

Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de Aplicaciones 2

Obligatorio 1

Evidencia de Clean Code y de la aplicación de TDD

Mauro Teixeira(211753) Rodrigo Hirigoyen (197396)

Docentes: Gabriel Piffaretti, Nicolas Fierro, Nicolas Hernandez

<https://github.com/mauroteix/Teixeira-Hirigoyen>

Entregado como requisito de la materia Diseño de Aplicaciones 2

6 de mayo de 2021

Índice

Evidencia de Clean Code	3
Nombres con sentido	3
Funciones	4
Comentarios	5
Formato	5
Objetos y estructuras de datos	6
Procesando errores	7
Clases	8
Evidencia de TDD y pruebas unitarias	9
Justificación de funcionalidades con TDD	10
Navegar entre las playlist	11
Dar de alta y baja contenido reproducible	14
Mantenimiento de un psicólogo con su respectiva información	18
Tests Unitarios y cobertura total de pruebas	21

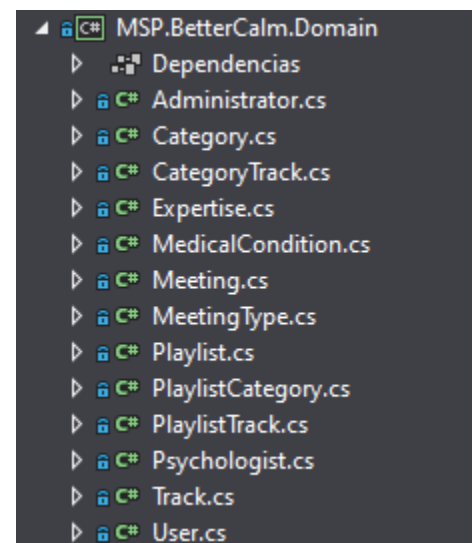
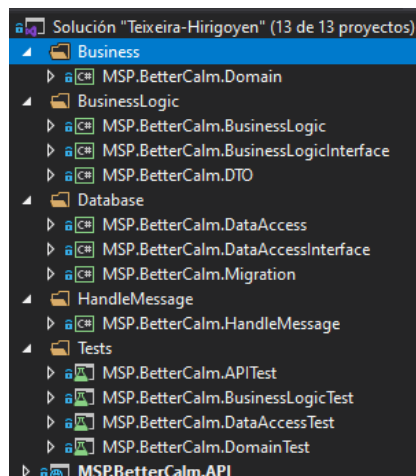
Evidencia de Clean Code

Para desarrollar nuestro obligatorio nos basamos en el libro Clean Code de Uncle Bob, en el cual durante 13 capítulos, nos explica con ejemplos las buenas prácticas que debemos seguir a la hora de escribir nuestro código. Para ejemplificar cómo utilizamos dichas técnicas, iremos haciendo un repaso capítulo a capítulo del libro y ejemplificando cómo aplicamos dichas técnicas en nuestro obligatorio. Intentaremos dar ejemplos con capturas de pantalla de los puntos más importantes de cada sección.

Nombres con sentido

Todo el obligatorio fue escrito en idioma inglés, desde el código mismo, hasta los mensajes de excepción y los endpoint de la api. Si bien entendemos que el sistema podría ser usado por personas de habla hispana, debíamos ser consistentes en el uso del lenguaje y elegimos el más universal. Todos los nombres utilizados para los proyectos, clases, variables, endpoints y tests son autodescriptivos con esto buscamos tener un código lo más legible posible y fácil de mantener para un futuro desarrollador que sea incluido en el proyecto.

```
public class Playlist
{
    51 referencias | 5/20 pasando | Rodrigo Hirigoyen, Hace 24 días | 1 autor, 1 cambio
    public int Id { get; set; }
    33 referencias | 4/15 pasando | Rodrigo Hirigoyen, Hace 24 días | 1 autor, 1 cambio
    public string Name { get; set; }
    27 referencias | 3/13 pasando | Rodrigo Hirigoyen, Hace 24 días | 1 autor, 1 cambio
    public string Description { get; set; }
    15 referencias | 3/3 pasando | Rodrigo Hirigoyen, Hace 24 días | 1 autor, 1 cambio
    public string Image { get; set; }
}
```



En cuanto a los tests, decidimos llamar al nombre de cada clase que describa qué clase está testeando. Omitimos la palabra test en los métodos dentro de cada clase, ya que consideramos que agregaría demasiadas palabras sin necesidad, ya que la clase y la etiqueta TestCase son suficientes para notar que se trata de tests.

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void GetPlaylist()...
```

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void GetPlaylistNotExist()...
```

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void AddPlaylistOk()...
```

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void AddPlaylistNameEmpty()...
```

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void AddPlaylistDescriptionLengthWrong()...
```

```
[TestMethod]
0 | 0 referencias | mauroteix, Hace 11 días | 1 autor, 1 cambio
public void AddPlaylistCategoryEmpty()
{
```

Funciones

Continuando en la misma línea que el capítulo anterior, utilizamos CamelCase con nombres descriptivos en las funciones, los cuales decidimos que podrían ser un poco largos en algunas ocasiones. Si bien priorizamos la claridad en los nombres ante el largo de las firmas, intentamos no tener firmas excesivamente largas. Nos focalizamos en cumplir SRP no solo en las clases sino también en las funciones, por este motivo es que en general nuestras funciones no superan las 20 o 30 líneas de código.

```
9 referencias | 0/7 pasando | mauroteix, Hace 4 días | 2 autores, 10 cambios
public void Add(Playlist playlist)
{
    ValidatePlaylist(playlist);
    Playlist play = ToEntity(playlist);
    _repository.Add(play);
}

2 referencias | mauroteix, Hace 4 días | 1 autor, 4 cambios
private void ValidatePlaylist(Playlist playlist)
{
    if (playlist.NameEmpty()) throw new FieldEnteredNotCorrect("The name cannot be empty");
    if (!playlist.DescriptionLength()) throw new FieldEnteredNotCorrect("The length of the description should not exceed 150 characters");
    if (playlist.PlaylistCategoryEmpty()) throw new FieldEnteredNotCorrect("A Playlist Category must be added");
    ValidateCategoriesId(playlist);
    ValidateTrackId(playlist);
    ValidateCategoryUnique(playlist);
    ValidateTrackUnique(playlist);
}
```

Por ejemplo, en la clase de lógica de las playlist, el método Add se asegura de que un la playlist tenga campos válidos, pero no es ese mismo metodo quien hace la validación ni quien agrega a la base de datos, si no que las responsabilidades se dividen en métodos auxiliares, cada uno con una función particular. En este caso existe un método que únicamente se encarga de validar una playlist, y se reutiliza para múltiples validaciones con el objetivo de reutilizar el código y no generar funciones con efectos secundarios. Nos aseguramos que las funciones reciban la cantidad mínima de parámetros posible, siendo 2 la cantidad máxima de parámetros en nuestras funciones. En su mayoría cuentan con 1 parámetro.

Comentarios

Una de las razones por las cuales nos esforzamos en usar nombres nemotécnicos es evitar el uso de comentarios, en general los comentarios reflejan la incapacidad de expresar ideas. Por lo que en nuestro código no hay ningún comentario más que la especificación de los endpoints en nuestro código.

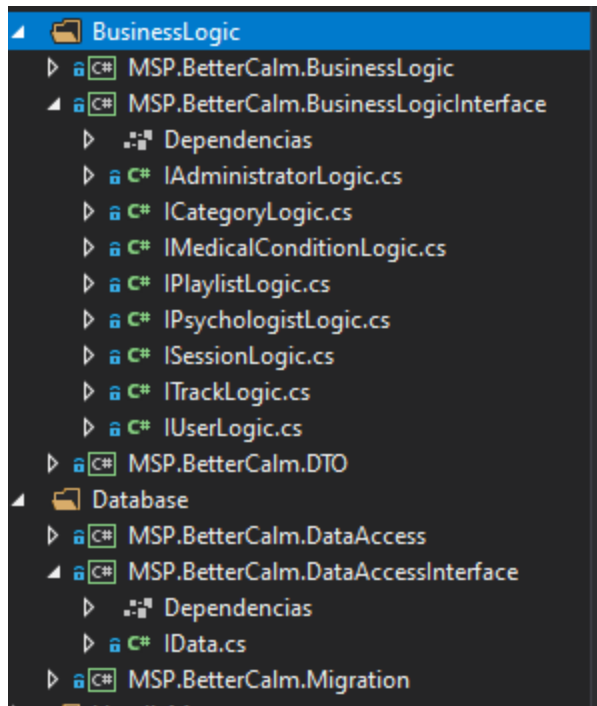
Formato

Cumplir con SRP hizo que el tamaño de nuestras clases sea nunca mayor a 200 líneas (a excepción de los tests). Ver imagen debajo(Última columna). Planteamos nuestras clases como una jerarquía piramidal de responsabilidades, donde en la parte superior se encuentran los métodos principales, y a medida que bajamos encontramos los métodos auxiliares utilizados por los métodos principales. Como autocrítica, debido a los nombres nemotécnicos y las excepciones, algunas líneas de código quedaron un poco más anchas de lo que nos hubiese gustado. Para finalizar, nos aseguramos de cumplir con todos los estándares de C# para el indentado de código.

Jerarquía	Indice de mantenimiento	Complejidad ciclomática	Profundidad de herencia	Acoplamiento de clases	Líneas de código fuente	Líneas de código ejecutable
Business\MSP.BetterCalm.Domain (Debug)	91	205	1	16	429	77
MSP.BetterCalm.Domain	91	205	1	16	429	77
Administrator	90	20	1	2	45	11
Category	93	11	1	4	24	4
CategoryTrack	90	13	1	3	31	8
Expertise	91	11	1	3	21	4
MedicalCondition	92	8	1	3	20	4
Meeting	92	15	1	4	22	4
meetingType	93	1	1	0	4	1
Playlist	90	23	1	4	47	11
PlaylistCategory	91	12	1	3	23	4
PlaylistTrack	91	12	1	3	24	4
Psychologist	91	21	1	5	37	7
Track	91	32	1	4	55	10
User	92	26	1	4	35	5
BusinessLogic\MSP.BetterCalm.BusinessLogic (Debug)	74	139	1	38	750	323
MSP.BetterCalm.BusinessLogic	74	139	1	38	750	323
AdministratorLogic	72	16	1	13	72	32
CategoryLogic	79	5	1	13	29	11
MedicalConditionLogic	89	5	1	8	22	6
PlaylistLogic	68	26	1	17	156	68
PsychologistLogic	68	46	1	20	227	113
SessionLogic	75	7	1	11	33	10
TrackLogic	68	26	1	16	160	72
UserLogic	75	8	1	7	27	11

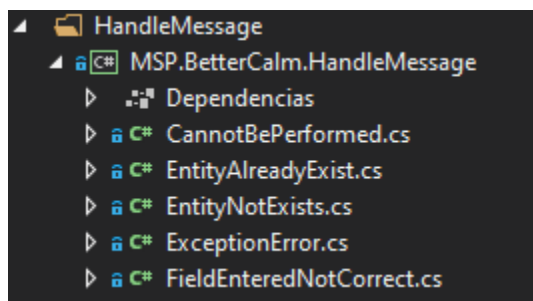
Objetos y estructuras de datos

Hicimos uso de interfaces para generar las dependencias entre interfaces y no entre implementaciones concretas. De esta forma nos aseguramos que nuestro proyecto sea extensible y mantenible en el futuro, reduciendo el acoplamiento entre módulos, aumentando la cohesión e intentando esconder implementaciones concretas. Como podemos ver en la imagen, utilizamos interfaces para la capa de acceso a datos y la lógica, siendo las mismas las más probables de sufrir cambios en el futuro. Mediante inyección de dependencias utilizamos las implementaciones concretas acordes a los requerimientos del obligatorio, pero sabemos que si en un futuro cambia la lógica de negocio o la forma de persistir los datos, nuestro sistema va a ser capaz de absorber y adaptar dichos cambios sin mayores riesgos ni esfuerzo (más que los que tenga la propia implementación de los nuevos cambios). De forma de reducir el acoplamiento entre clases, evitamos la concatenación de llamadas a métodos, también conocido como Train Wreck. Estos casos generalmente se dan cuando no hay suficiente encapsulamiento entre las clases. Intentamos evitar siempre estos casos.



Procesando errores

Creamos específicamente un paquete de manejo de excepciones en el cual manejamos las excepciones más comunes en nuestro proyecto. También utilizamos mensajes de error claros, junto con códigos HTTP acordes a cada respuesta del servidor. Estas decisiones fueron tomadas para brindarle al usuario la mayor información, lo más explicada posible y consistente a lo largo de toda la aplicación.



Estas excepciones son capturadas en la lógica y utilizadas en los controladores para que junto con el código de error más apropiado para cada caso, le brinden al usuario mensajes de error acordes a cada caso

```

[HttpGet()]
1 referencia | 1/1 pasando | Mauro Teixeira, Hace 16 días | 1 autor, 2 cambios
public IActionResult GetAll()
{
    try
    {
        return Ok(_categoryLogic.GetAll());
    }
    catch (EntityNotExists en)
    {
        return NotFound(en.MessageError());
    }
    catch (Exception e)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, e.Message);
    }
}

```

Evitamos pasar null como parámetro ni retornarlo.

Clases

SRP: Como mencionamos previamente, nuestras clases son por lo general pequeñas de máximo 150 líneas. El hecho de tener divididas las responsabilidades de cada una de ellas resultó en clases relativamente pequeñas, cumpliendo SRP.

OCP: El uso de interfaces y clases concretas que las implementan nos ayudan a que las mismas estén abiertas a la extensión y cerradas a la modificación.

Inversion dependency: Como también mencionamos previamente, nuestras clases dependen de abstracciones y no de implementaciones concretas. Esto facilita el unit testing por medio de mocks.

```

namespace MSP.BetterCalm.BusinessLogic
{
    4 referencias | Rodrigo Hirigoyen, Hace 2 días | 1 autor, 14 cambios
    public class PsychologistLogic : IPsychologistLogic
    {
        IData<Psychologist> _repository;
        IData<MedicalCondition> _repositoryMedicalCondition;
        2 referencias | Rodrigo Hirigoyen, Hace 3 días | 1 autor, 1 cambio
        public PsychologistLogic(IData<Psychologist> repository, IData<MedicalCondition> repositoryMedicalCondition)
        {
            _repository = repository;
            _repositoryMedicalCondition = repositoryMedicalCondition;
        }
    }
}

```


Evidencia de TDD y pruebas unitarias

Nuestro obligatorio fue desarrollado utilizando TDD, realizamos pruebas unitarias para nuestros proyectos, comenzando por escribir la prueba para cada uno de los métodos y properties, para luego escribir el código que pasa el test y finalmente refactorizar en caso que sea necesario. La técnica utilizada fue desde afuera hacia adentro, es decir, desde la base de datos hacia la api.

Librerías utilizadas para los tests y justificación:

- **NUnit**: Framework open source para .NET, lo utilizamos para escribir y probar nuestras pruebas unitarias.
- **Moq**: Librería que utilizamos para realizar los mocks. Nos fue realmente útil ya que en muchos casos queríamos probar un método puntual de una clase, pero para poder probarlo necesitábamos de un servicio brindado por una clase de otro proyecto. Entendemos que las pruebas unitarias (como su nombre lo dice) deben probar de a una cosa a la vez y cumplir FIRST. Por lo que requerimos de una librería que nos permita simular el comportamiento de dichas clases. Si bien se podría mockear cualquier comportamiento de nuestra solución, lo utilizamos para simular los comportamientos de nuestra capa de acceso a datos.

```
playlistList.Add(playlist);
repositoryPlaylist = new Mock<IData<Playlist>>();
repositoryCategory = new Mock<IData<Category>>();
repositoryTrack = new Mock<IData<Track>>();

repositoryPlaylist.Setup(r => r.GetAll()).Returns(playlistList);
repositoryCategory.Setup(r => r.GetAll()).Returns(categoryList);
repositoryTrack.Setup(r => r.GetAll()).Returns(trackList);

repositoryPlaylist.Setup(play => play.Get(0)).Returns(playlist);
repositoryPlaylist.Setup(play => play.Add(playlist));
playlistLogic = new PlaylistLogic(repositoryPlaylist.Object, repositoryCategory.Object, repositoryTrack.Object);
```

Esta librería nos fue realmente útil para poder simular las clases necesarias para realizar la inyección de dependencias que hace el Startup, pero en nuestros tests.

```
public void GetOnePlaylist()
{
    var mockPlaylist = new Mock<IPlaylistLogic>(MockBehavior.Strict);
    mockPlaylist.Setup(res => res.Get(playlistList[0].Id)).Returns(playlistList[0]);
    PlaylistController controller = new PlaylistController(mockPlaylist.Object);
    var result = controller.Get(playlistList[0].Id);

    mockPlaylist.VerifyAll();
    Assert.AreEqual(result.ToString(), new OkObjectResult("").ToString());
}
```

- **DbContextOptionsBuilder**: Si bien es un método provisto por la API de EFCore y no una librería de por sí, nos pareció pertinente destacar su uso ya que lo utilizamos para hacer las pruebas unitarias de nuestro DataAccess, permitiéndonos crear bases de datos en memoria y utilizar las mismas para testear nuestro métodos de los repositorios.

```
var options = new DbContextOptionsBuilder<BetterCalmContext>()
    .UseInMemoryDatabase(databaseName: "MSP.BetterCalmDatabase").Options;
var context = new BetterCalmContext(options);
listPlaylist.ForEach(cat => context.Add(cat));
context.SaveChanges();
repository = new PlaylistRepository(context);
var playlist = repository.Get(listPlaylist[0].Id);
context.Database.EnsureDeleted();
Assert.AreEqual(listPlaylist[0].Id, playlist.Id);
```

Justificación de funcionalidades con TDD

Si bien se realizaron test unitarios para el 90% de los métodos del obligatorio, realizamos la técnica de TDD de forma minuciosa para las funcionalidades que requería el obligatorio. Por decisión del equipo, siempre comenzamos con las clases del dominio, siguiendo por la capa de acceso a datos, luego la lógica para finalmente hacer los controladores de la api.











Como desarrolladores de este proyecto contamos con dos grandes beneficios al aplicar TDD. En primer lugar, documentamos nuestro trabajo:

- a. Las pruebas unitarias completas son documentación que evoluciona naturalmente con el sistema.
- b. La forma de documentar una clase es mediante la descripción de su comportamiento exterior, a través de un conjunto de casos de prueba
- c. La documentación está actualizada porque las pruebas deben correr.

En segundo lugar, facilitar el cambio.


- a. Para nosotros como desarrolladores que nos enfrentamos a una base completa de casos de prueba es más fácil introducir cambios sin introducir defectos.
- b. Ejecutar los mismos casos de prueba que sirvieron como criterio de satisfacción de las clases ya construidas da seguridad.
- c. Aumenta la confianza de nosotros al introducir el cambio y permite hacerlo en forma rápida.

Navegar entre las playlist

Merge pull request #5 from mauroteix/TestPlayList ...
 RodrigoHirigoyen committed 24 days ago
RegisterCategory Refactor
 RodrigoHirigoyen committed 24 days ago
RegisterTrack Refactor
 RodrigoHirigoyen committed 24 days ago
ImageEmpty Refactor
 RodrigoHirigoyen committed 24 days ago
RegisterImage Refactor
 RodrigoHirigoyen committed 24 days ago
DescriptionLengthFalse Refactor
 RodrigoHirigoyen committed 24 days ago
DescriptionLengthTrue Refactor
 RodrigoHirigoyen committed 24 days ago
RegisterDescription Refactor
 RodrigoHirigoyen committed 24 days ago
RegisterName Refactor
 RodrigoHirigoyen committed 24 days ago
NameEmpty Refactor
 RodrigoHirigoyen committed 24 days ago

Primero comenzamos por el dominio, si bien en los commit se puede observar que se comiteo directamente el refactor(Por un tema de practicidad), el proceso del mismo fue, red, green y por último el refactor que es el que se encuentra en la imagen.

Merge pull request [#23](#) from [mauroteix/TestPlaylistRepository](#) ...

 [mauroteix](#) committed 18 days ago

DeleteOnePlaylist Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 18 days ago

UpdatePlaylist Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 18 days ago


GetOnePlaylist Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 18 days ago










AddPlaylist Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 18 days ago



Merge pull request [#22](#) from [mauroteix/DataAccess](#) ...






 [mauroteix](#) committed 18 days ago

Luego continuamos con el repositorio de playlist(DataAcess), se realizaron los pasos(RED, GREEN, REFACTOR) como fueron mencionados anteriormente.

UpdatePlaylist Refactor  mauroteix authored and mauroteix committed 10 days ago
UpdatePlaylist Refactor  mauroteix authored and mauroteix committed 10 days ago
DeletePlaylistNotExists Refactor  mauroteix authored and mauroteix committed 10 days ago
DeletePlaylistIdNegative Refactor  mauroteix authored and mauroteix committed 10 days ago
DeletePlaylistOk Refactor  mauroteix authored and mauroteix committed 10 days ago
DeletePlaylist Refactor  mauroteix authored and mauroteix committed 10 days ago
DeletePlaylist Refactor  mauroteix authored and mauroteix committed 10 days ago
AddPlaylistError Refactor  mauroteix authored and mauroteix committed 10 days ago
AddOnePlaylist Refactor  mauroteix authored and mauroteix committed 10 days ago
Merge pull request #40 from mauroteix/PlayListLogic ...

En este caso seguimos con la capa lógica, aplicando TDD.

Update Refactor  mauroteix authored and mauroteix committed 5 days ago
Add Refactor  mauroteix authored and mauroteix committed 5 days ago


GetPlaylistsByCategory Refactor  mauroteix authored and mauroteix committed 7 days ago
ToEntity Refactor  mauroteix authored and mauroteix committed 7 days ago
ExistPlaylist added  mauroteix authored and mauroteix committed 7 days ago
ValidateCategoriesId added  mauroteix authored and mauroteix committed 7 days ago
ValidatePlaylist added  mauroteix authored and mauroteix committed 7 days ago

Por último terminamos con el TDD del controller, siguiendo los mismos pasos mencionados anteriormente. Esta secuencia se repite en todas las funcionalidades creadas de nuestro sistema. Esto permitió diseñar nuestro proyecto guiado mediante TDD, por lo que facilitó considerablemente el desarrollo.

Dar de alta y baja contenido reproducible

Al igual que con la playlist, comenzamos haciendo los tests para las properties de la entidad track.

Merge pull request #10 from mauroteix/TestPlaylistTrack ...


 mauroteix committed 20 days ago

RegisterPlaylistTrack Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 20 days ago

- Commits on Apr 13, 2021

RegisterPlaylistTrack Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 22 days ago

RegisterTrackId Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 22 days ago


RegisterTrack Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 22 days ago

RegisterPlaylistId Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 22 days ago

Merge pull request #1 from mauroteix/TestTrack ...





 mauroteix committed 25 days ago

SoundEmpty Refactor







 Mauro Teixeira authored and Mauro Teixeira committed 25 days ago

RegisterSound Refactor

 Mauro Teixeira authored and Mauro Teixeira committed 25 days ago

Merge pull request #21 from mauroteix/TestTrack ...  mauroteix committed 18 days ago
HoursEmpty Refactor  Mauro Teixeira authored and Mauro Teixeira committed 18 days ago
RegisterMinSeconds Refactor  Mauro Teixeira authored and Mauro Teixeira committed 18 days ago
RegisterHour Refactor  Mauro Teixeira authored and Mauro Teixeira committed 18 days ago





Luego continuamos con el TDD del repositorio de track

Merge pull request #24 from mauroteix/TrackRepository ...  RodrigoHirigoyen committed 17 days ago
DeleteTrack Refactor  RodrigoHirigoyen committed 17 days ago
DeleteTrack Refactor  RodrigoHirigoyen committed 17 days ago
UpdateTrack Refactor  RodrigoHirigoyen committed 17 days ago
GetTrack Refactor  RodrigoHirigoyen committed 17 days ago
AddTrack Refactor  RodrigoHirigoyen committed 17 days ago

Y continuamos con la lógica de track

Merge pull request #42 from mauroteix/TestTrackLogic <div> mauroteix committed 9 days ago </div>
AddTrackWithoutCategoryFail Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
AddTrackWithoutCategoryFail Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
AddTrackSoundEmpty Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
GetTrackNotExist Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
DeleteTrackNotExist Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
AddTrackNameEmpty Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
DeleteTrack Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
AddTrackOk Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
GetTrack Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>
CategoryTrackEmpty Refactor <div> mauroteix authored and mauroteix committed 9 days ago </div>

Por último hicimos TDD en el controlador de track

AddTrackError Refactor  mauroteix authored and mauroteix committed 9 days ago
AddOneTrack Refactor  mauroteix authored and mauroteix committed 9 days ago
GetOneTrackError Refactor  mauroteix authored and mauroteix committed 9 days ago
GetOneTrack Refactor  mauroteix authored and mauroteix committed 9 days ago

Mantenimiento de un psicólogo con su respectiva información

Primero comenzamos por el dominio del mismo


Psy/MedicalCondition Refactor  RodrigoHirigoyen committed 11 days ago
Psy/MedicalCondition Refactor  RodrigoHirigoyen committed 11 days ago
RegisterPsyId Refactor  RodrigoHirigoyen committed 11 days ago
RegisterPsy Refactor  RodrigoHirigoyen committed 11 days ago
RegisterMedicalConditionId Refactor  RodrigoHirigoyen committed 11 days ago
RegisterMedicalCondition Refactor  RodrigoHirigoyen committed 11 days ago

Luego continuamos por el repositorio del mismo

Merge pull request [#50](#) from mauroteix/PsyRepository ...

 RodrigoHirigoyen committed 6 days ago


DeletePsy Refactor

 RodrigoHirigoyen committed 6 days ago


UpdatePsy Refactor

 RodrigoHirigoyen committed 6 days ago


GetOnePsy Refactor

 RodrigoHirigoyen committed 6 days ago


AddPsy Refactor

 RodrigoHirigoyen committed 6 days ago













Create classs/Test

 RodrigoHirigoyen committed 6 days ago

Create classs/Test

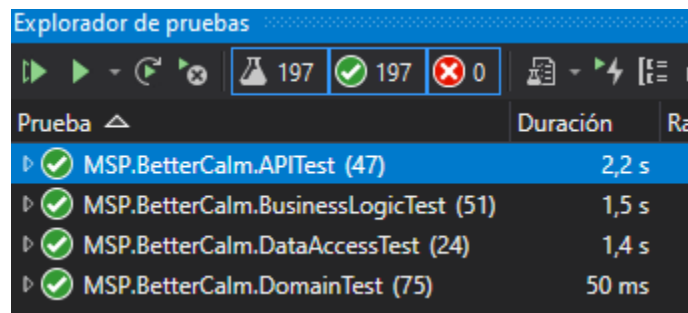
 RodrigoHirigoyen committed 6 days ago

Y por último podemos ver la lógica y el controller realizando TDD.

Expertise Equals psy  RodrigoHirigoyen committed 4 days ago
GetAll psy  RodrigoHirigoyen committed 4 days ago
Update psy  RodrigoHirigoyen committed 4 days ago
Exist psy  RodrigoHirigoyen committed 4 days ago
Validate psy  RodrigoHirigoyen committed 4 days ago
Star up  RodrigoHirigoyen committed 4 days ago
AddPsyErro Refactor  RodrigoHirigoyen committed 4 days ago
UpdatePsy Refactor  RodrigoHirigoyen committed 4 days ago
DeletePsyNotExist Refactor  RodrigoHirigoyen committed 4 days ago
DeletePsyNegative Refactor  RodrigoHirigoyen committed 4 days ago
DeletePsyOK Refactor  RodrigoHirigoyen committed 4 days ago
GetPsy Refactor  RodrigoHirigoyen committed 4 days ago

Tests Unitarios y cobertura total de pruebas

En total se realizaron 197 tests unitarios y se obtuvo una alta cobertura de los métodos en los distintos paquetes. Al momento de organizar los tests creímos que la mejor forma de hacerlo era crear varios proyectos de pruebas y dentro del mismo, crear las clases para cada uno de los proyectos a testear, de esta manera si hoy se desea hacer mantenimiento del mismo va a ser sencillo. Tenemos 4 proyectos de test, MSP.BetterCalm.APITest, MSP.BetterCalm.DomainTest, MSP.BetterCalm.BusinessLogicTest y MSP.BetterCalm.DataAccessTest para realizar los test de cada una de las clases. Como mencionamos anteriormente para el desarrollo de los tests y validación de los mismos utilizamos en algunos casos utilizamos Assert.Are Equals y en otros casos Assert.Throws para verificar el lanzamiento de excepciones.



En cuanto a la cobertura de pruebas, obtuvimos un porcentaje de 90,71% en promedio cumpliendo con lo solicitado. En la api nos muestra un porcentaje menor debido (Startup y program). Si bien realizamos TDD para el desarrollo de la aplicación, a último momento surgieron algunos cambios que no nos permitieron realizar las pruebas para los mismos, impactando la cobertura del paquete de BusinessLogic .

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
mauro_LAPTOP-00S4OKL9 2021...	400	9,29 %	3908	90,71 %
msp.bettercalm.api.dll	100	32,36 %	209	67,64 %
msp.bettercalm.apitest.dll	0	0,00 %	16	100,00 %
msp.bettercalm.businesslog...	140	17,37 %	666	82,63 %
msp.bettercalm.businesslog...	23	2,36 %	952	97,64 %
msp.bettercalm.dataaccess...	26	4,04 %	617	95,96 %
msp.bettercalm.dataaccesst...	1	0,18 %	570	99,82 %
msp.bettercalm.domain.dll	105	23,92 %	334	76,08 %
msp.bettercalm.domaintest...	0	0,00 %	514	100,00 %
msp.bettercalm.dto.dll	2	12,50 %	14	87,50 %
msp.bettercalm.handlemess...	3	15,79 %	16	84,21 %