

UNIVERSIDAD ORT URUGUAY

Facultad de Ingeniería

Obligatorio 1 Diseño de Aplicaciones 2

Entregado como requisito para la obtención del
título de Ingeniero en Sistemas

Martín Slepian - 266959

Leopoldo Perez - 257341

Eric Poplawski - 258327

Profesores: Nicolás Fierro, Alexander Wieler, Sofia
Duclos

2024

Link al repositorio: <https://github.com/IngSoft-DA2/257341-266959-258327>

Declaración de autoría

Nosotros, Martín Slepian, Leopoldo Perez y Eric Poplawski declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

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

Martín Slepian

7/10/2024

Leopoldo Perez

7/10/2024

Eric Poplawski

7/10/2024

Abstract

Este trabajo presenta el desarrollo de un sistema de software utilizando las metodologías Clean Code y TDD, con el objetivo de lograr un código altamente legible, mantenible y robusto. El proyecto fue implementado en C#, empleando entornos como Visual Studio y Rider, con un enfoque centrado en garantizar la calidad desde el inicio del desarrollo mediante pruebas automatizadas.

El uso de TDD fue clave para asegurar la detección temprana de errores y el desarrollo guiado por pruebas. En cada ciclo de desarrollo, se aplicaron las fases RED, GREEN y REFACTOR asegurando que cada funcionalidad fuera probada exhaustivamente antes de su implementación definitiva. Como resultado, se logró una cobertura de pruebas del 99% en promedio, destacándose la cobertura total en los controladores (100%), DataAccess (100%) y servicios (93%). Este enfoque proporcionó una base sólida para la evolución del sistema, permitiendo agregar nuevas funcionalidades sin comprometer la estabilidad del código existente.

En paralelo, se aplicaron rigurosamente los principios de Clean Code, promoviendo la modularidad, reutilización de componentes y nomenclatura clara. El uso de GitFlow facilitó un control eficiente de versiones y una colaboración ágil, asegurando que cada rama de desarrollo estuviera correctamente gestionada y revisada antes de su integración.

En nuestra opinión, el proyecto no solo cumplió con los requerimientos funcionales establecidos, sino que también estableció un estándar elevado en términos de calidad del código. Las prácticas de Clean Code, junto con la implementación de TDD, garantizan que el sistema sea fácilmente mantenible y extensible a largo plazo, minimizando el riesgo de errores y facilitando futuras mejoras.

En conclusión, este trabajo demuestra que la combinación de TDD y Clean Code no solo permite entregar software funcional de alta calidad, sino que también crea un entorno de desarrollo ágil y sostenible, preparado para afrontar nuevos desafíos y escalabilidad sin comprometer su integridad.

Índice

Declaración de autoría.....	2
Abstract.....	3
Índice.....	4
Evidencia de Clean Code y de la Aplicación de TDD.....	5
Principios del TDD.....	5
Reglas fundamentales del TDD.....	5
Estrategia de TDD (outside-in).....	6
Principios FIRST en las Pruebas Unitarias.....	7
GitFlow en el Desarrollo del Proyecto.....	8
Prácticas de Clean Code Implementadas.....	8
Nombres claros y descriptivos.....	8
Tamaño de los métodos.....	9
Código modular y reutilizable.....	9
Ejecución de Pruebas Unitarias y de Integración.....	10
Evidencia del ciclo TDD en funcionalidades priorizadas.....	10
Mantenimiento de cuentas de Administrador.....	10
Creación de una empresa.....	12
Acciones Soportadas/Detección de Movimiento.....	14
Cobertura de Pruebas.....	15
Conclusión.....	16

Evidencia de Clean Code y de la Aplicación de TDD

Principios del TDD

Test Driven Development (TDD) es una metodología que establece que antes de implementar una funcionalidad, se deben escribir las pruebas correspondientes. Esto asegura que el código sea probado de manera continua, evitando regresiones y asegurando su correcto funcionamiento.

El ciclo de RED-GREEN-REFACTOR es central en este enfoque:

1. **RED:** Se escribe una prueba que falla, ya que la funcionalidad no ha sido implementada.
2. **GREEN:** Se implementa el código mínimo necesario para que la prueba pase.
3. **REFACTOR:** Se refactoriza el código para mejorar su calidad, manteniendo su funcionalidad.

Este ciclo se aplicó de manera consistente en todas las funcionalidades, desde los controladores hasta los servicios. Por ejemplo, en algunos métodos olvidamos incluir validaciones para que ciertos atributos no fueran vacíos. Las pruebas detectaron estas omisiones al no lanzar la excepción esperada, lo que nos permitió corregir el método de inmediato.

Reglas fundamentales del TDD

1. No escribir código sin una prueba fallida
2. Escribir el código mínimo para que la prueba falle
3. Escribir el código mínimo para que la prueba pase

Por ejemplo, en el método `RegisterSecurityCamera`, la primera prueba que escribimos fue

`RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException`. En este caso, el objetivo era validar que se lanzara una excepción si la solicitud era nula.

Primero implementamos el código para que esta prueba fallara.

```
[TestMethod]
1 test OK Martin Slepian +1
public void RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException()
{
    var act :Func<RegisterSecurityCameraResponse> = () => _controller.RegisterSecurityCamera(request: null);

    act.Should().Throw<ControllerException>().WithMessage( expectedWildcardPattern: "Request cannot be null");
}
```

```
RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException [293 ms] Expected a <Domain.Exceptions.ControllerException> to be thrown, but found <System.NullReferenceException>.
Expected a <Domain.Exceptions.ControllerException> to be thrown, but found <System.NullReferenceException> Create breakpoint >:
System.NullReferenceException Create breakpoint : Object reference not set to an instance of an object.
at smarthome.WebApi.Controllers.DeviceController.RegisterSecurityCamera(RegisterSecurityCameraRequest request) in
C:\Users\martu\OneDrive\Escritorio\257341-266959-258327\SmartHome\smarthome.WebApi\Controllers\DeviceController.cs:line 29
at smarthome.UnitTests.DeviceControllerTest.<RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException>b__4_0() in
C:\Users\martu\OneDrive\Escritorio\257341-266959-258327\SmartHome\smarthome.UnitTests\DeviceControllerTest.cs:line 56
at FluentAssertions.Specialized.FunctionAssertions`1.InvokeSubject()
at FluentAssertions.Specialized.DelegateAssertions`2.InvokeSubjectWithInterception().
at FluentAssertions.Execution.LateBoundTestFramework.Throw(String message)
at FluentAssertions.Execution.TestFrameworkProvider.Throw(String message)
at FluentAssertions.Execution.DefaultAssertionStrategy.HandleFailure(String message)
at FluentAssertions.Execution.AssertionScope.FailWith(Func`1 failReasonFunc)
at FluentAssertions.Execution.AssertionScope.FailWith(Func`1 failReasonFunc)
at FluentAssertions.Execution.AssertionScope.FailWith(String message, Object[] args)
at FluentAssertions.Specialized.DelegateAssertionsBase`2.ThrowInternal[TException](Exception exception, String because, Object[] becauseArgs)
at FluentAssertions.Specialized.DelegateAssertions`2.Throw[TException](String because, Object[] becauseArgs)
at smarthome.UnitTests.DeviceControllerTest.RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException() in
C:\Users\martu\OneDrive\Escritorio\257341-266959-258327\SmartHome\smarthome.UnitTests\DeviceControllerTest.cs:line 58
at System.RuntimeMethodHandle.InvokeMethod(Object target, Void** arguments, Signature sig, Boolean isConstructor)
at System.Reflection.MethodBaseInvoker.InvokeWithNoArgs(Object obj, BindingFlags invokeAttr)
```

Ahora sí, pusimos el siguiente código mínimo para que la prueba pase.

```
if (request == null)
{
    throw new ControllerException( message: "Request cannot be null");
}
```

```
[TestMethod]
1 test OK Martin Slepian +1
public void RegisterSecurityCamera_WhenRequestIsNull_ShouldThrowAnException()
{
    var act :Func<RegisterSecurityCameraResponse> = () => _controller.RegisterSecurityCamera(request: null);

    act.Should().Throw<ControllerException>().WithMessage( expectedWildcardPattern: "Request cannot be null");
}
```

Estrategia de TDD (outside-in)

En este proyecto, se utilizó la estrategia de TDD outside-in. Esto significa que primero se hicieron las pruebas para los controladores, y luego se desarrolló la lógica interna en los servicios y repositorios. De esta forma, los requisitos se definen claramente desde la forma en que el sistema interactúa con el usuario, asegurando que la lógica interna cumpla con esas expectativas. Además, este enfoque ayuda a

crear un sistema más fácil de entender y mantener, ya que cada parte se prueba desde el inicio, garantizando que todo funcione correctamente.

Principios FIRST en las Pruebas Unitarias

Las pruebas en este proyecto siguen el principio FIRST:

- **Fast (rápidas):** Las pruebas se ejecutan rápidamente, permitiendo un desarrollo ágil.
- **Independent (independientes):** Las pruebas no dependen unas de otras.
- **Repeatable (repetibles):** Siempre producen el mismo resultado independientemente del entorno.
- **Self-Validating (auto-validables):** No requieren intervención manual para determinar si pasaron o fallaron.
- **Timely (oportunas):** Se escriben antes de implementar el código que prueban.

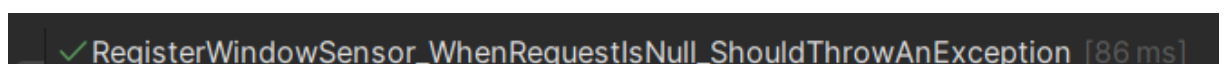


```
[TestMethod]
public void RegisterWindowSensor_WhenRequestIsNull_ShouldThrowAnException()
{
    var act :Func<RegisterWindowSensorResponse> = () => _controller.RegisterWindowSensor(request:null);

    act.Should().Throw<ControllerException>().WithMessage(expectedWildcardPattern: "Request cannot be null");
}
```

La foto de arriba es un ejemplo de un MSTest usando el principio FIRST.

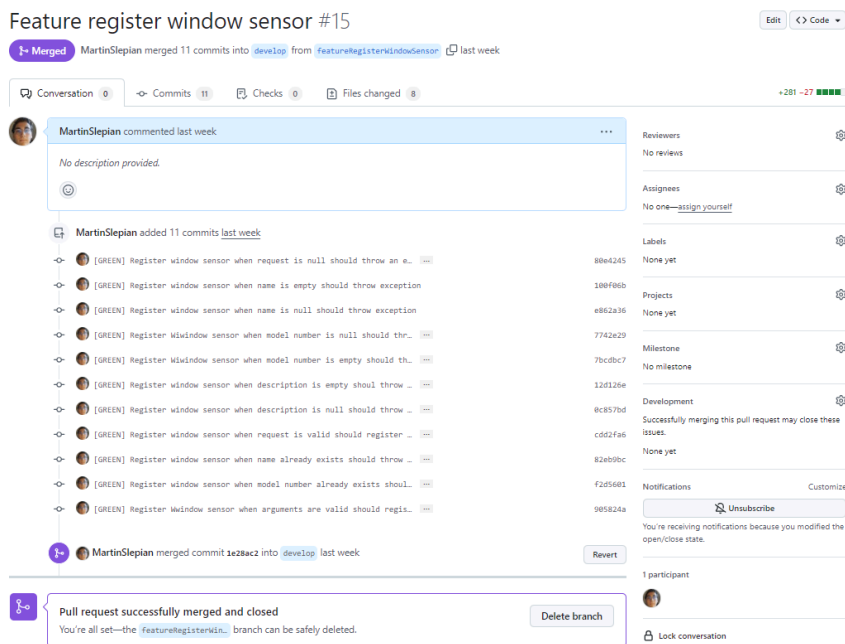
La prueba se ejecuta rápidamente (demora 86ms), esta prueba no depende de ninguna otra, siempre que la ejecutamos da el mismo resultado, se valida por sí sola (no tenemos que analizar ningún dato) y se escribió antes de hacer el código.



```
✓ RegisterWindowSensor_WhenRequestIsNull_ShouldThrowAnException [86 ms]
```

GitFlow en el Desarrollo del Proyecto

Para la gestión de versiones y la colaboración en equipo, utilizamos GitFlow, lo que nos permitió mantener una estructura clara en las ramas de desarrollo. Cada funcionalidad nueva fue desarrollada en una feature branch separada y, una vez completada y probada, se integró a la rama develop. Este proceso garantizó un flujo de trabajo organizado, facilitando la colaboración y evitando conflictos de código durante el desarrollo.



En esta imagen se puede observar el merge con Pull Request de la rama featurerRegisterWindowSensor con la rama develop.

Prácticas de Clean Code Implementadas

Nombres claros y descriptivos

Los nombres de las clases, métodos y variables fueron cuidadosamente seleccionados para reflejar su propósito de manera precisa. Evitamos abreviaciones y nombres ambiguos, siguiendo el principio de Clean Code de que los nombres deben comunicar la intención.

```
2 usages 1 implementation ericpoplawski
Device RegisterSecurityCamera(RegisterSecurityCameraArguments arguments, User userLoggedIn);
2 usages 1 implementation ericpoplawski
Device RegisterWindowSensor(RegisterWindowSensorArguments arguments, User userLoggedIn);
3 usages 1 implementation Martin Slepian
Task<List<Device>> GetDevices(int pageNumber, int pageSize, string deviceName, string companyName, string modelNumber,
    string mainPicture);
3 usages 1 implementation Martin Slepian
List<String> GetSupportedTypes();
```


Tamaño de los métodos

Nos aseguramos de que cada método realizará una única tarea. Si un método requería más de una responsabilidad, lo fragmentamos en otros más pequeños. Un ejemplo es el método `CreateCompany`, que delega la obtención del usuario logueado a `GetUserLogged`.

```
[HttpPost]
[Route(template: "api/company")]
[AuthorizationFilter(permission: "CreateCompany")]
6 usages 6 tests OK Martin Slepian +1
public CreateCompanyResponse CreateCompany(CreateCompanyRequest request)
{
    if (request == null)
    {
        throw new ControllerException(message: "Request cannot be null");
    }
    var userLogged = GetUserLogged();
    CreateCompanyArguments arguments = new CreateCompanyArguments(request.Name, request.Logo, request.RUT);
    Company companyCreated = _service.CreateCompany(arguments, userLogged);
    CreateCompanyResponse response = new CreateCompanyResponse(companyCreated);
    return response;
}
```

Código modular y reutilizable

Dividimos el código en componentes reutilizables, como servicios y repositorios. Esto facilita la reutilización del código y evita la duplicación, alineándose con el principio DRY (Don't Repeat Yourself).

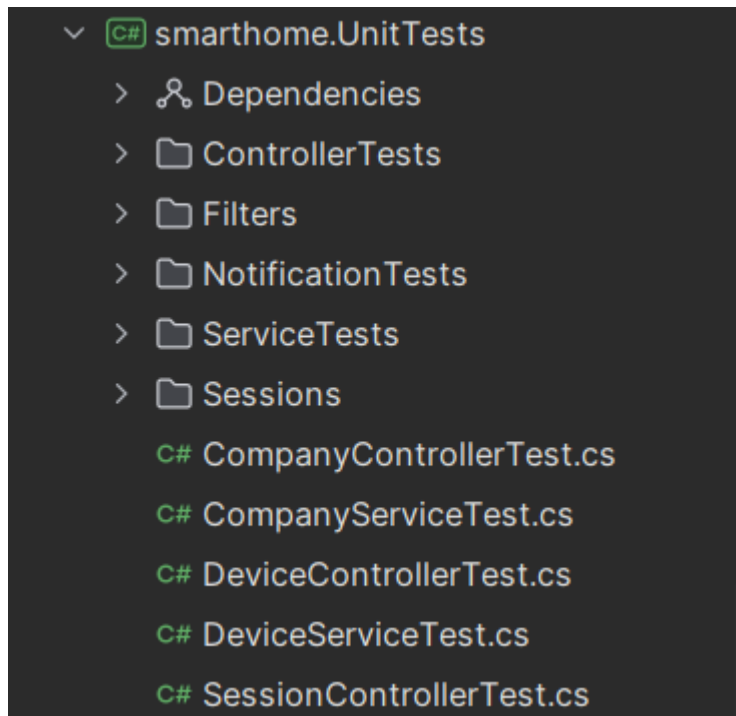
```
[HttpGet]
[Route(template: "api/device/supportedTypes")]
1 usage 1 test OK* Martin Slepian
public List<GetSupportedTypesResponse> GetSupportedTypes()
{
    List<String> supportedTypes = _service.GetSupportedTypes();
    List<GetSupportedTypesResponse> responses = new List<GetSupportedTypesResponse>();

    foreach (var type:string in supportedTypes)
    {
        responses.Add(item: new GetSupportedTypesResponse(type));
    }

    return responses;
}
```























Ejecución de Pruebas Unitarias y de Integración

Todas las pruebas se realizaron en un proyecto separado de la solución principal, llamado smartHome.UnitTests. Se probaron tanto casos exitosos como fallidos para asegurar que el código respondiera correctamente ante diferentes escenarios.



Evidencia del ciclo TDD en funcionalidades priorizadas

Mantenimiento de cuentas de Administrador

[Red]: Create, when request is correct, should create administrator ericoplowski committed 2 weeks ago	e8db72b	 
[Green]: Create, when password is null, should throw exception ericoplowski committed 2 weeks ago	19eab75	 
[Red]: Create, when password is null, should throw exception ericoplowski committed 2 weeks ago	b8934f8	 
[Green]: Create, when email is null, should throw exception ericoplowski committed 2 weeks ago	37b4c77	 
[Red]: Create, when email is null, should throw exception ericoplowski committed 2 weeks ago	fc3a751	 
[Green]: Create, when last name is null, should throw exception ericoplowski committed 2 weeks ago	577bc52	 
[Red]: Create, when last name is null, should throw exception ericoplowski committed 2 weeks ago	9815c27	 
[Green]: Create, when first name is null, should throw exception ericoplowski committed 2 weeks ago	ebe3b79	 
[Red]: Create, when first name is null, should throw exception ericoplowski committed 2 weeks ago	1869e56	 
[Green]: Create, when request is null, should throw exception ericoplowski committed 2 weeks ago	b4c5647	 
[Red]: Create, when request is null, should throw exception ericoplowski committed 2 weeks ago	c42ba48	 

[GREEN] Add Administrator when email does not exist should return user MartinSlepian committed 2 weeks ago	806cbf9		<>
Add Administrator when email does not exist should return user MartinSlepian committed 2 weeks ago	2fd996e		<>
[GREEN]: Add Administrator when Email Already exists should throw exception MartinSlepian committed 2 weeks ago	18f9fdf		<>
[RED]: Add Administrator when Email Already exists should throw exception MartinSlepian committed 2 weeks ago	2905847		<>
[Green]: Create, when request is correct, should create administrator ericoplowski committed 2 weeks ago	5f0c58a		<>
[REFACTOR - GREEN]: Create, when request is valid should create administrator ericoplowski committed 2 weeks ago	c07f279		<>
[RED - GREEN]: Create, when password is empty, should throw exception ericoplowski committed 2 weeks ago	f75c460		<>
[GREEN]: Create, when password is null, should throw exception ericoplowski committed 2 weeks ago	c0443cc		<>
[REFACTOR]: Create, when password is null, should throw exception ericoplowski committed 2 weeks ago	11eb757		<>
[RED - GREEN]: Create, when email is null, should throw exception ericoplowski committed 2 weeks ago	6a9ed10		<>
[GREEN]: Create, when email is null, should throw exception ericoplowski committed 2 weeks ago	d7053e4		<>
[RED]: Create, when email is null, should throw exception ericoplowski committed 2 weeks ago	4d041e7		<>
[RED- GREEN]: Create, when last name is null, should throw exception ericoplowski committed 2 weeks ago	3c9bcfc		<>
[GREEN]: Create, when last name is null, should throw exception ericoplowski committed 2 weeks ago	34af656		<>
[REFACTOR]: Create, when last name is null should throw exception ericoplowski committed 2 weeks ago	be62497		<>

En la foto se observan todos los commits hechos para la funcionalidad de crear un administrador. Aquí se pueden ver las 3 fases principales del TDD ya explicadas más arriba en el documento. Estas fases son RED, GREEN y REFACTOR.

AddAdministrator(CreateUserArguments)	100%	0/11
CreateAdministrator(CreateAdministratorRequest)	100%	0/8

En la foto se observa el porcentaje de la cobertura de líneas de código de la funcionalidad de crear un administrador tanto en el service (1era foto) como en el controller (2nda foto).

Taking away blankspaces ericoplowski committed last week	322484d		<>
[GREEN]: Delete administrator by id, when user does not have role administrator, should throw exception ericoplowski committed last week	ccb3d41		<>
[RED]: Delete administrator by id, when user does not have role administrator, should throw exception ericoplowski committed last week	68ee4a		<>
[GREEN]: Delete administrator by id, when user exists should delete user ericoplowski committed last week	46eddce		<>
[RED]: Delete administrator by id, when user exists should delete user ericoplowski committed last week	51e7e41		<>
[RED]: Delete administrator by id, when user does not exist should throw exception ericoplowski committed last week	38a1d7a		<>
[RED]: Delete administrator by id, when user does not exist, should throw exception ericoplowski committed last week	73dad81		<>
[GREEN]: Delete administrator by id, when id is valid should delete administrator ericoplowski committed last week	336cf98		<>
[RED]: Delete administrator by id, when id is valid should delete administrator ericoplowski committed last week	f3067c9		<>
[GREEN]: Delete administrator by id, when id is invalid guid should throw exception ericoplowski committed last week	a714f73		<>
[RED]: Delete administrator by id, when id is invalid guid should throw exception ericoplowski committed last week	9a364eb		<>































En la foto se observan todos los commits hechos para la funcionalidad de borrar un administrador. Aquí se pueden ver 2 de las 3 fases principales del TDD ya explicadas más arriba en el documento. Estas fases son RED y GREEN. No hay Refactors. Esto se debe a que no creemos que haya una forma más prolija de hacerlo.

DeleteAdministratorById(string)	100%	0/10
DeleteAdministratorById(string)	100%	0/6


En la foto se observa el porcentaje de la cobertura de líneas de código de la funcionalidad de borrar un administrador tanto en el service (1era foto) como en el controller (2nda foto).

Creación de una empresa

[GREEN] Create company when request is valid, should create company MartinSlepian committed 2 weeks ago	645bcce		<>
[RED] Create company when request is valid, should create company MartinSlepian committed 2 weeks ago	33c751b		<>















[GREEN] Create when name is empty, should throw an exception MartinSlepian committed 2 weeks ago	f683269		
[RED] Create when name is empty, should throw an exception MartinSlepian committed 2 weeks ago	4852965		
[GREEN] Create when Name already exists should throw an exception MartinSlepian committed 2 weeks ago	c134fb1		
[RED] Create when Name already exists should throw an exception MartinSlepian committed 2 weeks ago	fdb2b3b		
[REFACTOR] Put name validation, logo validation and RUT validation in separate functions MartinSlepian committed 2 weeks ago	a943736		
[GREEN] Create when Logo is empty should throw exception MartinSlepian committed 2 weeks ago	704ed8f		
[RED] Create when Logo is empty should throw exception MartinSlepian committed 2 weeks ago	d0ac44c		
[GREEN] Create when Logo already exists should throw exception MartinSlepian committed 2 weeks ago	a2388f6		
[RED] Create when Logo already exists should throw exception MartinSlepian committed 2 weeks ago	9dd8b8b		
[GREEN] Create when RUT is empty should throw exception MartinSlepian committed 2 weeks ago	9064c2e		
[RED] Create when RUT is empty should throw exception MartinSlepian committed 2 weeks ago	9571e12		
[GREEN] Create when RUT Already Exists Should throw exception MartinSlepian committed 2 weeks ago	2c8b45c		
[RED] Create when RUT Already Exists Should throw exception MartinSlepian committed 2 weeks ago	fa0a042		
[REFACTOR] Change the language of some methods of CompanyService MartinSlepian committed 2 weeks ago	cbdcdc2		
[REFACTOR] Change the method CreateCompany in CompanyController MartinSlepian committed 2 weeks ago	ee0b875		

En la foto se observan todos los commits hechos para la funcionalidad de crear una compañía. Aquí se pueden ver las 3 fases principales del TDD ya explicadas más arriba en el documento. Estas fases son RED, GREEN y REFACTOR.

 CreateCompany(CreateCompanyArguments,User)	100%	0/9
 CreateCompany(CreateCompanyRequest)	100%	0/10

En la foto se observa el porcentaje de la cobertura de líneas de código de la funcionalidad de crear una compañía tanto en el service (1era foto) como en el controller (2nda foto).

Acciones Soportadas/Detección de Movimiento

[GREEN]: Get hardware device by, when id match a device should return device ericoplowski committed 3 days ago	ecc18de		<>
[RED]: Get hardware device by id, when id match a device should return device ericoplowski committed 3 days ago	4a252bf		<>
[GREEN]: Get hardware device by id, when id does not match any device, should throw exception ericoplowski committed 3 days ago	8b6171e		<>
[RED]: Get hardware device by id, when id does not match any device should throw exception ericoplowski committed 3 days ago	46f9c2b		<>
[GREEN]: Create motion detection notification, when hardware device is not connected should throw exception ericoplowski committed 3 days ago	66d778e		<>
[RED]: Create motion detection notification, when hardware device is not connected should throw exception ericoplowski committed 3 days ago	154048		<>
[GREEN]: Create motion detection notification, when hardware device is not a security camera should throw exception ericoplowski committed 3 days ago	9a08a87		<>
[RED]: Create motion detection notification, when hardware device is not a security camera should throw exception ericoplowski committed 3 days ago	9a2d8c6		<>
[GREEN]: Create motion detection notification, when hardware device is null should throw exception ericoplowski committed 3 days ago	750a096		<>
[RED]: Create motion detection notification, when hardware device is null should throw exception ericoplowski committed 3 days ago	c0bd572		<>
[GREEN]: Execute motion detection notification, should throw ok ericoplowski committed 3 days ago	3c728fe		<>
[RED]: Execute motion detection notification, should return ok ericoplowski committed 3 days ago	703d470		<>
[GREEN]: Create motion detection notification when checks are passed should return response ericoplowski committed 3 days ago	fc83532		<>
[RED]: Create motion detection notification when checks are passed should return response ericoplowski committed 3 days ago	6e275ac		<>

En la foto se observan todos los commits hechos para la funcionalidad de detectar movimiento y generar una notificación para los miembros del hogar configurados.

Aquí se pueden ver 2 de las 3 fases principales del TDD ya explicadas más arriba en el documento. Estas fases son RED y GREEN. No hay Refactors. Esto se debe a que no creemos que haya una forma más prolija de hacerlo.

CreateMotionDetectionNotification(string)	100%	0/35
ExecuteMotionDetectionNotification(string)	100%	0/4

En la foto se observa el porcentaje de la cobertura de líneas de código de la funcionalidad de detectar movimiento y generar una notificación para los miembros del hogar configurados tanto en el service (1era foto) como en el controller (2nda foto). Solo mostramos la cobertura de los 2 métodos principales.

Asociar dispositivos al hogar

[REFACTOR] Create_WhenStreetsEmpty_ShouldThrowAnException Leopoldo4703 committed 5 days ago	f3f6a98	📄 <>
Commits on Sep 26, 2024		
[GREEN] Add device to home, when arguments are not null, should add device to home Leopoldo4703 committed last week	98be6f8	📄 <>
[GREEN] Get user home permissions, when arguments are not null, should return permissions Leopoldo4703 committed last week	e242c23	📄 <>
Commits on Sep 25, 2024		
[GREEN] Add user to home, when arguments are not null, should add user to home Leopoldo4703 committed last week	c6ae128	📄 <>
[GREEN] Add device to home, when user has permission, should add device to home Leopoldo4703 committed last week	6e36439	📄 <>
[RED] Add device to home, when user has permission, should add device to home Leopoldo4703 committed last week	8098c1a	📄 <>
[GREEN] Add device to home, when user does not have permission, should throw an exception Leopoldo4703 committed last week	5852849	📄 <>
[RED] Add device to home, when user does not have permission, should throw an exception Leopoldo4703 committed last week	85e3686	📄 <>
[GREEN] Add device to home, when request is null, should throw an exception Leopoldo4703 committed last week	12449bb	📄 <>
[RED] Add device to home, when request is null, should throw an exemption Leopoldo4703 committed last week	1568388	📄 <>

En la foto se observan todos los commits hechos para la funcionalidad de asociar dispositivos a un hogar. Aquí se pueden ver las 3 fases principales del TDD ya explicadas más arriba en el documento. Estas fases son RED, GREEN y REFACTOR.

🔍 AddDeviceToHome(AddDeviceToHomeArguments)	100%	0/7
> 🔍 AddDeviceToHome(string, AddDeviceToHomeRequest)	100%	0/12

En la foto se observa el porcentaje de la cobertura de líneas de código de la funcionalidad de asociar dispositivos a un hogar tanto en el service (1era foto) como en el controller (2nda foto).

Cobertura de Pruebas

Alcanzamos una cobertura de código entre el 90% y el 100%, asegurando que todas las funcionalidades clave estuvieran completamente cubiertas por pruebas. La cobertura incluye tanto pruebas unitarias como de integración.

Total	99%	59/4588
> smarthome.DataAccess	100%	0/89
> smarthome.Domain	99%	1/753
> smarthome.UnitTests	99%	6/2843
> smarthome.WebApi	97%	8/311
> smarthome.BussinessLogic	93%	44/592

Tener en cuenta que excluimos de los tests las migraciones y el program que no se testean.

Conclusión

La implementación de TDD y las prácticas de Clean Code aseguraron un código robusto, mantenible y extensible, con detección temprana de errores gracias a pruebas unitarias e integradas. El diseño modular y desacoplado permite una escalabilidad sencilla, facilitando la integración de nuevas funcionalidades sin comprometer el sistema existente. Además, la alta legibilidad del código garantiza que futuros desarrolladores puedan modificar y expandir el sistema de manera eficiente, asegurando su evolución sostenible y manteniendo su estabilidad a largo plazo.

Referencias Bibliográficas

- [1] Diseño-de-apliaciones-2, "Diseño de aplicaciones 2," [Online]. Available: <https://diseño-de-apliaciones-2.notion.site/Dise-o-de-aplicaciones-2-299517e56d4f4b97ad2e7048610f6e0e>. Accessed on: Oct. 03, 2024.
- [2] SharePoint, "Clase 1 - GIT," [Online]. Available: [Clase 1 - GIT.pptx \(sharepoint.com\)](#). Accessed on: Oct. 03, 2024.
- [3] SharePoint, "Clase 4 - TDD," [Online]. Available: [Clase 4 - TDD.pptx \(sharepoint.com\)](#). Accessed on: Oct. 03, 2024.
- [4] SharePoint, "03.10_TDD," [Online]. Available: [03.10_TDD.pptx \(sharepoint.com\)](#). Accessed on: Oct. 03, 2024.
- [5] SharePoint, "03.20_Refactoring," [Online]. Available: [03.20_Refactoring.pptx \(sharepoint.com\)](#). Accessed on: Oct. 03, 2024.
- [6] SharePoint, "07.00_SOLID," [Online]. Available: [07.00_SOLID.pptx \(sharepoint.com\)](#). Accessed on: Oct. 03, 2024.