

**Universidad ORT Uruguay**

**Facultad de Ingeniería**



*Obligatorio 1 Diseño de Aplicaciones 2*

Enzo Izquierdo (283145)

Manuel Graña (285727)

Martín Salaberry (294238)

[Repositorio](#)

2024

# Índice

<b>Índice</b>	<b>2</b>
<b>Introducción</b>	<b>3</b>
<b>Evidencia de Clean Code</b>	<b>3</b>
<b>Estrategia de TDD</b>	<b>3</b>
Enfoque Seguido	3
Funcionalidades Prioritarias (*)	4
Mantenimiento de cuentas de administrador:	4
Creación de una empresa:	5
Detección de movimiento:	6
Asociar dispositivos al hogar:	6
<b>Evidencia de TDD</b>	<b>7</b>
<b>Cobertura de Pruebas</b>	<b>7</b>

# Evidencia de Clean Code y de la aplicación de TDD

## Introducción

A lo largo del desarrollo de este proyecto, se ha seguido un enfoque que prioriza la claridad y la calidad del código, así como la creación de pruebas automatizadas desde las fases iniciales. En las siguientes secciones, se detallará cómo se aplicaron los principios de Clean Code y cómo se adoptó Test-Driven Development (TDD) para implementar las funcionalidades prioritarias.

El documento incluye ejemplos de código limpio, refactorizaciones clave y evidencia de pruebas unitarias en las que se utilizó TDD, siguiendo un enfoque Outside-In.

## Evidencia de Clean Code

### Principios Seguidos

Estos son algunos de los principios que seguimos a lo largo de nuestro proyecto:

**Uso de interfaces:** Implementamos inyección de dependencias para desacoplar clases y facilitar la prueba y mantenimiento.

```
services.AddScoped<IBusinessRepository, BusinessRepository>();  
services.AddScoped<IOwnedDeviceRepository, OwnedDeviceRepository>();  
services.AddScoped<IAuthService, AuthService>();  
services.AddScoped<IUserService, UserService>();  
services.AddScoped<IHomeOwnerService, HomeOwnerService>();
```

```
[ApiController]  
[Route("homes")]  
[AuthenticationFilter]  
public class HomeController(IHomeOwnerService homeOwnerService) : ControllerBase  
{
```

**Funciones pequeñas:** En los refactors extrajimos funciones para que queden pequeñas.

```
public Guid CreateHome(CreateHomeArgs args)
{
    EnsureCreateHomeModelIsValid(args);
    EnsureAddressIsUnique(args.Address);
    EnsureUserExists(args.HomeOwnerId);
    var user = _userRepository.Get(Guid.Parse(args.HomeOwnerId));
    var home = new Home(user, args.Address, args.Latitude, args.Longitude, args.MaxMembers);
    _homeRepository.Add(home);
    return home.Id;
}

private void EnsureAddressIsUnique(string address)
{
    var home = _homeRepository.GetByAddress(address);
    if (home != null)
    {
        throw new ArgumentException("Address is already in use");
    }
}

private void EnsureUserExists(string homeOwnerId)
{
    if (!_userRepository.Exists(Guid.Parse(homeOwnerId)))
    {
        throw new ArgumentException("User does not exist");
    }
}
```

**Pocos argumentos:** Utilizamos models o Dtos para encapsular los argumentos en structs y así mantener la cantidad de parámetros de las funciones (salvo excepciones).

```
public void Notify(NotificationArgs args)
{
    EnsureOwnedDeviceExists(args.HardwareId);
    var ownedDevice = OwnedDeviceRepository.GetByHardwareId(args.HardwareId);
    EnsureOwnedDeviceIsNotNull(ownedDevice);
    var home = ownedDevice.Home;
    var shouldReceiveNotification = new HomePermission(HomePermission.GetNotifications);
    NotifyUsersWithPermission(args, home, shouldReceiveNotification, ownedDevice);
}
```

```
public struct NotificationArgs
{
    public string HardwareId { get; set; }
    public DateTime Date { get; set; }
    public string Event { get; set; }
}
```

**Evitar comentarios innecesarios:** El código debe explicarse por sí mismo, sin necesidad de comentarios redundantes.

## Estrategia de TDD

### Enfoque Seguido

Durante el desarrollo, adoptamos un enfoque mayoritariamente **Outside-In**. Comenzamos implementando las clases de servicio, encargadas de gestionar la lógica de negocio y comunicarse con los repositorios y clases del dominio. A medida que avanzábamos en las pruebas, fuimos desarrollando el dominio correspondiente. Aplicamos inyección de dependencias en los servicios, lo que nos llevó a crear primero las interfaces de los repositorios, y posteriormente, sus implementaciones. Finalmente, implementamos los controladores, ya que en las primeras fases del proyecto no teníamos claridad sobre su estructura, por lo que decidimos dejarlos para el final.

### Funcionalidades Prioritarias (\*)

#### Mantenimiento de cuentas de administrador:

Primero implementamos los servicios y las clases del dominio, luego los repositorios y por último el endpoint.

([Pull request con la implementación del servicio y repositorio](#))

Empezando por el servicio:

- Create\_WhenAlreadyExists\_ThrowsException
  - [\[red\]](#) [\[green\]](#)
- Create\_WhenArgumentsAreValid\_CreatesAdmin
  - [\[red\]](#) [\[green\]](#): Typo, le pusimos red

- Delete\_WhenDoesNotExist\_ThrowsException
  - [\[red\]](#) [\[green\]](#)
- Delete\_WhenArgumentsAreValid\_DeletesAdmin
  - [\[red\]](#) [\[green\]](#)
- Pruebas de la clase Admin
  - [\[red\]](#) [\[green\]](#)
- Create\_WhenArgumentsHaveEmptyFields\_ThrowsException
  - [\[red\]](#) [\[green\]](#)

## Refactors

- [refactor: Extract admin existence validation in EnsureAdminExists method](#)
- [refactor: Extract username validation in EnsureAdminUsernameIsUnique method](#)
- [refactor: Extract empty string validation into a separate method in AdminService](#)
- [refactor: remove unused mock setup in test](#)

## Formato del mail

- [\[red\]](#) [\[green\]](#) [refactor: fix regions in tests](#)

Luego implementamos las funciones que crean dueños de empresa, y nos dimos cuenta que los administradores y estos comparten atributos, por lo que los centralizamos en una clase user. [refactor: merged Admin and BusinessOwner into user, added role enum as a placeholder](#)

Siguiendo implementamos las pruebas para crear admins ([\[red\]](#) [\[green\]](#)) y para borrar admins ([\[red\]](#)[\[green\]](#)), pruebas a funciones que habíamos trabajado usando mocks anteriormente.

Estas son las primeras iteraciones de las pruebas, no ponemos todos los commits para simplificar la explicación.

Y por último implementamos los endpoints para crear admins ([\[red\]](#) [\[green\]](#)) y para borrar admins ([\[red\]](#)[\[green\]](#)). ([Pull request con la implementación del endpoint](#))

## Creación de una empresa:

En este caso empezamos por el servicio, luego el repositorio y por último el endpoint.

([Pull request con la implementación del servicio y repositorio](#))

Empezando por BusinessOwnerService:

- CreateBusiness\_WhenOwnerExists\_CreatesBusiness
  - [\[red\]](#) [\[green\]](#)
- CreateBusiness\_WhenOwnerAlreadyHasBusiness\_ThrowsException
  - [\[red\]](#) [\[green\]](#)
- CreateBusiness\_WhenOwnerDoesNotExist\_ThrowsException
  - [\[green\]](#) Solo green porque aplicamos implementación obvia
- CreateBusiness\_WhenBusinessRutAlreadyExists\_ThrowsException
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)

Luego seguimos por BusinessRepository:

- Add\_ValidBusiness\_AddsBusiness
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)
- Add\_WhenBusinessAlreadyExists\_ThrowsException
  - [\[red\]](#) [\[green\]](#)

Por último, implementamos el endpoint ([Pull request con la implementación del endpoint](#)):

- CreateBusiness\_WhenCalledWithValidRequest\_ReturnsCreatedResponse
  - [\[red\]](#) [\[green\]](#)
- CreateBusiness\_WhenCalledWithValidRequest\_ReturnsCorrectRut
  - [\[green\]](#) [\[refactor\]](#) Solo green porque aplicamos implementación obvia

## Detección de movimiento:

### [Pull request con los cambios](#)

En este caso, como empezamos por los servicios, ya teníamos el método CreateNotification en NotificationService para crear notificaciones en general, que usaremos más adelante.

Además este utiliza la función Add de NotificationRepository, que ya habíamos implementado y probado.

En la primera iteración agregamos el endpoint, y luego hicimos un refactor extrayendo el método que crea los argumentos de que se le pasan a NotificationService.

- [\[red\]](#) [\[green\]](#) [\[refactor\]](#)

Luego, implementamos el metodo Notify en NotificationService, el cual es utilizado por el método del endpoint.

- Notify\_WhenCalledWithNonExistentDevice\_ShouldThrowArgumentException
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)
- Notify\_WhenCalledWithExistentDevice\_ShouldCreateNotificationForEachUserWithPermissions
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)

## Asociar dispositivos al hogar:

Primero empezamos implementando el servicio HomeOwnerService y las clases del dominio que fueran necesarias, luego el endpoint y por último los métodos del repositorio OwnedDeviceRepository. Hacerlo de esta manera fue posible gracias a los Mocks.

Pruebas implementadas en el servicio ([Pull request con la implementación del servicio](#)):

- AddDevicesToHome\_WhenArgumentsAreValid\_AddsDevice
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)
- AddDevicesToHome\_WhenHomeIdIsNotAGuid\_ThrowsException
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)
- AddDevicesToHome\_WhenDeviceIdIsNotAGuid\_ThrowsException
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#)

Luego implementamos el endpoint ([Pull request con la implementación del endpoint](#)):

- AddDevices\_WhenCalledWithValidRequest\_ReturnsOkResponse
  - [\[red\]](#) [\[green\]](#) [\[refactor\]](#) [\[refactor\]](#)

Implementamos una prueba más en el servicio que se nos había pasado por alto:

- AddDevicesToHome\_WhenAtLeastOneDevicesIsAlreadyAdded\_ThrowsException
  - [\[red\]](#) [\[green\]](#)

Y por último implementamos el repositorio que es usado por el servicio ([Pull request con la implementación del repositorio](#)):

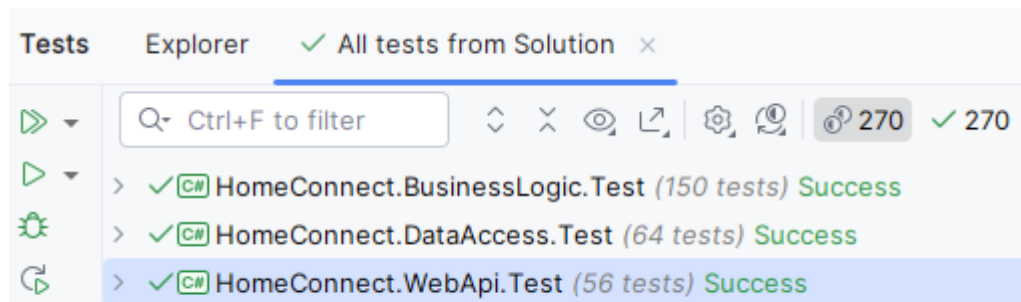
- Add\_WhenOwnedDeviceDoesNotExist\_AddsOwnedDevice
  - [\[green\]](#) Solo green porque aplicamos implementación obvia



No verificamos que exista porque eso lo hacemos con la en el servicio, llamando al método `EnsureDevicesAreNotAdded` que llama a `Exists` del repositorio.

- `Exists_WhenOwnedDeviceExists_ReturnsTrue`
  - [\[green\]](#) Solo green porque aplicamos implementación obvia

## Evidencia de TDD



## Cobertura de Pruebas

- Informe de Cobertura de Código

El porcentaje de cobertura alcanzó el 98% cuando revisamos los *Coverage Results* en Rider. Las líneas que están sin cubrir refieren mayormente a lo relacionado con los modelos (getters, setters) y a las clases/atributos creadas para hacer uso de las convenciones de Entity Framework Core.




Symbol	Coverage (%)	Uncov...
▼ Total	98%	39/2308
▼ HomeConnect.DataAccess	99%	3/398
▼ HomeConnect.DataAccess	99%	3/398
> Repositories	100%	0/330
> Context	96%	3/68
▼ HomeConnect.WebApi	99%	7/712
▼ HomeConnect.WebApi	99%	7/712
> Pagination	100%	0/6
> Filters	100%	0/198
> Controllers	99%	7/508
▼ HomeConnect.BusinessLogic	98%	29/1198
▼ BusinessLogic	98%	29/1198
> PagedData<T>	100%	0/9
> Admins.Services	100%	0/64
> Users	99%	1/160
> Notifications	99%	1/82
> Devices	99%	2/188
> BusinessOwners	99%	1/181
> Roles.Entities	97%	1/37
> HomeOwners	96%	16/377
> Auth	93%	7/100

Si analizamos la *code coverage* de Github, alcanzamos un 94% de cobertura [\[link\]](#)

Package	Line Rate	Branch Rate	Health
HomeConnect.BusinessLogic	97%	97%	✓
HomeConnect.Data Access	100%	87%	✓
HomeConnect.Web Api	88%	78%	—
Summary	94% (1632 / 1733)	90% (294 / 326)	✓
















- Análisis de Cobertura (Desviaciones y Justificaciones)

En la clase Context, no están cubiertos por los tests estos tres DbSets, ya que no contamos con un repositorio que los acceda directamente. Sin embargo, son necesarios para persistir las entidades relacionadas en la base de datos de manera indirecta.

>  Permissions	<div><div></div></div> 50%	1/2
>  Cameras	<div><div></div></div> 50%	1/2
>  HomePermissions	<div><div></div></div> 50%	1/2

Luego, en las distintas entidades, tenemos algunas propiedades creadas para garantizar el correcto funcionamiento de EF Core y hacer uso de sus convenciones. Por ejemplo, no está contemplado el uso de los constructores vacíos que suelen ser requeridos por Entity Framework para funcionar correctamente.

Ejemplo:

▼  Member	<div><div>80%</div><div></div></div>	12/59
>  User	<div><div>100%</div><div></div></div>	0/2
>  HomePermissions	<div><div>100%</div><div></div></div>	0/2
>  Home	<div><div>100%</div><div></div></div>	0/2
 Member(User)	<div><div>100%</div><div></div></div>	0/8
 Member(User,Lis	<div><div>100%</div><div></div></div>	0/9
 AddPermission(H	<div><div>100%</div><div></div></div>	0/4
 EnsurePermission	<div><div>100%</div><div></div></div>	0/5
 EnsurePermission	<div><div>100%</div><div></div></div>	0/5
>  DeletePermission	<div><div>100%</div><div></div></div>	0/5
>  HasPermission(H	<div><div>100%</div><div></div></div>	0/4
>  Id	<div><div>50%</div><div></div></div>	1/2
>  UserId	<div><div>0%</div><div></div></div>	2/2
>  Homeld	<div><div>0%</div><div></div></div>	2/2
 Member()	<div><div>0%</div><div></div></div>	7/7