

# **Universidad ORT Uruguay**

## **Facultad de Ingeniería**

**Bernard Wand Polak**

**Licenciatura en Sistemas - 2023**

## **Diseño de aplicaciones 2**

**Obligatorio I (Evidencia de la aplicación de TDD y Clean Code)**

**Agustín Fioritto  
Joaquín Calvo  
Chiara Sosa**

**Nro est. 238927  
Nro est. 203832  
Nro est. 241699**

# ÍNDICE

<b>Aplicación de TDD.....</b>	<b>3</b>
<b>Principios de Clean Code.....</b>	<b>5</b>
<b>Informe de cobertura.....</b>	<b>8</b>

## Aplicación de TDD

Para la implementación de TDD, se realizó como primera instancia, la creación de las pruebas las cuales dan error por no tener implementaciones en la capa lógica (lo que vamos a testear).

```
[TestMethod]
0 referencias
public void GetProducts()
{
    var products = new List<Product>();

    Product aux1 = new Product(1, "teclado", 4000, "mecanico", 14, 14, "negro");
    Product aux2 = new Product(2, "mouse", 2000, "mecanico", 14, 14, "negro");
    products.Add(aux1);
    products.Add(aux2);

    mock.Setup(m => m.GetProducts()).Returns(products);

    var result = _controller.GetProducts();

    Assert.IsInstanceOfType(result, typeof(OkObjectResult));
    var ok = (OkObjectResult)result;
    var res = ok.Value as List<Product>;
    Assert.IsNotNull(res);
    Assert.AreEqual(products.Count, res.Count);
}
```

El siguiente paso es la implementación mínima del código que deseamos testear , para que el test logre pasar.

```
public IActionResult GetProducts()
{
    try
    {
        var products = _productService.GetProducts();
        return Ok(products);
    }
    catch (ProductManagementException ex)
    {
        Log.Error(ex, $"Error al obtener productos: {ex.Message}");
        return BadRequest($"Error al obtener productos: {ex.Message}");
    }
    catch (Exception e)
    {
        Log.Error(e, "Error inesperado al obtener los productos: {ErrorMessage}", e.Message);
        return BadRequest($"Error inesperado al obtener los productos: {e.Message}");
    }
}
```

Luego volvemos a probar el test para observar el comportamiento del código que generamos recientemente. Si todo está correcto el test no debería tener errores de sintaxis y este mismo debería pasar correctamente.

```
[TestMethod]
0 | 0 referencias
public void GetProducts()
{
    var products = new List<Product>();

    Product aux1 = new Product(1, "teclado", 4000, "mecanico", 14, 14, "negro");
    Product aux2 = new Product(2, "mouse", 2000, "mecanico", 14, 14, "negro");
    products.Add(aux1);
    products.Add(aux2);

    mock.Setup(m => m.GetProducts()).Returns(products);

    var result = _controller.GetProducts();

    Assert.IsInstanceOfType(result, typeof(OkObjectResult));
    var ok = (OkObjectResult)result;
    var res = ok.Value as List<Product>;
    Assert.IsNotNull(res);
    Assert.AreEqual(products.Count, res.Count);
}
```

✓ GetProductIDNotFound	< 1 ms	< Entorno local de Windows >
✓ GetProducts	1 ms	< Entorno local de Windows >

Una forma adicional de mostrar el TDD realizado por el equipo, es en los commits con nombres descriptivos de las etapas del desarrollo de pruebas.

(Implementación GetProducts)

Implementación mínima  
joaquincalvo • 1 minute ago

implementacionMinima getPro...  
joaquincalvo • 2 hours ago

CreacionProductControllerTest  
joaquincalvo • 3 hours ago

(Implementación CreateUser y UpdateUserInformation)

Implementacion de UpdateUserInform...

 fiioro25 • Sep 13, 2023

---

UpdateUserInformationTest

 fiioro25 • Sep 13, 2023

---

Implementacion CreateUser

 fiioro25 • Sep 13, 2023

---

CreateUserTest

 fiioro25 • Sep 13, 2023

---

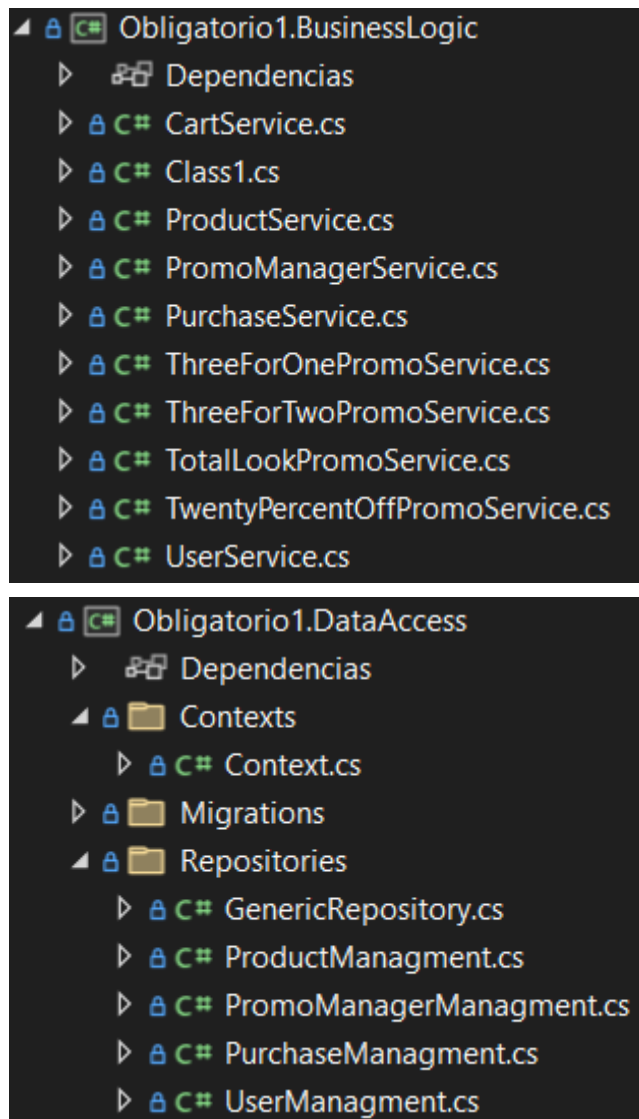
## Principios de Clean Code

Uno de los principios de Clean Code que procuramos seguir, es el de mantenibilidad de nuestro código. De esta manera logramos que los cambios que realicemos y las actualizaciones, afecten lo menos posible a nuestro código, buscando que al momento de por ejemplo realizar un merge en Github, no surjan conflictos.

Otro principio el cual aplicamos a nuestro código, es el de coherencia. Aquí, buscamos que todas las definiciones, términos (ya sea en variables, métodos y paquetes) y formatos sean uniformes y coherentes.

Para ello se siguió una nomenclatura similar para BusinessLogic y DataAccess.

En las capturas siguientes observamos el nombramiento de las clases según los objetos del dominio tanto para BusinessLogic como DataAcces, con la diferencia en su terminación, indicando que se implementan distintas cosas.



Luego para métodos que cumpliesen funciones similares, se optó por ponerles un nombre similar pero variando según la clase del dominio que utilicen.

En las capturas siguientes se puede observar dos métodos, uno para obtener productos mediante un ID y el otro para usuarios. Si bien el código tiene diferencias, las funciones tienen un rol similar, por lo que optamos por una nomenclatura similar.

```

public User GetUserByID(int userID)
{
    User? user = _userManager.GetUserByID(userID);

    if (user == null)
    {
        throw new UserException($"Usuario con ID {userID} no encontrado.");
    }

    return user;
}

```

```

public Product GetProductByID(int prodID)
{
    Product? prod = productsManagement.GetProductByID(prodID);

    if (prod == null)
    {
        throw new ProductManagementException("Producto no encontrado");
    }

    return prod;
}

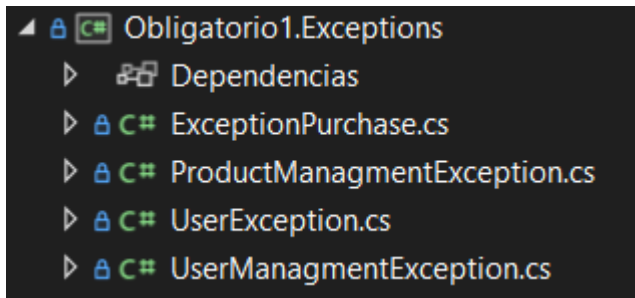
```

Siguiendo con Clean Code, se buscó una baja complejidad en el código, creando los métodos utilizados de manera simple y sin muchas responsabilidades (se busca evitar anidamiento y excesivas estructuras de control complejas). Adicionalmente, para evitar la complejidad, fue que se implementaron las *IBusinessLogic* y *IDataAccess*, las cuales contienen interfaces a sus respectivas clases, buscando generar así un menor acoplamiento.

En base a los comentarios, nuestro código solo los presenta en métodos los cuales ayuda a la persona que esté revisando el código a entenderlo, pero son escasos ya que los nombres empleados en variables, métodos y clases son lo bastante descriptivos como para indicar su funcionamiento.

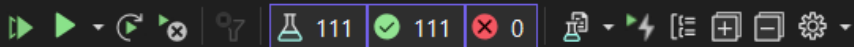
Por último se encuentran los test y las excepciones. Para cumplir con Clean Code, se debe mantener un flujo de test fáciles de probar y además debe implementar las excepciones para toda parte de código aplicable, las cuales ambas tareas fueron implementadas en este proyecto.

Fueron creadas excepciones para manejar cada entidad del proyecto, tales como usuario, producto y compras entre otras.



## Informe de cobertura

A continuación se muestra el informe de cobertura de los tests de nuestra solución, logrando una cobertura general de un 85%. La razón por la cual este valor no llega a un 90% se debe a que ciertos tests relacionados al carrito y cálculo de las promociones quedaron sin ejecutar por problemas de conexión con la base de datos. Para la próxima entrega, éste será uno de los primeros bugs a solucionar para lograr continuar con el desarrollo de la aplicación y obtener un porcentaje de cobertura que supere el 90%.

Explorador de pruebas			
			
Serie de pruebas finalizada: 111 pruebas (Superadas: 111; Con errores: 0; Omitidas: 0) ejecutadas en 256 ms			
Prueba	Dur...	Rasgos	Mensaje de error
▲  Obligatorio1.BusinessLogic.Test ...	123...		
▲  Obligatorio1.BusinessLogic.Test	123...		
▸  CartServiceTest (8)	90 ms		
▸  ProductServiceTest (6)	16 ms		
▸  ThreeForOnePromoTest (14)	2 ms		
▸  ThreeForTwoPromoTest (14)	< 1 ms		
▸  TotalLookPromoTest (13)	< 1 ms		
▸  TwentyPercentOffPromoTest ..	< 1 ms		
▸  UserServiceTest (14)	15 ms		
▲  Obligatorio1.DataAccess.Test (1)	2 ms		
▲  Obligatorio1.DataAccess.Test (...)	2 ms		
▲  UnitTest1 (1)	2 ms		
TestMethod1	2 ms		
▲  Obligatorio1.WebApi.Test (34)	109...		
▲  Obligatorio1.WebApi.Test (34)	109...		
▸  CartControllerTest (2)	57 ms		
▸  ProductControllerTest (7)	33 ms		
▸  UserControllerTest (25)	19 ms		



Resultados de la cobertura de código				
Usuario_DESKTOP-DKSHPT5_2023-10-05.18				
Buscar				
Hierarchy	Covered (%Blocks)	Not Covered (%Blocks)	Covered (%Lines)	Not Covered (%Lines)
▸ Usuario_DESKTOP-DKSHPT5_2023-10-05.18_45.coverage	84,68%	15,32%	75,80%	21,26%
▸ Usuario_DESKTOP-DKSHPT5_2023-10-05.18_45.coverage	84,68%	15,32%	75,80%	21,26%
▸ obligatorio1.exceptions.dll	20,00%	80,00%	16,67%	83,33%
▸ obligatorio1.webapi.test.dll	100,00%	0,00%	100,00%	0,00%
▸ obligatorio1.businesslogic.test.dll	97,03%	2,97%	93,33%	0,11%
▸ obligatorio1.domain.dll	72,28%	27,72%	71,70%	28,30%
▸ obligatorio1.webapi.dll	31,37%	68,63%	29,09%	70,15%
▸ obligatorio1.dataaccess.test.dll	100,00%	0,00%	100,00%	0,00%
▸ obligatorio1.businesslogic.dll	78,85%	21,15%	74,90%	23,52%