

**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 2](#_Toc83924348)

[1.1 Suposiciones y decisiones de diseño 2](#_Toc83924349)

[1.2 Descripción general del diseño 4](#_Toc83924350)

[1.3 Diagrama general de paquetes 4](#_Toc83924351)

[1.4 Diagrama de componentes 4](#_Toc83924352)

[1.5 Asignación de responsabilidades 4](#_Toc83924353)

[1.6 Mecanismos de inyección de dependencias, fabricas, patrones y principios de diseño 4](#_Toc83924354)

[1.7 Descripción del mecanismos de acceso a datos utilizado 4](#_Toc83924355)

[1.8 Descripción del manejo de excepciones 4](#_Toc83924356)

[1.9 Diagrama de secuencia de … 4](#_Toc83924357)

# Descripción del diseño

## Suposiciones y decisiones de diseño

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.

## Descripción general del diseño

Para la solución del obigatorio 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** as los cuales tienen nombre del proyecto , testers y desarrolladores asignados e incidentes.

El sistema cuenta con un modulo 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 pertecene y modificar el estado del mismo (activo/resuelto).

## Diagrama general de paquetes

ESTA EN EL GIT

## Diagrama de componentes

ESTA EN EL GIT

## Asignación de responsabilidades

### Logica

Dentro de este paquete tenemos 3 carpetas para agrupar las clases :

* DTOs
* Excepciones
* Gestores

//FOTO DIAGRAMA DE CLASES LOGICA

### LogicaFabrica y LogicaInterfaz

Estos paquetes tienen 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 modulo 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.

//FOTO DE DIAGRAMA DE CLASES DATOS

### DatosFabrica y DatosInterfaz

Estos paquetes tienen como responsabilidad hacer la inyección de dependencia entre la web API y la capa de acceso a datos. De esta manera logramos que el modulo 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. // VER

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

## Descripción de los mecanismos de acceso a datos utilizado

## Descripción del manejo de excepciones

Excepciones en LOGICA

Excepciones en DATOS

Excepciones en WEBAPI

## Diagrama de secuencia de LOGIN