OBLIGATORIO 1 - SmartHome

Diseño 2

Evidencia de Clean Code y de la aplicación de TDD

Link al repositorio:

https://github.com/IngSoft-DA2/281542-281835-281535

Integrantes:

- Matias Corvetto (281535)
- Alejo Fraga (281542)
- Sebastian Vega (281835)

Docentes:

- Daniel Acevedo
- Federico Gonzales

Septiembre 2024

Universidad ORT Uruguay

Facultad de Ingeniería

Abstract

Se busca evidenciar la correcta aplicación de TDD en el obligatorio y los seguimientos a los criterios de Clean Code. En la siguiente documentación se va a apreciar que la técnica de TDD fue aplicada correctamente, aunque la cobertura no llega a un 100% ya que se utilizaron shadow properties para facilitar el mapeo de EF Core. Además de que luego de refactorizar métodos para tener en cuenta casos bordes (como si un parámetro es null), se cometió la desatención de no refactorizar los test, agregando pruebas para esos casos bordes.
Las directrices de Clean Code fueron seguidas en todo el obligatorio, como se van a poder ver en unos ejemplos, aunque por obvias razones se refuerza más viendo el código fuente de la aplicación.

Índice

1. Estrategia de TDD utilizada

En esta entrega se aplicó la estrategia de TDD conocida como Inside-Out, la cual se caracteriza por comenzar con las pruebas de las clases del dominio, es decir, aquellas que representan los conceptos principales y las reglas de negocio del sistema. Este enfoque busca establecer primero una base sólida en los componentes centrales de la aplicación antes de abordar otros niveles más abstractos o dependientes.

El proceso comienza diseñando y escribiendo pruebas unitarias para validar el comportamiento de estas clases del dominio, asegurándose de que su funcionalidad cumpla con los requisitos establecidos. Una vez que estas clases están bien probadas y se ha verificado su correcto funcionamiento, se avanza hacia la siguiente fase, que consiste en probar los servicios que interactúan con dichas clases.

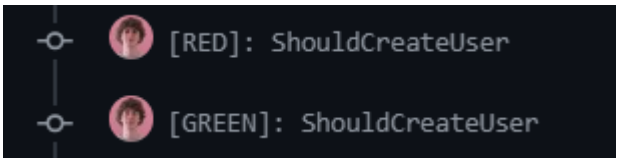
De esta manera, la estrategia Inside-Out asegura que los servicios dependen de un núcleo confiable y bien probado, reduciendo la posibilidad de errores en las capas superiores del sistema. Este enfoque no solo permite detectar y corregir problemas desde las primeras etapas del desarrollo, sino que también facilita una integración más fluida y estructurada de los diferentes componentes del sistema.

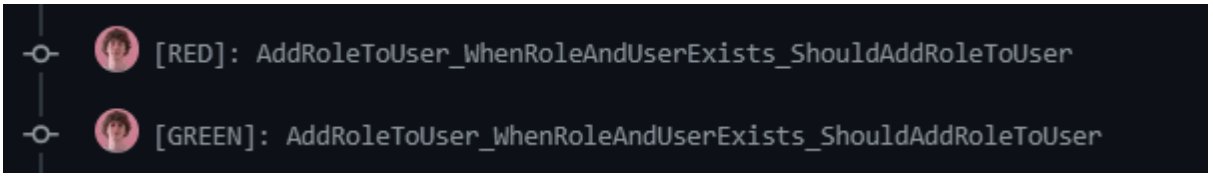
2. Evidencia de TDD en requerimientos

2.1 Mantenimiento de cuentas de administrador

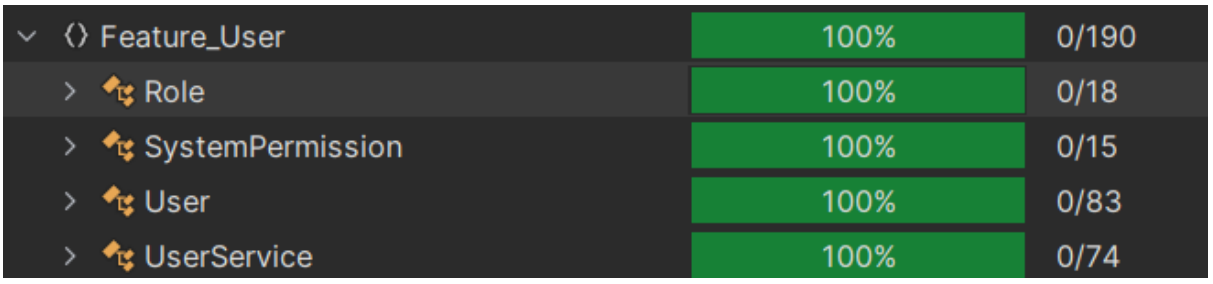
Se siguió la técnica de TDD, y la funcionalidad fue testeada en su totalidad, cubriendo casos borde, casos de error y casos exitosos. Esto se refleja claramente en la cobertura de esta funcionalidad. Aunque el requerimiento menciona "administrador", no hacemos distinción entre Admin y cualquier otro User a nivel de clases. La diferencia entre ellos radica en el atributo Role, por lo que, además de los commits y la cobertura del método Add, también incluimos la cobertura del método AddRole.

Commits:



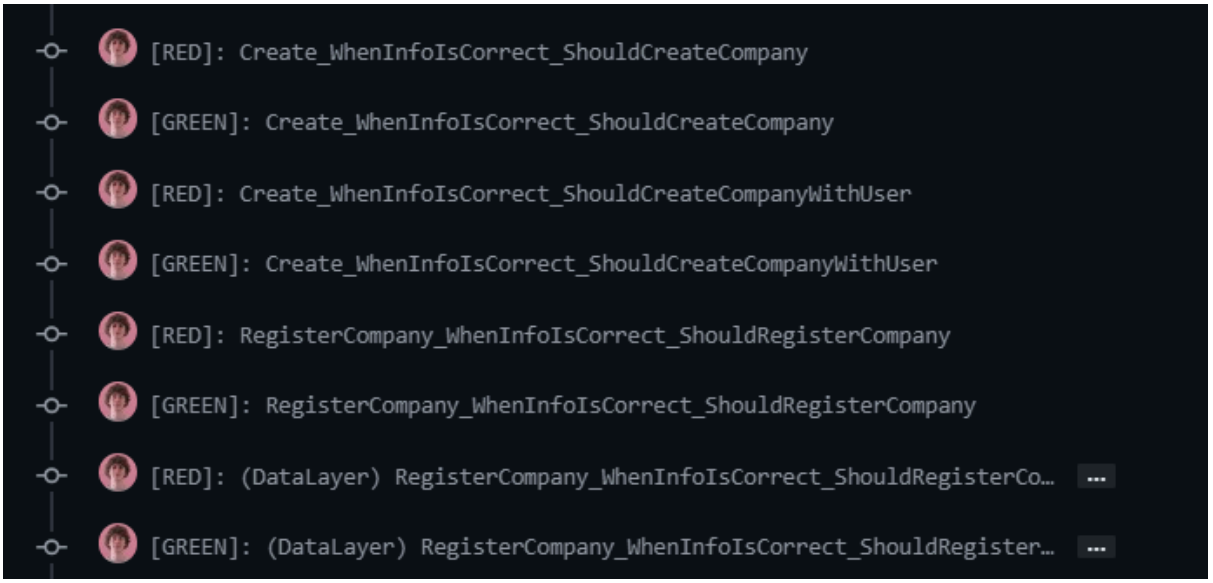


Cobertura:



2.2 Creación de una empresa

Commits:

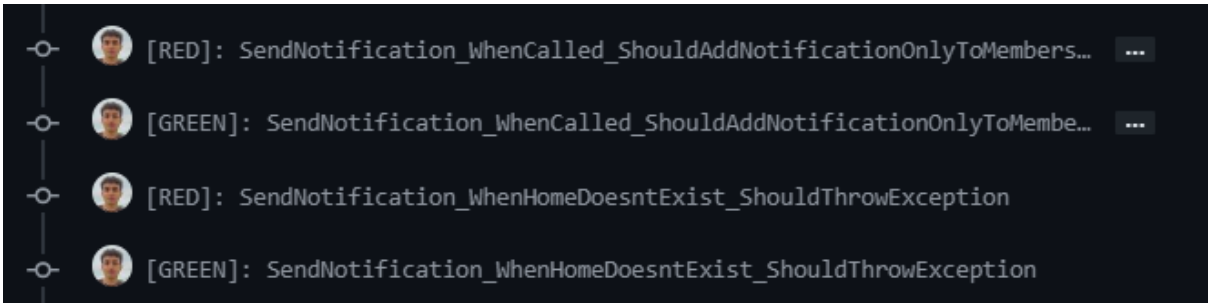


Cobertura:

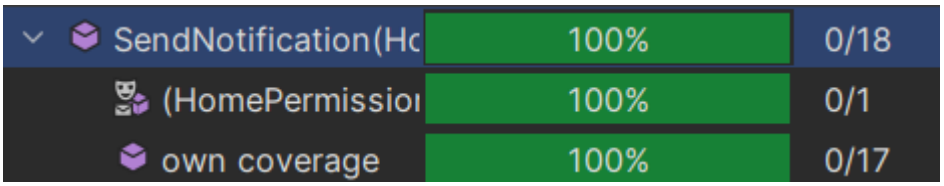
2.3 Detección de movimiento

Cuando un sensor detecta movimiento y envía la información al servidor, lo que hacemos es notificar a los miembros del hogar en el que se produjo el evento. Solo aquellos miembros con los permisos adecuados pueden recibir las notificaciones. Por esta razón, hemos incluido la cobertura y los commits del método SendNotifications, que gestiona este proceso.

Commits:

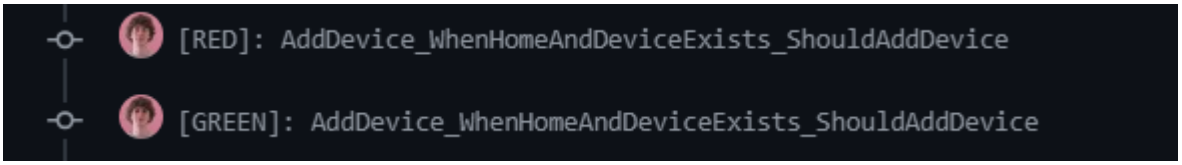


Cobertura:



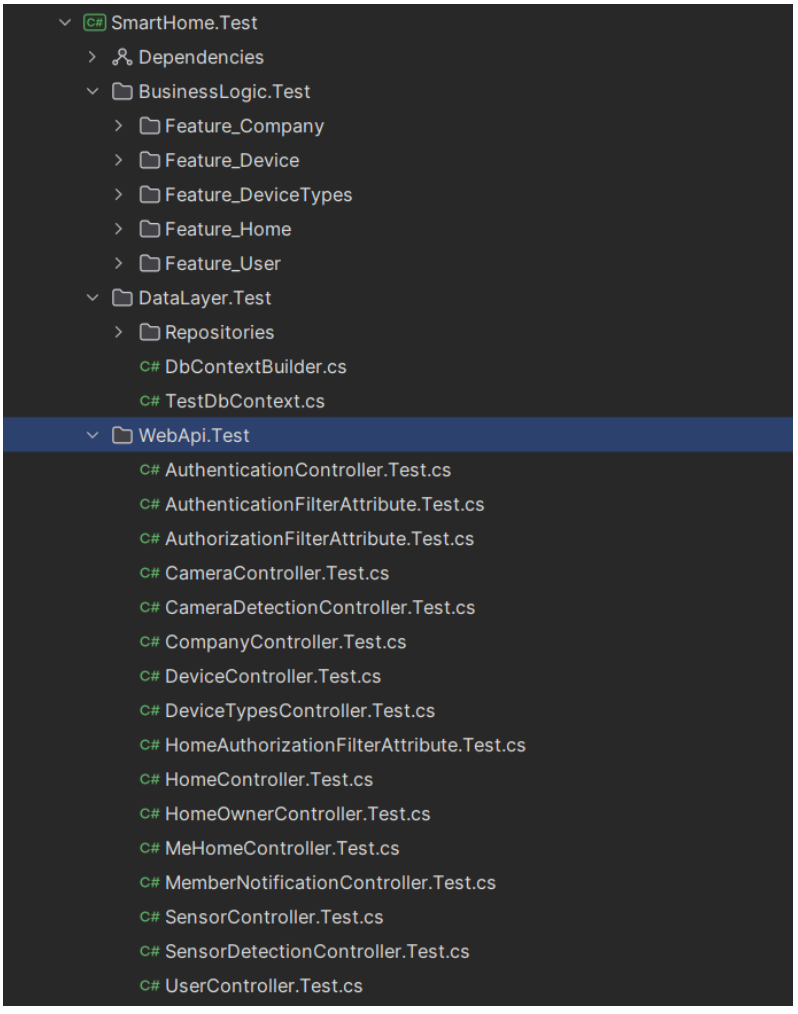
2.4 Asociar dispositivos al hogar

Commits:



Cobertura:

3. Estructura del proyecto de tests

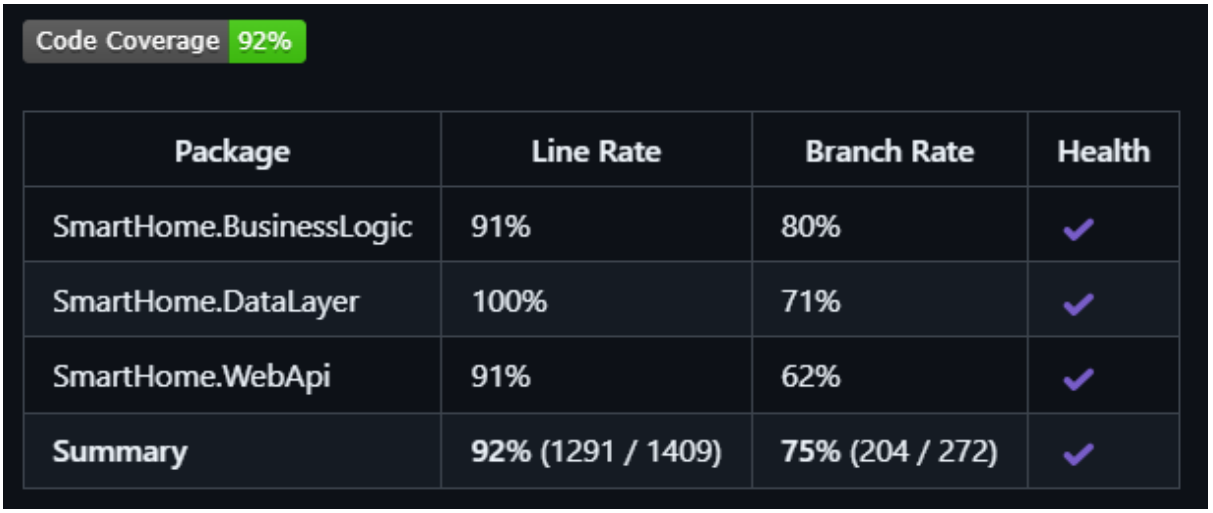


La estructura del proyecto de pruebas se organizó de manera que refleje los proyectos de desarrollo, separando los tests en tres directorios principales: BusinessLogic, DataLayer y WebApi. Dentro de BusinessLogic.Test, cada "Feature" cuenta con su propio directorio,

donde se dividen las pruebas en diferentes clases de test (Test Class), separando los tests de las clases de dominio de los tests de los servicios.

El directorio de DataLayer.Test está enfocado en las pruebas de los repositorios, además de contener la configuración del contexto de pruebas para garantizar un entorno adecuado para los tests. Finalmente, en WebApi.Test, únicamente se realizan pruebas sobre los Controllers y Filters, dado que estos elementos están relacionados directamente con la API y su interacción con los demás componentes. Este esquema proporciona una clara separación de responsabilidades y facilita el mantenimiento y escalabilidad del proyecto de pruebas.

4. Cobertura del proyecto



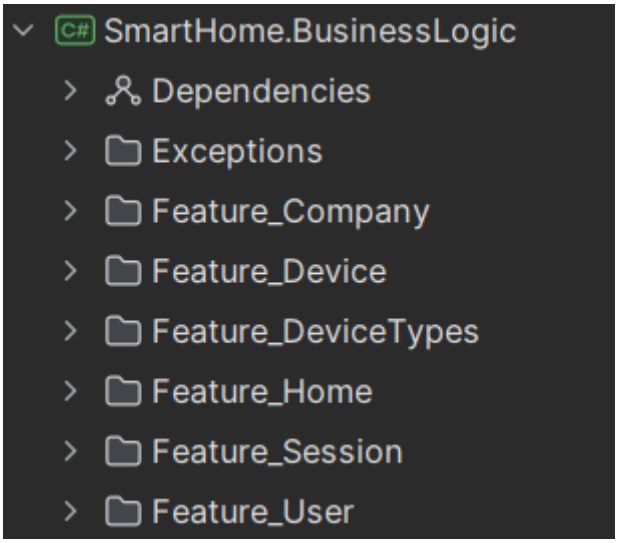
No se logró alcanzar el 100% de cobertura de pruebas debido a la presencia de shadow properties en las clases del dominio, las cuales se utilizan para facilitar el mapeo de las entidades y sus relaciones con Entity Framework Core. Estas propiedades no están directamente expuestas en el código, lo que dificulta su inclusión en las pruebas unitarias.

Asimismo, durante algunos refactorings, se añadieron validaciones para verificar que los parámetros ingresados a los métodos no sean nulos, pero estas validaciones no fueron incluidas en las pruebas unitarias, lo que impacta en la cobertura.

Sin embargo, todos los métodos y atributos accesibles fueron debidamente testeados siguiendo la metodología TDD, asegurando que la lógica principal del dominio y los servicios asociados funcionen correctamente dentro de lo que se puede probar.

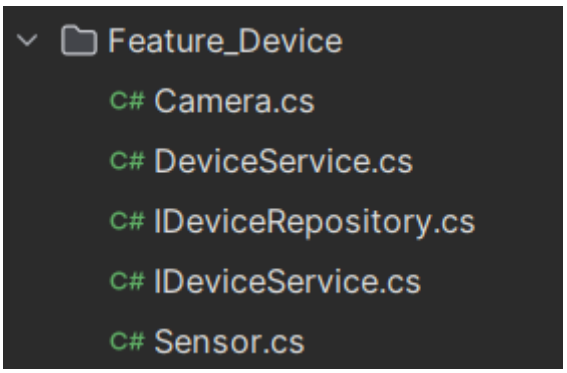
5. Ejemplos de la aplicación de Clean Code

5.1 Alta Cohesión en BusinessLogic



Debido a la estructura de BusinessLogic que utilizamos, logramos una alta cohesión dentro de cada uno de los directorios que la conforman. En lugar de separar las clases del dominio y los servicios en diferentes módulos, agrupamos todo lo relacionado a una Feature específica en un solo directorio. Esto significa que las clases dentro de cada directorio están altamente cohesionadas, ya que todas cumplen un propósito común y se relacionan directamente con la misma funcionalidad del sistema. Un buen ejemplo de esta organización es el caso de Feature_Device, donde las clases de dominio y los servicios trabajan juntos de manera coherente dentro del mismo espacio, simplificando su gestión y mantenimiento.

Este enfoque está alineado con los principios de Clean Code, donde se promueve la cohesión y se busca que cada módulo o componente tenga una única responsabilidad bien definida. Al agrupar las funcionalidades relacionadas en un solo lugar, reducimos la complejidad y facilitamos la comprensión del código.



5.2 Los métodos tienen una única responsabilidad

Los métodos de nuestra aplicación están diseñados para realizar una única tarea específica, y esa tarea se lleva a cabo de manera eficiente y correcta. Esto sigue los principios de Clean Code, ya que una de las reglas fundamentales de este enfoque es que cada función o método debe tener una sola responsabilidad bien definida. Cumplir con este principio de Single Responsibility garantiza que el código sea más claro, fácil de entender y de mantener.

Cuando un método se enfoca en hacer una única cosa, se evita la complejidad innecesaria, lo que facilita su prueba y depuración. Además, al mantener los métodos simples y directos, se reduce el riesgo de introducir errores, ya que cada componente del sistema tiene un propósito claro y no interfiere con otros aspectos del código. Esto también promueve la reutilización, ya que los métodos especializados y enfocados pueden ser más fácilmente utilizados en diferentes partes del sistema.

Ejemplo de un método que cumple SRP:

```
2+4 usages   2 tests OK   Matias +2
public void Add(Sensor newSensor)
{
    if (IsDeviceDuplicated(newSensor))
    {
        throw new ArgumentException( message: "Model number already in use");
    }

    deviceRepository.Add(newSensor);
}
```

6. Declaracion de autoria

Nosotros, Sebastian Vega, Alejo Fraga y Matias Corvetto, 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 realizamos el Primer Obligatorio de Diseño de Aplicaciones 2;

• 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.

Matias Corvetto

8/10/24