

Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de aplicaciones 2

Obligatorio 1

TDD y Clean Code

Rodrigo Conze - 268505

Sebastián Borjas – 303433

Agustina Martínez – 303804

Docente: Pablo Geymonat

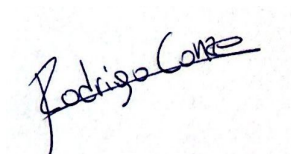
Abril 2024

Declaración de Autoría

Montevideo, Uruguay, 2 de Mayo de 2024.

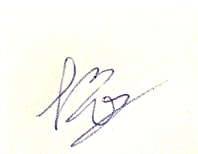
Nosotros, Rodrigo Conze, Sebastián Borjas y Agustina Martínez, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el primer obligatorio de la asignatura Diseño de Aplicaciones II;
- 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



Rodrigo Conze

02/05/2024



Sebastián Borjas

02/05/2024



Agustina Martínez

02/05/2024

Abstract

En este documento, presentamos evidencia de la aplicación de los principios de Clean Code y técnicas de Desarrollo Guiado por Pruebas, en el desarrollo de nuestro proyecto BuildingAdmin. El proyecto se ha desarrollado siguiendo las mejores prácticas de Clean Code para garantizar la legibilidad y mantenibilidad. Además, se ha aplicado el enfoque de TDD para asegurar la corrección y confiabilidad del código.

Palabras claves

Clean Code, Desarrollo Guiado por Pruebas, TDD, Pruebas Unitarias, Pruebas de Integración, Cobertura de Código, Refactorización, Documentación.

Índice

Declaración de Autoría.....	2
Abstract.....	3
Palabras claves.....	4
Índice.....	5
Introducción:.....	6
Estrategia de TDD seguida.....	6
Reporte de Cobertura de Pruebas.....	7
Estructura del Proyecto.....	7
Funcionalidades Priorizadas (*).....	9
Informe de Cobertura de Pruebas para las funcionalidades priorizadas:.....	9
Análisis de cobertura por componente:.....	9
Evidencia de que se aplicó TDD.....	10
Análisis de cobertura por componente:.....	14

Introducción:

El desarrollo del proyecto BuildingAdmin, se adhiere a los principios de Clean Code y sigue un enfoque de Desarrollo Guiado por Pruebas.

Se ha generado un informe completo de cobertura de pruebas para todas las pruebas desarrolladas en BuildingAdmin. El informe proporciona información sobre el porcentaje de código cubierto por las pruebas, destacando áreas que pueden requerir pruebas adicionales.

La estructura del proyecto mantiene una clara separación entre los proyectos de prueba y los proyectos de solución. Esta separación asegura que las pruebas sean independientes y no interfieran con el código principal de la solución.

Para las funcionalidades prioritarias (*), se ha dado un enfoque especial a dos aspectos cruciales. En primer lugar, los commits proporcionan una clara evidencia de la implementación del Desarrollo Guiado por Pruebas (TDD), mostrando la evolución del ciclo de desarrollo. Además, se ha realizado un exhaustivo análisis de cobertura de pruebas, asegurándose de alcanzar un porcentaje de cobertura entre el 90% y el 100% de las líneas de código, y cualquier desviación de estos valores ha sido meticulosamente investigada y documentada.

Estrategia de TDD seguida

La estrategia de TDD seguida en el desarrollo de BuildingAdmin involucró tanto un enfoque de hacia adentro hacia afuera (inside – out) como hacia afuera hacia adentro (outside – in)

Comenzamos con una lógica interna, basándonos en hacia adentro hacia fuera, centrándonos en las pruebas del dominio para validar y desarrollar funcionalidades esenciales. Este enfoque nos permitió establecer un dominio sólido y funcional, garantizando que nuestras pruebas reflejen correctamente el comportamiento esperado. Conjunto con esto, adoptamos una estrategia de branching donde cada test y clase del dominio tenía su propia branch, lo que facilitó el seguimiento y la gestión de cambios específicos en cada componente.

Posteriormente, cambiamos nuestra estrategia hacia fuera hacia adentro, enfocándonos en las interfaces de los servicios y la exposición de la funcionalidad a través de la API. Para la creación de la WebAPI, seguimos un proceso TDD que comenzó con la creación de un proyecto de pruebas

dedicado para la WebAPI. Posteriormente, creamos pruebas para los métodos del controlador que aún no estaban implementados, lo que nos llevó a la implementación del propio controlador. Durante este proceso, intentamos inicialmente mockear el IService, lo que resaltó la necesidad de definir interfaces para nuestros servicios. Luego, creamos la interfaz IService con los métodos necesarios y procedimos a implementar los métodos del controlador utilizando esta interfaz de servicio. Finalmente, realizamos llamadas a estos métodos en nuestras pruebas y verificamos todas las llamadas y aserciones para asegurar su correcto funcionamiento. Este enfoque iterativo nos permitió desarrollar la funcionalidad de la WebAPI de manera estructurada y probada.

Esta transición también implicó un cambio en nuestra estrategia de branching, pasando de branches específicas para cada clase de proyecto a branches que abarcan varios proyectos, lo que refleja una mayor integración y colaboración entre los componentes de la aplicación.

Reporte de Cobertura de Pruebas

La cobertura de pruebas es un indicador fundamental de la efectividad y calidad del conjunto de pruebas desarrolladas para un proyecto. A continuación, se presenta un informe de la cobertura de pruebas para las funcionalidades del proyecto BuildingAdmin.

Los porcentajes de cobertura indican que la mayoría del código ha sido probado exhaustivamente, lo que sugiere estabilidad y confiabilidad en el sistema. Sin embargo, las áreas con menor cobertura, como en webapi.dll, requieren atención adicional para garantizar un nivel óptimo de calidad y fiabilidad en todo el sistema.

Jerarquía	Cubiertas (líneas)	No cubiertas (líneas)	Cubiertas (%líneas)
agust_AGUSTINA_2024-05-02.12_34_12.coverage	6010	277	95,49%
testservices.dll	1518	55	96,50%
webapi.dll	293	35	89,33%
domain.dll	279	2	98,94%
services.dll	517	15	96,82%
iservices.dll	26	0	100,00%
testdataaccess.dll	1183	14	98,83%
testwebapi.dll	1241	0	100,00%
dto.dll	308	83	77,97%
dataaccess.dll	46	0	100,00%
testdomain.dll	599	73	89,14%

Estructura del Proyecto

Para garantizar la escalabilidad, mantenibilidad y claridad en el desarrollo de BuildingAdmin, se ha establecido una estructura de proyecto bien definida y organizada. Esta estructura se basa en

principios de arquitectura de software, separación de preocupaciones y buenas prácticas de desarrollo.

El proyecto se divide en varios componentes, cada uno con su propósito específico y claramente definido. En primer lugar, se mantiene una separación nítida entre los proyectos de prueba y los de la solución principal. Esto facilita la identificación y ejecución de pruebas unitarias y de integración, así como el mantenimiento de la base de código en general.

Dentro de la solución, se encuentran distintos proyectos que abarcan desde la capa de acceso a datos hasta la interfaz de usuario. Cada proyecto está diseñado para cumplir una función específica, como el acceso a datos, la lógica de dominio, la implementación de servicios y la exposición de la API a través de la Web.

La estructura del proyecto sigue un enfoque modular, lo que permite una fácil extensión y modificación en el futuro. Además, se han aplicado principios de diseño limpio y patrones de arquitectura que promueven la cohesión y la baja dependencia entre los diferentes componentes del sistema.

A continuación, se detalla la estructura del proyecto y la función de cada componente dentro de la solución de BuildingAdmin.

- DTO: Contiene los objetos de transferencia de datos (DTO) utilizados para la comunicación entre capas.
 - In: DTOs de entrada utilizados como parámetros en las solicitudes.
 - Out: DTOs de salida utilizados para representar la respuesta de las solicitudes.
- DataAccess: Contiene la lógica de acceso a datos y las migraciones de la base de datos.
 - Migrations: Carpeta que almacena las migraciones de la base de datos.
 - appsettings.json: Archivo de configuración de la base de datos.
- Domain: Contiene las entidades del dominio, excepciones y tipos de datos.
 - DataTypes: Enumeraciones y tipos de datos utilizados en el dominio.
 - Exceptions: Excepciones personalizadas utilizadas en el dominio.
- Factory: Contiene la implementación de la fábrica utilizada para crear instancias de servicios.
- IDataAccess: Define las interfaces para el acceso a datos.
- IServices: Define las interfaces para los servicios de la aplicación.

- Services: Contiene la implementación de los servicios de la aplicación.
- TestDataAccess: Contiene las clases de repositorio de pruebas para el acceso a datos.
- TestDomain: Contiene las pruebas de las entidades del dominio.
- TestServices: Contiene las pruebas de los servicios de la aplicación.
- TestWebApi: Contiene las pruebas de los controladores de la WebAPI.
 - Controllers: Controladores de prueba para las acciones de la API.
- WebApi: Contiene la implementación de la WebAPI.
 - Constants: Constantes utilizadas en la aplicación.
 - Controllers: Controladores de la API que manejan las solicitudes HTTP.
 - Filters: Filtros utilizados para la autenticación y manejo de excepciones.
 - Properties: Archivos de configuración y de inicio del programa.

Funcionalidades Priorizadas (*)

Informe de Cobertura de Pruebas para las funcionalidades priorizadas:

Invitar encargados a unirse a la plataforma: Esta funcionalidad permite a un administrador crear invitaciones para personas adjudicándoles el rol de encargado. La cobertura de pruebas para esta funcionalidad es del 100% en todos los componentes involucrados, desde el controlador hasta el repository. Esto significa que todas las líneas de código relacionadas con esta funcionalidad están cubiertas por pruebas, lo que garantiza una sólida base de pruebas para su correcto funcionamiento.

Eliminar invitaciones: En esta funcionalidad, un administrador puede eliminar invitaciones no aceptadas. Al igual que la funcionalidad anterior, la cobertura de pruebas es del 100% en todos los componentes relevantes, lo que asegura una adecuada cobertura de pruebas para su correcto funcionamiento.

Modificar invitación: En esta funcionalidad el administrador podrá dar más prórroga de tiempo para aceptar una invitación en aquellas invitaciones que no fueron aceptadas y están por vencer en un día o estén vencidas. Para esta funcionalidad también mantenemos una cobertura del 100% en los componentes que involucran la funcionalidad.

Análisis de cobertura por componente:

Componente	Cobertura
InvitationController	100,00%

ModifyInvitation(int, DTO.In.ModifyInvitationInput)	100,00%
AcceptInvitation(DTO.In.AcceptInvitationInput)	100,00%
DeleteInvitation(int)	100,00%
InvitationService	100,00%
CreateInvitation(Domain.Invitation)	100,00%
ModifyInvitation(int, System.DateTime)	100,00%
DeleteInvitation(int)	100,00%
InvitationRepository	100,00%
ModifyInvitationInput	100,00%
CreateInvitationInput	100,00%
RejectInvitationInput	100,00%
AcceptInvitationOutput	100,00%
Invitation (Domain)	100,00%


Evidencia de que se aplicó TDD

Para dejar evidencia mediante los commits de que se aplicó TDD vamos a mostrar las diferentes branch que involucran toda la funcionalidad, demostrando que en cada una de ellas se hizo énfasis en aplicar TDD.

Feature/data access invitation repository #23

Merged agustinamartinez... merged 11 commits into `develop` from `feature/dataAccess_invitationRepository`

Conversation 0 Commits 11 Checks 0 Files changed 3

 seaborjas added 11 commits last week

- feat(TestInvitationRepository): [GREEN] test add invitation
- feat(TestInvitationRepository): [GREEN] test update invitation
- feat(TestInvitationRepository): [GREEN] test get invitation
- feat(TestInvitationRepository): [REFACTOR] loading data to test
- feat(TestInvitationRepository): [GREEN] test get all invitation
- feat(TestInvitationRepository): [GREEN] test get all invitation with ...
- feat(TestInvitationRepository): [GREEN] test delete invitation
- feat(TestInvitationRepository): [REFACTOR] delete save() of some oper...
- feat(TestInvitationRepository): [GREEN] test clean up
- feat(TestInvitationRepository): [GREEN] test check connection
- feat(TestInvitationRepository): [GREEN] test save to database

Feature/domain invitation #8

Merged seaborjas merged 43 commits into `develop` from `feature/domain_invitation`

Conversation 0 Commits 43 Checks 0 Files changed 7

seaborjas added 5 commits 2 weeks ago

- feat(Invitation): [RED] set email attribute
- feat(Invitation): [GREEN] set email attribute
- feat(Invitation): [REFACTOR] set email attribute
- feat(Invitation): [RED] set empty email attribute exception
- feat(Invitation): [GREEN] set empty email attribute exception
- feat(Invitation): [REFACTOR] set empty email attribute exception
- feat(Invitation): [RED] set wrong email format attribute exception
- feat(Invitation): [GREEN] set wrong email format attribute exception
- feat(Invitation): [REFACTOR] set wrong email wiwhout at
- feat(Invitation): [RED] set email wihout dot
- feat(Invitation): [GREEN] set email wihout dot
- feat(Invitation): [REFACTOR] set email wihout dot
- feat(Invitation): [RED] set wrong email
- feat(Invitation): [GREEN] set wrong email
- feat(Invitation): [REFACTOR] set wrong email
- feat(Invitation): [RED] get expiration date

Feature/services_invitation #57

Merged agustinamartinez... merged 44 commits into `develop` from `feature/services_invitation`

Conversation 5 Commits 44 Checks 0 Files changed 10

agustinamartinez1044 added 29 commits 3 days ago

- feat(Invitation): [RED] get name
- feat(Invitation): [GREEN] get name
- feat(Invitation): [RED] set empty name
- feat(Invitation): [GREEN] set empty name
- feat(Invitation): [RED] null name
- feat(Invitation): [GREEN] null name
- feat(Invitation): [GREEN] set name
- feat(Invitation): [REFACTOR] attribute name
- feat(CreateInvitationInput): add attribute name
- feat(InvitationService): [RED] create invitation
- feat(InvitationService): [GREEN] create invitation
- feat(InvitationService): [RED] create invitation already exist
- feat(InvitationService): [GREEN] create invitation already exist

Feature/webapi invitations #35

Merged agustinamartinez1044 merged 46 commits into develop from feature/webapi_invitations

Conversation 0 Commits 46 Checks 0 Files changed 14

rodrigoconze added 30 commits last week

- feat(InvitationController): [RED] test create invitation
- feat(InvitationController): [GREEN] test create invitation
- feat(InvitationController): [RED] test create invitation without body
- feat(InvitationController): [GREEN] test create invitation without body
- feat(InvitationController): [REFACTOR] test create invitation without...
- feat(InvitationController): [RED] test create invitation without mail
- feat(InvitationController): [GREEN] test create invitation without mail
- feat(InvitationController): [RED] create invitation with empty mail
- feat(InvitationController): [GREEN] create invitation with empty mail
- feat(InvitationController): [REFACTOR] create invitation with empty mail
- feat(InvitationController): [RED] create invitation without name
- feat(InvitationController): [GREEN] create invitation without name
- feat(InvitationController): [RED] create invitation with empty name
- feat(InvitationController): [GREEN] create invitation with empty name
- feat(InvitationController): [REFACTOR] create invitation with empty name
- feat(InvitationController): [RED] create invitation with gone expirat...

Para ver un ejemplo de evolución del ciclo de TDD que aplicamos vamos a mostrar los siguientes casos:

feat(Invitation): [RED] set email attribute

seaborjas committed 2 weeks ago

commit 82644c:d2ba0c:f42f98f92abf98b00bfeb560cf2

Domain/Invitation.cs

```

1 + namespace Domain
2 + {
3 +     public class Invitation
4 +     {
5 +         public string Email { get { return "email"; } set { } }
6 +     }
7 + }

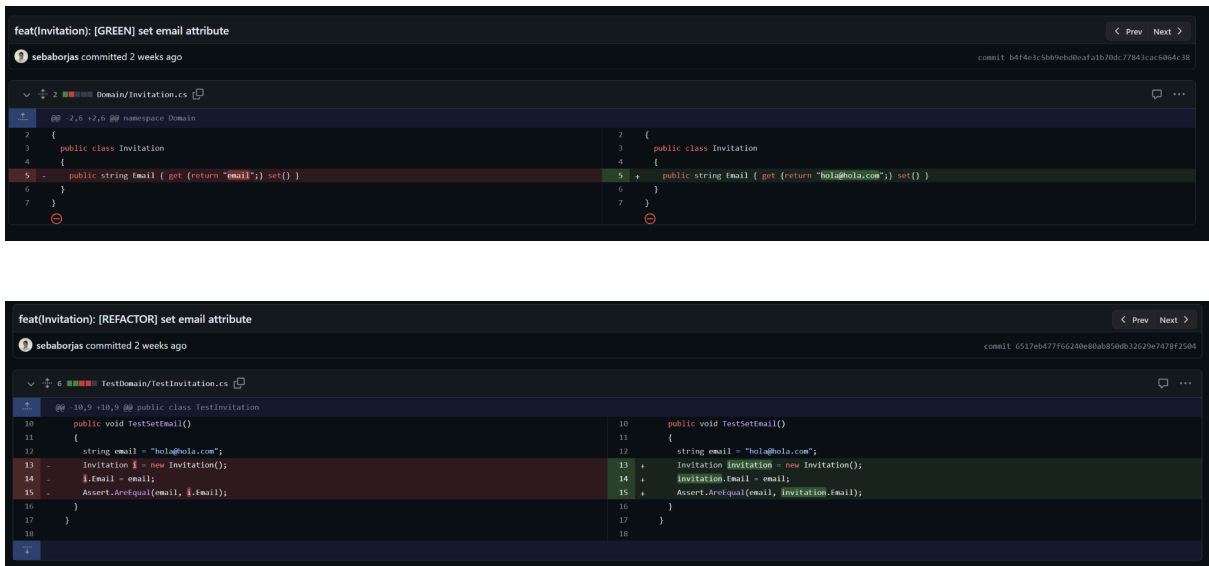
```

TestDomain/TestInvitation.cs

```

1 + using Domain;
2 + using Exceptions;
3 +
4 + namespace TestDomain
5 + {
6 +     [TestClass]
7 +     public class TestInvitation
8 +     {
9 +         [TestMethod]
10 +         public void TestSetEmail()
11 +         {
12 +             string email = "hol@hola.com";
13 +             Invitation i = new Invitation();
14 +             i.Email = email;
15 +             Assert.AreEqual(email, i.Email);
16 +         }
17 +     }
18 + }
19 + }

```



Mantenimiento de edificio: Esta funcionalidad permite a un encargado mantener un edificio (crear, modificar y eliminar). La cobertura de pruebas para esta funcionalidad es del 100% en los controladores y servicios relacionados. Sin embargo, en el servicio BuildingService, la cobertura es del 92.04%, lo que indica que hay algunas líneas de código que no están siendo cubiertas por las pruebas.

Análisis de cobertura por componente:

Componente	Cobertura
BuildingController	100,00%
DeleteBuilding(int)	100,00%
ModifyBuilding(int, DTO.In.ModifyBuildingInput)	100,00%
CreateBuilding(DTO.In.CreateBuildingInput)	100,00%
BuildingService	92,04%
DeleteBuilding(int)	100,00%
CreateBuilding(Domain.Building)	100,00%
ModifyBuilding(int, Domain.Building)	100,00%

BuildingRepository	100,00%
CreateBuildingInput	100,00%
ModifyBuildingInput	100,00%
CreateBuildingOutput	100,00%
Building (Domain)	100,00%

Feature/domain building #6


Merged agustinamartinez... merged 72 commits into develop from feature/domain_building

Conversation 0

Commits 72


Checks 0

Files changed 2




agustinamartinez1044


added 30 commits 2 weeks ago




feat(Building): [RED] attribute id




feat(Building): [GREEN] attribute id




feat(Building): [REFACTOR] attribute id




feat(Building): [RED] attribute name




feat(Building): [GREEN] attribute name




feat(Building): [REFACTOR] attribute name




feat(Building): [REFACTOR] attribute name




feat(Building): [RED] invalid id




feat(Building): [GREEN] invalid id




feat(Building): [RED] empty name




feat(Building): [GREEN] empty name




feat(Building): [RED] invalid name




feat(Building): [GREEN] invalid name



feat(Building): [RED] attribute expenses



feat(Building): [GREEN] attribute expenses



feat(Building): [REFACTOR] attribute expenses

Feature/webapi buildings #41

Merged agustinamartinez... merged 43 commits into `develop` from `feature/webapi_buildings`

Conversation **0** Commits **43** Checks **0** Files changed **8**

- feat(BuildingController): [RED] create building
- feat(BuildingController): [GREEN] create building
- feat(BuildingController): [RED] create building without body
- feat(BuildingController): [GREEN] create building without body
- feat(BuildingController): [RED] create building without Name
- feat(BuildingController): [GREEN] create building without Name
- feat(BuildingController): [RED] create building with empty name
- feat(BuildingController): [GREEN] create building with empty name
- feat(BuildingController): [REFACTOR] create building with empty name
- feat(BuildingController): [RED] create building without address
- feat(BuildingController): [GREEN] create building without address
- feat(BuildingController): [RED] create building with empty address
- feat(BuildingController): [GREEN] create building with empty address
- feat(BuildingController): [REFACTOR] create building with empty address
- feat(BuildingController): [RED] create building without location
- feat(BuildingController): [GREEN] create building without location

Feature/services building #51

Merged rodrigoconze merged 32 commits into `develop` from `feature/services_building` yesterday

Conversation **1** Commits **32** Checks **0** Files changed **7**

- refactor(IService): change IBuildingServices to IBuildingService
- feat(BuildingService): [RED] create building
- feat(BuildingService): [GREEN] create building
- feat(BuildingService): [RED] create building already exist
- feat(BuildingService): [GREEN] create building already exist
- feat(BuildingService): [RED] delete building
- feat(BuildingService): [GREEN] delete building
- feat(BuildingService): [RED] delete building that not exist
- feat(BuildingService): [GREEN] delete building that not exist
- feat(BuildingService): [RED] modify building
- feat(BuildingService): [GREEN] modify building
- feat(BuildingService): [RED] modify building that not exist
- feat(BuildingService): [GREEN] modify building that not exist
- feat(BuildingService): [REFACTOR] modify building

De la misma forma que en la funcionalidad anterior mostramos como se ha seguido el ciclo de TDD en lo que respecta a todos los componentes de la funcionalidad.

En conclusión, el Desarrollo Guiado por Pruebas ha sido fundamental en nuestro proceso de desarrollo. Nos ha brindado una estructura sólida para escribir código, asegurando que nuestras funcionalidades estén respaldadas por pruebas automatizadas desde el principio. Esto ha resultado en un código más estable, con menos errores y una mayor confianza en su funcionamiento.

Además, el TDD nos ha permitido iterar de manera eficiente sobre nuestras implementaciones, ya que cada ciclo nos ha llevado a una mejor comprensión de los requisitos y a una mayor claridad en la solución. Esto se refleja en la calidad de nuestro código y en la capacidad de adaptarnos a los cambios y nuevas funcionalidades de manera ágil.