

Pruebas de Unidad y Cobertura de Código

Índice

1. Objetivos

1.1. Objetivo general

Implementar Pruebas de Unidad (Unit Tests) para medir la cobertura de las mismas (Code Coverage) en una aplicación real.

1.2. Objetivos específicos

- Seleccionar el framework adecuado para implementar las pruebas de unidad en la aplicación seleccionada
- Seleccionar la herramienta de cobertura más adecuada para medir la cobertura de código fuente de la aplicación seleccionada
- Implementar pruebas de unidad a partir de la identificación de casos de prueba utilizando las técnicas de Path Coverage
- Llegar a 70% de cobertura de código de línea (Statement coverage) en la aplicación seleccionada

2. Descripción de la Aplicación bajo prueba (AUT)

2.1. Contexto de la aplicación

La aplicación consiste en una API desarrollada en .NET, que cuenta con 2 recursos, atletas y disciplinas, así gestionando información sobre el deporte atletismo.

Una disciplina contiene un identificador único, un nombre, las reglas de la misma, la fecha en la que se creó, el récord mundial femenino y masculino, una lista de atletas y opcionalmente una imagen que se puede subir al momento de crear una disciplina y se guarda como archivo estático en la aplicación.

```
{  
  "id": 1,  
  "name": "200M",  
  "rules": "200M has to be run in a straight line plus curve",
```

```
    "creationDate": "1990-08-21T00:00:00",  
    "femaleWorldRecord": 20.85,  
    "maleWorldRecord": 20.58,  
    "athletes": []  
}
```

Un atleta pertenece a una disciplina, y una disciplina puede estar relacionada a varios atletas (relación n:1). Un atleta contiene un identificador único, nombre, nacionalidad, un booleano para indicar si está o no activo, número de competencias en las que estuvo, género, mejor marca personal, mejor marca de temporada, fecha de nacimiento y puntos que tiene en el ranking de la disciplina a la que pertenece.

```
{  
  "name": "Femke Bol",  
  "nationality": "Netherlands",  
  "isActive": true,  
  "numberOfCompetitions": 9,  
  "gender": "M",  
  "personalBest": 53.05,  
  "seasonBest": null,  
  "birthDate": "1995-04-28T00:00:00",  
  "points": 250  
}
```

La API cuenta con 2 roles de seguridad: administrador y usuario. La única acción que puede realizar el usuario es ver la lista de disciplinas; todo lo demás requiere de permiso de administrador (loguearse como administrador).

La API cuenta con los siguientes endpoints:

DISCIPLINAS:

- GetDisciplines: (GET) ruta: <http://localhost:5077/api/disciplines>
obtiene la lista de todas las disciplinas de la bd con sus datos respectivos
- GetDiscipline: (GET) ruta: <http://localhost:5077/api/disciplines/{id}>
obtiene la información de una disciplina especificada por el id
- DeleteDiscipline: (DELETE) ruta: <http://localhost:5077/api/disciplines/{id}>
Elimina una disciplina de la bd, a través de su id
- UpdateDiscipline: (PUT) ruta: <http://localhost:5077/api/disciplines/{id}>
Actualiza datos de una disciplina existente
- CreateDiscipline: (POST) ruta: <http://localhost:5077/api/disciplines>
Crea una disciplina a partir del BODY del método POST

- Race: (GET) ruta: [http://localhost:5077/api/disciplines/{id}/race/?gender=\[M/F\]&podium=\[true/false\]](http://localhost:5077/api/disciplines/{id}/race/?gender=[M/F]&podium=[true/false])
Simula una carrera de una disciplina con un determinado género y con opción de devolver solamente el podio (3 primeros lugares), o resultados de todos los atletas que compitieron.
- GetWorldRankings: (GET) ruta: [http://localhost:5077/api/disciplines/{id}/worldRankings/?gender=\[M/F/ALL\]](http://localhost:5077/api/disciplines/{id}/worldRankings/?gender=[M/F/ALL])
Obtiene el ranking mundial de una disciplina determinada, para hombres, mujeres o ambos, clasificando y ordenando a los atletas por los puntos que tienen.

ATLETAS

- GetAthletes: (GET) ruta: <http://localhost:5077/api/disciplines/{id}/athletes>
Obtiene la lista de atletas que pertenecen a una determinada disciplina
- GetAthlete: (GET) ruta: <http://localhost:5077/api/disciplines/{id}/athletes/{id}>
Obtiene información de un atleta determinado de una disciplina
- CreateAthlete: (POST) ruta: <http://localhost:5077/api/disciplines/{id}/athletes>
Crea un atleta en una disciplina determinada, a través del BODY de la del método.
- UpdateAthlete: (PUT) ruta: <http://localhost:5077/api/disciplines/{id}/athletes/{id}>
Actualiza datos correspondientes a un atleta de una disciplina
- DeleteAthlete: (DELETE) ruta: <http://localhost:5077/api/disciplines/{id}/athletes/{id}>
Elimina un atleta de una disciplina

SEGURIDAD

- CreateRoleAsync: (POST) ruta: <http://localhost:5077/api/auth/role>
Crea un rol de usuario
- CreateUserRoleAsync: (POST) ruta: <http://localhost:5077/api/auth/User>
Asigna un rol determinado a un usuario determinado
- LoginAsync: (POST) ruta: <http://localhost:5077/api/auth/login>
Realiza el login de un usuario existente
- RegisterAsync: (POST) ruta: <http://localhost:5077/api/auth/User>
Registra un nuevo usuario

2.1.1. Cómo hacer correr el proyecto

Prerrequisitos:

- Tener instalado Visual Studio con soporte a aplicaciones REST de dotnetcore
- Tener instalada la versión 3.1 del framework donetcore

Para la base de datos:

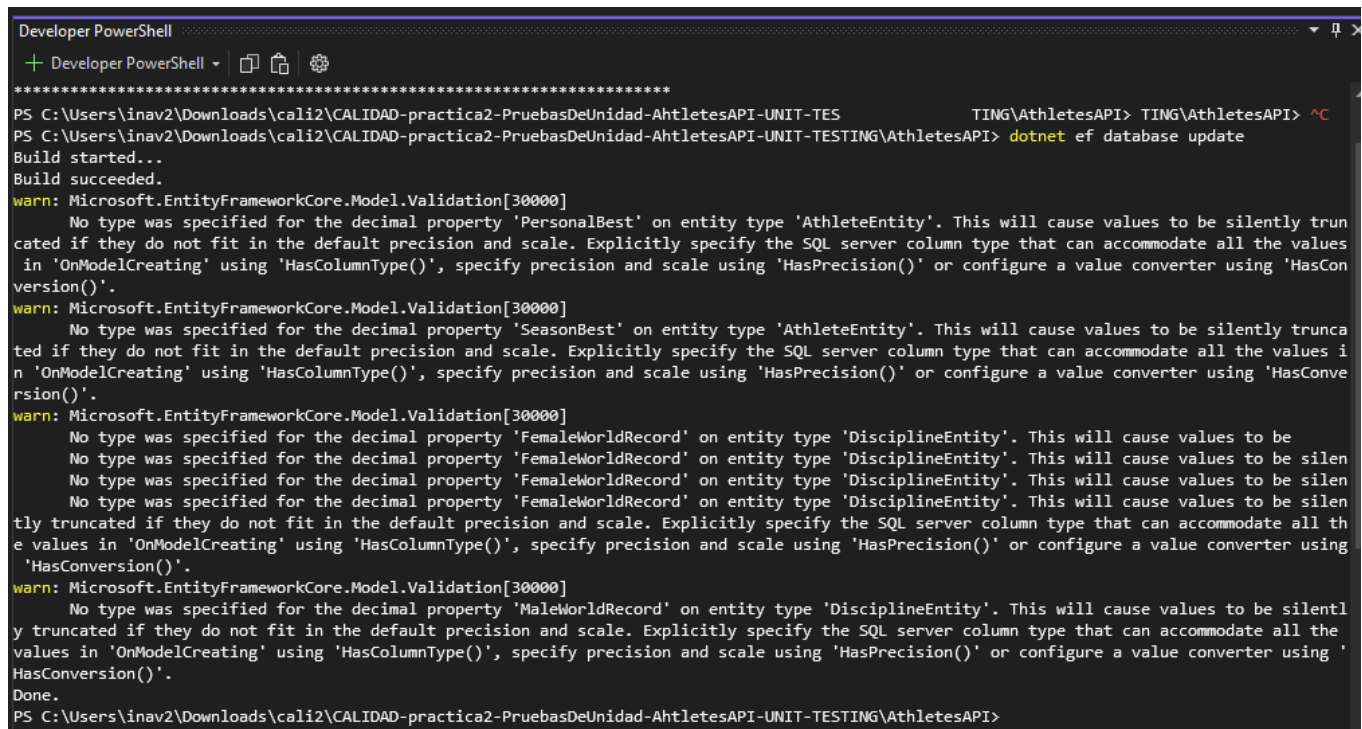
Tener instalado localdb para Visual Studio o cambiar la cadena de conexión en `appsettings.json`: `ConnectionStrings: AthleteApi` para utilizar una base de datos externa

```
"ConnectionStrings": {
  "AthleteApi": "Data Source=
(localdb)\MSSQLLocalDB;Database=AthleteAPI;Trusted_Connection=True;"
}
```

Hacer correr las migraciones de base de datos con los siguientes comandos:

```
dotnet tool install --global dotnet-ef
dotnet ef database update
```

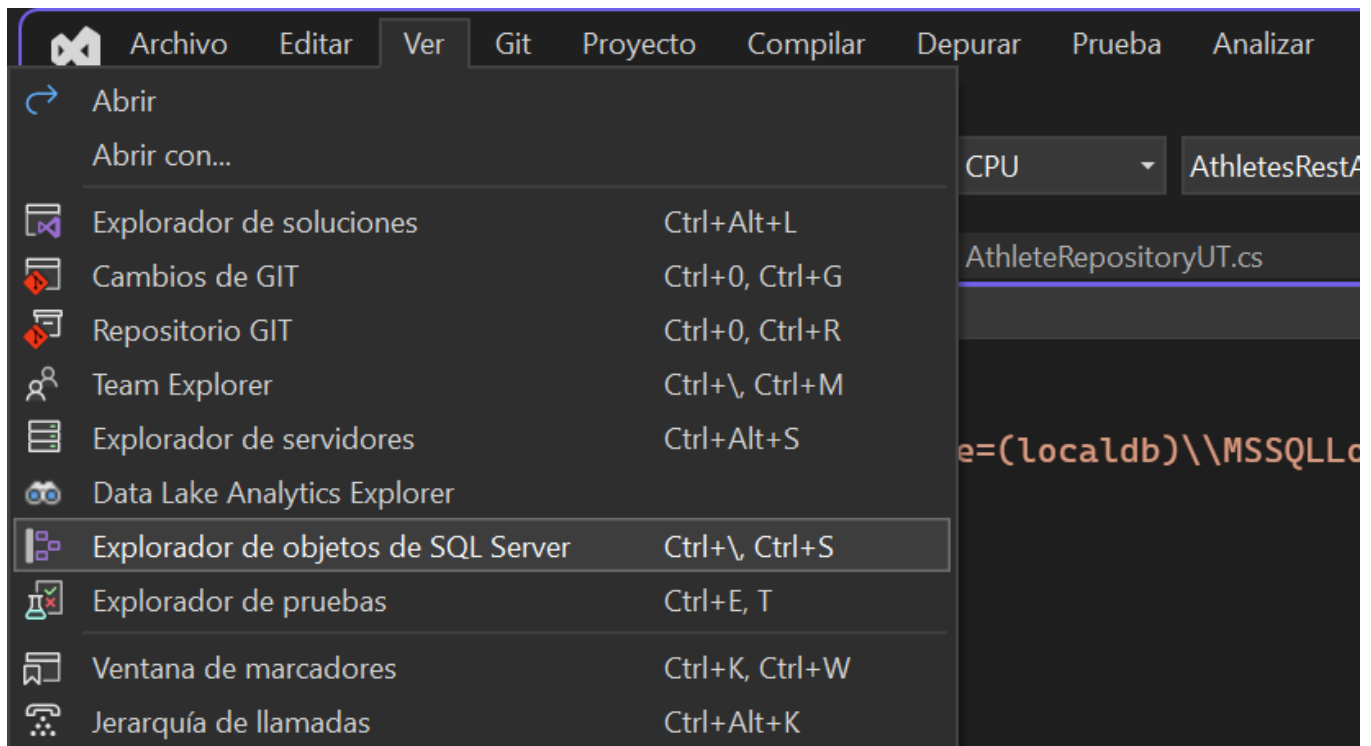
desde la terminal, dentro del proyecto, es decir en la ruta `\CALIDAD-practica2-PruebasDeUnidad-AhtletesAPI-UNIT-TESTING\AthletesAPI` como se muestra en la siguiente imagen:



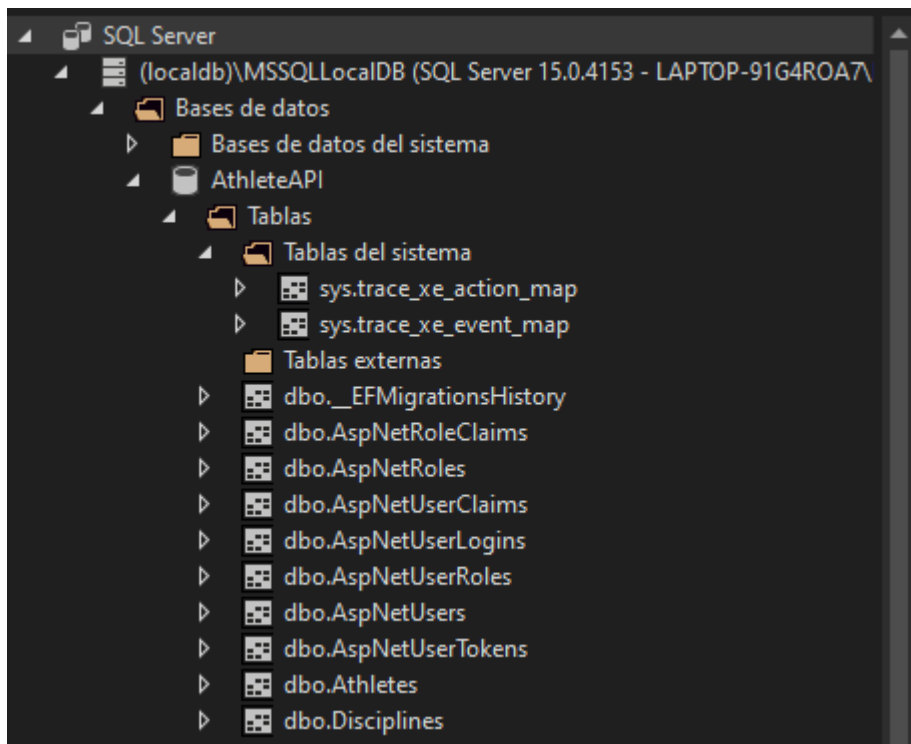
```
Developer PowerShell
+ Developer PowerShell | [ ] [ ] [ ]
*****
PS C:\Users\inav2\Downloads\cali2\CALIDAD-practica2-PruebasDeUnidad-AhtletesAPI-UNIT-TESTING\AthletesAPI> dotnet ef database update
Build started...
Build succeeded.
warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal property 'PersonalBest' on entity type 'AthleteEntity'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType()', specify precision and scale using 'HasPrecision()' or configure a value converter using 'HasConversion()'.
warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal property 'SeasonBest' on entity type 'AthleteEntity'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType()', specify precision and scale using 'HasPrecision()' or configure a value converter using 'HasConversion()'.
warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal property 'FemaleWorldRecord' on entity type 'DisciplineEntity'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType()', specify precision and scale using 'HasPrecision()' or configure a value converter using 'HasConversion()'.
warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal property 'MaleWorldRecord' on entity type 'DisciplineEntity'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType()', specify precision and scale using 'HasPrecision()' or configure a value converter using 'HasConversion()'.
Done.
PS C:\Users\inav2\Downloads\cali2\CALIDAD-practica2-PruebasDeUnidad-AhtletesAPI-UNIT-TESTING\AthletesAPI>
```

Es normal que salgan warnings, pero al final debe salirnos Done.

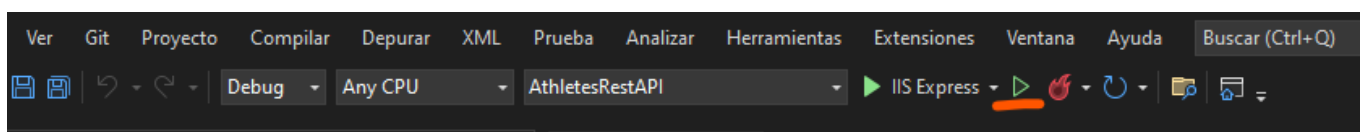
Una vez corridos los comandos, si se quiere ver la base de datos, entrar a Ver-> Explorador de objetos de SQL Server:



y se deberían mostrar las tablas de la base de datos AthleteAPI:

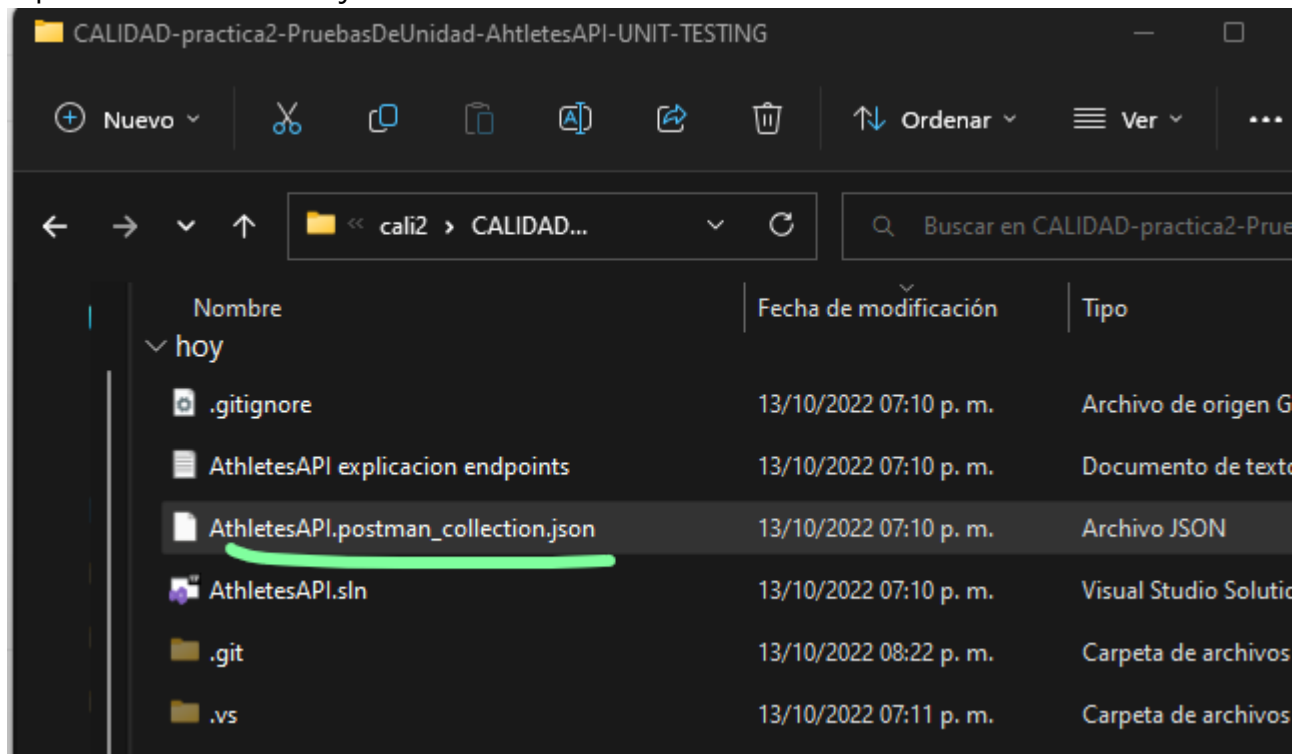


Una vez realizados los pasos anteriores satisfactoriamente, se puede proceder a correr la aplicación a través de Visual Studio:

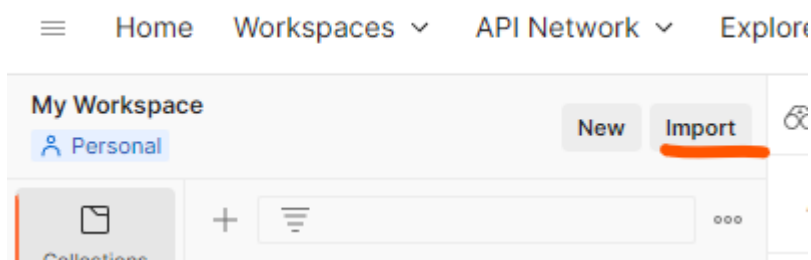


2.1.2. Cómo funciona

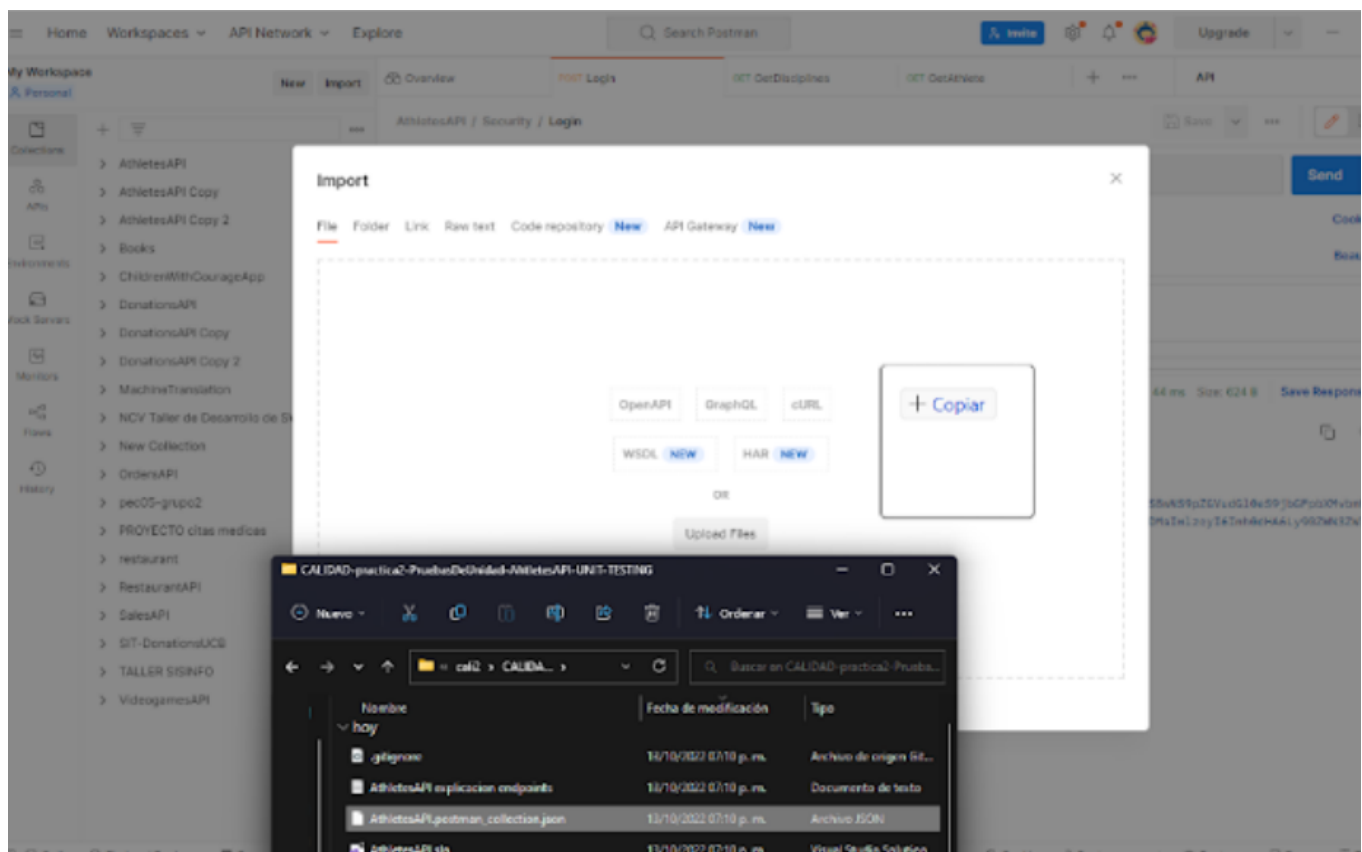
Para probar el funcionamiento de la API, se provee la documentación de Postman en el repositorio, en formato json:



Para importar el archivo desde Postman, hacer click en import:



Después se debe arrastrar el archivo e importar



Import



Select files to import · 1/1 selected

NAME	FORMAT	IMPORT AS
AthletesAPI	Postman Collection v2.1	Collection

Cancel

Import

La aplicación debe estar corriendo para probar desde postman. Se debe setear la variable de entorno athletesAPI a <http://localhost:5077> desde postman, o bien reemplazar la variable por la

ruta, es decir reemplazar

AthletesAPI / Disciplines / GetDisciplines

GET	▼	{{athletesAPI}}/api/disciplines
-----	---	---------------------------------

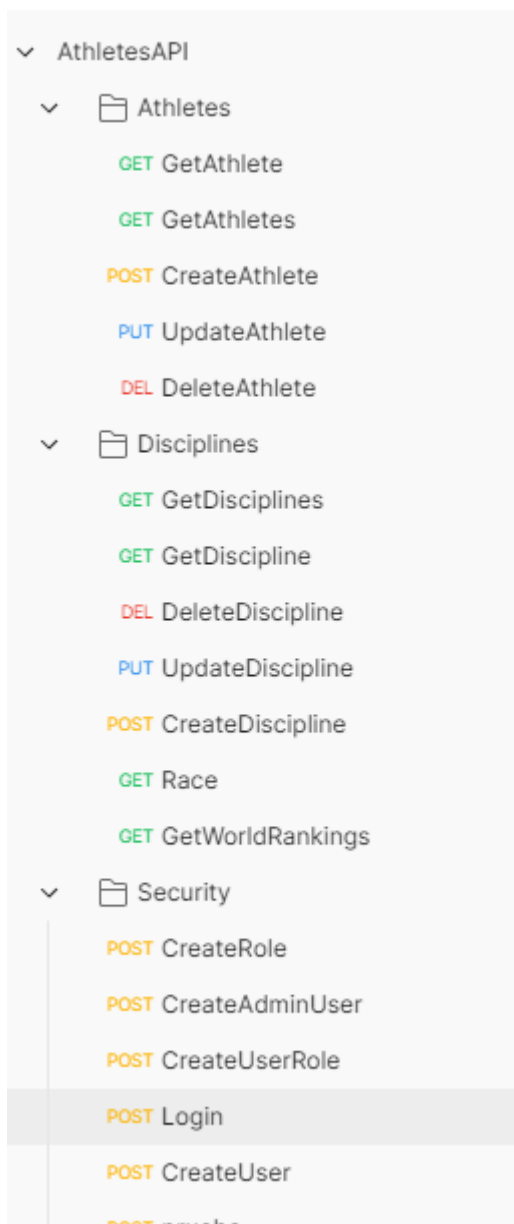
por

AthletesAPI / Disciplines / GetDisciplines

GET	▼	http://localhost/5077/api/disciplines
-----	---	---------------------------------------

en todos los endpoints.

Se podrán ver todos los endpoints posibles y proceder a probar:



Cabe destacar que para probar cualquier endpoint fuera del GetDisciplines, requiere login de administrador para obtener el token. Para esto se puede usar un usuario que ya tiene asignado el rol de administrador para el login:

GET ▼ http://localhost:5077/api/disciplines

Params Authorization ● Headers (7) Body Pre-request Script Tests Settings

Type Bearer To... ▼

The authorization header will be automatically generated when you send the request. Learn more about

Token eyJh...

Heads up! These parameters hold sensitive data. To keep your environment secure, we recommend using variables. Learn more

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ ↺

```
1 [
2   {
3     "id": 1,
4     "name": "100M",
5     "rules": "Short race",
6     "creationDate": "1992-10-02T00:00:00",
7     "femaleWorldRecord": 10.54,
8     "maleWorldRecord": 9.58,
9     "athletes": [],
10    "imagePath": null
11  }
12 ]
```

2.2. TechStack

2.2.1. Tecnologías usadas

El proyecto está codificado en c#, a través del framework .NETCore, versión 3.1.

Para el manejo de la base de datos, se utilizaron los paquetes NuGet de EntityFrameworkCore:

- microsoft.entityframeworkcore
- microsoft.entityframeworkcore.design
- microsoft.entityframeworkcore.sqlserver

Estos 3 paquetes permiten seguir el enfoque "Code First", permitiendo mapear las clases de entidad a la base de base de datos.

El paquete automapper.extensions.microsoft.dependencyinjection permite mapear una entidad (representación de un registro de base de datos) a un modelo (representación de la entidad en la aplicación) y viceversa.

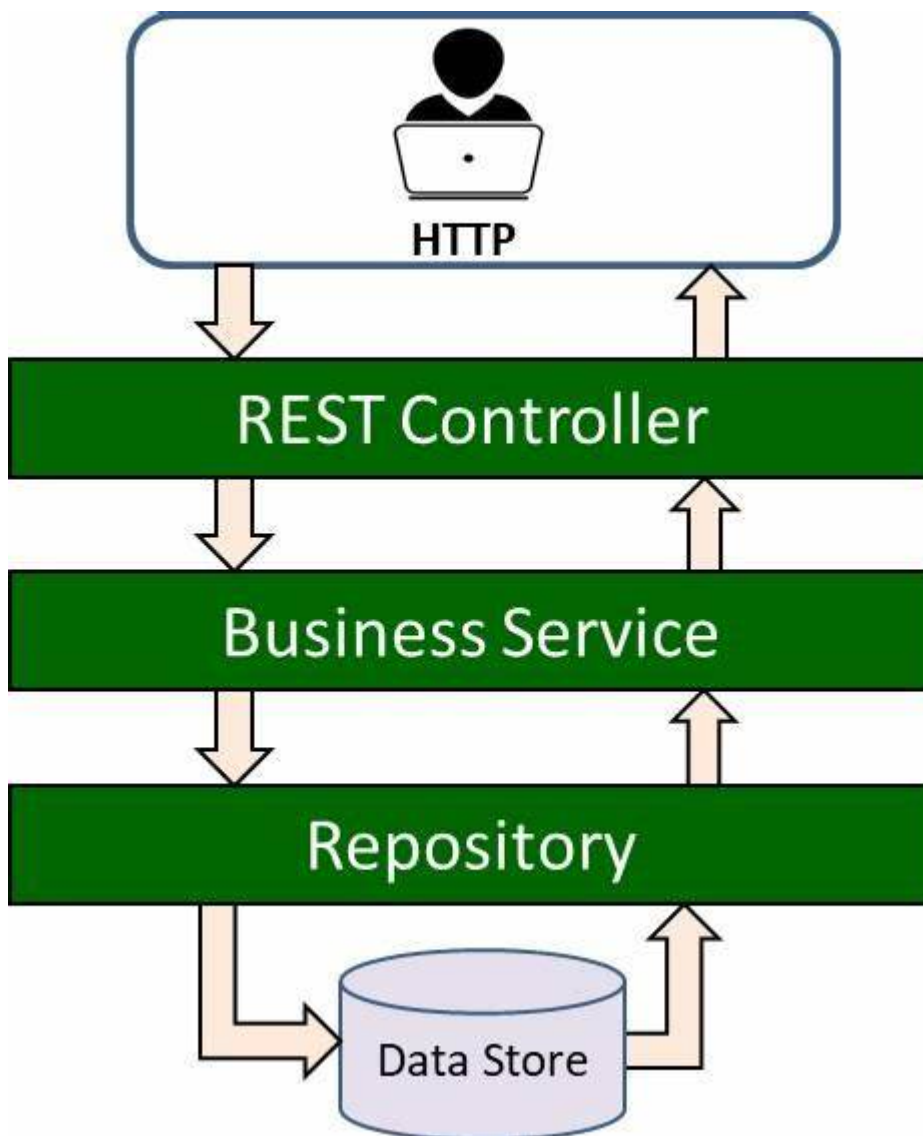
La API tiene implementada seguridad, a través del manejo de tokens (JWT), con un tiempo

de vida de 2 horas, implementada a través de los paquetes NuGet proporcionados por el framework de NetCore, específicamente:

- microsoft.aspnetcore.authentication.jwtbearer (para el manejo de tokens)
- microsoft.aspnetcore.identity (para el manejo de usuarios y roles)
- microsoft.aspnetcore.identity.entityframeworkcore (para el manejo de las tablas de usuario y rol en base de datos, creadas a partir del código fuente)

2.2.2. Estructura del proyecto

El proyecto tiene una estructura de MVC (modelo vista controlador), y cuenta con 3 capas:



CONTROLADORES: 1 por cada recurso, encargados de manejar los endpoints y manejar los datos en formato JSON

SERVICIOS: 1 por cada recurso, se encargan de toda la lógica de negocio y comunicación entre controlador y repositorio

REPOSITORIO: 1 para toda la aplicación, se encarga de interactuar con la base de datos, recuperando e insertando información

2.3. Scope de la AUT

La aplicación bajo prueba tiene un total de:

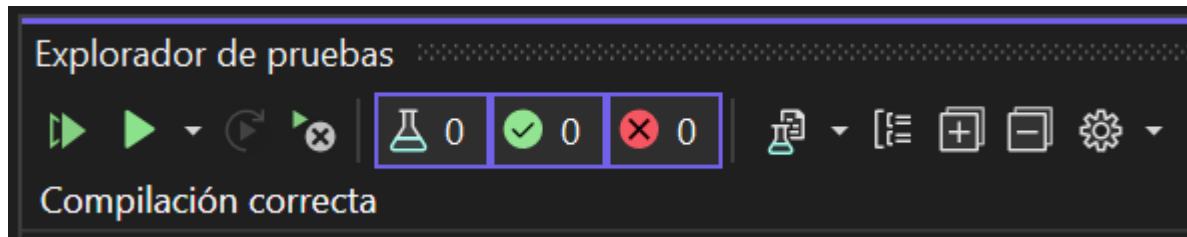
- 31 archivos de clase que pueden ser probados
- 1023 líneas de código que pueden ser probadas
- 57 métodos que pueden ser probados

El total de líneas (2137) de la aplicación no puede ser probada, o al menos no tiene sentido probar ya que esta contiene archivos de migraciones de base de datos, archivos de configuración como `Program.cs` o `Startup.cs`. Estos archivos fueron excluidos para el diseño de las pruebas y el calculo de la cobertura.

La aplicación bajo prueba, incluyendo solamente lo que se puede probar tiene un total de:

- 28 archivos de clase que pueden ser probados
- 943 líneas de código que pueden ser probadas
- 57 métodos que pueden ser probados

3. Estado inicial



Como se observa en la imagen de arriba, al inicio el proyecto no contaba con ninguna prueba de unidad implementada. Por evidentes razones la cobertura era de un 0%.

Information	Line coverage	Branch coverage	Method coverage
Parser: Cobertura	Covered lines: 0	Covered branches: 0	Method coverage is only available for sponsors.
Assemblies: 1	Uncovered lines: 3082	Total branches: 184	Upgrade to PRO version
Classes: 38	Coverable lines: 3082	Branch coverage: 0%	
Files: 44	Total lines: 4697		
Coverage date: 14/10/2022 - 15:31:47	Line coverage: 0%		

Name	Line coverage					Branch coverage		
	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage
AthletesRestAPI	0	3082	3082	4697	0%	0	184	0%
AthletesAPI.Program	0	8	8	26	0%	0	0	
AthletesAPI.Startup	0	69	69	147	0%	0	2	0%
AthletesAPI.WeatherForecast	0	4	4	15	0%	0	0	
AthletesRestAPI.Controllers.AthletesController	0	76	76	202	0%	0	6	0%
AthletesRestAPI.Controllers.AuthController	0	43	43	93	0%	0	16	0%
AthletesRestAPI.Data.AthleteDbContext	0	14	14	62	0%	0	0	
AthletesRestAPI.Data.Entity.AthleteEntity	0	12	12	39	0%	0	0	
AthletesRestAPI.Data.Entity.DisciplineEntity	0	8	8	28	0%	0	0	
AthletesRestAPI.Data.Repository.AthleteRepository	0	80	80	365	0%	0	34	0%
AthletesRestAPI.Exceptionss.IncompleteRequestException	0	1	1	13	0%	0	0	
AthletesRestAPI.Exceptionss.InvalidElementOperationException	0	1	1	12	0%	0	0	
AthletesRestAPI.Exceptionss.NoAthletesToRaceException	0	1	1	12	0%	0	0	
AthletesRestAPI.Exceptionss.NotFoundElementException	0	1	1	12	0%	0	0	
AthletesRestAPI.Migrations.AthleteDbContextModelSnapshot	0	328	328	362	0%	0	0	
AthletesRestAPI.Migrations.AthletesPoints	0	334	334	381	0%	0	0	
AthletesRestAPI.Migrations.enableCascadeDeleting	0	105	105	140	0%	0	0	
AthletesRestAPI.Migrations.imagesAthletes	0	340	340	387	0%	0	0	
AthletesRestAPI.Migrations.ImagesDiscipline	0	337	337	384	0%	0	0	
AthletesRestAPI.Migrations.InitialMigration	0	138	138	177	0%	0	0	
AthletesRestAPI.Migrations.security	0	507	507	574	0%	0	0	
AthletesRestAPI.Models.AthleteFormModel	0	1	1	13	0%	0	0	
AthletesRestAPI.Models.AthleteModel	0	12	12	43	0%	0	0	
AthletesRestAPI.Models.DisciplineFormModel	0	1	1	13	0%	0	0	
AthletesRestAPI.Models.DisciplineModel	0	8	8	26	0%	0	0	
AthletesRestAPI.Models.RaceAthleteModel	0	6	6	20	0%	0	0	
AthletesRestAPI.Models.RaceInfoModel	0	15	15	34	0%	0	0	
AthletesRestAPI.Models.Security.CreateRoleViewModel	0	2	2	16	0%	0	0	
AthletesRestAPI.Models.Security.CreateUserRoleViewModel	0	2	2	13	0%	0	0	
AthletesRestAPI.Models.Security.LoginViewModel	0	2	2	20	0%	0	0	
AthletesRestAPI.Models.Security.RegisterViewModel	0	3	3	24	0%	0	0	
AthletesRestAPI.Models.Security.UserManagerResponse	0	4	4	15	0%	0	0	
AthletesRestAPI.Models.ShortAthleteModel	0	8	8	23	0%	0	0	
AthletesRestAPI.Services.AthleteService	0	81	81	195	0%	0	12	0%
AthletesRestAPI.Services.DisciplineService	0	261	261	354	0%	0	82	0%
AthletesRestAPI.Services.FileService	0	17	17	33	0%	0	2	0%
AthletesRestAPI.Services.Security.UserService	0	136	136	203	0%	0	22	0%
DisciplinesRestAPI.Controllers.DisciplinesController	0	103	103	190	0%	0	8	0%
RestaurantRestAPI.Data.AutomapperProfile	0	13	13	31	0%	0	0	

4. Estado final

Al finalizar el diseño e implementación de las pruebas de unidad se realizaron **89 Unit tests**

Explorador de pruebas			
<div> ▶ ▶ ↺ ✖ 🧪 89 ✅ 89 ❌ 0 🔍 ⌵ ⌶ ⌵ ⚙️ </div>			
Serie de pruebas finalizada: 89 pruebas (Superadas: 89; Con errores: 0; Omitidas: 0) ejecutadas en 1,7 s			
Prueba	Duración	Rasgos	Mensaje de error
📁 ✅ UnitTesting (89)	2,6 s		
▶️ ✅ UnitTesting (1)	4 ms		
📁 ✅ UnitTesting.AthleteRepositoryUT (6)	1 s		
▶️ ✅ AthleteRepositoryUT (6)	1 s		
📁 ✅ UnitTesting.ControllersUT (21)	144 ms		
▶️ ✅ AthletesControllerUT (9)	75 ms		
▶️ ✅ AuthControllerUT (12)	69 ms		
📁 ✅ UnitTesting.ServicesUT (61)	1,4 s		
▶️ ✅ AthleteServiceUT (13)	446 ms		
▶️ ✅ DisciplineServiceUT (33)	778 ms		
▶️ ✅ UserServiceUT (15)	211 ms		

Con una cobertura final de 80% se probaron:

- 757/943 líneas
- 38/57 métodos
- 28/28 clases (archivos)

Line coverage					Branch coverage			
Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage	
757	186	943	2137	80.2%	134	206	65%	

Information

Parser: Cobertura
 Assemblies: 1
 Classes: 28
 Files: 28
 Coverage date: 14/10/2022 - 15:41:48

Line coverage

Covered lines: 757
 Uncovered lines: 186
 Coverable lines: 943
 Total lines: 2137
 Line coverage: 80.2%

Branch coverage

Covered branches: 134
 Total branches: 206
 Branch coverage: 65%

Y aunque no nos basamos en la métrica de branch coverage, se consiguió un 65%.

Name	Line coverage					Branch coverage			
	Covered	Uncovered	Coverable	Total	Percentage	Covered	Total	Percentage	
AthletesRestAPI	757	186	943	2137	80.2%	134	206	65%	
AthletesRestAPI.Controllers.AthletesController	40	36	76	202	52.6%	3	6	50%	
AthletesRestAPI.Controllers.AuthController	43	0	43	93	100%	16	16	100%	
AthletesRestAPI.Data.AthleteDbContext	14	0	14	62	100%	0	0		
AthletesRestAPI.Data.Entity.AthleteEntity	12	0	12	39	100%	0	0		
AthletesRestAPI.Data.Entity.DisciplineEntity	8	0	8	28	100%	0	0		
AthletesRestAPI.Data.Repository.AthleteRepository	59	26	85	367	69.4%	5	36	13.8%	
AthletesRestAPI.Exceptions.IncompleteRequestException	1	0	1	13	100%	0	0		
AthletesRestAPI.Exceptions.InvalidElementOperationException	1	0	1	12	100%	0	0		
AthletesRestAPI.Exceptions.NoAthletesToRaceException	1	0	1	12	100%	0	0		
AthletesRestAPI.Exceptions.NotFoundElementException	1	0	1	12	100%	0	0		
AthletesRestAPI.Models.AthleteFormModel	0	1	1	13	0%	0	0		
AthletesRestAPI.Models.AthleteModel	27	2	29	63	93.1%	11	22	50%	
AthletesRestAPI.Models.DisciplineFormModel	0	1	1	13	0%	0	0		
AthletesRestAPI.Models.DisciplineModel	8	0	8	26	100%	0	0		
AthletesRestAPI.Models.RaceAthleteModel	6	0	6	20	100%	0	0		
AthletesRestAPI.Models.RaceInfoModel	19	0	19	39	100%	0	0		
AthletesRestAPI.Models.Security.CreateRoleViewModel	2	0	2	16	100%	0	0		
AthletesRestAPI.Models.Security.CreateUserRoleViewModel	2	0	2	13	100%	0	0		
AthletesRestAPI.Models.Security.LoginViewModel	2	0	2	20	100%	0	0		
AthletesRestAPI.Models.Security.RegisterViewModel	3	0	3	24	100%	0	0		
AthletesRestAPI.Models.Security.UserManagerResponse	4	0	4	15	100%	0	0		
AthletesRestAPI.Models.ShortAthleteModel	8	0	8	23	100%	0	0		
AthletesRestAPI.Services.AthleteService	82	0	82	196	100%	12	12	100%	
AthletesRestAPI.Services.DisciplineService	265	0	265	359	100%	65	82	79.2%	
AthletesRestAPI.Services.FileService	0	17	17	33	0%	0	2	0%	
AthletesRestAPI.Services.Security.UserService	136	0	136	203	100%	22	22	100%	
DisciplinesRestAPI.Controllers.DisciplinesController	0	103	103	190	0%	0	8	0%	
RestaurantRestAPI.Data.AutomapperProfile	13	0	13	31	100%	0	0		

4.1. Comparación estado inicial y final

Métrica	Inicial	Final
Statement Coverage	0%	80%
Líneas probadas	0/943	757/943
Métodos probados	0/57	38/57
Pruebas unitarias	0	89

5. Flujo de Trabajo

5.1. Elección de Framework para .NET

Una vez seleccionada la aplicación para realizar las pruebas, se buscó el framework adecuado para realizar los tests. Se encontraron 3 frameworks principales para testing en dotnet:

- Microsoft (MSTest)
- Nunit
- xUnit

Cada integrante se encargó de buscar reseñas y características para uno de los 3, y posteriormente probarlo:

5.1.1. Microsoft (MSTest)

Opiniones y reseñas

Las reseñas y opiniones de distintos foros denotan una falta de apoyo de la comunidad hacia MSTest para pruebas con .NET. Principalmente influenciado por la poca flexibilidad y la dependencia que tenía en versiones iniciales de un server TFS y del IDE Visual Studio. Aunque se reconoce que tiene una gran cantidad de Asserts y una construcción robusta, soportada por Microsoft. Y la sintaxis de MSTest attributes mas simplificada comparada con Nunit attributes

Sin embargo la dificultad de sintaxis a comparación de otros frameworks y los factores previamente mencionados lo hacen la opción menos favorita.

I wouldn't go with MSTest. Although it's probably the most future proof of the frameworks with Microsoft behind **it's not the most flexible solution**. It won't run stand alone without some hacks. So running it on a **build server other than TFS (Team Foundation Server, now Azure DevOps) without installing Visual Studio is hard**. The visual studio test-runner is actually slower than Testdriven.Net + any of the other frameworks. And because the releases of this framework are tied to releases of Visual Studio there are less updates and if you have to work with an older VS you're tied to an older MSTest. (stackoverflow, <https://stackoverflow.com/a/261323>, 2009)

For me xunit is a little bit simpler (initialize fields in constructor instead of using annotations), but that's totally minor. Use any of them, it won't make that much difference anyway. It's not worth the time you put into comparing the two. Just pick what you know or what seems to be simpler (reddit, <https://www.reddit.com/r/csharp/comments/m119j3/comment/gqazpfp/>?, 2009)

reddit (https://www.reddit.com/r/dotnet/comments/87dqdi/mstest_or_xunit/)

Even Microsoft is using XUnit in their new tutorials. I'd prefer it over MSTest and NUnit. Not a huge difference though. All can work as good testing frameworks.

I've always used MSTest for its native TFS compatibility. The TestCategory and ExpectedException have always worked for me & my teams on NetFx or dotnet. TFS 2015 on-prem has some issues that require a little more manual configuration and coverage is still lacking. (5 years ago, 2017)

MSTest supports Assrt.Inconclusive. This is essential for integration testing when you need to know the difference between a failed test and a test that can't be run because a prerequisite is broken. (5yr ago, 2017)

Historically you had to have vs installed to run mstest tests, which was a pain for CI. I believe that has since changed, but I still will reach for xunit or nunit every time. In reality I don't think there's that much difference these days. I'd worry more about your mocking framework, far more coupling happens there.

xUnit is, by default, leaner and cleaner than both NUnit and MSTest when it comes to writing unit tests. This feature is one of many that enables us to help write cleaner tests. (10 months ago, 2022)

xUnit is, by default, leaner and cleaner than both NUnit and MSTest when it comes to writing unit tests. This feature is one of many that enables us to help write cleaner tests. (10 months ago, 2022)

Características

CARACTERÍSTICAS	Microsoft MSTest
Descripción general	MSTest is the Microsoft test framework for all .NET languages. It's extensible and works with both .NET CLI and Visual Studio. (Microsoft Open Source)
Soporte actualizaciones	.NET 5 or higher. Supported platforms: <ul style="list-style-type: none">- .NET 4.5.0+- .NET Core 1.0+ (Universal Windows Apps 10+) (Visual Studio 2017)- .NET 5.0 (Visual Studio 2019)- .NET 5.0 Windows.18362+ (WinUI Desktop Apps) (Visual Studio 2019)- ASP.NET Core 1.0+ (Visual Studio 2017)
Plataformas (windows, linux, etc)	MSTest V2 - This is a fully supported, open source and cross-platform implementation of the MSTest test framework with which to write tests targeting .NET Framework, .NET Core and ASP.NET Core on Windows, Linux, and Mac. (https://github.com/Microsoft/testfx)
Comunidad	MSTest no es muy apreciado por la comunidad debido a su dependencia de Microsoft y a que este ligado a los <u>realeses</u> de VS. También su primera versión le trajo mala fama por su dependencia de un <u>server</u> TFS y de VS, actualmente se <u>mejora</u> en ese aspecto, pero la comunidad se <u>decanta</u> ya por el resto de Frameworks. Incluso la nueva documentación de MS realiza tutoriales en <u>xUnit</u> en lugar de MSTest

	alternativas en XUnit en lugar de MSTest
Documentación	Bien organizada y con guías. https://github.com/Microsoft/testfx-docs , https://learn.microsoft.com/en-us/visualstudio/test/using-microsoft-visualstudio-testtools-unittesting-members-in-unit-tests?view=vs-2022 , https://learn.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting?view=visualstudiosdk-2022
<u>Flexibilidad</u>	La primera versión era dependiente de TFS y Visual Studio
Ventajas	<ul style="list-style-type: none"> - Documentación bien organizada y centralizada por MS - Permite class scoped y method scoped parallelization - <u>Integración</u> nativa con VS - test coverage integrado
Desventajas	<ul style="list-style-type: none"> - Poco soporte de la comunidad - Primeras versiones dependientes de Visual Studio - Mayor variedad de <u>parametros</u> para correr solo test seleccionados (Priority, TestCategory) (<u>Demás</u> frameworks si soportan, pero con una mayor complejidad de sintaxis) - No permite parameter scoped parallelization - Menor cantidad de assertions (29, contra 42 de NUnit) - Complejidad de sintaxis TestInitialize a <u>comparacion</u> de xUnit que te permite inicializar en el constructor - Menor cantidad de actualizaciones por estar ligado a VS
Diferencias destacables	- simplicidad de la sintaxis de comandos para ejecutar selected test (por prioridad, etc)

Proof of concept (POC)

Para probar el framework se siguió la siguiente guía de la documentación oficial de Microsoft [Unit testing C# with MSTest and .NET - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-mstest-and-net-net?view=net-6.0)

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    [TestClass]
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService _primeService;

        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService();
        }
    }
}
```

```

[TestMethod]
public void IsPrime_InputIs1_ReturnFalse()
{
    bool result = _primeService.IsPrime(1);

    Assert.IsFalse(result, "1 should not be prime");
}
}
}

```

5.1.2. Nunit

Reseñas

En slant se encontró una comparación de este framework para comparar pros y contras

PROS

- PRO Better support for integration and system test
- PRO Excellent availability of learning resources: Due to it's popularity and active development, plenty of learning resources (such as detailed documentation and various tutorials) exist for learning NUnit.
- PRO Widely used: NUnit is one of the most popular testing frameworks for .NET. Other than giving a certain sense of security in the continuation of the project, it also means that there are a lot of third-party resources, guides and tutorials available for NUnit.
- PRO Built-in fluent Assertions: It has more readable Assertions out of the box like `Assert.That(myClass.MyMethod(null), Throws.ArgumentNullException.With.Message.Contains("param"));` NUnit also has good tutorials in using the variants for parameterized tests: e.g. its easy to find an example how to correctly use `[TestCaseSource(typeof(myTestCaseEnumerator))]`.

CONS

- CON Slow to adapt: NUnit is developed slowly; some developers complain that it is no longer able to keep up with new testing approaches.

En lambda test se encontró una comparación con XUnit y MSTest, sus puntos más importantes son:

- NUnit has been downloaded more than 126 million times from NuGet.org. This shows the popularity of NUnit within the .NET user community. As of writing this article, there were close to 24,000 questions tagged as NUnit on Stackoverflow. It is a popular test framework

used for test automation. If you are planning to perform Test-Driven Development (TDD) with C#, you should have a close look at the NUnit framework.

- When we do NUnit vs. XUnit vs. MSTest, extensibility plays an important role in choosing a particular test framework. The choice might depend on the needs of the project, but in some scenarios, extensibility can turn the tables around for a particular test framework. When compared to MSTest and NUnit frameworks, xUnit framework is more extensible since it makes use of [Fact] and [Theory] attributes. Many attributes that were present in NUnit framework e.g. [TestFixture], [TestFixtureSetup], [TestFixtureTearDown], [ClassCleanup], [ClassInitialize], [TestCleanup], etc. are not included in the xUnit framework.
- In NUnit parallelism is possible at the level of Children (child tests are executed in parallel with other tests), Fixtures (descendants of test till the level of Test Fixtures can execute in parallel), Self (test itself can be executed in parallel with other tests), and All (test & its descendants can execute in parallel with others at the same level).
- The NUnit uses [SetUp], [TearDown] pairs whereas MSTest uses [TestInitialize], [TestCleanup] pairs for setting up the activities related to initialization & de-initialization of the test code.

Características

CARACTERÍSTICAS	Xunit.net Tiara
Descripción general	Testing framework para todos los lenguajes .Net con soporte de .NET 5 y .NET Core frameworks
Soporte actualizaciones	.NET 4 o mejores y Visual Studio 2017 o mejores
Plataformas (windows, linux, etc)	Corre sobre todo lo que pueda correr Visual Studio (windows, mac, etc)
Comunidad	Nunit es parte de la .NET Foundation que provee <u>guías</u> y soporte para el uso de la herramienta
Documentación	Documentación clara para todas las operaciones que se quieran realizar
Flexibilidad	<ul style="list-style-type: none"> - Corre sobre Visual Studio - Tiene dependencia con Selenium y NUnit test framework
Ventajas	<ul style="list-style-type: none"> - Ayuda a crear test de manera paralela, secuencial y ordenada-Puede ser ejecutado vía <u>líneas</u> de comandos con diferentes parámetros para correr los test - NUnit Assertions ayudan a mejorar la ejecución de los PASS/FAIL - Los test pueden ser ejecutados vía líneas de comandos con diferentes parámetros para usarlos al correr los test - Mejor soporte e integración para el sistema - Ampliamente usado por muchos desarrolladores (30 años)
Desventajas	<ul style="list-style-type: none"> - Lento de adaptar y mantenerse al ritmo con nuevos test approaches - No es compatible con .NET Core 2
Diferencias destacables	<ul style="list-style-type: none"> - Usa TestFixture(nav, versión, os) para recibir parámetros de navegador, versión del navegador y sistema operativo - Utiliza funciones SetUp y TearDown para inicializar y cerrar servidores - Al usarse con Visual Studio todas las corridas, configuraciones y resultados con test se pueden manejar desde la IDE de Visual Studio

5.1.3. xUnit

Reseñas

En stackOverflow se encontraron múltiples reseñas, sobre todo en inglés:

-xUnit.net is more modern, more TDD adherent, more extensible, and also trending in .NET Core development. It's also well documented.

-I am not sure how you found out that xUnit is "well documented", as it clearly doesn't.

- How is the documentation "better"? It barely exists
- XUnit doesn't work with `Console.WriteLine()`
- a bad documented green sandbox with broken understanding of TDD, bad documentation and lack of cool features.
- posted that around 4 years ago and I've been using xunit since then up until now. the documentation and the community is way better than before and it's more mature now

En reddit se encontraron comentarios como:

- I'd definitely go with xUnit since it has all the new sauce.
- Spend an hour with each and then pick the one that feels most intuitive. Or just pick XUnit if you want the current populist choice. It really doesn't matter.
- either choice is fine. I use both; NUnit at the office, and XUnit for my personal projects.
- One reason I like xUnit more than NUnit is that xUnit discourages per-test setup and teardown by only allowing constructor setup and teardown.
- I personally prefer xUnit, because of their reliance on language features instead of attributes
- They're both good choices. I think it's really a matter of taste.

Se puede notar que varias personas piensan que XUnit es un buen framework para testing, y un porcentaje menor cree que no.

Características

CARACTERÍSTICAS	Xunit.net Tiara
Descripción general	gratis, open-source, nace en 2007
Soporte actualizaciones	soporte de .Net core hasta la última versión
Plataformas (windows, linux, etc)	windows, mac, linux
Comunidad	orientado a la comunidad, comunidad extensa, foros abiertos, ejemplos y recursos, más de 7500 respuestas en foros
Documentación	buena, orientada a guía para tareas específicas
<u>Flexibiilidad</u>	<ul style="list-style-type: none"> -Bastante flexible ya que permite agregar atributos, tipos de datos, assert -Compatible con otros XUnit Frameworks -Permite extender clase Assert
Ventajas	<ul style="list-style-type: none"> - crea una instancia nueva de la clase para cada test, lo cual garantiza que cada test se ejecute aisladamente <ul style="list-style-type: none"> - Extensible para TDD - Ejecución paralela de tests - soporte de <u>DataDrivenTests</u> (múltiples entradas pasadas a 1 método de test en) <ul style="list-style-type: none"> - instalación sencilla - * creado para MEJORAR NUnit - Assert.Throws permite testear un set específico de código para lanzar una excepción y retorna la excepción satisfactoria para que se puedan realizar asserts sobre la instancia de excepción <ul style="list-style-type: none"> - Terminología intuitiva - apoyo de la comunidad - Comentarios:I would go with xUnit since it is more extensible and has fewer attributes, making the code clean & easy to maintain. xUnit was created to succeed NUnit, a popular unit testing library that is part of the .NET framework. Although the .NET framework has evolved since NUnit was first written, xUnit leverages some of its advanced features to write cleaner tests that are easier to debug and run than in NUnit.s xUnit framework provides much better isolation of tests in comparison to NUnit and MSTest frameworks. For each test case, the test class is instantiated, executed, and is discarded after the execution. This ensures that the tests can be executed in any order as there is reduced/no dependency between the tests
Desventajas	<ul style="list-style-type: none"> - no tiene mocks incluidos - reciente, menos conocido - documentación no estructurada - no tiene disponibilidad de obtener test context
Diferencias destacables	<ul style="list-style-type: none"> - no requiere atributo para una clase de test - no usa [SetUp] ni [TearDown] - se puede simular el setup en el constructor ya que este se instancia en cada test - para Teardown se puede utilizar <u>Idisposable.Dispose</u> - ASSERTIONS: -no Throws.Nothing exception

	-no Fail -no <u>Is.GreaterThan</u>
--	---------------------------------------

Proof of Concept (POF)

Para probar el framework, se creó un proyecto pequeño: tiaraRS/POC-Xunit-.net [tiaraRS/POC-Xunit-.net \(github.com\)](https://github.com/tiaraRS/POC-Xunit-.net).

5.1.4. Elección - xUnit

Después de una extensa investigación se concluyó que **xUnit** seria el framework utilizado para implementar las pruebas de unidad.

Las razones principales son:

- La facilidad en la sintaxis, permite evitar tareas de setup, pudiendo inicializarse los atributos en el constructor de la clase de prueba.
- El uso de attributes `[Fact]` para test individuales y `[Theory]` para test a los que se pueden pasar diferentes parámetros de entrada.
- Es el framework con mejor soporte y mas usado por la comunidad, incluso por la misma documentación oficial de Microsoft
- Permite la ejecución de pruebas en paralelo a nivel de método.
- Permite seleccionar que pruebas ejecutar (ejecución parcial)

Muchas de las reseñas y opiniones encontradas, señalan que los 3 frameworks mencionados tienen características muy similares y no destacan mucho sobre los otros. Se realizó una tabla general para comparar las características más destacables de los 3:

CARACTERÍSTICAS

CARACTERÍSTICAS	Microsoft	Nunit	Xunit
Facilidad de uso	Simple	Simple	simple
Facilidad de sintaxis	Uso de attributes	Con decoradores de SetUp TearDown y Fact	sintaxis amigable, lo único los decorators de Theory y Fact que pueden confundir al principio
Comodidad	Con interfaces de prueba integradas en visual studio	Con interfaces de prueba integradas en visual studio	cómodo, interfaz para ejecutar pruebas desde visual studio
Features útiles	Capacidad de correr pruebas en paralelo y compatible com Moq	Capacidad de correr pruebas en paralelo y compatible com Moq	Gran variedad de asserts, compatible con Moq en Visual Studio

5.1.5. Guía para integrar proyecto a Xunit

Para poder integrar XUnit a la aplicación desde Visual Studio, y ejecutar las pruebas, se siguió la siguiente guía: [Getting Started: .NET Framework with Visual Studio > xUnit.net](#)

Al ser una API con la estructura mencionada anteriormente, se tienen dependencias entre capas, por lo tanto, para poder probar cada método aisladamente, se deben utilizar mocks: [Mocks para pruebas de unidad .NET - Presentaciones de Google](#).

.Net provee el paquete Moq para poder crear mocks.

Para utilizar moq, primero se crea un mock de tipo de clase/interface que queramos, en este caso el repositorio:

```
var repositoryMock = new Mock<IAthleteRepository>();
```

Posteriormente, se indica al mock qué método quiere simular (a través del Setup), y qué queremos que devuelva al ejecutar el método que simula (Returns):

```
repositoryMock.Setup(r => r.GetDisciplineAsync(1, false)).ReturnsAsync(disciplineEntity100M);
```

Por último, le pasamos la dependencia del mock de repositorio al servicio, que es la clase que probaremos:

```
var disciplinesService = new DisciplineService(repositoryMock.Object, mapper);
```

5.2. Elección de Herramienta para Cobertura - Coverlet

5.2.1. Opciones de cobertura

Existen varias herramientas para calcular la cobertura de código sin embargo la mayoría de estas son de pago o requiere alguna licencia o suscripción.

Algunas de las herramientas son:

- NCover
- Cake
- Coverage levels
- Test well
- Fine Code Coverage
- Coverlet

Las herramientas mas soportadas en foros y por la comunidad son:

- Visual Studio integrated coverage, sin embargo a esta solo se puede acceder en la version Enterprise del IDE.
- dotCover de JetBrains que de igual manera es de pago
- NCover que proporciona información detallada sobre la cobertura ya que la herramienta en su totalidad esta centrada en ello, de igual manera es de pago.

Entre las opciones que consideramos (gratuitas):

- Fine Code Coverage: permitió una instalación sencilla como extension de VisualStudio, con la métrica de statement coverage.
- Coverlet: la instalación si es un poco mas complicada, sin embargo microsoft usa la herramienta en su documentación, brindando tutoriales para su instalación, siendo gratuita, permite la generación de reportes en diferentes formatos. Destacando el HTML, que de manera interactiva te permiten ver las líneas y métricas de cada Archivo o clase. Y en su version de partner brinda incluso las métricas a nivel de método. Finalmente la ventaja principal es que permite ver en el IDE las líneas resaltadas: en verde las cubiertas y en rojo las no cubiertas.

```
14 referencias | 0/10 pasando
public async Task<IEnumerable<AthleteEntity>> GetAthletesAsync(int disciplineId)
{
    IQueryable<AthleteEntity> query = _dbContext.Athletes;
    query = query.AsNoTracking();

    query = query.Where(a => a.Discipline.Id == disciplineId);
    var result = await query.ToListAsync(); //aquí va recién a bd

    return result;
}

4 referencias | 0/2 pasando
public async Task UpdateAthleteAsync(int athleteId, AthleteEntity athlete, int disciplineId)
{
    var athleteToUpdate = await _dbContext.Athletes.FirstOrDefaultAsync(a => a.Id == athleteId && a.Discipline.Id == disciplineId);
    // no modifco el discipline id
    // var athleteToUpdate = _athletes.FirstOrDefault(a => a.Id == athleteId && a.DisciplineId==disciplineId);
    athleteToUpdate.Gender = athlete.Gender ?? athleteToUpdate.Gender;
    athleteToUpdate.BirthDate = athlete.BirthDate ?? athleteToUpdate.BirthDate;
    athleteToUpdate.Name = athlete.Name ?? athleteToUpdate.Name;
    athleteToUpdate.IsActive = athlete.IsActive ?? athleteToUpdate.IsActive;
    athleteToUpdate.Nationality = athlete.Nationality ?? athleteToUpdate.Nationality;
    athleteToUpdate.NumberOfCompetitions = athlete.NumberOfCompetitions ?? athleteToUpdate.NumberOfCompetitions;
    athleteToUpdate.PersonalBest = athlete.PersonalBest ?? athleteToUpdate.PersonalBest;
    athleteToUpdate.SeasonBest = athlete.SeasonBest ?? athleteToUpdate.SeasonBest;
    athleteToUpdate.Points = athlete.Points ?? athleteToUpdate.Points;
}
```

Por todo ello la elección de Coverlet es clara.

5.2.2. Comparación entre Fine Code Coverage y Coverlet, con captura de ejemplo

Algo interesante que pudimos notar al poder comparar ambas herramientas, es que mostraban distintos porcentajes de cobertura de línea. Después de indagar, nos dimos cuenta de que la herramienta Fine Code Coverage no tomaba en cuenta los archivos de migraciones, en cambio la herramienta Coverlet si lo hacía. Tomamos el criterio de que los archivos de migraciones no se deben tomar en cuenta ya que no son parte de la aplicación como tal. Para corregir esto en el análisis de Coverlet, se agregó el atributo `[ExcludeFromCoverage]`

```
[ExcludeFromCodeCoverage]
public partial class InitialMigration : Migration
{
```

a cada clase de las migraciones.

5.2.3. Guía para instalación Coverlet

Prerrequisitos

- Tener instalado `dotnet`
- Tener creado un proyecto de testing

Instalación

1. Añadir `coverlet.msbuild` a cada proyecto de testing (`.csproj`) mediante el NuGet package manager o con el comando (correr el comando a nivel de `.csproj`).

```
dotnet add package coverlet.msbuild
```

Esto modificara el archivo `.csproj` instalando el paquete.



coverlet.msbuild por tonerdo

Coverlet is a cross platform code coverage library for .NET, with support for line, branch and method coverage.

3.1.2

```
<PackageReference Include="coverlet.msbuild" Version="3.1.2">
  <IncludeAssets>runtime; build; native; contentfiles; analyzers;
  buildtransitive</IncludeAssets>
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

2. En un powershell con privilegios de administrador ejecutar

```
dotnet tool install -g dotnet-reportgenerator-globaltool
dotnet tool install dotnet-reportgenerator-globaltool --tool-path tools
dotnet new tool-manifest
dotnet tool install dotnet-reportgenerator-globaltool
```

Respuesta esperada

```
C:\WINDOWS\system32> dotnet tool install -g dotnet-reportgenerator-globaltool
Puede invocar la herramienta con el comando siguiente: reportgenerator
La herramienta "dotnet-reportgenerator-globaltool" (versión '5.1.10') se
instaló correctamente.
```

```
C:\WINDOWS\system32> dotnet tool install dotnet-reportgenerator-globaltool --
tool-path tools
```

```
Puede invocar la herramienta con el comando siguiente: reportgenerator
La herramienta "dotnet-reportgenerator-globaltool" (versión '5.1.10') se
instaló correctamente.
```

```
C:\WINDOWS\system32> dotnet new tool-manifest
```

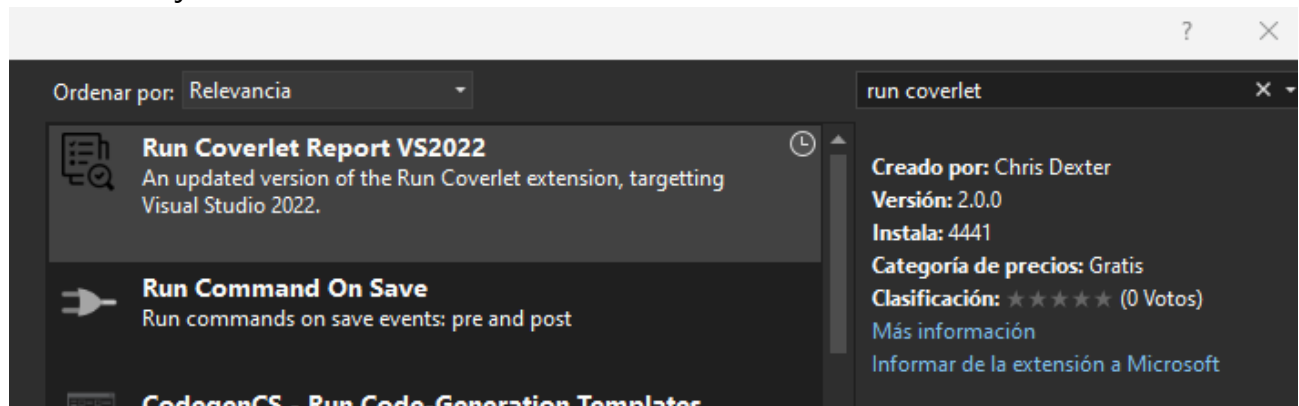
```
La plantilla "Archivo de manifiesto de la herramienta local de dotnet" se creó
correctamente.
```

```
C:\WINDOWS\system32> dotnet tool install dotnet-reportgenerator-globaltool
Puede invocar la herramienta desde este directorio con los comandos siguientes:
"dotnet tool run reportgenerator" o "dotnet reportgenerator".
```

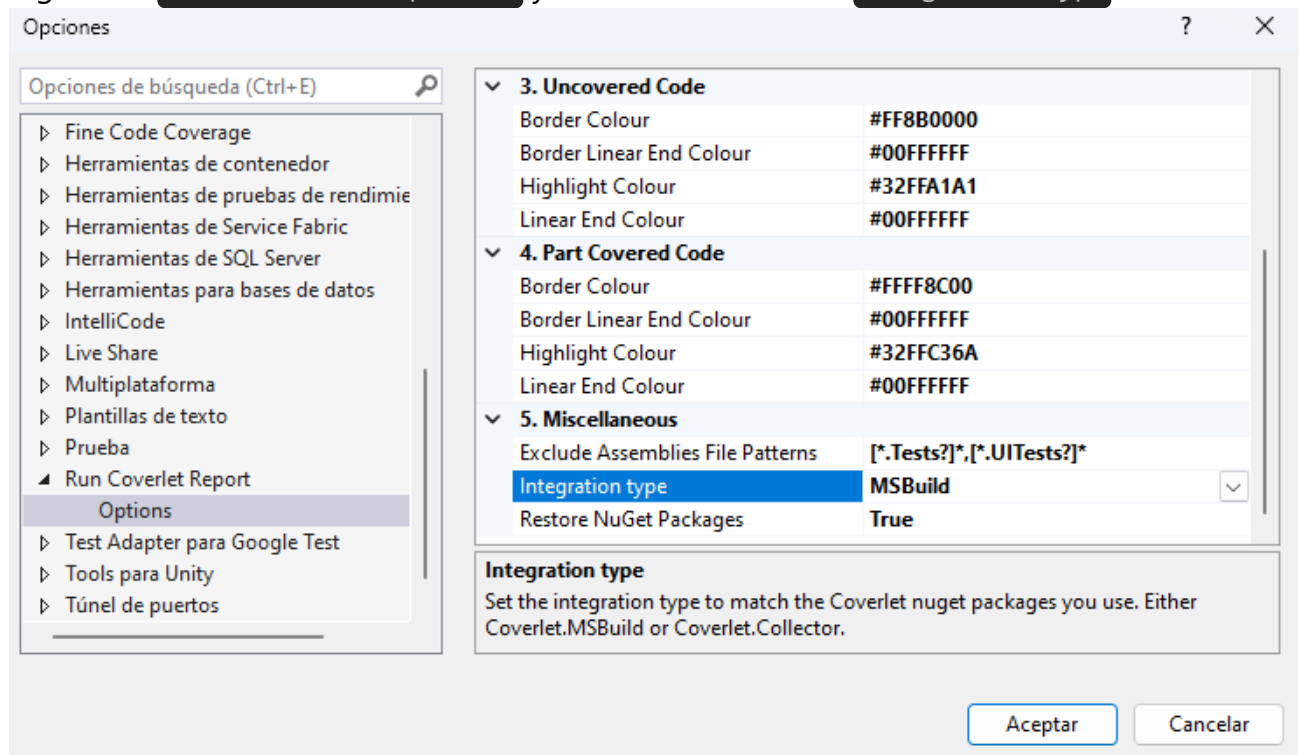
```
La herramienta "dotnet-reportgenerator-globaltool" (versión "5.1.10") se
instaló correctamente. Se ha agregado la entrada al archivo de manifiesto
```

```
C:\WINDOWS\system32\.config\dotnet-tools.json.  
C:\WINDOWS\system32>
```

3. Instalar la extensión **Run Coverlet Report** en Visual Studio, reiniciar el IDE para que los cambios surjan efecto.



Ingresar a **Herramientas > Opciones** y modificar el valor de **Integration type**



Instalación completa.

Uso y generación de reportes

Para obtener el reporte se deberá correr todos los test por lo menos una vez, para inicializar de manera correcta las referencias a los proyectos de test. (solo necesario la primera vez).

Para correr el análisis de cobertura ir a **Herramientas > Run Code Coverage**. Esto generara los files del reporte y los abrirá en Visual Studio.

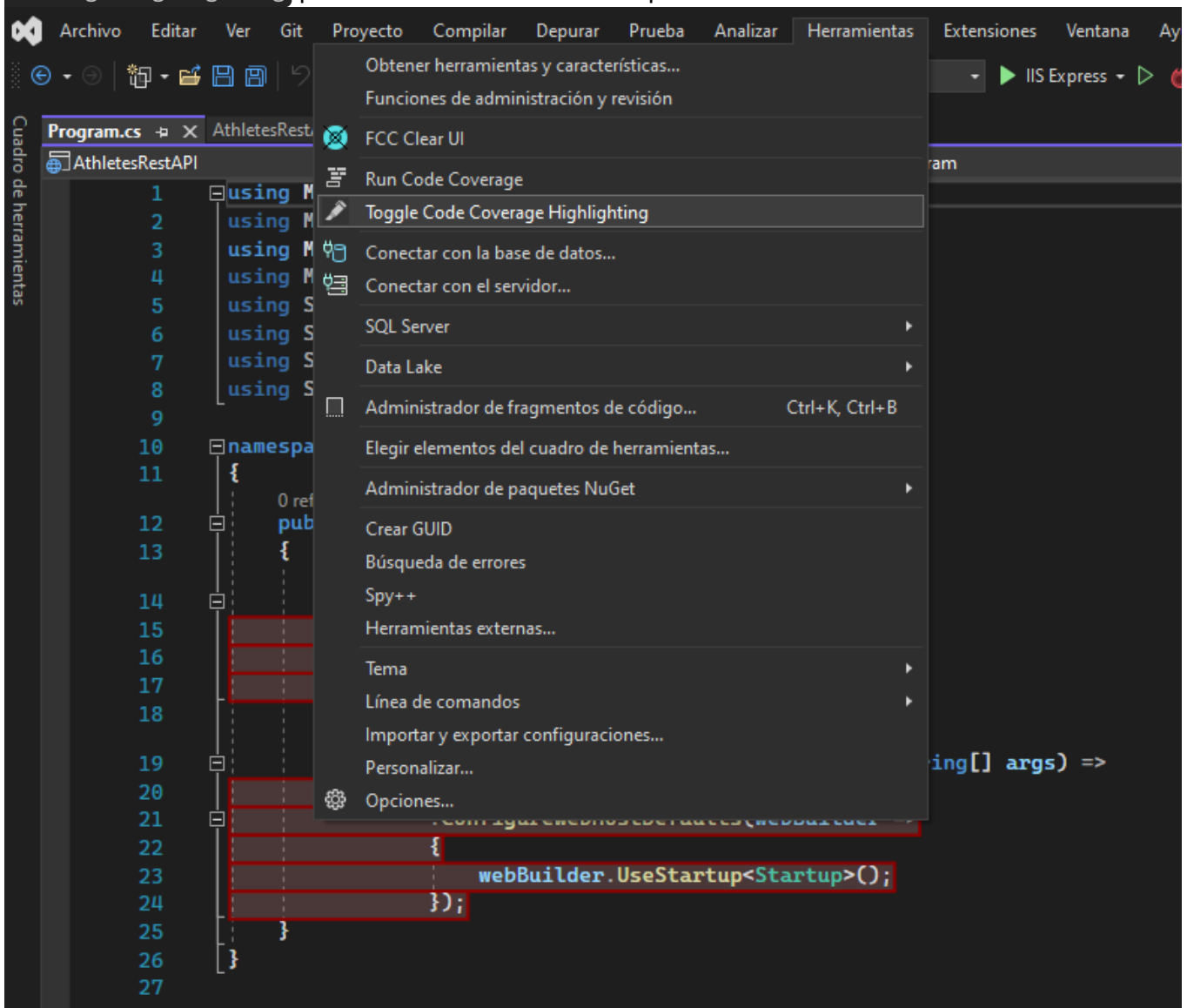
Program.cs Summary - Coverage Report

Dirección URL: C:\Users\Kevin\AppData\Local\Temp\9ec29fe-7ff4-4c3b-be9d-82b43a3d121\coverageReport\index.html

Para ayudar a proteger su seguridad, el explorador web impidió que este archivo mostrara contenido activo que podría obtener acceso al equipo. Haga clic aquí para ver más opciones...

Name	Line coverage			Total	Percentage	Branch coverage		
	Covered	Uncovered	Coverable			Covered	Total	Percentage
AthletesRestAPI	128	2954	3082	4697	4.1%	8	184	4.3%
AthletesAPI Program	0	8	8	26	0%	0	0	
AthletesAPI Startup	0	69	69	147	0%	0	2	0%
AthletesAPI WeatherForecast	0	4	4	15	0%	0	0	
AthletesRestAPI.Controllers.AthletesController	0	76	76	202	0%	0	6	0%
AthletesRestAPI.Controllers.AuthController	0	43	43	93	0%	0	16	0%
AthletesRestAPI.Data.AthleteDbContext	0	14	14	62	0%	0	0	
AthletesRestAPI.Data.Entity.AthleteEntity	0	12	12	39	0%	0	0	
AthletesRestAPI.Data.Entity.DisciplineEntity	8	0	8	28	100%	0	0	
AthletesRestAPI.Data.Repository.AthleteRepository	0	80	80	365	0%	0	34	0%
AthletesRestAPI.Exceptions.IncompleteRequestException	0	1	1	13	0%	0	0	
AthletesRestAPI.Exceptions.InvalidElementOperationException	0	1	1	12	0%	0	0	
AthletesRestAPI.Exceptions.NoAthletesToRaceException	0	1	1	12	0%	0	0	
AthletesRestAPI.Exceptions.NotFoundElementException	1	0	1	12	100%	0	0	
AthletesRestAPI.Migrations.AthleteDbContextModelSnapshot	0	328	328	362	0%	0	0	
AthletesRestAPI.Migrations.AthletesPoints	0	334	334	381	0%	0	0	
AthletesRestAPI.Migrations.enableCascadeDeleting	0	105	105	140	0%	0	0	
AthletesRestAPI.Migrations.ImagesAthletes	0	340	340	387	0%	0	0	
AthletesRestAPI.Migrations.ImagesDiscipline	0	337	337	384	0%	0	0	
AthletesRestAPI.Migrations.InitialMigration	0	138	138	177	0%	0	0	
AthletesRestAPI.Migrations.security	0	507	507	574	0%	0	0	

Para ver las líneas resaltadas directamente en el IDE seleccionar **Herramientas > Toggle Code Coverage Highlighting** para activar o desactivar la opción.

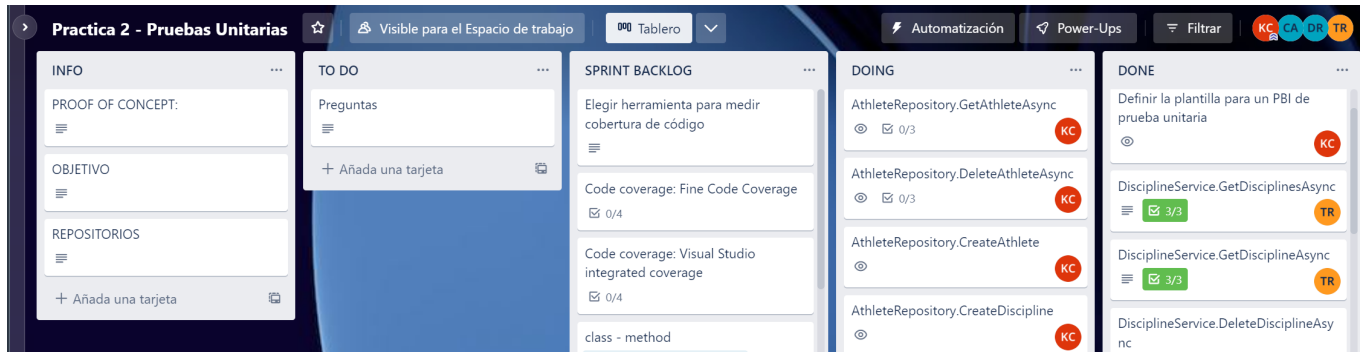


guia extension

5.3. Descripción de flujo de trabajo

5.3.1. Manejo del Trello

Para el trello se manejaron 5 columnas, la de información, la de cosas pendientes, el sprint backlog, las tareas en proceso y las tareas concluidas.



5.3.2. Flujo de Trabajo

Cada uno tenía la tarea de aumentar la cobertura de código en un 25%, para alcanzar el objetivo deseado. Para poder cumplir el objetivo planteado, se utilizó la técnica para determinar casos de prueba que se utiliza en Path Coverage. Por teoría sabemos que 100% de path coverage garantiza 100% de statement coverage, que es la métrica que utiliza nuestra herramienta de cobertura, por lo tanto, al utilizar esta técnica, garantizamos que incrementemos el % de statement coverage. Para aplicar la técnica, cada uno se creaba una rama y se tenían 3 checks:

- Realizar el grafo para determinar la complejidad ciclomática, para poder saber cuántos casos de prueba se necesitan como máximo para un método.

- Determinar los casos de prueba a partir del grafo, siguiendo la heurística del camino más largo, e ir cambiando la última decisión para hallar todos los caminos posibles.

- Implementar los casos de prueba en el código fuente

Una vez cumplidas estas 3 checks, se verifica que realmente el % de cobertura haya subido, y las líneas que esperamos estén en verde (toggle code coverage highlight). Posteriormente, se unen los cambios al main y se verifica que todos los tests pasen. Finalmente, se realiza el push a github.

6. Casos de prueba y pruebas de unidad

Se diseñaron e implementaron las pruebas de unidad identificando primero los test cases mediante Path coverage, realizándose los grafos y test cases para cada uno de los métodos testeados. Al ser 38 métodos, incluirlos, haría el documento muy extenso, por lo que se optó en mostrarlo por GitHub.

[CALIDAD-practica2-PruebasDeUnidad-AthletesAPI-UNIT-TESTING/6. Casos de prueba y pruebas de unidad.md](#) at

[main · tiaraRS/CALIDAD-practica2-PruebasDeUnidad-AhtletesAPI-UNIT-TESTING \(github.com\)](https://github.com/tiaraRS/CALIDAD-practica2-PruebasDeUnidad-AhtletesAPI-UNIT-TESTING)

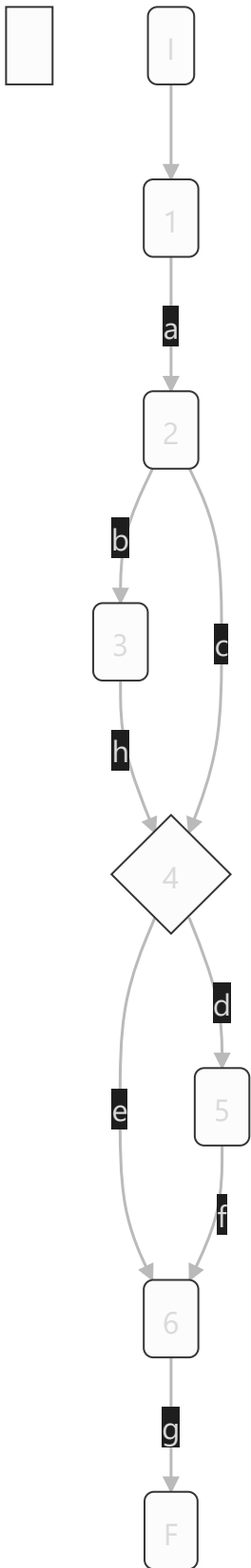
En este repositorio se muestran todos los test diseñados e implementados (incluye el código y los grafos y caminos). A continuación se muestra un ejemplo de lo realizado para cada método que se probó.

DisciplineService.CheckSeasonBest

Código

```
public bool CheckSeasonBest(AthleteModel athlete, Decimal mark, string discipline)
{
    bool seasonBest = _markComparer[discipline](athlete.SeasonBest, mark); //1
    if (athlete.SeasonBest == null) //2
    {
        seasonBest = true; //3
    }
    if (seasonBest) //4
    {
        athlete.SeasonBest = mark; //5
    }
    return seasonBest; //6
}
```

Grafo



Complejidad ciclo matica

Numero de regiones

$$v(G) = Rv(G) = 3$$

Numero de nodos y aristas

$$v(G) = E - N + 2v(G) = 9 - 8 + 2$$

Numero de decisiones

$$v(G) = P + 1v(G) = 2 + 1$$

Casos de prueba

athlete, mark, discipline

	Camino	Entrada	TC	Salida
1	l-1a- 2b- 3h- 4d-5f- 6g-F	<code>athlete</code> ={Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=null} <code>mark</code> =51.79 <code>discipline</code> ='400MH'	<code>athlete.SeasonBest=null</code> -> <code>seasonBest = true</code>	true
2	l-1a- 2c-4d- 5f-6g- F	<code>athlete</code> ={Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=52} <code>mark</code> =51.79 <code>discipline</code> ='400MH'	<code>athlete.SeasonBest!=null</code> -> <code>seasonBet = true</code>	true
3	l-1a- 2c-4e- 6g-F	<code>athlete</code> ={Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=52} <code>mark</code> =52.79 <code>discipline</code> ='400MH'	<code>athlete.SeasonBest!=null</code> -> <code>seasonBet = false</code>	false

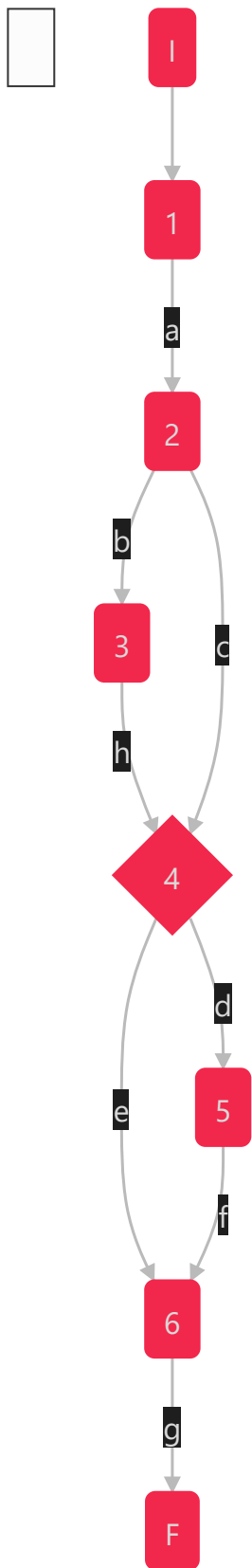
TC1: Verificar que si la atleta {Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=null}, inicialmente sin mejor marca de temporada, realiza una marca de 51.79 en la disciplina 400MH, devuelva true

TC2: Verificar que si la atleta {Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=52}, con mejor marca de temporada 52, realiza una marca de 51.79 en la disciplina 400MH, mejor a su mejor marca de temporada previa, devuelva true

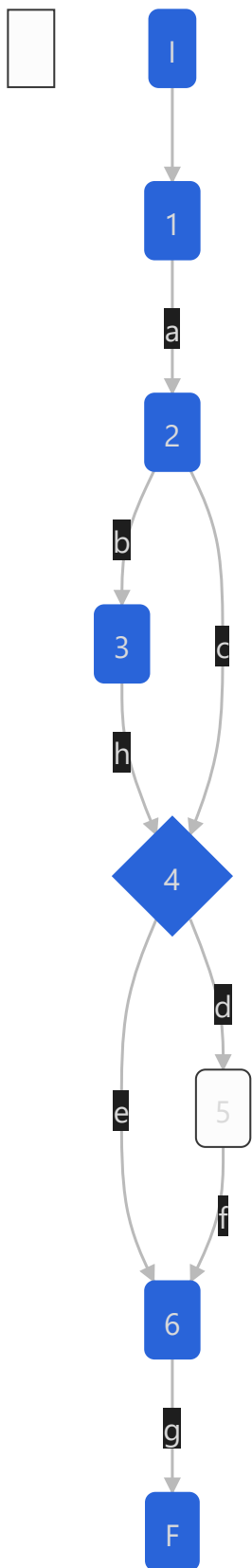
TC3: Verificar que si la atleta {Id=1,Nationality='USA', Name='Sydney Maclaughlin', Gender='f', Points=1000, PersonalBest=52.75, SeasonBest=52}, con mejor marca de temporada 52, realiza

una marca de 52.79 en la disciplina 400MH, peor a su mejor marca de temporada previa,
devuelva false

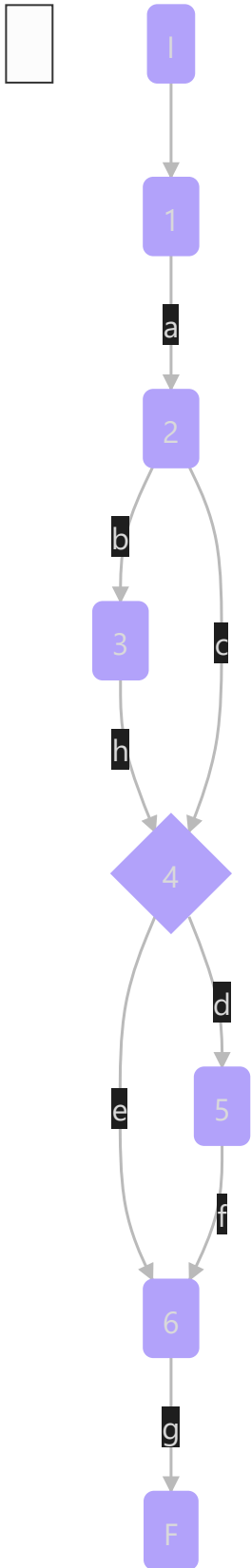
Camino 1



Camino 2



Camino 3



Pruebas unitarias

```
//tc1  
[Fact]  
public void CheckSesasonBest_SeasonBestNull_ReturnsTrue()
```

```

{
    var config = new MapperConfiguration(cfg =>
cfg.AddProfile<AutomapperProfile>());
    var mapper = config.CreateMapper();
    var sydney = new AthleteModel()
    {
        Id = 1,
        Nationality = "USA",
        Name = "Sydney MacLaughlin",
        Gender = Gender.F,
        Points = 1000,
        PersonalBest = 52.75m,
        SeasonBest = null
    };
    var disciplineName = "400MH";
    var mark = 51.76m;
    var repositoryMock = new Mock<IAthleteRepository>();
    var disciplinesService = new DisciplineService(repositoryMock.Object,
mapper);

    var result =
disciplinesService.CheckSeasonBest(sydney, mark, disciplineName);
    Assert.True(result);
}
//tc2
[Fact]
public void CheckSeasonBest_SeasonBestImproved_ReturnsTrue()
{
    var config = new MapperConfiguration(cfg =>
cfg.AddProfile<AutomapperProfile>());
    var mapper = config.CreateMapper();
    var sydney = new AthleteModel()
    {
        Id = 1,
        Nationality = "USA",
        Name = "Sydney MacLaughlin",
        Gender = Gender.F,
        Points = 1000,
        PersonalBest = 52m,
        SeasonBest = 51.79m
    };
    var disciplineName = "400MH";
    var mark = 51.76m;
    var repositoryMock = new Mock<IAthleteRepository>();
    var disciplinesService = new DisciplineService(repositoryMock.Object,
mapper);

```

```

        var result = disciplinesService.CheckSeasonBest(sydney, mark,
disciplineName);
        Assert.True(result);
    }

    //tc3
    [Fact]
    public void CheckSesasonBest_SeasonBestNotImproved_ReturnsFalse()
    {
        var config = new MapperConfiguration(cfg =>
cfg.AddProfile<AutomapperProfile>());
        var mapper = config.CreateMapper();
        var sydney = new AthleteModel()
        {
            Id = 1,
            Nationality = "USA",
            Name = "Sydney MacLaughlin",
            Gender = Gender.F,
            Points = 1000,
            PersonalBest = 52.75m,
            SeasonBest = 52m
        };
        var disciplineName = "400MH";
        var mark = 52.79m;
        var repositoryMock = new Mock<IAthleteRepository>();
        var disciplinesService = new DisciplineService(repositoryMock.Object,
mapper);

        var result = disciplinesService.CheckSeasonBest(sydney, mark,
disciplineName);
        Assert.False(result);
    }
}

```

7. Conclusiones

8. Bibliografía

<https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>

<https://www.slant.co/options/1756/~nunit-review>

<https://docs.nunit.org/>