

3. Clean Code y TDD

En esta sección del documento mostraremos evidencia de que el desarrollo de la aplicación se hizo siguiendo el enfoque TDD lo máximo que se pudo y se siguieron las guías establecidas en Clean Code. Como herramienta de cobertura de pruebas utilizamos CoCoverage en Visual Studio. Y adicionalmente corrimos Coverlet en Visual Studio Code.

3.1. Análisis de cobertura de pruebas:

Se alcanzó la cobertura deseada para todos los paquetes, menos para DataAccess. Para ese caso particular no se alcanzó la cobertura deseada ya que dentro del mismo está incluido el paquete Migrations el cual no tiene pruebas unitarias y hace que la cobertura se muestre inferior a la buscada. A continuación se muestra un reporte de análisis de cobertura de pruebas para cada paquete:

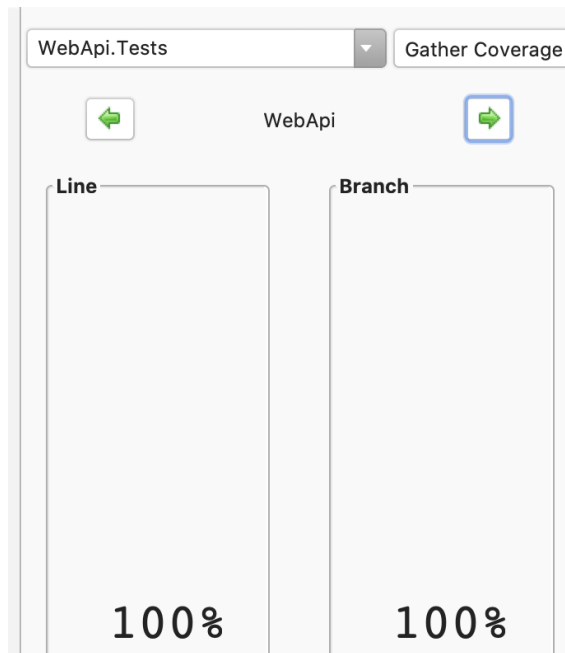


Figura 3.1: Cobertura WebApi en Visual Studio

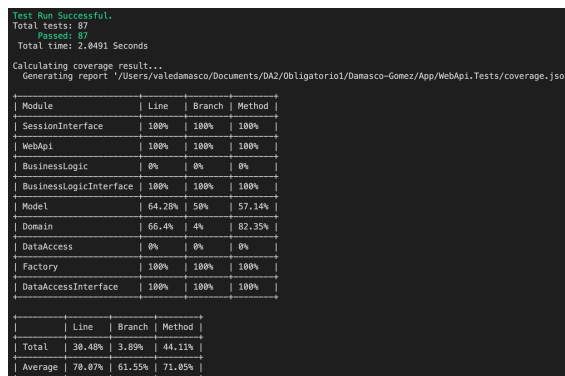


Figura 3.2: Cobertura WebApi en Visual Studio Code

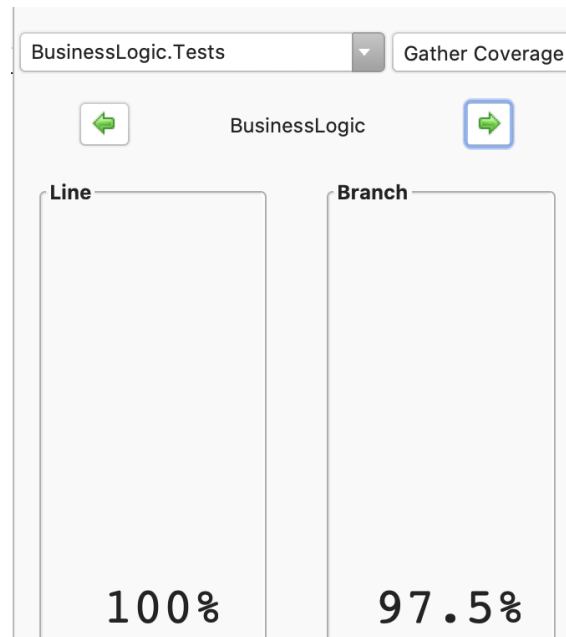


Figura 3.3: Cobertura BussinessLogic en Visual Studio

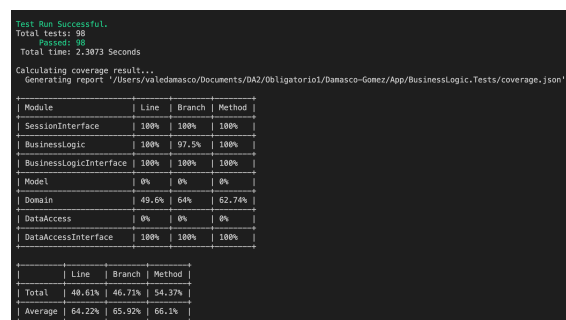


Figura 3.4: Cobertura BussinessLogic en Visual Studio Code

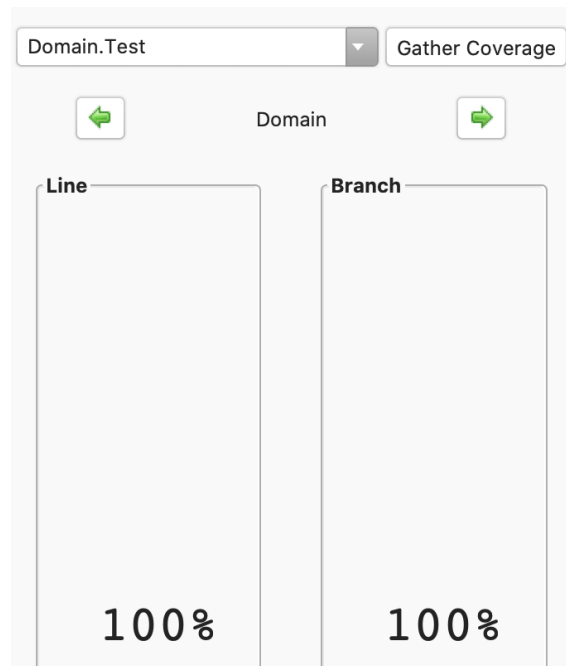


Figura 3.5: Cobertura Domain en Visual Studio

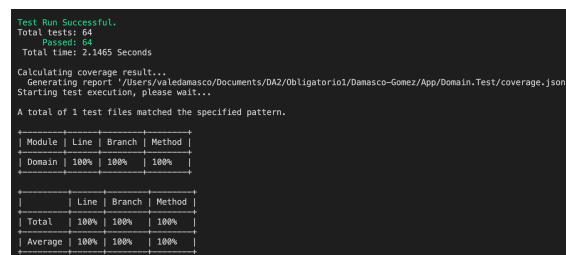


Figura 3.6: Cobertura Domain en Visual Studio Code

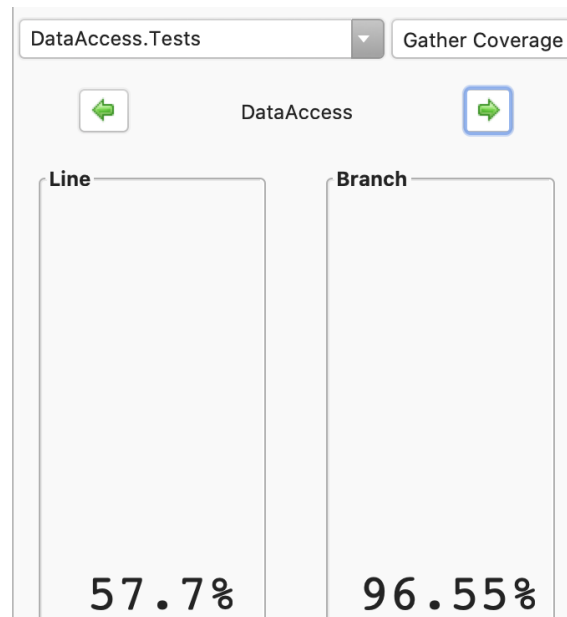


Figura 3.7: Cobertura DataAccess en Visual Studio

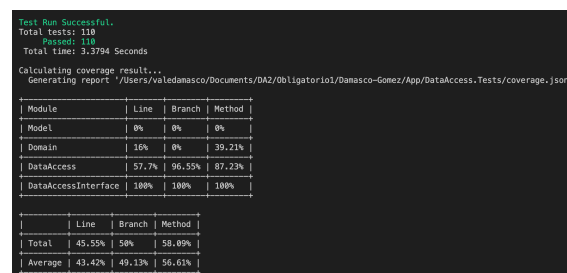


Figura 3.8: Cobertura DataAccess en Visual Studio Code

3.2. Evidencia de TDD:

Empleamos TDD de forma estricta para WebApi. No así para el resto de las clases, para las cuales se trató de cumplir TDD pero se realizó de manera mas flexible. A medida que se iban identificando funcionalidades necesarias se trató de generar los test con los métodos y el mínimo código para que pasen para luego realizar refactoring. Algunos test fueron añadidos al final y no al principio del ciclo por la necesidad de llegar a una cobertura mas alta y gracias a los reportes de los analizadores que nos indicaban si alguna sentencia estaba sin testear.

Globalmente la evidencia de TDD se muestra mediante la realización de los paquetes de test de forma simultánea a la creación de las clases. Para luego realizar el refactorio de los métodos como de las pruebas unitarias.



Figura 3.9:



Figura 3.10:



Figura 3.11:



Figura 3.12:

↓	Add Test classes of the controllers	✓	vdamasco	f78eeeb	Sep 20, 2020 at 15:17
↓	Add webapi.test reference into webapi	✓	vdamasco	5f0168d	Sep 20, 2020 at 14:42
↓	Adding WebApi.Test proyect	✓	vdamasco	18d4cdb	Sep 20, 2020 at 14:41
↓	Config the route in the controllers	✓	vdamasco	c763de7	Sep 20, 2020 at 14:36
↓	Adding reference from webapi to model	✓	vdamasco	9556fcd	Sep 20, 2020 at 14:34
↓	Create Model class for inputs and outputs of webapi	✓	vdamasco	6c76124	Sep 20, 2020 at 14:33
↓	Add all controller classes	✓	vdamasco	2aa10fc	Sep 20, 2020 at 14:24

Figura 3.13:

↓	Controller-region HttpDelete	✗	yuligomez	228cb41	Sep 26, 2020 at 13:02
↓	Http Put	✗	yuligomez	80d26ba	Sep 26, 2020 at 12:31
↓	more Post test	✗	yuligomez	ad99d37	Sep 26, 2020 at 12:18
↓	add TestPostFailSameRegion	✗	yuligomez	e00415c	Sep 26, 2020 at 12:15
↓	test PostOk	✗	yuligomez	fe5057f	Sep 26, 2020 at 12:11

Figura 3.14:

↓	Merge branch 'logic-test' into develop	✓	vdamasco	22a4855	Sep 28, 2020 at 22:27
↓	test initialize BookingTest	✗	yuligomez	30a7289	Oct 1, 2020 at 20:21
↓	refactoring HoseLogic	✗	yuligomez	fd51148	Oct 1, 2020 at 19:57
↓	refactoring houseLogic	✗	yuligomez	2fcabb4	Oct 1, 2020 at 19:56
↓	refactoring CategoryLogic	✗	yuligomez	b5d3d45	Oct 1, 2020 at 19:28
↓	refactoring CategoryLogic	✗	yuligomez	9075cb6	Oct 1, 2020 at 19:28
↓	refactor HouseLogicTets	✗	yuligomez	012a30b	Oct 1, 2020 at 19:20
↓	refactoring PersonLogic test	✗	yuligomez	bb07418	Oct 1, 2020 at 19:04
↓	Domain tests	✗	yuligomez	95e45fb	Sep 30, 2020 at 19:17
↓	new logic tests	✗	yuligomez	20736e0	Sep 30, 2020 at 18:50
↓	more logic test	✗	yuligomez	f9754a2	Sep 30, 2020 at 18:32
↓	test correction	✗	yuligomez	5380131	Sep 29, 2020 at 20:49
↓	more test passed	✗	yuligomez	893a205	Sep 28, 2020 at 22:21
↓	Merge branch 'develop' into logic-test	✗	yuligomez	fba6e15	Sep 28, 2020 at 21:01

Figura 3.15:

- Buscar hospedajes para un cierto punto turístico con los parámetros especificados

↓	Clean code	✓	vdamasco	64f338f	Oct 3, 2020 at 11:29
↓	Add update house and test	✓	vdamasco	9a6b235	Oct 3, 2020 at 11:27
↓	Merge branch 'House-controller-gethousesby' into update-method-fix	✓	vdamasco	eaf1a35	Oct 3, 2020 at 10:53
↓	TestCalculateTotalPrice and CalculateTotalPrice	✗	yuligomez	1ec8421	Oct 2, 2020 at 21:35
↓	adding Model	✗	yuligomez	fb58443	Oct 2, 2020 at 21:09
↓	refactor TestGetByIdTouristPoint	✗	yuligomez	8a64432	Oct 2, 2020 at 20:35
↓	refactoring getHousesByTouristPoint	✗	yuligomez	c6229b0	Oct 2, 2020 at 20:20
↓	refactoring getHousesByTouristPoint	✗	yuligomez	ec83f1a	Oct 2, 2020 at 20:20
↓	Update data base with touristpoint id	✓	vdamasco	35099c4	Oct 2, 2020 at 19:58
↓	TestGetHousesBy , function GetHousesBy in HouseController	✗	yuligomez	4dcbe96	Oct 2, 2020 at 19:34

Figura 3.16: GetHousesBy

- Realizar una reserva de un hospedaje. Para esta funcionalidad se deja evidencia de que primero se creo el controlador de Booking y sus respectivos test para cada endpoint definido dentro del mismo. Con lo cual incluye la reserva de un hospedaje mediante el método POST.

↓	Merge branch 'creation-controller-booking' into develop	✓	vdamasco	9691e19	Sep 26, 2020 at 12:15
↓	origin/creation-controller-booking Add delete method	✓	vdamasco	dc4021c	Sep 26, 2020 at 12:15
↓	Add method put	✓	vdamasco	f6f01d4	Sep 26, 2020 at 12:14
↓	Add post method with test	✓	vdamasco	9848006	Sep 26, 2020 at 12:13
↓	Add test and method get by booking controller	✓	vdamasco	31e1846	Sep 26, 2020 at 12:11

Figura 3.17: Post ../bookings

- Cambiar el estado de una reserva, indicando una descripción : El refactorio para esta funcionalidad se muestra a continuación



The image shows a screenshot of a commit history table, likely from a version control system like Git. The table has four columns: a commit icon, a commit message, a commit hash, and a commit date. The commit messages are: 'update interface booking logic', 'Update booking logic update', 'Add some comments for later', 'Fix booking controller test', and 'Improving logic add and update booking'. The commit hashes are: '3518a1c', '5ea672c', 'daa953b', '248bbde', and '59713d9'. The commit dates are: 'Oct 6, 2020 at 10:19', 'Oct 6, 2020 at 10:19', 'Oct 5, 2020 at 22:12', 'Oct 5, 2020 at 22:00', and 'Oct 5, 2020 at 21:52'. The last commit is highlighted in blue.



	update interface booking logic	3518a1c	Oct 6, 2020 at 10:19
	Update booking logic update	5ea672c	Oct 6, 2020 at 10:19
	Add some comments for later	daa953b	Oct 5, 2020 at 22:12
	Fix booking controller test	248bbde	Oct 5, 2020 at 22:00
	Improving logic add and update booking	59713d9	Oct 5, 2020 at 21:52

Figura 3.18: Put ../bookings

3.3. Evidencia de clean code:

En esta sección se muestran evidencias de haber seguido Clean Code para nuestro desarrollo de código.

3.3.1. Nombres significativos

Los nombres de variables se hicieron en notación húngara, además se definieron lo más informativos posible. Los nombres de los métodos comienzan con mayúscula y respetan notación húngara.

```
[HttpGet("{id}",Name="GetPerson")]
2 references
public IActionResult GetBy([FromRoute]int id)
{
    Person elementPerson = this.personLogic.GetBy(id);
    PersonBasicModel personBasicModel = new PersonBasicModel(elementPerson);
    return Ok(personBasicModel);
}
```

Figura 3.19:

3.3.2. Funciones

Los métodos nos reciben mas de dos parámetros. Para limitar la cantidad de parámetros en las firmas de las funciones englobamos los mismos en DTOs. De esta manera logramos cumplir Clean Code y en consecuencia logramos un sistema mas seguro al no exponer nuestras entidades.

```
[HttpPut("{id}")]
[AuthorizationFilter]
3 references
public IActionResult Put([FromRoute]int id,[FromBody]CategoryModel categoryModel)
{
    Category newCategory = categoryModel.ToEntity();
    newCategory = this.categoryLogic.Update(id,newCategory);
    CategoryBasicInfoModel categoryInfoModel = new CategoryBasicInfoModel(newCategory);
    return CreatedAtRoute("GetCategory", new {Id = categoryInfoModel.Id} ,categoryInfoModel);
}
```

Figura 3.20:

3.3.3. Identación

Procedimos a realizar la indentación sugerida por los lineamientos de clean code , aunque no aseguramos que cada línea de código cumpla con este punto.

3.3.4. Importaciones correctas

Solo fueron importados los namespace necesarios en cada clase. Fueron removidas las importaciones innecesarias realizando refactorio del código..

```

[HttpGet("{id}",Name="GetPerson")]
2 references
public IActionResult GetBy([FromRoute]int id)
{
    Person elementPerson = this.personLogic.GetBy(id);
    PersonBasicModel personBasicModel = new PersonBasicModel(elementPerson);
    return Ok(personBasicModel);
}

```

Figura 3.21:

3.3.5. Bajo Acoplamiento

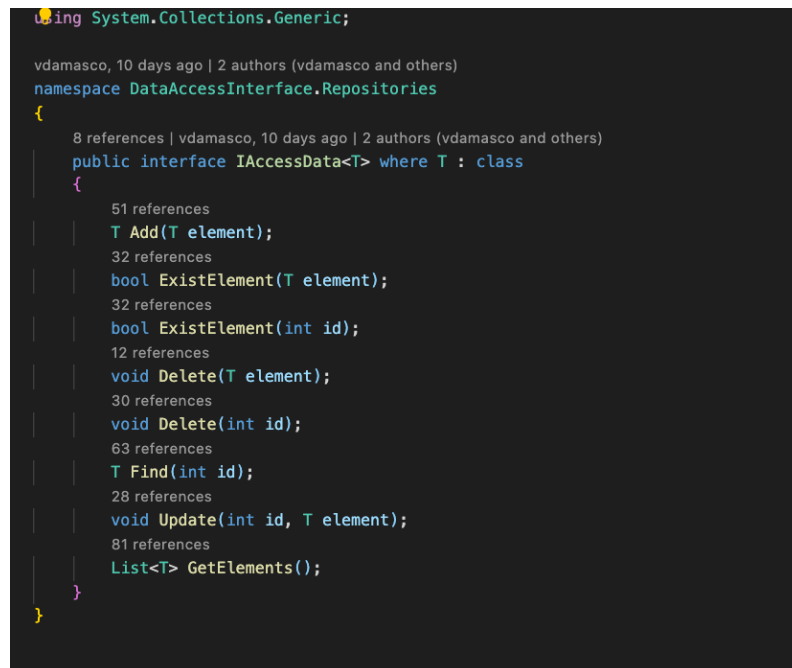
Para lograr un bajo acoplamiento y alinearnos a Clean Code evitamos el acoplamiento a implementaciones específicas y utilizamos interfaces en vez de ello.

3.3.6. Sistema

En esta sección mostramos como evidencia de Clean code que separamos responsabilidades de cada clase de manera de cumplir SRP. Separando la lógica de negocio , las entidades del sistema , los repositorios y la WebApi. Utilizamos el patrón facade evidenciado por nuestra WebApi. Asimismo empleamos inyección de dependencias imponiendo que un objeto no es responsable de instanciar sus dependencias sino que lo delegamos a las interfaces. Tanto nuestra lógica de negocio, como nuestros repositorios, como la sesión se acoplan a una interfaces.

3.3.7. No duplicación

Para nuestro código practicamos la ideología de la no duplicación de código. De esta manera evitamos re trabajo y complejidad adicional innecesaria generando código limpio. Dejamos evidencia de este punto mostrando que dentro del paquete de repositorios mantenemos una clase abstracta `DataAccess` genérica y cada repositorio hereda de esta clase , de esta manera re usan los métodos definidos en la clase genérica.



```
using System.Collections.Generic;

vdamasco, 10 days ago | 2 authors (vdamasco and others)
namespace DataAccessInterface.Repositories
{
    8 references | vdamasco, 10 days ago | 2 authors (vdamasco and others)
    public interface IAccessData<T> where T : class
    {
        51 references
        T Add(T element);
        32 references
        bool ExistElement(T element);
        32 references
        bool ExistElement(int id);
        12 references
        void Delete(T element);
        30 references
        void Delete(int id);
        63 references
        T Find(int id);
        28 references
        void Update(int id, T element);
        81 references
        List<T> GetElements();
    }
}
```

Figura 3.22:

3.3.8. Test

Los test fueron separados en paquetes independientes, los mismos son: `WebApi.Tests`, `Domain.Test`, `DataAccess.Test`, `BusinessLogic.Test`. De esta manera logramos que cada paquete tenga la única responsabilidad de testar a lo que referencia cada nombre respectivamente. Se llegó a una cobertura alta. Se pretendió seguir TDD al máximo y en la medida de lo posible. Cada test tiene bien diferenciadas sus partes (Arrange, Act y Assert. Se hizo uso de mocks para las pruebas unitarias de `WebApi` y para la lógica de negocio.

```

[TestMethod]
0 references | Run Test | Debug Test
public void TestGetAllBookingsOk()
{
    mock.Setup(m => m.GetAll()).Returns(bookingsToReturn);
    IEnumerable<BookingBasicModel> bookingModels = bookingsToReturn.Select(m => new BookingBasicModel(m));

    var result = controller.Get();

    var okResult = result as OkObjectResult;
    var bookings = okResult.Value as IEnumerable<BookingBasicModel>;
    mock.VerifyAll();
    Assert.IsTrue(bookingModels.SequenceEqual(bookings));
}

```

Figura 3.23:

3.3.9. Manejo de errores

Para realizar un correcto manejo de errores utilizamos una clase `ExceptionHandler` de manera de centralizar el catcheo de errores de la aplicación y encapsularlos en una única clase que tiene esta única responsabilidad.

```

namespace Filters
{
    1 reference | You, 3 days ago | 1 author (You)
    public class ExceptionFilter : IExceptionHandler
    {
        0 references
        public void OnException(ExceptionContext context)
        {
            try
            {
                throw context.Exception;
            }
            catch (ArgumentException e)
            {
                context.Result = new ContentResult()
                {
                    StatusCode = 400,
                    Content = e.Message.ToString()
                };
            }
            catch (Exception)
            {
                You, 5 days ago * catch(Exception) in ExceptionFilter
                context.Result = new ContentResult()
                {
                    StatusCode = 500,
                    Content = "Server error"
                };
            }
        }
    }
}

```

Figura 3.24: