Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones II
Obligatorio I
Evidencia TDD y Cleancode

Juan Pablo Barrios - 206432
Juan Ignacio Irabedra - 212375

Entregado como requisito de la materia Diseño de aplicaciones 2

15 de octubre de 2020

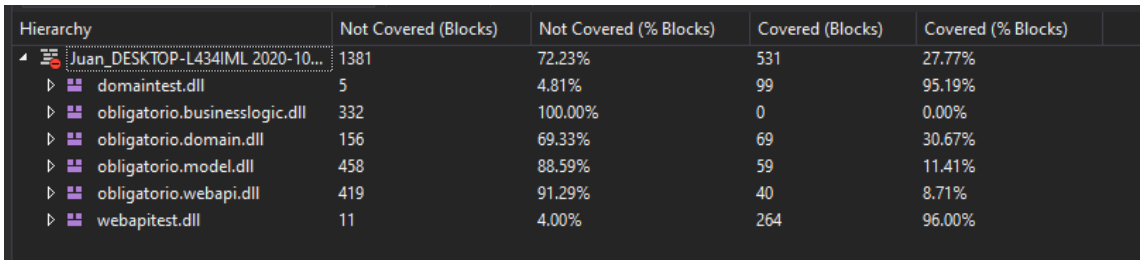
Contents

1. Evidencia de TDD:

Comenzamos aplicando TDD. Nos enfrentamos a dificultades con la tecnología y esto nos atrasó muchísimo. Decidimos no aplicar TDD. Las pruebas unitarias son una característica del código profesional. En este aspecto el nuestro **no** lo es. Preferimos hacer énfasis en que el obligatorio más o menos funcione para que se nos puedan evaluar muchas otras dimensiones de este proyecto.

Esto tuvo como consecuencia muy baja cobertura de pruebas unitarias, estamos al tanto de que esto va en contra de la rúbrica de evaluación. También sabemos que TDD es mencionado en Clean Code. De todos modos optamos por dedicar una sección a CleanCode sin tener en cuenta las pruebas a los efectos de defender todos aquellos demás aspectos que creemos son valiosos de Clean Code y que sí llevamos a la práctica.

De todos modos, dejamos la cobertura de código para pruebas unitarias como muestra de transparencia.



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Juan_DESKTOP-L434IML 2020-10...	1381	72.23%	531	27.77%
domaintest.dll	5	4.81%	99	95.19%
obligatorio.businesslogic.dll	332	100.00%	0	0.00%
obligatorio.domain.dll	156	69.33%	69	30.67%
obligatorio.model.dll	458	88.59%	59	11.41%
obligatorio.webapi.dll	419	91.29%	40	8.71%
webapitest.dll	11	4.00%	264	96.00%

Figure 1.1: Cobertura de código por pruebas unitarias

2. Evidencia de Clean Code:

Para el desarrollo de esta apliación se decidió seguir como guia el libro de Clean Code escrito por Robert C. Martin. Intentamos seguir lo más posible este documento a lo largo del desarollo de la aplicación. Algunos de los estándares que nos parecieron más importantes y que intentamos seguir para todo el trabajo fueron los siguientes:

- Nombres signifacntes: Esto lo aplicamos para variables, métodos, interfaces, clases y paquetes.

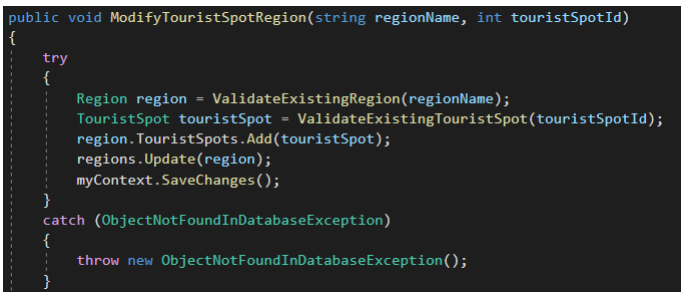


Figure 2.1: Diagrama de estructura compuesta

- Limitación en la cantidad de parámetros recibidos por una función. Ninguna función en nuestro sistema recibe más de 3 parámetros como sugiere Clean Code.

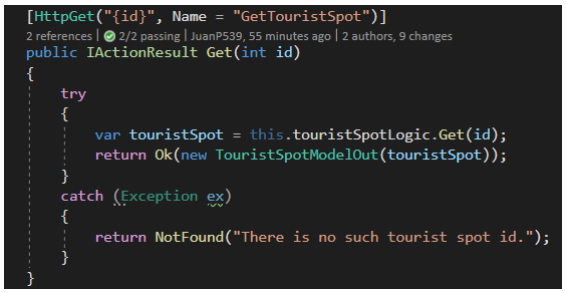


Figure 2.2: Diagrama de estructura compuesta

- Comentarios, evitamos hacerlos. Los comentarios que se encuentran en el documento, consideramos que casi todo el código escrito es perfectamente entendible sin necesidad de comentarios extras.

```
public void Add(Category newEntity)
{
    if (AlreadyExistsByName(newEntity))
    {
        throw new RepeatedObjectException();
    }
    else
    {
        categories.Add(newEntity);
        myContext.SaveChanges();
    }
}
```

Figure 2.3: Diagrama de estructura compuesta

```
public void ModifyTouristSpotRegion(string regionName, int touristSpotId)
{
    try
    {
        Region region = ValidateExistingRegion(regionName);
        TouristSpot touristSpot = ValidateExistingTouristSpot(touristSpotId);
        region.TouristSpots.Add(touristSpot);
        regions.Update(region);
        myContext.SaveChanges();
    }
    catch (ObjectNotFoundInDatabaseException)
    {
        throw new ObjectNotFoundInDatabaseException();
    }
}
```

Figure 2.4: Diagrama de estructura compuesta

- Densidad vertical y horizontal: Estos conceptos fueron tomados muy en cuenta al momento de realizar los métodos.

Hubo un método solo que no respeta Clean Code y es el siguiente:

Este método lo planeábamos refactorizar para que cumpliera Clean Code pero por temas de tiempo no lo pudimos hacer. Pära la siguiente entrega vamos a eliminar este método y re escribirlo.

```
[HttpGet]
1 reference | 0/1 passing | Juan2520, 53 minutes ago | 2 authors, 11 changes
public IActionResult Get()
{
    if (Request.QueryString.Value is null)
    {
        return Ok(this.touristSpotLogic.GetAll().Select(ts => new TouristSpotModelOut(ts)));
    }
    else
    {
        List<TouristSpotModelOut> touristSpots = new List<TouristSpotModelOut>();
        string arguments = Request.QueryString.Value.Split('?')[1];
        List<string> criteria = arguments.Split('&').ToList<string>();
        string sortingRegion = "";
        bool queryStringHasCategory = false;
        foreach (var param in criteria)
        {
            string regionName = param.Split('=')[0];
            if (regionName != "regionName")
            {
                string value = param.Split('=')[1].Replace("%22", "");
                value = value.Replace("%20", " ");
                touristSpots.AddRange(
                    touristSpotLogic.FindByCategory(value).Select(ts => new TouristSpotModelOut(ts)));
                queryStringHasCategory = true;
            }
            else
            {
                if (sortingRegion != "")
                {
                    return BadRequest("Remember to select only one region.");
                }
                sortingRegion = param.Split('=')[1].Replace("%22", "");
                sortingRegion = sortingRegion.Replace("%20", " ");
            }
        }
        List<TouristSpotModelOut> touristSpotsWithNoDuplicates = new List<TouristSpotModelOut>();
        foreach (var ts in touristSpots)
        {
            if (!touristSpotsWithNoDuplicates.Contains(ts))
            {
                touristSpotsWithNoDuplicates.Add(ts);
            }
        }
        sortingRegion = sortingRegion.Trim();
        if (sortingRegion is null || sortingRegion == "")
        {
            return BadRequest("You need to specify the region.");
        }
        List<TouristSpotModelOut> touristSpotbyRegion = new List<TouristSpotModelOut>();
        touristSpotbyRegion.AddRange(
                    touristSpotLogic.FindByRegion(sortingRegion).Select(ts => new TouristSpotModelOut(ts)));
        List<TouristSpotModelOut> touristSpotsReturn = new List<TouristSpotModelOut>();
        foreach (var item in touristSpotbyRegion)
        {
            if (touristSpotsWithNoDuplicates.Contains(item))
            {
                touristSpotsReturn.Add(item);
            }
        }
        if (queryStringHasCategory)
        {
            return Ok(touristSpotsReturn);
        }
        else
        {
            return Ok(touristSpotbyRegion);
        }

    }
}
```

Figure 2.5: Diagrama de estructura compuesta

Bibliography

[1] "Universidad ORT Uruguay". (2020) "Letra del primer obligatorio de Diseño de Aplicaciones I, semestre impar 2020". [Online]. Available: "http://www.aulas.ort.edu.uy"

[2] R. C. Martin, *Clean Code: A handbook of agile software craftsmanship*, 1st ed. Prentice Hall, 2009.

[3] Martin Fowler. (2013-9-5) Tell, dont ask. [Online]. Available: https://martinfowler.com/bliki/TellDontAsk.html