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]
  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]
  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 métodо.
- 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
  `json "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:      **![](https://lh4.googleusercontent.com/HbJQQzDW09DL0usIBeJgnbObuymlbbOyOKKp3ToCO6cXHNE5u6fH4YW1uQQQVwq7eg7zPKmS9EoFCOk7G47b1MKztHX62bWLjZasrk4TMnohPkQNNyjNezaOp1DM6x0GeNUqWQDdYwk4Qsu-kn9CIGPPm2A3OlPgVbPrnBOrfwJpPTvatG-p_eM9Pg)**
  ```

```
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

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

por

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:

Una vez logueado satisfactoriamente, se devuelve un token. Se debe copiar este token al campo de Authentication -> Bearer Token -> BearerToken en cada endpoint, por ejemplo:

Una vez realizado esto, se puede probar cada endpoint:

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 del 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 de la aplicación no puede ser probada, o al menos no tiene sentido probar ya que esta contiene archivos de migraciones.
3. Estado inicial

Como se observa en la imagen de arriba, al inicio el proyecto no contaba con ninguna prueba de unidad implementada.

4. Estado final (incluir número de líneas, métodos y archivos probados frente al total)
   4.1. Comparación estado inicial y final

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, incluyendo Microsoft, Nunit y Xunit. Cada integrante se encargó de buscar reseñas y características para uno de los 3, y posteriormente probarlo:

5.1.1. Microsoft Incluir opiniones/reseñas, tabla de características y un ejemplo/prueba
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(myTesCaseEnumerator))].

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 guias y soporte para el uso de la herramienta
Documentación
Documentación clara para todas las operaciones que se quieran realizar
Flexibilidad

- Corre sobre Visual Studio
- Tiene dependencia con Selenium y NUnit test framework

Ventajas

- Ayuda a crear test de manera paralela, secuencial y ordenada-Puede ser ejecutado vía lineas 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

- Lento de adaptar y mantenerse al ritmo con nuevos test approaches

- No es compatible con .NET Core 2
  Diferencias destacables

- 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 Incluir
  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
Flexibiilidad
-Bastante flexible ya que permite agregar atributos, tipos de datos, assert
-Compatible con otros XUnit Frameworks
-Permite extender clase Assert
Ventajas

- crea una instancia nueva de la clase para cada test, lo cual garantiza que cada test se ejecute aisladamente
- Extensible para TDD
- Ejecución paralela de tests
- soporte de DataDrivenTests (múltiples entradas pasadas a 1 método de test en)
- 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
- 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
- no tiene mocks incluidos
- reciente, menos conocido

- documentación no estructurada
- no tiene disponibilidad de obtener test context
  Diferencias destacables
- 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 Idisponsable.Dispose
- ASSERTIONS:
  -no Throws.Nothing exception
  -no Fail
  -no Is.GreaterThan

Prueba
Para probar el framework, se creó un proyecto pequeño: tiaraRS/POC-Xunit-.net (github.com)
5.1.4. Elección - proveer ventajas y razón clara de porqué elegimos XUNIT
Se realizó una tabla general para comparar las características más destacables de los 3:

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:
(explicación de uso de mocks en esta estructura).
.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();
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
5.2.1. Opciones de cobertura + breve explicación
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]

a cada clase de las migraciones.

5.2.3. Guía para instalación Coverlet

Prerrequisitos

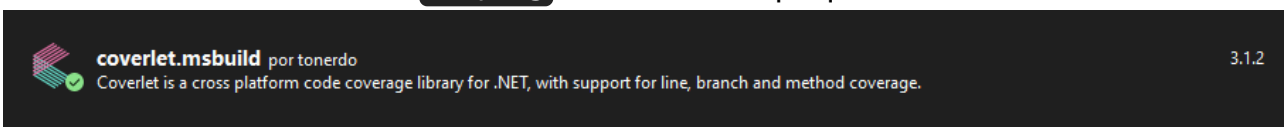
- Tener instalado `dotnet`
- Tener creado un proyecto de testing

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.


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

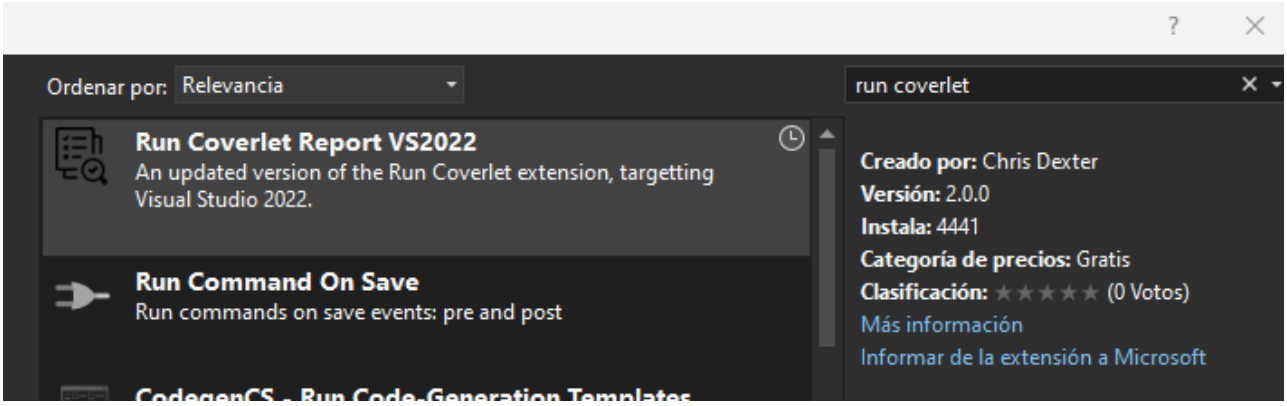
2. En un powershell con privilegios de administrador ejecutar

   ```
   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
   ```

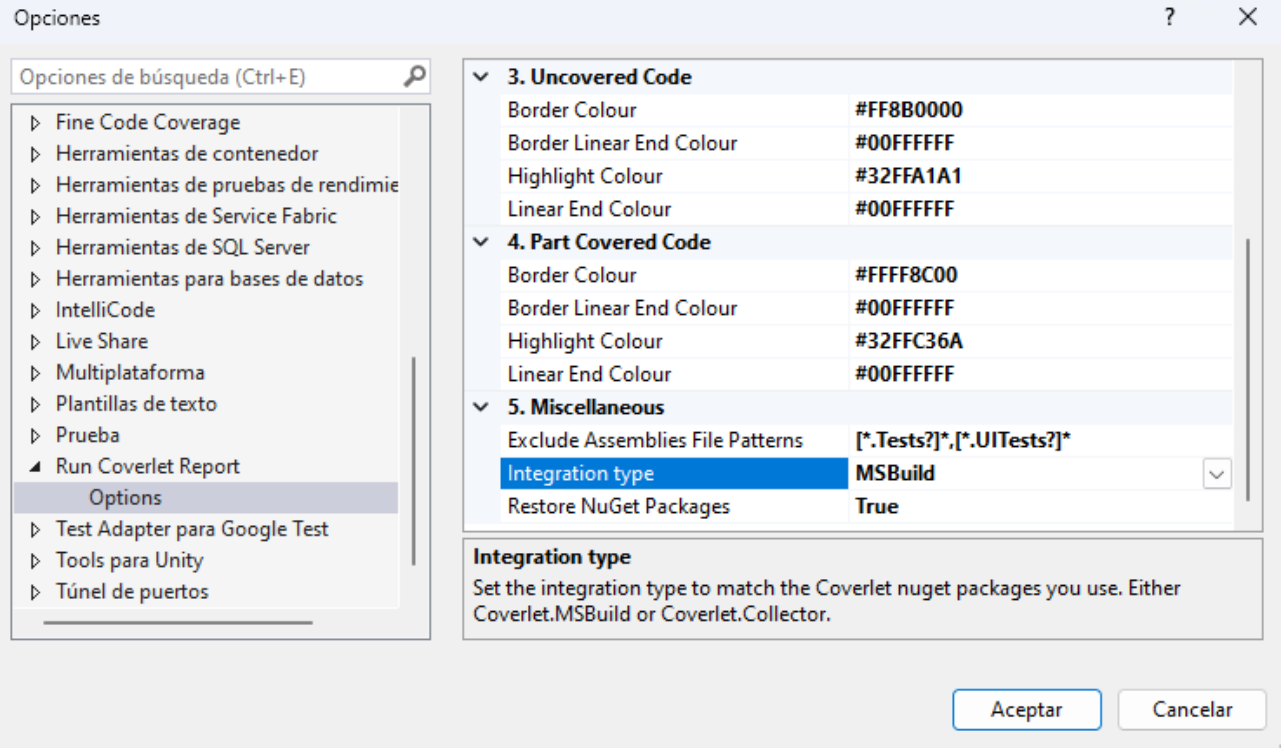
   Respuesta esperada

```
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`

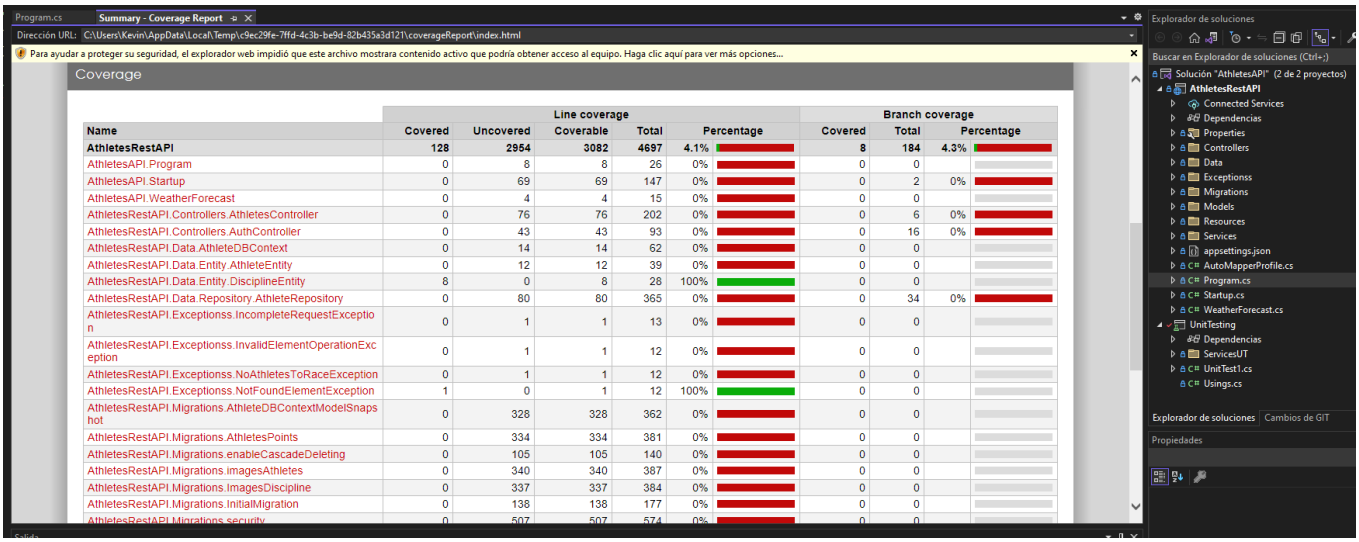


Instalación completa.

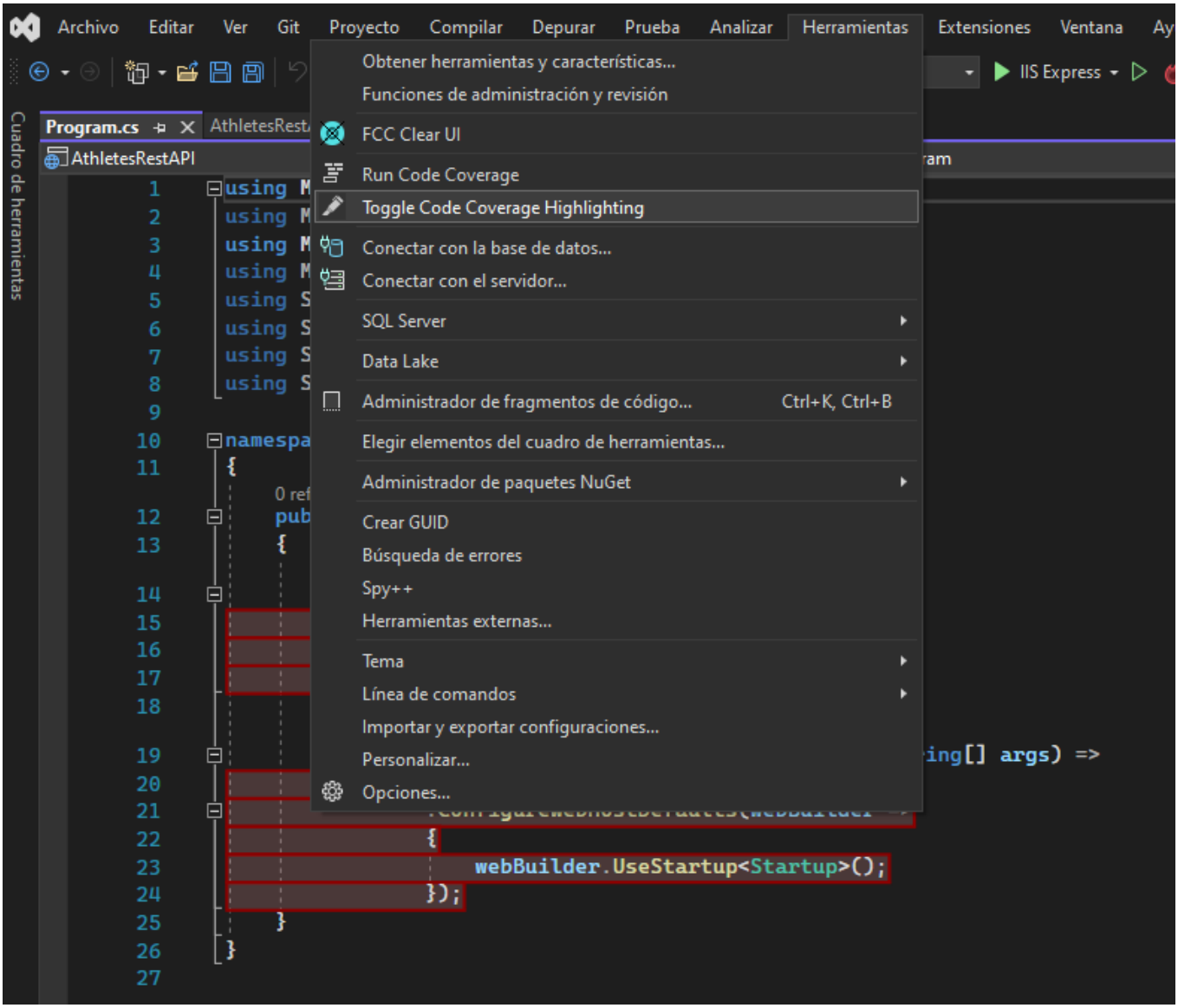
Uso y generación de reportes

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.



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 incremente 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 estos 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.

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/

3. Code coverage tool

3.1 Conclusiones y elección final

3.2. Guía de instalación y uso Coverlet

Prerrequisitos

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

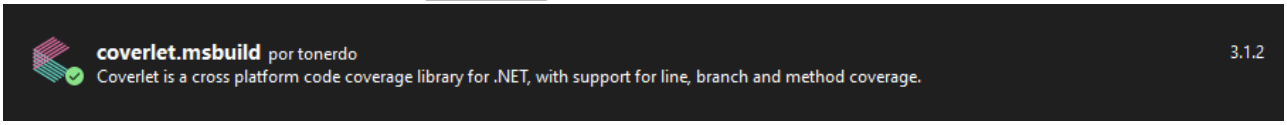
Instalación

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.




```
<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

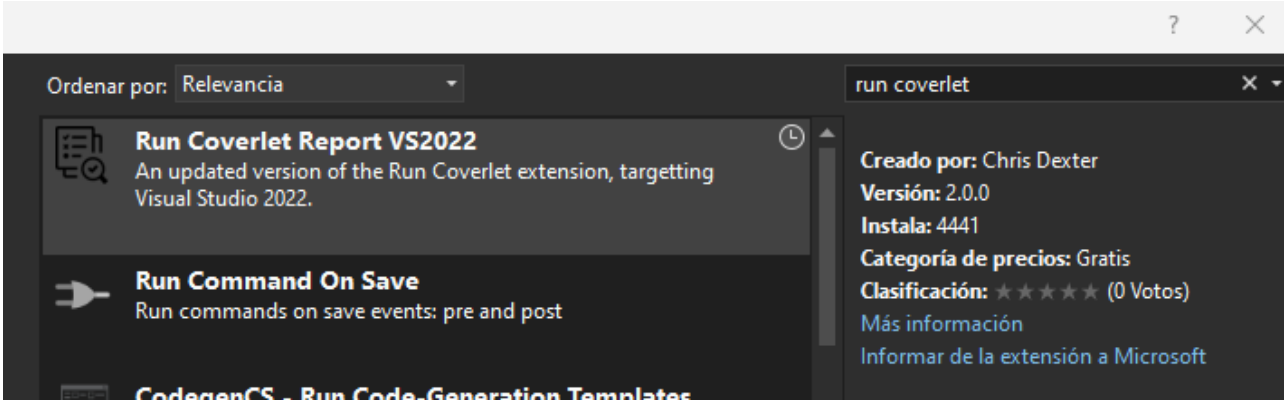
```
dotnet tool install -g dotnet-reportgenerator-globaltool
dotnet tool install dotnet-reportgenerator-globaltool --tool-path
tools
dotnet new tool-manifest
dotnet tool install dotnet-reportgenerator-globaltool
```

Respuesta esperada

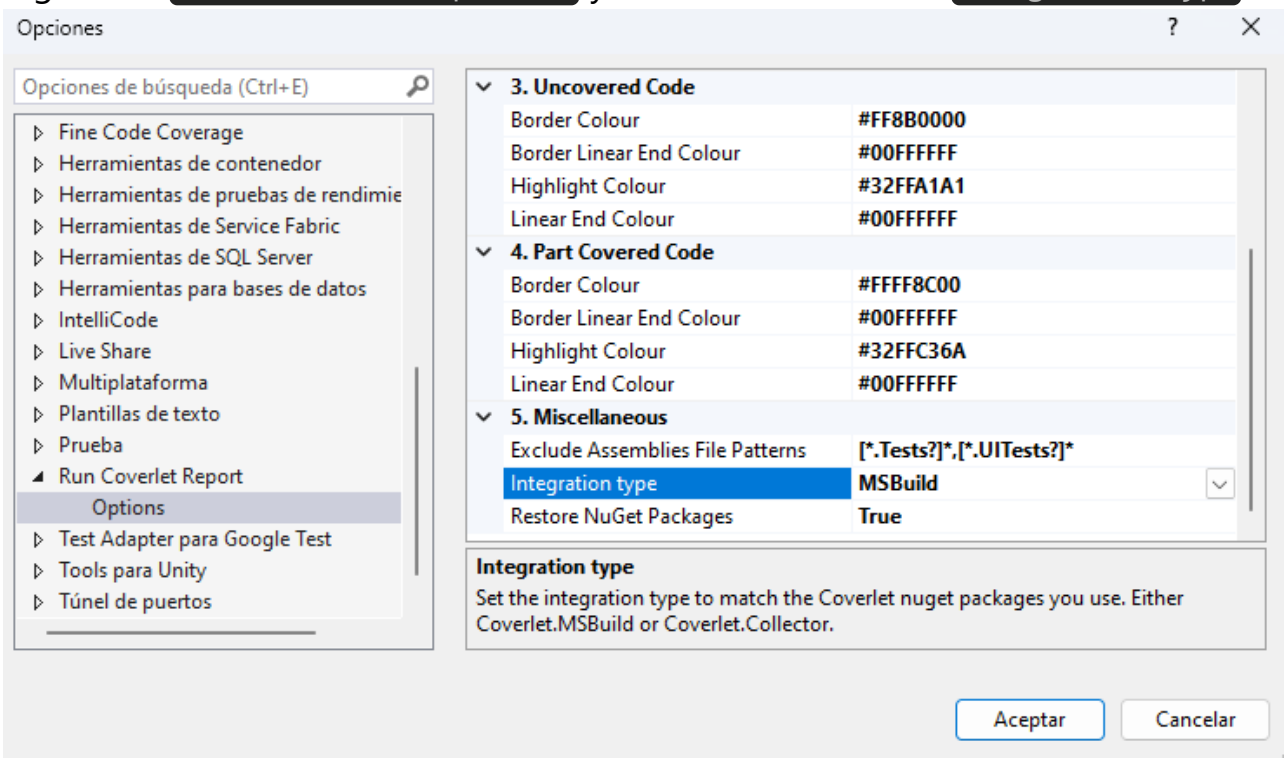
```
C:\WINDOWS\system32> dotnet tool install -g dotnet-reportgenerator-
globaltool
Puede invocar la herramienta con el comando siguiente: reportgenerator
La herramienta "dotnet-reportgenerator-globaltool" (versión '5.1.10')
se instaló correctamente.
C:\WINDOWS\system32> dotnet tool install dotnet-reportgenerator-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



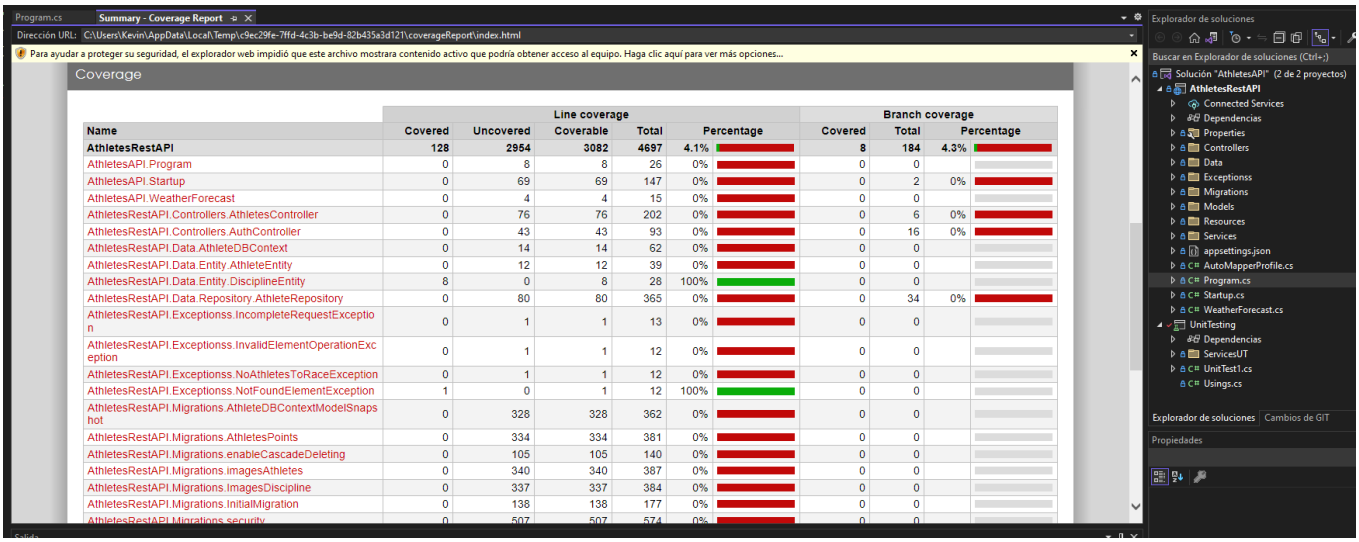
Ingresar a `Herramientas > Opciones` y modificar el valor de `Integration type`



Instalación completa.

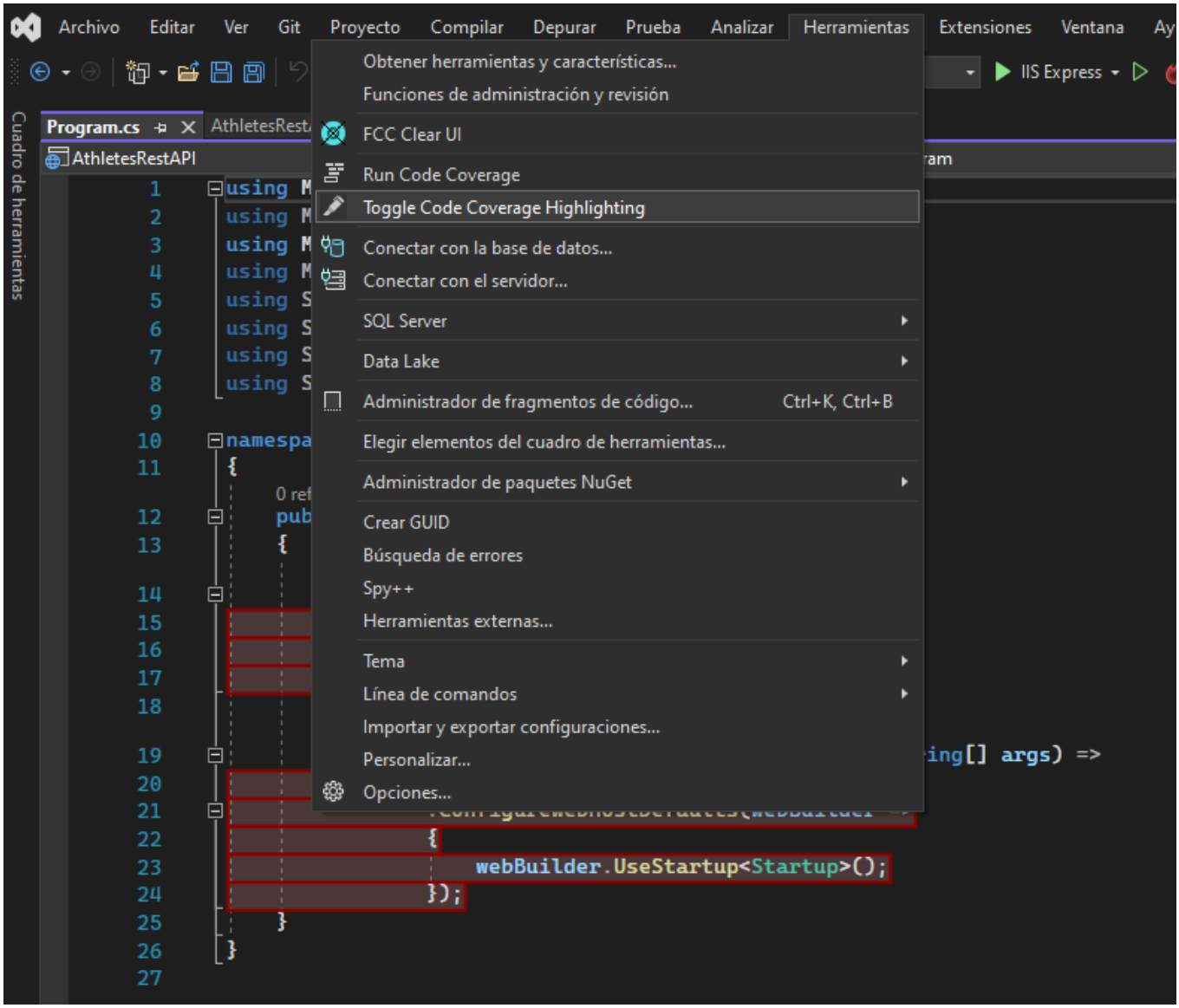
Uso y generación de reportes

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.



	Line coverage					Branch coverage		
Name	Covered	Uncovered	Coverable	Total	Percentage	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.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	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