

Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones II

Obligatorio I

Evidencia de Clean Code y de la aplicación de TDD

Juan Manuel Gallicchio 233335

Federico Carbonell 224359

<https://github.com/ORT-DA2/CarbonellGallicchio>

Evidencia de Clean Code y de la aplicación de TDD	4
Pruebas unitarias	4
Proyectos de pruebas	5
Justificación y evidencia de TDD	6
La decisión del equipo fue realizar TDD en la totalidad del proyecto, siempre comenzamos desde los controllers hasta llegar a la capa de acceso a datos. Por lo que la evidencia presentada serán capturas parciales de los commits realizados. Al utilizar la metodología Git Flow es probable que se encuentren commits mezclados.	6
Evidencia de commits	6
Ver las principales categorías y navegar entre su contenido y playlists.	6
Agendar consultas con un psicólogo	6
Iniciar sesión en el sistema	7
Dar de alta y baja contenido reproducible	7
Mantenimiento de un psicologo con su respectiva información	7
Un administrador también puede realizar el mantenimiento de los administradores del sistema.	8
Análisis de cobertura de pruebas	9
Cobertura Session Logic:	10
Cobertura Business Logic:	10
Cobertura Data Access:	10
Cobertura Paquetes de excepciones:	10
Cobertura Paquete Validator:	10
Cobertura Adapter:	10
Cobertura Models:	10
Cobertura Domain:	10
Cobertura WebApi:	10
Evidencia de clean code	11
Nombres nemotécnicos	11
Funciones chicas	11
Pruebas unitarias	11
Uso de excepciones	12
Bajo acoplamiento	12
Inyección de dependencias	12
Single Responsibility Principle	13
Repetición innecesaria	13

Evidencia de Clean Code y de la aplicación de TDD

La finalidad de este documento es dejar en evidencia la metodología utilizada para el desarrollo de la aplicación y demostrar cómo el código cumple con los estándares de Clean Code.

La práctica utilizada fue TDD, consiste en escribir inicialmente las pruebas unitarias luego escribir el código necesario para que pase la prueba satisfactoriamente y finalmente refactorizar el código con el propósito de mejorarlo. Avanzando en el documento se encontrará la cobertura de pruebas de los tests realizados.

Pruebas unitarias

Como se explicó anteriormente, la metodología se basa en realizar pruebas unitarias para el código. Debido a esto es que la mayoría de nuestras clases y métodos presentan las mismas.

Todas las pruebas cumplen con el principio First, este indica cinco características que deben cumplir las pruebas para ser consideradas de calidad.

Las pruebas son de rápida ejecución, esto nos permite ejecutar una gran cantidad de pruebas en un corto tiempo, gracias a esto se pueden ejecutar frecuentemente y detectar bugs.

Cada una de estas es independiente a todas las demás pruebas que componen los proyectos de pruebas, esto nos evita tener que realizar regresiones y tener los problemas localizados fácilmente.

Los resultados de las pruebas son independientes del ambiente en el cual se ejecutan, no dependen de configuraciones particulares.

Son auto validantes, no es necesario realizar acciones para validar el resultado de las mismas. Disparando la ejecución éstas se ejecutan automáticamente y podemos realizar otras tareas en simultáneo.

Las pruebas en su totalidad fueron desarrolladas antes de realizar el código de la aplicación.

Proyectos de pruebas

Para aumentar la separación y evitar dependencias decidimos generar un proyecto de pruebas por capa del sistema. Cada uno independiente a los demás.

La carpeta de Tests esta compuesta por AdapterTests, BusinessLogicTests, DataAccessTests, SessionLogicTests, ValidatorTests y WebApiTests.

Para los proyectos se utilizó el framework MSTests y para la implementación de las pruebas independientemente de estos se realizaron siguiendo la estructura Arrange, Act, Assert. Esto nos provee una mejor organización, se basa en inicialmente preparar los objetos necesarios para el test, luego ejecutar la preparación y finalmente verificar los resultados.

Para lograr que los tests sean independientes de las implementaciones de ciertas clases utilizamos Mocks, mediante el framework Moq. De esta manera, por ejemplo, nos permite realizar los test de los controllers y no depender de los adapters. Utilizamos los métodos Setup() para configurar el comportamiento de dichos objetos y luego mediante un .Throws() o .Returns() configurar la respuesta de dicha llamada. Una vez finalizada la configuración y ejecutado el método del test a probar, se verifica que la configuración de los mocks hayan sido correctas mediante el método .VerifyAll().

En la siguiente imagen se puede notar la explicación dada previamente, en este caso estamos mockeando el adapter de la clase audio content.

```
[TestMethod]
public void TestGetAudioContentOk()
{
    int audioContentId = 1;
    AudioContentBasicInfoModel audioContentToReturn = new AudioContentBasicInfoModel()
    {
        Id = audioContentId,
        Name = "Canción",
        Duration = TimeSpan.MaxValue,
        CreatorName = "Juan",
        ImageUrl = "www.unaimagen.com",
        AudioUrl = "www.audio.com"
    };
    Mock<IAudioContentLogicAdapter> mock = new Mock<IAudioContentLogicAdapter>(MockBehavior.Strict);
    mock.Setup(m => m.GetById(audioContentId)).Returns(audioContentToReturn);
    AudioContentController controller = new AudioContentController(mock.Object);

    var result = controller.Get(audioContentId);
    var okResult = result as OkObjectResult;
    var audioContent = okResult.Value as AudioContentBasicInfoModel;

    Assert.AreEqual(audioContentToReturn.Id, audioContent.Id);
}
```

Justificación y evidencia de TDD

La decisión del equipo fue realizar TDD en la totalidad del proyecto, siempre comenzamos desde los controllers hasta llegar a la capa de acceso a datos. Por lo que la evidencia presentada serán capturas parciales de los commits realizados. Al utilizar la metodología Git Flow es probable que se encuentren commits mezclados.

Evidencia de commits

Ver las principales categorías y navegar entre su contenido y playlists.

```
▶ 🐛 Refactor of GetPlaylistByCategory tests on category controller and on category logic. +0 -5 -0 · You, 16 days ago
▶ 🐛 Moving AutoMapper to BusinessLogic and refactor category controller tests, lack TestGetPlaylistByCategoryNotExistentId +0 -12 -0 · You, 19 days ago
▶ 🐛 Refactor of TestGetAllCategoriesOk on category controller. Changing business logic to return models +0 -1 -0 · You, 19 days ago
▶ 🐛 Refactor GetPlaylistByCategory on CategoryLogic, added private function to validate playlists nullability +0 -1 -0 · You, 20 days ago
▶ 🐛 Adding GetPlaylistByCategory not existent id test on CategoryLogic. Red phase +0 -2 -0 · You, 20 days ago
▶ 🐛 Adding GetPlaylistByCategory on CategoryLogic test. Teorically red but signature method already exists. +0 -1 -0 · You, 20 days ago
▶ 🐛 Refactor GetPlaylistByCategory controller test. Green phase +0 -2 -0 · You, 20 days ago
▶ 🐛 Adding GetPlaylistByCategory controller test. Red phase +0 -1 -0 · You, 20 days ago
▶ 🐛 Adding BetterCalmContext and FirstMigration +8 -17 -0 · Juan Manuel Gallicchio, 24 days ago
▶ 🐛 Refactoring, changing folder BetterCalm for WebApi because it's the right name now +0 -17 -1 · Juan Manuel Gallicchio, 24 days ago
▶ 🐛 Refactoring. Adding new project for each layer test to apply SRP. +4 -3 -0 · Juan Manuel Gallicchio, 24 days ago
▶ 🐛 - Add definitions and methods for GetPlayListByCategory in their respective layers ... +0 -6 -0 · You, 24 days ago
▶ 🐛 Refactor GetPlaylistByCategory, moving responsibilities to business layer +0 -2 -0 · You, 24 days ago
▶ 🐛 - Override equals of CategoryBasicInfoModel ... +0 -2 -0 · You, 24 days ago
▶ 🐛 Modifying unused atributes on get all categories method +0 -3 -0 · You, 24 days ago
▶ 🐛 Refactor of Get Categories test on Category controller +0 -1 -0 · You, 24 days ago
▶ 🐛 Start Refactor of CategoryBusinessLogic. Moving the responsibility of Business Layer to Data Access +1 -6 -0 · You, 24 days ago
▶ 🐛 TestGetAllCategories on Logic layer. Red phase (We have already created the repository and the interface that implements it) +1 -3 -0 · You, 25 days ago
▶ 🐛 Adding AutoMapper instantiation to StartUp and moving corresponding behavior to logic layer +0 -11 -0 · Juan Manuel Gallicchio, a month ago
▶ 🐛 Refactoring. Adding necessary layers for the correct chosen architecture. Red step from TDD. +16 -4 -0 · Juan Manuel Gallicchio, a month ago
▶ 🐛 Refactor: changing WebApi project name and adding a layer for Domain and for Model +7 -9 -2 · Juan Manuel Gallicchio, a month ago
▶ 🐛 Adding GetPlaylistByCategory and the corresponding tests +1 -3 -0 · Juan Manuel Gallicchio, a month ago
Over a month ago
▶ 🐛 Adding Test project and first test for CategoryController +4 -1 -0 · Juan Manuel Gallicchio, a month ago
```




Agendar consultas con un psicólogo

```
▶ 🐛 Adding Agendas migration and registering IAgendaLogic in ServiceFactory +2 -3 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Refactor GetAvailablesByProblematicId name to GetAllByProblematicId because it represent more accurate the behavior. Also adding TestClass decorate in ConsultationLo...
▶ 🐛 Green phase: adding validation for List +0 -2 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Red phase: adding TestGetAvailableByProblematicIdAndDateNoPsychologist +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Refactor, fixing test +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase: implementing Assign with IsAvaible asignation +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Red phase: TestAssingAgendaWithCountFourShouldNotBeAvaible +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase: implementing Assign in AgendaLogic +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Red phase: TestAssingAgendaOk +0 -3 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase: Adding implementation of GetAvailableByProblematicIdAndDate +0 -4 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Red phase: TestGetAvailableByProblematicIdAndDateOk +0 -2 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase: Adding GetAvailablesByProblematicId implementation +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Red phase: TestGetAvailablesByProblematicId adding method without implementationm and test +0 -3 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase: TestCreateAgendaByPsychologistIdAndDate, adding Add implementation +0 -1 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Green phase for GetAgendaByPsychologistIdAndDate and adding TestCreateAgendaByPsychologistIdAndDate red phase +0 -3 -0 · Juan Manuel Gallicchio, 2 days ago
▶ 🐛 Adding TestGetAgendaByPsychologistIdAndDate and all necessary classes +4 -0 -0 · Juan Manuel Gallicchio, 2 days ago
```

Iniciar sesión en el sistema

- ▶ 🟡 Green phase: adding GetByEmailAndPassword to get Administrator +0 ~2 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Implement Add on ConsultationLogic. Green phase +0 ~1 -0 · You, 3 days ago
- ▶ 🟡 Adding TestAddConsultationOk on AdministratorLogicTest and the corresponding clases to compile. Red phase +5 ~0 -0 · You, 3 days ago
- ▶ 🟡 Red phase: adding TestLoginCorrectlyGetAdminId +0 ~2 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Green phase: adding implementation for Add in SessionLogics +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Red phase: adding TestLoginCorrectly in SessionLogicsTests +0 ~2 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Add HttpPost decorator on Post of ConsultationController +0 ~1 -0 · You, 3 days ago
- ▶ 🟡 Making required Password property in SessionModel +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Adding TestPostSessionEmptyPassword it does return ok but Password argument must be required +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Making required Email property in SessionModel +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Adding TestPostSessionEmptyEmail it does return ok but Email argument must be required +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Implement Post on ConsultationController. Green phase +0 ~1 -0 · You, 3 days ago
- ▶ 🟡 Green phase: Adding try/catch in SessionController Post with ArgumentException to return 400 in case argument is invalid +0 ~1 -0 · Juan Manuel Gallicchio, 3...
- ▶ 🟡 Red phase: adding TestPostSessionIncorrectPasswordForEmail +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Adding TestPostConsultationOk on ConsultationControllerTest and the corresponding clases to compile. Red phase +6 ~0 -0 · You, 3 days ago
- ▶ 🟡 Green phase: Adding try/catch in SessionController Post with NullObjectMappingException to return 404 in case object not exist +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Red phase: adding TestPostSessionEmailNotExistent +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Green phase: adding Post implementation in SessionController +0 ~1 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Red phase: creating TestPostSessionOk on SessionControllerTest and all clases only to compile +6 ~0 -0 · Juan Manuel Gallicchio, 3 days ago
- ▶ 🟡 Merge branch 'feature/authorization-filter' into feature/web-api ... +8 ~11 -0 · Juan Manuel Gallicchio, 3 days ago



















Dar de alta y baja contenido reproducible

- ▶ 🟡 Refactoring AudioContentLogic +0 ~1 -0 · Juan Manuel Gallicchio, 13 days ago   
- ▶ 🟡 Green phase: adding Create implementation in AudioContentLogic to pass TestCreateAudioContentOk +0 ~2 -0 · Juan Manuel Gallicchio, 13 days ago
- ▶ 🟡 Red phase: adding TestCreateAudioContentOk and the corresponding methods only to compile +0 ~6 -0 · Juan Manuel Gallicchio, 13 days ago
- ▶ 🟡 Green phase: adding GetById implementation in AudioContentLogic to pass TestGetByIdAudioContentOk +0 ~1 -0 · Juan Manuel Gallicchio, 13 days ago
- ▶ 🟡 Red phase: adding AudioContentLogicTests, in this case TestGetAudioContentOk and the corresponding clases and methods only to compile +4 ~1 -0 · J...
- ▶ 🟡 Green phase: adding Update implementation in AudioContentController to pass TestUpdateAudioContentOk +0 ~1 -0 · Juan Manuel Gallicchio, 13 days ago

Mantenimiento de un psicologo con su respectiva información

- ▶ 🟡 Implementation of Update on Repository. Green phase +0 ~1 -0 · You, 11 days ago
- ▶ 🟡 Fix TestUpdatePsychologistOk on PsychologistRepositoryTests. Still Red phase +0 ~1 -0 · You, 11 days ago
- ▶ 🟡 Adding TestUpdatePsychologistOk on PsychologistRepositoryTests. Red phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Implementation of Add on Repository and modify TestAddPsychologistOk. Green phase +0 ~2 -0 · You, 12 days ago
- ▶ 🟡 Adding TestAddPsychologistOk on PsychologistRepositoryTests and DbSet of Psychologist. Red phase +1 ~1 -0 · You, 12 days ago
- ▶ 🟡 Fixing names on corresponding Update tests on PsychologistLogicTests +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding validation for not existent Psychologist in PsychologistLogic to pass TestUpdatePsychologistNotExistent. Green phase +0 ~5 -0 · You, 12 days ago
- ▶ 🟡 Adding TestUpdatePsychologistNotExistent on PsychologistLogicTests. Red phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding validation for null object on method Delete on PsychologistLogic . Green phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding TestDeletePsychologistNotExistentId on PsychologistLogicTests. Red phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding validation for null object on method GetById on PsychologistLogic . Green phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding TestGetPsychologistNotExistentId on PsychologistLogicTests. Red phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Add Tests on PsychologistControllerTests ... +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Implement catching argument exception on PsychologistController Update method. Green phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding TestUpdatePsychologistInvalidName on PsychologistLogicTests. Red phase +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Adding required attributes on PsychologistModel +0 ~1 -0 · You, 12 days ago
- ▶ 🟡 Add Tests on PsychologistControllerTests ... +0 ~1 -0 · You, 13 days ago
- ▶ 🟡 - Fixing namespaces ... +0 ~3 -0 · You, 13 days ago
- ▶ 🟡 Implement catching exceptions on PsychologistController Update method. Green phase +0 ~1 -0 · You, 13 days ago
- ▶ 🟡 Adding TestUpdatePsychologistNotFound on PsychologistLogicTests. Red phase +0 ~1 -0 · You, 13 days ago

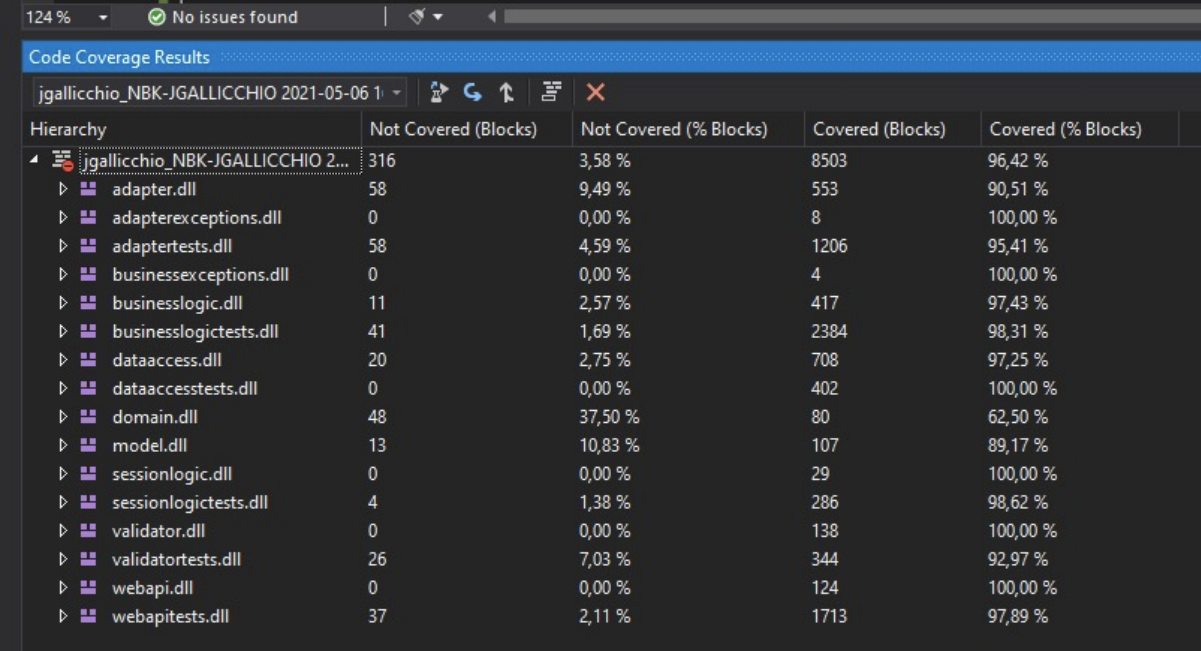
Un administrador también puede realizar el mantenimiento de los administradores del sistema.

- ▶  Add validation of exceptions on GetById On Administrator controller. Green phase. +0 ~1 -0 · You, 3 days ago
- ▶  Adding TestGetAdministratorNotExistent on AdministratorController. Red phase +0 ~1 -0 · You, 3 days ago
- ▶  Add migrations and snapshot for Administrator +2 ~1 -0 · You, 3 days ago
- ▶  Adding DbSet of Administrators on Context and Administrator logic and interface on service factory. +0 ~2 -0 · You, 3 days ago
- ▶  Add validation on Update on AdministratorLogic. Green phase. +0 ~2 -0 · You, 3 days ago
- ▶  Add validation on DeleteById on AdministratorLogic. Green phase. +0 ~1 -0 · You, 3 days ago
- ▶  Adding TestDeleteAdministratorNotExistentId on AdministratorLogicTest. Red phase +0 ~1 -0 · You, 3 days ago
- ▶  Add validation on GetById on AdministratorLogic. Green phase. +0 ~1 -0 · You, 5 days ago
- ▶  Adding TestGetAdministratorNotExistentId on AdministratorLogicTest. Red phase +0 ~1 -0 · You, 5 days ago
- ▶  Implement Update on AdministratorLogic. Green phase +0 ~1 -0 · You, 5 days ago
- ▶  Adding TestUpdateAdministratorOk on AdministratorLogicTest and the corresponding clases to compile. Red phase +0 ~3 -0 · You, 5 days ago
- ▶  Implement DeleteById on AdministratorLogic. Green phase +0 ~1 -0 · You, 5 days ago
- ▶  Adding TestDeleteAdministratorOk on AdministratorLogicTest and the corresponding clases to compile. Red phase +0 ~3 -0 · You, 5 days ago
- ▶  Implement Add on AdministratorLogic. Green phase +0 ~1 -0 · You, 5 days ago
- ▶  Fixing Add method on IAdministratorLogic and AdministratorLogicTest +0 ~2 -0 · You, 5 days ago
- ▶  Adding TestCreateAdministratorOk on AdministratorLogicTest and the corresponding clases to compile. Red phase +0 ~3 -0 · You, 5 days ago
- ▶  Implement GetById on AdministratorLogic. Green phase +0 ~1 -0 · You, 5 days ago
- ▶  Adding TestGetAdministratorByIdOk on AdministratorLogicTest and the corresponding clases to compile. Red phase +4 ~0 -0 · You, 5 days ago

Análisis de cobertura de pruebas

Para la ejecución de la cobertura de pruebas decidimos utilizar [ExcludeFromCodeCoverage] en Startup, Program y Migrations. Ya que no son probados por lo que agregarlo en los resultados no aporta valor

Logramos obtener un gran porcentaje de cobertura fluctuando entre noventa y cien por ciento en la mayoría de los proyectos. El único proyecto que no estuvo los mismos resultados es Domain.



The screenshot shows the 'Code Coverage Results' window in Visual Studio. The title bar indicates '124 %' zoom and 'No issues found'. The window displays a table with the following columns: Hierarchy, Not Covered (Blocks), Not Covered (% Blocks), Covered (Blocks), and Covered (% Blocks). The data is organized into a tree structure under the project 'jgallicchio_NBK-JGALLICCHIO 2021-05-06 1'. The table lists various DLLs and their corresponding coverage statistics.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
jgallicchio_NBK-JGALLICCHIO 2021-05-06 1	316	3,58 %	8503	96,42 %
adapter.dll	58	9,49 %	553	90,51 %
adapterexceptions.dll	0	0,00 %	8	100,00 %
adaptestests.dll	58	4,59 %	1206	95,41 %
businessexceptions.dll	0	0,00 %	4	100,00 %
businesslogic.dll	11	2,57 %	417	97,43 %
businesslogictests.dll	41	1,69 %	2384	98,31 %
dataaccess.dll	20	2,75 %	708	97,25 %
dataaccesstests.dll	0	0,00 %	402	100,00 %
domain.dll	48	37,50 %	80	62,50 %
model.dll	13	10,83 %	107	89,17 %
sessionlogic.dll	0	0,00 %	29	100,00 %
sessionlogictests.dll	4	1,38 %	286	98,62 %
validator.dll	0	0,00 %	138	100,00 %
validatortests.dll	26	7,03 %	344	92,97 %
webapi.dll	0	0,00 %	124	100,00 %
webapitests.dll	37	2,11 %	1713	97,89 %

Analizando los porcentajes que se encuentran debajo, creemos que más allá de que en el Dominio el porcentaje cubierto no es bueno en la totalidad, si fue cubierto lo que era relevante de probar y el porcentaje mostrado no es el que realmente importa. Igualmente se alcanzó una excelente cobertura general del proyecto lo que ayuda a poder realizar modificaciones en un futuro y asegurar que el cambio no genere problemas.

Cobertura Session Logic:

Se obtuvo una excelente cobertura, le dimos importancia al set de pruebas de dicho proyecto ya que es el encargado de mantener las sesiones de usuario y realizar las validaciones correspondientes.

Cobertura Business Logic:

Aunque no logramos la totalidad de cobertura es un muy buen resultado, creemos que cubrimos satisfactoriamente todo el código siendo la lógica de negocio una parte crucial del sistema.

Cobertura Data Access:

Al igual que en Business Logic, en la Data Access se logra una cobertura muy buena. Y también cumple un rol importante en el sistema.

Cobertura Paquetes de excepciones:

Obtuvimos la totalidad de cobertura, si bien no presentan demasiada lógica es importante saber que su funcionamiento es el esperado.

Cobertura Paquete Validator:

Obtuvimos un gran resultado de cobertura, es de suma importancia que su funcionamiento sea el correcto ya que es el encargado de validar las entidades del sistema.

Cobertura Adapter:

Presenta una de las coberturas más bajas, sin embargo es un resultado totalmente satisfactorio y se encuentra en los rangos de excelencia.

Cobertura WebApi:

Se obtuvo una excelente cobertura, le dimos importancia al set de pruebas de dicho proyecto.

Evidencia de clean code

Nombres nemotécnicos

Para los nombres de variables utilizamos camelcase, además se definieron lo más informativos posible. Los nombres de los métodos comienzan con mayúscula y respetan la notación húngara.

```
public void Add(AdministratorModel administrator)
{
    try
    {
        administratorModelValidator.Validate(administrator);
        Administrator administratorIn = mapper.Map<Administrator>(administrator);
        administratorLogic.Add(administratorIn);
    }
    catch (AlreadyExistException e)
    {
        throw new EntityAlreadyExistException(e.Message);
    }
}
```

Funciones chicas

Las funciones tienen una única responsabilidad, las funciones de mayor tamaño probablemente no cumplan lo indicado. Esto nos genera una mantenibilidad más simple y un mejor entendimiento al leer el código. Tampoco se debe abusar de los parámetros, lo ideal sería que sean menos de tres.

```
public Session Add(string email, string password)
{
    Administrator admin = administratorLogic.GetByEmailAndPassword(email, password);
    Session newSession = new Session()
    {
        Token = Guid.NewGuid(),
        Administrator = admin
    };
    return sessionRepository.Add(newSession);
}
```

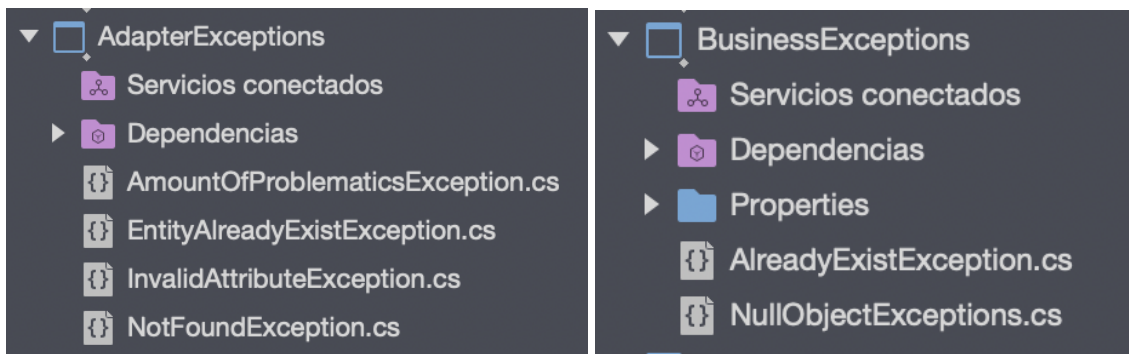
Pruebas unitarias

Para las pruebas unitarias seguimos el ciclo TDD comentado anteriormente. Hasta no tener hecha una prueba no se realiza la implementación del código.

Uso de excepciones

Para este proyecto decidimos crear nuestras propias excepciones para tener mayor control y proporcionar una mejor descripción para que sea más sencillo encontrar por que se ha lanzado dicha excepción.

Creamos distintos proyectos de excepciones para evitar la dependencia y separar las capas.



Bajo acoplamiento

Para lograr un bajo acoplamiento generamos interfaces para evitar la dependencia de clases concretas y gracias a esto disminuir el acoplamiento.

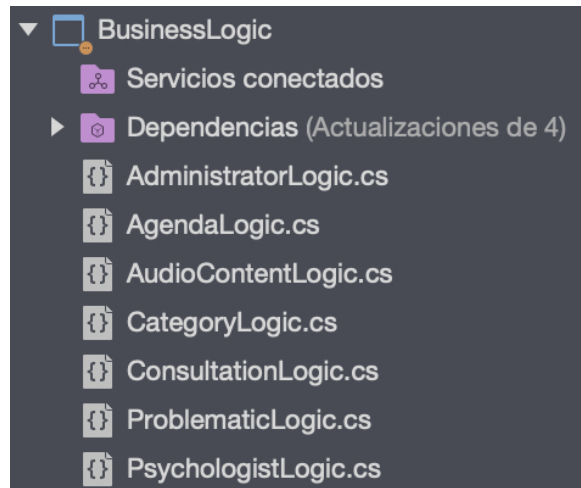
Inyección de dependencias

Empleamos inyección de dependencias para que un objeto no sea el responsable de instanciar sus dependencias sino que lo delegamos a las interfaces. Tanto nuestros adapters, la lógica de negocio, las validaciones, nuestros repositorios y la sesión se le inyectan dependencias.

```
public class PsychologistLogicAdapter : IPsychologistLogicAdapter
{
    private readonly IPsychologistLogic psychologistLogic;
    private readonly IMapper mapper;
    private readonly IValidator<PsychologistModel> psychologistModelValidator;
    public PsychologistLogicAdapter(IPsychologistLogic psychologistLogic, IMapper mapper,
        IValidator<PsychologistModel> psychologistModelValidator)
    {
        this.psychologistLogic = psychologistLogic;
        this.mapper = mapper.Configure();
        this.psychologistModelValidator = psychologistModelValidator;
    }
}
```

Single Responsibility Principle

Siguiendo el principio es que se creó una clase de Business Logic, Adapter y Validator por entidad. Esto nos garantiza que solamente se realicen acciones sobre cada entidad mediante su correspondiente clase de una capa específica.



Repetición innecesaria

Para evitar la repetición de gran cantidad de código decidimos crear una interfaz de repositorio genérico y la implementación de la misma genérica. Ya que la implementación de las funciones son prácticamente iguales sin importar el tipo de entidad a persistir.

```
namespace DataAccessInterface
{
    public interface IRepository<T> where T : class
    {
        List<T> GetAll(Expression<Func<T, bool>> predicate = null);
        T GetById(int id);
        T Get(Expression<Func<T, bool>> predicate);
        T Add(T domain);
        void Delete(T domain);
        void Update(T domain);
        bool Exists(Expression<Func<T, bool>> predicate);
    }
}
```