empresa.

## Gráfico, Diagrama Descripción generada automáticamenteANEXO I DIAGRAMA DE COMPONENTES

## ANEXO II Diagrama de interacción relevantes

**LOGIN**

Diagrama

Descripción generada automáticamente

**AGREGAR DESARROLLADORES Y TESTER A PROYECTO**

Interfaz de usuario gráfica

Descripción generada automáticamente

**Diagrama

Descripción generada automáticamenteALTA INIDENTE**

**Diagrama

Descripción generada automáticamenteCAMBIAR ESTADO A INCIDENTE (ACTIVO/RESUELTO)**