

**Diseño de Aplicaciones 2**  
Primer Obligatorio



Cristian Palma 208443 Federico Alonso 182999

**Evidencia de Clean Code y de la aplicación de TDD**

**Grupo N6A**

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

**Índice**

[1. Aplicación de TDD 3](#_Toc84521193)

[1.1 Descripción de la estrategia de TDD seguida 3](#_Toc84521194)

[1.2 Informe de cobertura para todas las pruebas desarrolladas 4](#_Toc84521195)

[1.2.1 Cobertura Paquete de Datos 4](#_Toc84521196)

[1.2.2 Cobertura Paquete Dominio 4](#_Toc84521197)

[1.2.3 Cobertura Paquete Lógica 5](#_Toc84521198)

[1.2.4 Controladores de la WebAPI 5](#_Toc84521199)

[2. Evidencia de Clean Code 6](#_Toc84521200)

[2.1 Nombres según Clean Code 6](#_Toc84521201)

[2.1.1 Variables y Properties 6](#_Toc84521202)

[2.1.2 Métodos 6](#_Toc84521203)

[2.1.3 Clases 6](#_Toc84521204)

[2.2 Formato según Clean Code 7](#_Toc84521205)

[2.3 Manejo de errores 7](#_Toc84521206)

# Aplicación de TDD

## Descripción de la estrategia de TDD seguida

Para el desarrollo del obligatorio se hizo utilizando TDD (Test-Driven Development) siguiendo los siguientes pasos:

1. Escribir las pruebas.
2. Hacer el código mínimo para que la prueba sea fallida.
3. Hacer la implementación mínima para que las pruebas pasen.
4. Refactor para mejorar el código de las etapas anteriores.

Las pruebas antes mencionadas cumplen con el principio FIRST:

* FAST. Las pruebas deben ser rápidas de ejecutarse (aislarlas de factores que las puedan enlentecer como llamadas de red)
* INDEPENDENT. Las pruebas no deben depender una de la otra. Se debe poder correr las pruebas en cualquier orden y estas deberían pasar.
* REPEATABLE. Se deben poder repetir en cualquier ambiente.
* SELF-VALIDATING. Deben tener output booleano, las propias pruebas nos deben decir si pasan o no.
* TIMELY. Las pruebas se deben escribir antes que el código que las haga pasar.

Cabe destacar que esta práctica la podemos ver reflejada en los commits realizados en GitHub, como se muestra el siguiente ejemplo a continuación:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**(\*)** Cuando se comienza con una funcionalidad, esta se va desarrollando con sus respectivas pruebas.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

.

**(\*)** Además de las etapas de desarrollo (que llamamos funcionalidad), pasamos luego por una etapa de refactor.

## Informe de cobertura para todas las pruebas desarrolladas

### 1.2.1 Cobertura Paquete de Datos

% Cubiertos

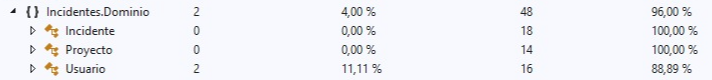
Tabla

Descripción generada automáticamente con confianza media

(\*) El método funciona utilizando la base de datos real, pero cuando aplicamos en las pruebas Inmemory falla. La descripción se encuentra en errores conocidos.

### 1.2.2 Cobertura Paquete Dominio

% Cubiertos



(\*) El 11% no cubierto corresponde a la lista virtual de proyectos que se usa para generar la relación N a N en EF.

### 1.2.3 Cobertura Paquete Lógica

Imagen que contiene Tabla

Descripción generada automáticamente % Cubiertos

### 1.2.4 Controladores de la WebAPI

Imagen que contiene Gráfico

Descripción generada automáticamente % Cubiertos

# Evidencia de Clean Code

## Nombres según Clean Code

* Se utilizan nombres que revelen intención y nombres pronunciables.
* Elegir sustantivos y no verbos para los nombres de las Clases.
* Para los métodos elegir verbos que indiquen lo que el método hace

### Variables y Properties

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

### Métodos

Texto

Descripción generada automáticamente

### Clases

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteCaptura de pantalla de un celular

Descripción generada automáticamente

* 1. Formato según Clean Code

Para el formato se tuvo en cuenta las siguientes recomendaciones:

* Leer de arriba hacia abajo tanto en importancia como en la lógica de las llamadas.
* Funciones que no tengan ningún argumento, pero en el caso de haber lo “normal” sería tolerar hasta 2.
* Evitar comentarios.
* La Metáfora del diario
* Dejar que el código respire
* Indentar consistentemente y agrupar correctamente

Texto

Descripción generada automáticamente

## Manejo de errores

Como se puede apreciar en la imagen anterior para el manejo de errores se favorecer el uso de excepciones en lugar de retornar códigos de error, como enteros o enumerados.