

Diseño de aplicaciones 2

OBLIGATORIO 1



Facundo Pujol 226033
Serafín Revetria 209143

Facultad ORT Uruguay

GitHub: <https://github.com/ORT-DA2/DA2-Pujol-Revetria.git>

Indice

[Descripción de la estrategia de TDD seguida](#)

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

[Pruebas unitarias](#)

Evidencia de Clean Code y de la aplicación de TDD

Comprendemos que seguimos los parámetros establecidos de Clean Code. Y esto se ve reflejado en varios puntos, como por ejemplo: Se intentó que todos los métodos hicieran una única cosa, creando métodos privados utilizados dentro de otros, utilizamos CamelCase en todas las variables utilizadas, también utilizamos Pascal Case para los métodos y propiedades. Se crearon excepciones personalizadas para los respectivos errores. Se limitó en la mayoría de los casos los parámetros por método a dos. Los tests chequean una sola funcionalidad.

Evidencia:

En esta primera imagen se ve un ejemplo de un método privado únicamente usado para dividir funcionalidades. También podemos notar que todas las variables creadas están en camelCase, también se ve que los métodos a los que se llama están en PascalCase al igual que las propiedades.

8 references |  6/6 passing | Sera Revetria, 3 hours ago | 2 authors, 6 changes

```
public Booking AddBooking(Booking booking)
{
    if (booking.Guests == null)
    {
        throw new NullInputException("Booking Guests");
    }
    if (booking.AccommodationId == 0)
    {
        throw new NullInputException("Booking Accommodation");
    }

    var accommodation = this.accommodationRepository.GetById(booking.AccommodationId);
    if (accommodation == null)
    {
        throw new NotFoundException("Booking Accommodation");
    }
    var tourist = this.touristRepository.GetByEmail(booking.HeadGuest.Email);
    if (tourist != null)
    {
        booking.HeadGuest = tourist;
    }
    double totalprice = CalculateTotalPrice(booking);
    booking.TotalPrice = totalprice;
    var newBooking = this.bookingRepository.Add(booking);
    return newBooking;
}
```

1 reference | Facundo, 9 days ago | 1 author, 2 changes

```
private double CalculateTotalPrice(Booking booking)
{
    double total = 0;
    foreach (Guest guest in booking.Guests)
    {
        total += guest.Amount * guest.Multiplier;
    }
    return total;
}
```

En la imagen anterior también podemos ver como no se pasan mas de dos parámetros por método.

Aquí se evidencia la utilización y creación de excepciones específicas.

```

namespace Domain
{
    13 references | Facundo, 1 day ago | 1 author, 2 changes
    public class APIException : Exception
    {
        2 references | Facundo, 4 days ago | 1 author, 1 change
        public int StatusCode { get; set; }

        11 references | Facundo, 1 day ago | 1 author, 2 changes
        public APIException(string message, int statusCode) : base(message)
        {
            StatusCode = statusCode;
        }
    }

    70 references | Facundo, 1 day ago | 1 author, 3 changes
    public class NullInputException : APIException
    {
        23 references | Facundo, 1 day ago | 1 author, 2 changes
        public NullInputException(string entity) : base(String.Format("The {0} cannot be null", entity), 400)
        { }
    }

    7 references | Facundo, 1 day ago | 1 author, 3 changes
    public class AlreadyExistsException : APIException
    {
        3 references | Facundo, 1 day ago | 1 author, 2 changes
        public AlreadyExistsException(string entity) : base(String.Format("The {0} Already Exists", entity), 400)
        { }
    }

    1 reference | Facundo, 1 day ago | 1 author, 3 changes
    public class FutureDateException : APIException
    {
        0 references | Facundo, 1 day ago | 1 author, 2 changes
        public FutureDateException(string entity) : base(String.Format("The {0} entered is in the future", entity), 400)
        { }
    }
}

```

Y por ultimo aqui podemos ver como los test testean una sola funcionalidad, también se evidencia que hay un solo Assert por test.

```

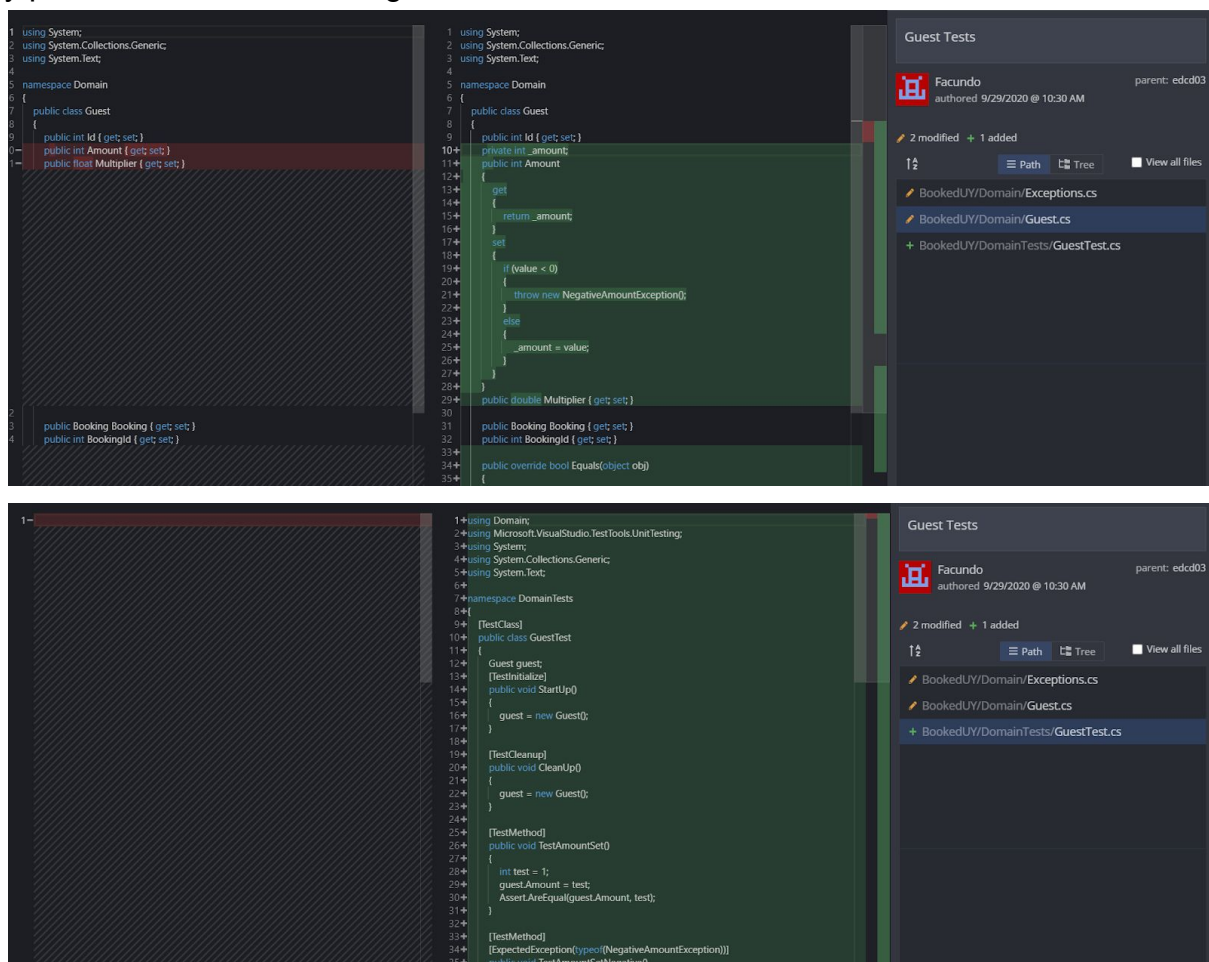
[TestMethod]
✓ | 0 references | Sera Revetria, 19 hours ago | 1 author, 2 changes
public void AddAccommodationTest()
{
    int testId = 5;
    List<CategoryTouristicSpot> categories = new List<CategoryTouristicSpot>();
    CategoryTouristicSpot category = new CategoryTouristicSpot()
    {
        CategoryId = 1,
        TouristicSpotId = 5
    };
    categories.Add(category);
    TouristicSpot touristicSpot = new TouristicSpot()
    {
        Id = testId,
        Name = "abm",
        RegionId = 3,
        Categories = categories
    };
    var mock = new Mock<ITouristicSpotRepository>(MockBehavior.Strict);
    mock.Setup(p => p.Add(It.IsAny<TouristicSpot>())).Returns(touristicSpot);
    mock.Setup(p => p.GetByName(It.IsAny<string>())).Returns<Accommodation>(null);
    var logic = new TouristicSpotLogic(mock.Object);
    var result = logic.AddTouristicSpot(touristicSpot);
    mock.VerifyAll();
    Assert.IsTrue(result.Equals(touristicSpot));
}

```

Para concluir, consideramos que nuestro código es fácil de leer y que no requiere de mayor esfuerzo para comprenderlo, por lo que podemos decir que el seguimiento de Clean Code fue exitoso y nos facilitará parte del próximo obligatorio.

Descripción de la estrategia de TDD seguida

La utilización de la metodología TDD fue lograda parcialmente, si tomamos en cuenta todo el desarrollo del sistema. Nuestro método de prueba fue Inside-Out desde el principio hasta el final, donde se comenzó por el paquete Domain, seguido de la DataAccess, etc. Se empezó por el desarrollo del Dominio con las entidades de negocio. En el inicio se realizó correctamente siguiendo las pautas de TDD, generado una prueba seguido de código que hace que esa prueba sea satisfactoria, y posteriormente refactoring del método.



Guest Tests



Facundo

authored 9/29/2020 @ 10:30 AM

parent: edcd03

2 modified + 1 added



2

≡ Path

📁 Tree

View all files

BookedUY/Domain/Exceptions.cs

BookedUY/Domain/Guest.cs

+ BookedUY/DomainTests/GuestTest.cs

```
1 using DataAccessInterface;
2
3 using System;
4 using System.Collections.Generic;
5 using System.Text;
6
7 namespace DataAccess.Repositories
8 {
9     public class TouristicSpotRepository : ITouristicSpotRepository
10     {
11
12     }
13 }
```

```
1 using DataAccessInterface;
2 using Domain;
3 using Microsoft.EntityFrameworkCore;
4 using System;
5 using System.Collections.Generic;
6 using System.Text;
7
8 namespace DataAccess.Repositories
9 {
10     public class TouristicSpotRepository : ITouristicSpotRepository
11     {
12         private readonly DbSet<TouristicSpot> spots;
13         private readonly DbContext bookUYContext;
14
15         public TouristicSpotRepository(DbContext bookUYContext)
16         {
17             this.bookUYContext = bookUYContext;
18             this.spots = bookUYContext.Set<TouristicSpot>();
19         }
20
21         public IEnumerable<TouristicSpot> GetAll()
22         {
23             return this.spots;
24         }
25     }
26 }
27
```

GetAll Code-Test Spots



Facundo

authored 9/30/2020 @ 10:33 AM

parent: fae7ae

1 modified + 1 added



2

≡ Path

📁 Tree

View all files

+ BookedUY/DataAccess... /TouristicSpotRepositoryTest.cs

BookedUY/DataAccess... /TouristicSpotRepository.cs

```
1-
2-
3-
4-
5-
6-
7-
8-
9-
10-
11-
12-
13-
14-
15-
16-
17-
18-
19-
20-
21-
22-
23-
24-
25-
26-
27-
28-
29-
30-
31-
32-
33-
34-
35-
36-
37-
38-
39-
40-
41-
42-
43-
44-
45-
46-
47-
48-
49-
50-
51-
52-
53-
54-
55-
56-
57-
58-
59-
60-
61-
62-
63-
64-
65-
66-
67-
68-
69-
70-
71-
72-
73-
74-
75-
76-
77-
78-
79-
80-
81-
82-
83-
84-
85-
86-
87-
88-
89-
90-
91-
92-
93-
94-
95-
96-
97-
98-
99-
100-
```

```
1 using DataAccess.Context;
2 using DataAccess.Repositories;
3 using Domain;
4 using Microsoft.EntityFrameworkCore;
5 using Microsoft.VisualStudio.TestTools.UnitTesting;
6 using System;
7 using System.Collections.Generic;
8 using System.Linq;
9 using System.Text;
10
11 namespace DataAccess.Tests
12 {
13     [TestClass]
14     public class TouristicSpotRepositoryTest
15     {
16         [TestMethod]
17         public void TestGetAllSpotsOK()
18         {
19             List<TouristicSpot> spotsToReturn = new List<TouristicSpot>()
20             {
21                 new TouristicSpot()
22                 {
23                     Id=1,
24                     Name="Villa Serrana",
25                     Accommodations=null,
26                     Description="Villa Serrana es un poblado ubicado en el departamento de Lavalle",
27                     "s 25 kilometros al noreste de la capital departamental.", +
28                     "Minas, entre los valles de los arroyos Penitente y Marmarajá.",
29                     Region=null,
30                     RegionId=2,
31                     Categories=null,
32                 },
33                 new TouristicSpot()
34                 {
35                     Id=2,
36                     Name="Colonia del Sacramento",
37                     Accommodations=null,
38                     Description="Colonia del Sacramento es un poblado ubicado en el departamento de Lavalle",
39                     "s 25 kilometros al noreste de la capital departamental.", +
40                     "Minas, entre los valles de los arroyos Penitente y Marmarajá.",
41                     Region=null,
42                     RegionId=2,
43                     Categories=null,
44                 },
45             };
46         }
47     }
48 }
```

GetAll Code-Test Spots



Facundo

authored 9/30/2020 @ 10:33 AM

parent: fae7ae

1 modified + 1 added



2

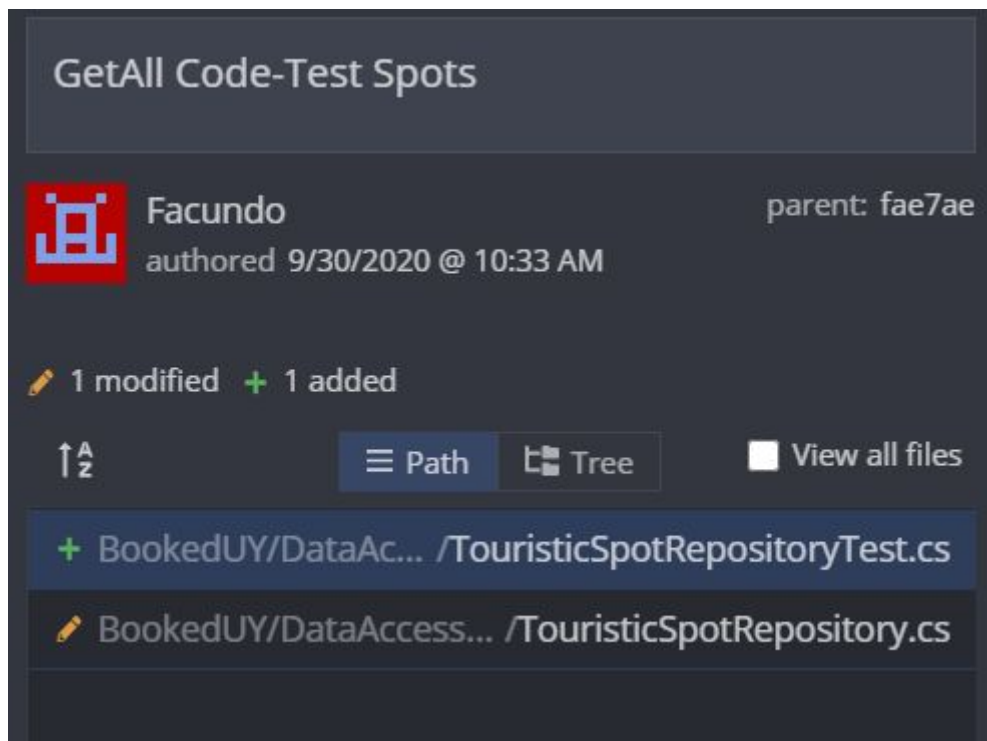
≡ Path

📁 Tree

View all files

+ BookedUY/DataAccess... /TouristicSpotRepositoryTest.cs

BookedUY/DataAccess... /TouristicSpotRepository.cs



En las etapas tempranas no hubo ninguna anomalía en el TDD. A medida que se avanzó en el desarrollo y el tiempo se fue reduciendo por ende el desarrollo pasó a ser más inestable. Nos apuramos para desarrollar y lograr código que cumpliera con las funcionalidades, perdimos de vista parcialmente el desarrollo mediante TDD y de ser ordenados al momento de hacer commits. En muchas partes del desarrollo las pruebas quedaron separadas del código en el repositorio. Por lo que en las últimas etapas de desarrollo no cumplimos con TDD. Un ejemplo claro puede ser el controller de WepApi de Administrator que quedó sin testear porque la autenticación fue de lo último en implementar ya que generó problemas y no tuvimos tiempo de aplicar TDD. Comprendemos que hubo una mala administración de tiempo y que tuvimos deficiencias en nuestra aplicación de TDD.

Informe de cobertura para todas las pruebas desarrolladas

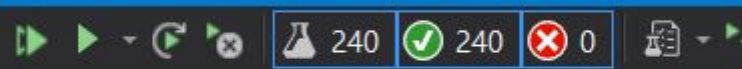


Se logró una cobertura total de un 94%, la cual consideramos aceptable ya que la mayoría del código está probado. Esto es importante ya que reducimos la posibilidad de que el cliente/usuario sea quien se tope con un bug o error no esperado. La clase con menor cobertura es web api y esto se debe a lo explicado en el punto anterior. En el resto de las clases se evidencia que se logró una alta cobertura. En aquellas que tienen una menor

cobertura se debe a una falta de tiempo y a soluciones de código a último momento.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
seraf_LAPTOP-KIT4ND5L 2020...	369	5.74%	6056	94.26%
businesslogic.dll	13	5.75%	213	94.25%
businesslogic.tests.dll	90	5.13%	1665	94.87%
dataaccess.dll	29	2.94%	956	97.06%
dataaccess.tests.dll	2	0.14%	1478	99.86%
domain.dll	35	8.68%	368	91.32%
domaintests.dll	117	17.81%	540	82.19%
webapi.dll	27	20.00%	108	80.00%
webapi.dtos.dll	56	19.38%	233	80.62%
webapi.tests.dll	0	0.00%	495	100.00%

Pruebas unitarias

Se realizaron un total de 240 tests, distribuidos de la siguiente manera.

Test Explorer		
		
240  240  0		
Test	Duration	Trait
BusinessLogic.Tests (45)	382 ms	
DataAccess.Tests (53)	1.6 sec	
DomainTests (127)	34 ms	
WebApi.Tests (15)	308 ms	

Se realizaron pruebas de integración en el proyecto DataAcces, en el Dominio se realizaron pruebas unitarias y en el resto se realizaron mocks, los cuales sirven para solo testear la clase en cuestión y no depender de los resultados de otras clases. Esto genera una ventaja a la hora de testear ya que a la hora de investigar un error, las opciones se reducen a la clase en cuestión, descartando todas las demás clases con las que esta clase interactua.