Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones 2
Descripción del diseño
Obligatorio 1

Emiliano Barboza (147067)
Mauricio Dalgalarrondo (189280)

Docentes:
Ignacio Valle
Daniel Acevedo

Entregado como requisito de la materia Diseño de aplicaciones 2
https://github.com/ORT-DA2/barboza-dalgalarrondo

15 de octubre de 2020

Declaraciones de autoría

Nosotros, Emiliano Barboza y Mauricio Dalgalarrondo, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos la materia
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Resumen

Este documento del obligatorio 1 plantea poner en práctica los conocimientos vistos hasta el momento en el curso del punto de vista la aplicación de TDD y Clean Code visto en diseño de aplicaciones 1:

1. TDD
2. Clean code
3. Code coverage

Palabras claves

- TDD, Test Development Driven, consta de una técnica para el desarrollo de software mediante pruebas unitarias.
- Code coverage, harramienta de reporte de cobertura de los test unitarios.
- Outside-In, se trata de una metodología aplicada a TDD. Consiste en que el desarrollador comience creando la funcionalidad final del sistema y luego mediante rafactorizaciones ir creando pequeños componentes que dan soporte. El camino resulta más como ir explorando las necesidades. Esto resulta bueno, cuando se conoce la idea general, pero no los detalles de la implementación.

Índice general

1. Descripción del Proyecto

1.1. Introducción y Objetivos

El objetivo de este proyecto es crear una api rest para dar soporte a la marca Uruguay Natural.

Las tecnologías invlucradas son:

- Web Services (REST API) - Se expondrá las apis en un IIS.
- Microsoft Visual Studio Code (lenguaje C)
- Microsoft SQL Server Express 2017
- Postman
- NET Core SDK 3.1 / ASP.NET Core 3
- Entity Framework Core 3.1.3

1.2. Alcance de la Aplicación

El nuevo sistema, deberá dar acceso a 3 tipos de usuarios posibles en la plataforma (turistas, administradores y super administradores). Para los cuales se listan los requerimientos funcionales para los mismos.
Requerimientos funcionales:

- RF1 - Búsqueda de puntos turísticos por región y por categoría:.
- RF2 - Elegir un punto turístico y realizar una búsqueda de hospedajes.
- RF3 - Dado un hospedaje, realizar una reserva.

Requerimientos no funcionales

- RNF1 - Independencia de librerías.

  Se debe diseñar la solución que al modificar el código fuente minimice el impacto del cambio en los componentes físicos de la solución.

  Cada paquete lógico debe ser implementado en un assembly independiente.

- RNF2 - Acceso a las funcionalidades mediante HTTP

  Acceso mediante web service.

- RNF3 - Persistencia en base de datos.

  El diseño debe contemplar el modelado de una solución de persistencia adecuada para el problema utilizando Entity Framework (Code First).

  Se espera que como parte de la entrega se incluya dos respaldos de la base de datos: uno vacío y otro con datos de prueba.

  Se debe entregar el archivo .bak y también el script .sql para ambas bases de datos.

- RNF4 - Mantenibilidad

  Estar en un repositorio Git.

  Haber sido escrito utilizando TDD

  Se debe utilizar el framework Moq para realizar los Mocking.

- RNF5 - Control de versiones

  GitFlow

2. Descripción de la estrategia de TDD

Elegimos la estrategia de Outside-In, debido a que nuestro conocimiento mayor consistía en saber qué apis eran necesarias, por lo tanto decidimos empezar desde afuera.

Siguiendo con las reglas de TDD (toda implementación requiere ser una necesidad generada por un test), comenzamos creando nuestra WebApi.Test.
En la misma fuimos primeramente creando el controller junto al test que fuimos necesitando. Luego que ese test estaba escrito y fallaba (red), podíamos pasar a la siguiente etapa que era la implementación (green).

Finalmente, luego de varios ciclos aplicando esta técnica fuimos viendo las oportunidades de refactoreo (refactor), dado a que fuimos viendo que en varias ocaciones el código se repetía.

Por ende este fue el flujo de trabajo aplicado en todo el cíclo de desarrollo de la api.

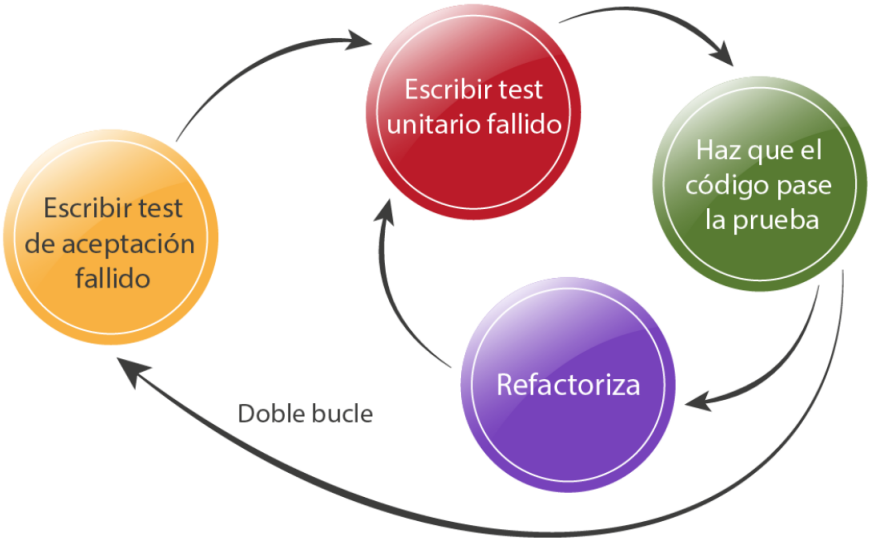

Figura 2.1: Ciclo TDD

2.1. Evidencia cantidad de test unitarios creados en el proceso

Se crearon un total de 177 test unitarios, los cuales se dividieron de la siguiente forma por proyecto:

- AuthorizationDataAccess.Test - 14
- AuthorizationEngine.Test - 5
- Helper.Test - 11
- NaturalUruguayDataAccess.Test - 60
- NaturalUruguayEngine.Test - 48
- WebApi.Test - 39

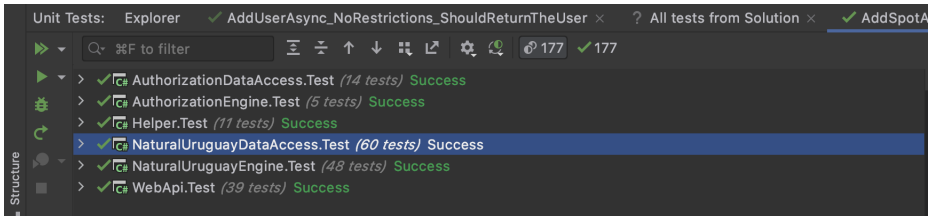

Figura 2.2: Evidencia TDD.

2.2. Nomenclatura para nombrar test unitarios

Seguimos las recomendaciones del artículo, donde especifica como buena práctica para el nombre de un test:

- NombreMetodo_EstadoActual_EstadoFinalEsperado

2.3. Pruebas de aplicación de TDD para los casos marcados con *

2.3.1. (*)Buscar hospedajes para un cierto punto turístico con los parámetros especificados.

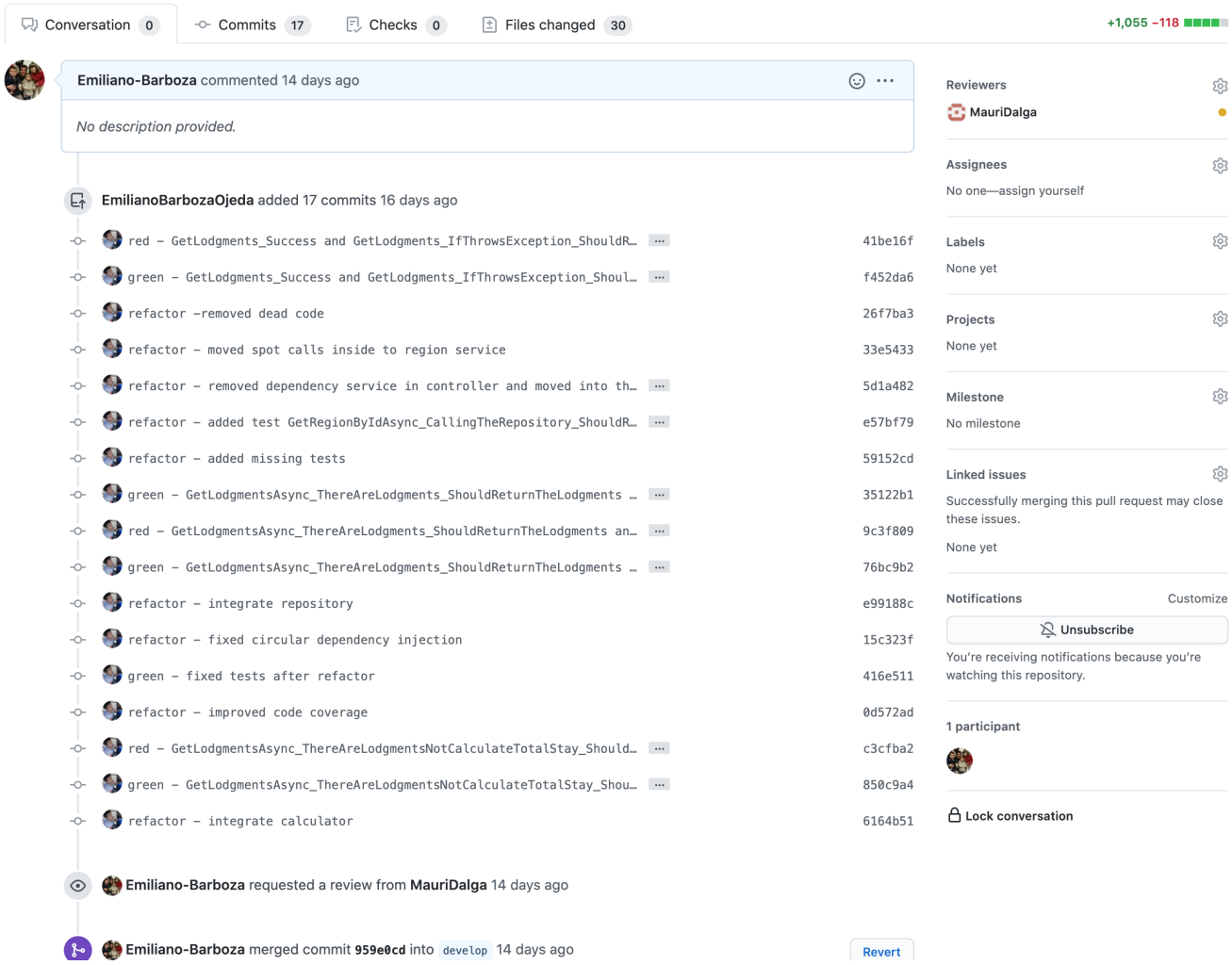

Figura 2.3: Buscar hospedajes. Pull Request

2.3.2. (*) Realizar una reserva de un hospedaje.

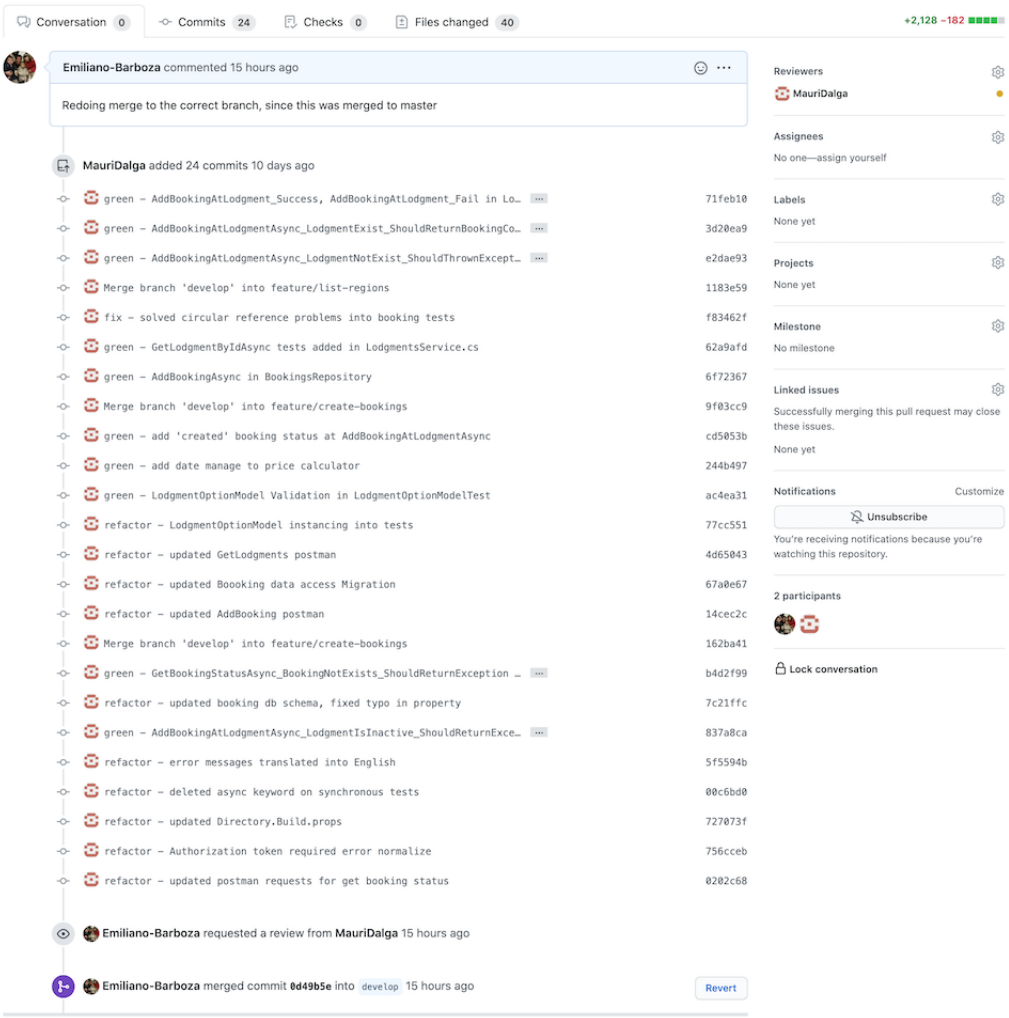

Figura 2.4: Realizar una reserva. Pull Request

2.3.3. (*) Dar de alta un nuevo hospedaje o borrar uno existente, para un punto turístico existente.

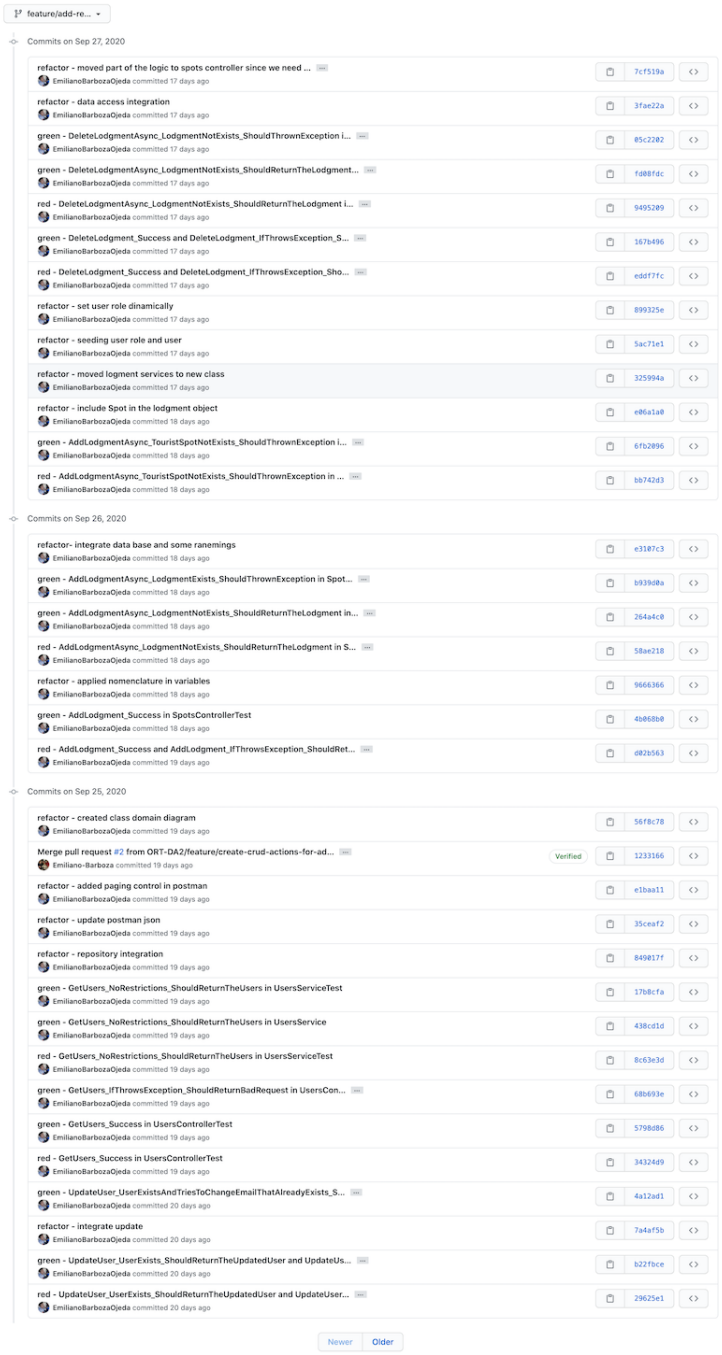

Figura 2.5: Dar de alta un nuevo hospedaje o borrar uno existente. Pull Request

2.3.4. (*) Modificar la capacidad actual de un hospedaje.

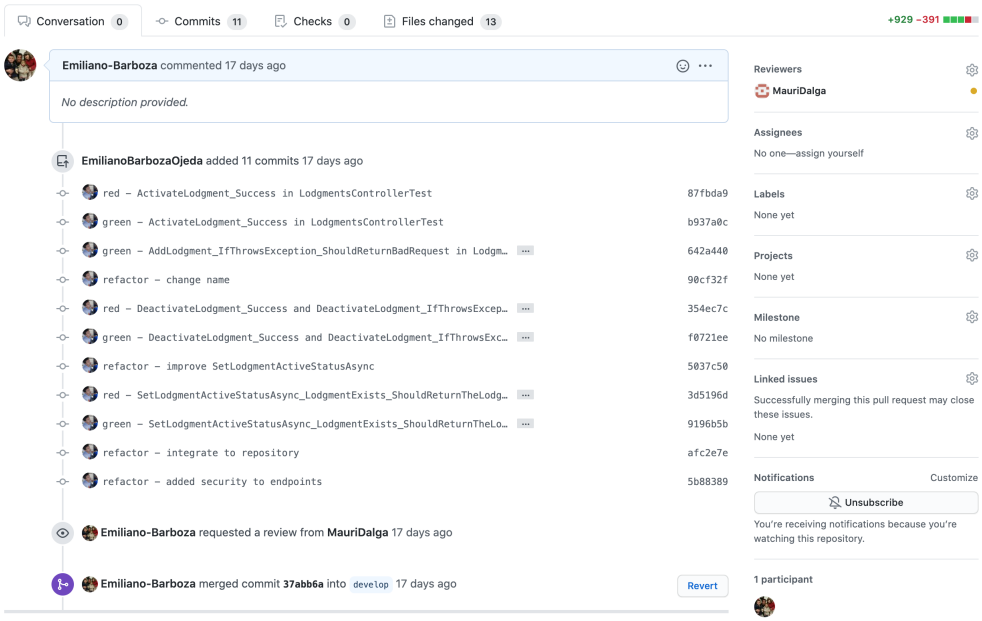

Figura 2.6: Modificar la capacidad actual de un hospedaje. Pull Request

2.3.5. (*) Cambiar el estado de una reserva, indicando una descripción.

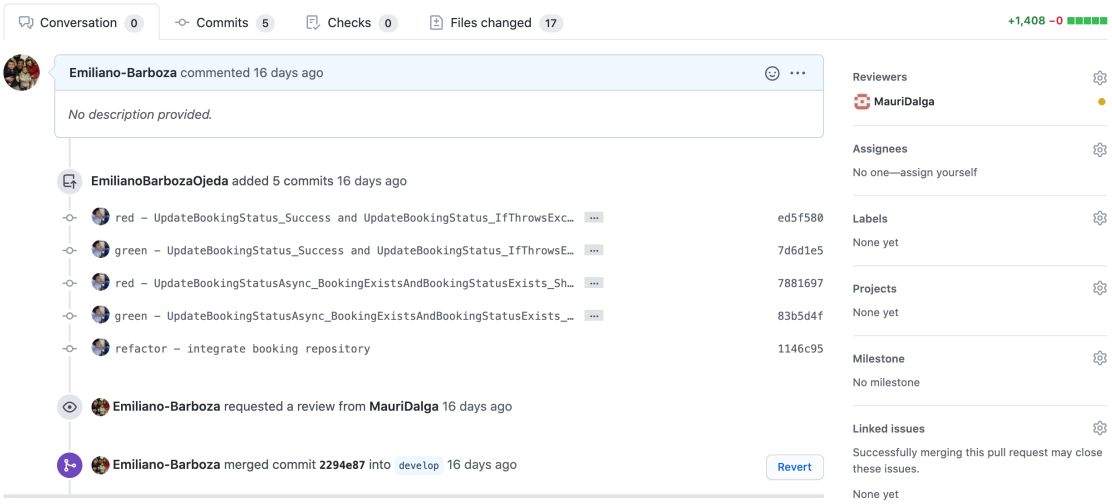

Figura 2.7: Cambiar el estado de una reserva. Pull Request

3. Clean Code

3.1. Nomenclatura

Se siguió el estandard de C sharp declarado en el siguiente repositorio con el fin de seguir buenas prácticas.

3.2. Largo de métodos

Los métodos no superan las 30 líneas en los métodos más largos, respetando la legibilidad en la laectura.

3.3. Parámetros

Declaramos un máximo de 3 parámetros en los métodos.

3.4. Nombres

Los nombres son nemotécnicos y hacen referencia a los objetos que conyevan.

3.5. Excepciones

Se manejan las expceciones tiradas por base de datos de forma que las mismas no lleguen al cliente de forma cruda

3.6. Lugares donde se infringieron criterios de Clean code

Lugares como el repositorio, Setup de web api y los paquetes de test, fueron excluídos para este criterio debido a que en algunos casos es muy conveniente la nomenclatura Lambda, la cual va en contra de Clean Code.

3.7. Paquetes de Test

Como estrategia de test unitarios se decidió que los proyectos de test se nombren de forma tal que sea el "NombreDelProyecto.Test".

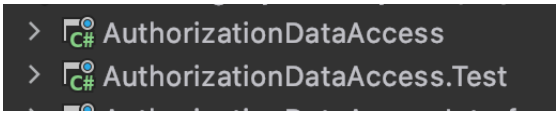

Figura 3.1: Convención nombres de paquetes de test.

4. Code coverage

Declaramos haber logrado un 95 % de code coverage.
Identificamos que faltaron casos de prueba para lograr un mejor desempeño en el paquete de Domain el cual quedó con un 70 %, siendo así el menos cubierto.

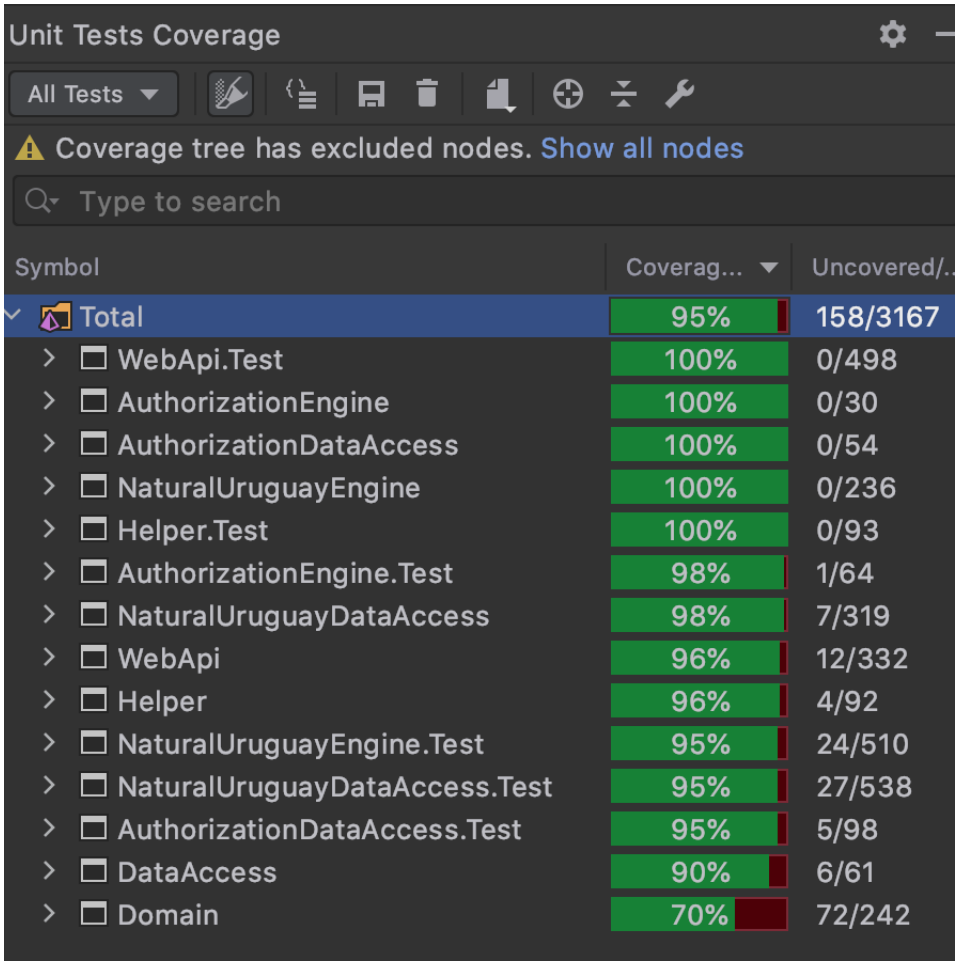

Figura 4.1: Convención nombres de paquetes de test.

4.1. Paquetes y clases excluídas

- Migrations dentro del DataAccess, estas con las migraciones que se corren con el comando dotnet.

- Migrations, este es el proyecto del cual arrancamos las migraciones.
- Factory, aquí únicamente ocurren la inyección de dependencias.
- Clase Program y Setup del proyecto WebApi, estas no tiene sentido testearlas ya que es el arranque de la aplicación y configuraciones de la misma.
- WrappedDbException, implemtanta la clase Exception y nada más, por lo cual no tiene sentido testearla ya que sería igual que testear Exception.
- NaturalUruguayContext, es el acceso a la base de datos.

Bibliografía

[1] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf