

**Diseño de Aplicaciones 2**  
Primer 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](#_Toc84457094)

[1.1 Descripción general del trabajo 3](#_Toc84457095)

[1.2 Errores conocidos 3](#_Toc84457096)

[1.3 Diagrama general de paquetes 4](#_Toc84457097)

[1.3.1 Dominio 5](#_Toc84457098)

[1.3.2 Lógica 5](#_Toc84457099)

[1.3.3 LogicaFabrica y LogicaInterfaz 6](#_Toc84457100)

[1.3.4 Datos 6](#_Toc84457101)

[1.3.5 DatosFabrica y DatosInterfaz 6](#_Toc84457102)

[1.4 Modelo de tablas de la estructura de la base de datos. 7](#_Toc84457103)

[1.5 Diagrama de interacción relevantes 8](#_Toc84457104)

[1.6 Justificación del diseño 9](#_Toc84457105)

[1.6.1 Mecanismos de inyección de dependencias, fábricas, patrones y principios de diseño 9](#_Toc84457106)

[1.6.2 Descripción del mecanismo de acceso a datos utilizado 9](#_Toc84457107)

[1.6.3 Descripción del manejo de excepciones 9](#_Toc84457108)

[1.6.4 Decisiones de diseño propias 9](#_Toc84457109)

[1.7 Diagrama de componentes 10](#_Toc84457110)

# Descripción del diseño

## Descripción general del trabajo

Para la solución del obligatorio se implementa una WEB API como backend, la cual define las funcionalidades necesarias para administrar los incidentes en los proyectos de software cumpliendo con todos los requerimientos exigidos en la letra del obligatorio.

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

Los administradores del sistema pueden realizar altas de **usuarios** registrando su nombre, apellido, nombre de usuario, contraseña, dirección de correo electrónico y su rol. También pueden crear **proyectos**  los cuales tienen nombre del proyecto , testers y desarrolladores asignados e incidentes.

El sistema cuenta con un módulo que permite importar incidentes en formato XML y en archivo de texto.

Los administradores tienen acceso a los reportes de cantidad de bugs por proyecto y cantidad de bugs resueltos por un desarrollador.

Los usuarios con el rol tester, pueden ver todos los incidentes de todos los proyectos a los cuales pertenece y también puede crear, modificar y eliminar incidentes.

Los usuarios con rol desarrollador pueden visualizar los bugs de los proyectos a los cuales pertenece y modificar el estado de este (activo/resuelto).

## Errores conocidos

* Por falta de tiempo para esta versión del producto no se utilizó la clase mapper y se hicieron los mapeos manualmente.
* //////
* /////
* /////
* /////
* ///
* //
* //
* /
* /
* /
* /
* /
* /
* /
* /

## Diagrama general de paquetes

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

En el presente diagrama podemos observar el bajo acoplamiento que existe entre las capas principales del backend:

* WebApi
* Logica
* Datos
* Dominio

Para ellos se utilizaron paquetes de interfaces y fabricas que en más adelante vamos a detallar.

### Dominio

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

Diagrama

Descripción generada automáticamente

### Lógica

En este paquete se encuentran los gestores de cada repositorio, la implementación de las interfaces de la capa ILogicaInterfaz y las excepciones. También se encuentra en la carpeta fuentes, las clases necesarias para poder importar Incidentes. Se trata de implemetnaciones de la interfaz IFuente, donde las clases FuenteXML y FuenteTXT desarrollan la lógica para ejecutar las importaciones tanto de archivos TXT o XML.

Diagrama

Descripción generada automáticamente

### LogicaFabrica y 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.

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

### 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

### DatosFabrica y DatosInterfaz

El paquete DatosInterfaz provee una serie de interfaces de repositorios, las cuales serán implementadas en el paquete de Datos. De esta manera y junto con la inyección de dependencias logramos desacoplar la capa de acceso a datos de la capa de la lógica.

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

Estos paquetes tienen como responsabilidad 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.

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

La base de datos representa los objetos del sistema y la relación N a N entre usuarios y proyectos:

Diagrama

Descripción generada automáticamente con confianza media

## Diagrama de interacción relevantes

**LOGIN**

Diagrama

Descripción generada automáticamente

**AGREGAR DESARROLLADORES Y TESTER A PROYECTO**

Interfaz de usuario gráfica

Descripción generada automáticamente

## Justificación del diseño

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

////////

### 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 distintos específicos a cada clase, sino que según el rol podían acceder a distintas funcionalidades 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. **//NO SERIA NECESARIO ELIMINAR // BUGS ASOCIADOS?**
4. Los incidentes pueden tener dos estados: activos o resueltos.
5. Se creó una clase **Validaciones**, que sirve para validar atributos de diferentes clases con las siguientes validaciones:
6. Validar largos de textos
7. Validar passwords
8. Validar emails
9. No se puede agregar a un proyecto un tester o desarrollador que previamente no haya sido dado de alta en la BD.
10. Cuando un desarrollador cambia el estado de un incidente a resuelto, automáticamente se la asigna a dicho desarrollador el atributo Desarrollador Id**.**
11. Para utilizar la funcionalidad de exportar incidentes como no se aclaró que rol tiene permitido usar este recurso, se dejo sin verificar el control por rol.
12. Un usuario puede tener solo un rol.

## Diagrama de componentes

Diagrama

Descripción generada automáticamente