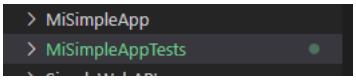
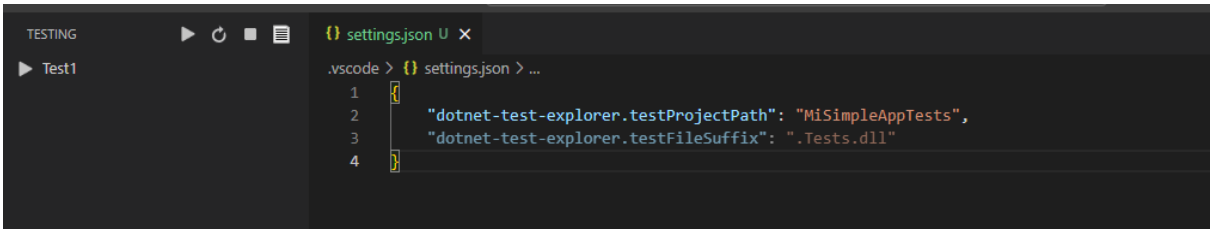
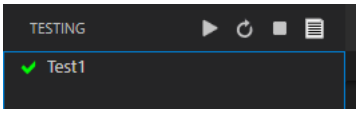
TP9




```
PS D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3> git clone https://github.com/ingsoft3ucc/MiSimpleApp.git
>> cd MiSimpleApp
>> code .
Cloning into 'MiSimpleApp'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
PS D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiSimpleApp> cd ..
>> dotnet new nunit -n MiSimpleAppTests
La plantilla "NUnit 3 Test Project" se creó correctamente.

Procesando acciones posteriores a la creación...
Restaurando D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiSimpleAppTests\MiSimpleAppTests.csproj:
  Determinando los proyectos que se van a restaurar...
  Se ha restaurado D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiSimpleAppTests\MiSimpleAppTests.csproj (en 24,29 sec).
Restauración realizada correctamente.


PS D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3> cd MiSimpleAppTests
>> dotnet add package NUnit
>> dotnet add package NUnit.ConsoleRunner
>> dotnet add reference ../MiSimpleApp/MiSimpleApp.csproj
>> cd ..
>> code .
  Determinando los proyectos que se van a restaurar...
  Writing C:\Users\Hp\AppData\Local\Temp\tmpA592.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Agregando PackageReference para el paquete "NUnit" al proyecto "D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiSimpleAppTests\MiSimpleAppTests.csproj".
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/nunit/index.json
```





Creamos nuestros Tests

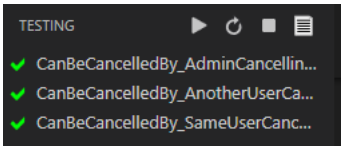
```
namespace MiSimpleAppTests;

[TestFixture]
public class Tests
{
    [SetUp]
    public void Setup()
    {
    }

    [Test]
    public void CanBeCancelledBy_AdminCancelling_ReturnsTrue()
    {
        //Arrange

        User user=new User();
        user.IsAdmin=true;
        Reservation reservation=new Reservation();

        //Act

        bool result=reservation.CanBeCancelledBy(user);

        //Assert

        //Assert.IsTrue(result);
        Assert.That(result,Is.True);
        //Assert.That(result==true);
    }


    [Test]
    public void CanBeCancelledBy_SameUserCancelling_ReturnsTrue()
    {
        //Arrange
        User user=new User();
        Reservation reservation=new Reservation();
        reservation.MadeBy=user;

        //Act
        bool result=reservation.CanBeCancelledBy(user);

        //Assert
        Assert.That(result,Is.True);

    }


    [Test]
    public void CanBeCancelledBy_AnotherUserCancelling_ReturnsFalse()
    {
        //Arrange
        User user=new User();
        Reservation reservation=new Reservation();
        reservation.MadeBy=user;

        //Act
        User newUser=new User();
        bool result=reservation.CanBeCancelledBy(newUser);

        //Assert
        Assert.That(result,Is.False);

    }

}
```

Este código parece ser una serie de pruebas unitarias escritas en C# utilizando el framework NUnit. Las pruebas están diseñadas para probar el comportamiento de la clase `Reservation` y su método `CanBeCancelledBy`.

A continuación, desglosaré el código:

1. Se utiliza el espacio de nombres `MiSimpleAppTests`.

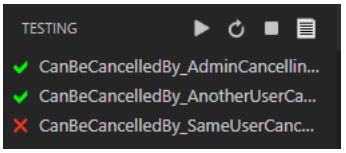
2. Se define una clase llamada `Tests` que contendrá las pruebas.
3. En el método `Setup`, se establece la configuración inicial para cada prueba. En este caso, el método está vacío, por lo que no realiza ninguna configuración adicional.
4. Hay tres métodos de prueba etiquetados con `[Test]`, que son métodos de prueba unitaria:
    - `CanBeCancelledBy_AdminCancelling_ReturnsTrue`: Esta prueba verifica si un administrador puede cancelar una reserva. Se crea un usuario (`User`) y una reserva (`Reservation`). Luego, se llama al método `CanBeCancelledBy` de la reserva con el usuario administrador y se verifica si el resultado es `true`.
    - `CanBeCancelledBy_SameUserCancelling_ReturnsTrue`: Esta prueba verifica si un usuario puede cancelar su propia reserva. Se crea un usuario y una reserva, y se establece que la reserva fue realizada por el mismo usuario. Luego, se llama al método `CanBeCancelledBy` con el mismo usuario y se verifica si el resultado es `true`.
    - `CanBeCancelledBy_AnotherUserCancelling_ReturnsFalse`: Esta prueba verifica si un usuario distinto al que realizó la reserva puede cancelarla. Se crea un usuario y una reserva, y se establece que la reserva fue realizada por el mismo usuario. Luego, se crea un nuevo usuario y se llama al método `CanBeCancelledBy` con el nuevo usuario. Se verifica que el resultado sea `false`.
5. Dentro de cada prueba, se siguen los siguientes pasos comunes:
    - `Arrange`: Se crean las instancias necesarias de objetos (por ejemplo, usuarios y reservas) y se establece cualquier estado necesario para la prueba.
    - `Act`: Se realiza la acción que se va a probar, que en este caso es llamar al método `CanBeCancelledBy` en la reserva.
    - `Assert`: Se verifica el resultado esperado de la acción realizada. En este caso, se utiliza la clase `Assert` de NUnit para comprobar si el resultado es `true` o `false`.

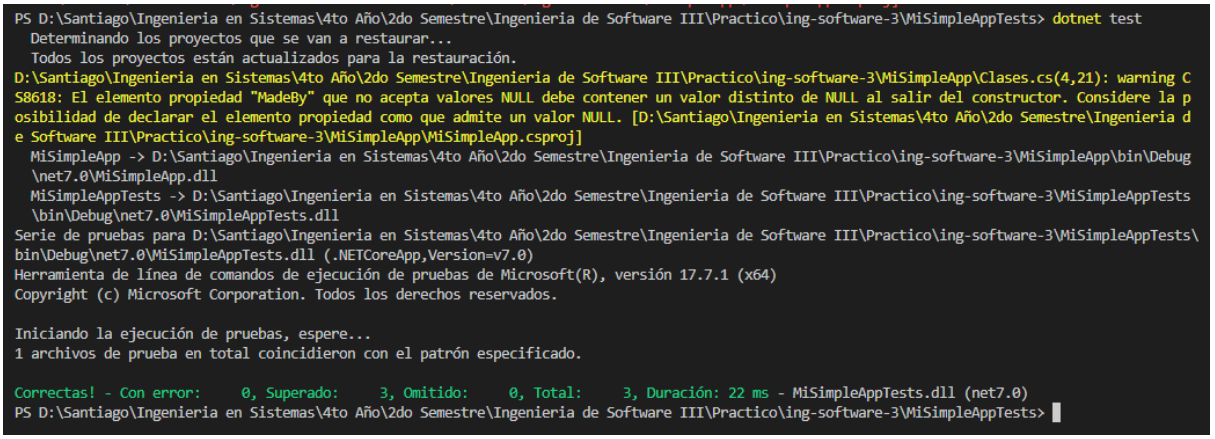


cambiamos un true por False

```
public class Reservation
{
    1 reference
    public User MadeBy { get; set; }

    0 references
    public bool CanBeCancelledBy(User user)
    {
        if (user.IsAdmin)
            return false;
        if (MadeBy==user)
            return true;
        return false;
```

y el resultado de los test cambia:





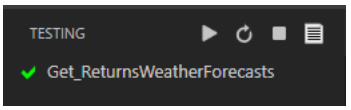

```
using Microsoft.Extensions.Logging;
using NUnit.Framework;
using SimpleWebAPI.Controllers;
using System;
using System.Linq;

namespace SimpleWebAPI.Tests
{
    [TestFixture]
    public class WeatherForecastControllerTests
    {
        [Test]
        public void Get_ReturnsWeatherForecasts()
        {
            // Arrange
            ILogger<WeatherForecastController> logger = new LoggerFactory().CreateLogger<WeatherForecastController>();
            var controller = new WeatherForecastController(logger);

            // Act
            var result = controller.Get();

            // Assert
            Assert.NotNull(result);
            Assert.AreEqual(5, result.Count());
        }
    }
}
```

Este código representa una prueba unitaria en C# utilizando el framework NUnit para la clase `WeatherForecastController`. La prueba verifica el comportamiento del método `Get` en el controlador `WeatherForecastController` de una API web.

A continuación, se desglosa el código:

1. Se utiliza el espacio de nombres `Microsoft.Extensions.Logging`, `NUnit.Framework`, `SimpleWebAPI.Controllers` y otros según sea necesario.
2. Se define una clase llamada `WeatherForecastControllerTests`, que contiene la prueba unitaria.
3. Dentro de la prueba `[Test]`, llamada `Get_ReturnsWeatherForecasts`, se realizan los siguientes pasos:
    - **Arrange**: En esta sección, se prepara el entorno para la prueba. Se crea un objeto `logger` de tipo `ILogger<WeatherForecastController>` utilizando una instancia de `LoggerFactory`. Luego, se crea una instancia del controlador `WeatherForecastController` pasando el `logger` como parámetro en el constructor.
    - **Act**: En esta sección, se ejecuta la acción que se va a probar. En este caso, se llama al método `Get` del controlador y se almacena el resultado en la variable `result`.
    - **Assert**: En esta sección, se realizan afirmaciones para verificar si el resultado de la acción es el esperado. Se utilizan las siguientes afirmaciones:
        - `Assert.NotNull(result)`: Se asegura de que el resultado no sea nulo, lo que significa que se espera una respuesta válida del método `Get`.

- `Assert.AreEqual(5, result.Count())`: Se verifica que el número de elementos en el resultado sea igual a 5. Esto es específico para esta prueba y depende de la lógica interna del método `Get` del controlador.




```
using System;
using System.Net.Http;
using System.Net.Http.Json;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;

namespace MyApp
{
    class Program
    {
        static async Task Main()
        {
            var serviceProvider = new ServiceCollection()
                .AddHttpClient()
                .AddTransient<IApiService, ApiService>()
                .BuildServiceProvider();

            var apiService = serviceProvider.GetRequiredService<IApiService>();

            var myModels = await apiService.GetMyModelsAsync();
            foreach (var model in myModels)
            {
               Console.WriteLine($"Id: {model.Id}, Title: {model.Title}");
            }
        }
    }
}
```

Este código muestra un programa de consola en C# que utiliza el patrón de inyección de dependencias (DI) y el cliente HTTP para consumir una API web y mostrar los resultados en la consola. Aquí hay una descripción de lo que hace el código:

1. Se incluyen los espacios de nombres necesarios para trabajar con HTTP, JSON y la inyección de dependencias.
2. Se define la clase `Program` que contiene el método `Main`, que es el punto de entrada del programa.
3. Dentro del método `Main`, se realiza lo siguiente:
   - Se crea un `ServiceProvider` utilizando `ServiceCollection`. Esto establece un contenedor de inyección de dependencias que se utilizará para registrar y resolver servicios.
   - Se agrega un cliente HTTP utilizando `AddHttpClient()`. Esto permite realizar solicitudes HTTP a una API web.
   - Se registra una implementación de la interfaz `IApiService` como `ApiService` utilizando `AddTransient()`. Esto significa que cada vez que se solicite un servicio que implemente `IApiService`, se creará una nueva instancia de `ApiService`.
   - Se construye el proveedor de servicios llamando a `BuildServiceProvider()`.
   - Se obtiene una instancia de `IApiService` del proveedor de servicios. Esto se hace llamando a `GetRequiredService<IApiService>()`.
   - Se llama al método `GetMyModelsAsync()` en `apiService` para obtener una colección de modelos desde una API web. Esto parece ser una operación asincrónica.
   - Se itera a través de los modelos obtenidos e imprime sus propiedades `Id` y `Title` en la consola.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Http;
```

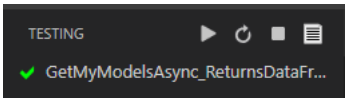
```
using System.Net.Http.Json;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;
using Moq;
using NUnit.Framework;
using Moq.Protected;

namespace MiNotSoSimpleAppTests
{
    public class ApiServiceTests
    {
        [Test]
        public async Task GetMyModelsAsync_ReturnsDataFromHttpClient()
        {
            // Arrange
            var serviceCollection = new ServiceCollection();
            var mockResponse = new HttpResponseMessage(HttpStatusCode.OK)
            {
                Content = new StringContent("[{ \"UserId\": 1, \"Id\": 1, \"Title\": \"Test Title\", \"Body\": \"Test Body\" }]")
            };

            var mockHttpMessageHandler = new Mock<HttpMessageHandler>();
            mockHttpMessageHandler.Protected()
                .Setup<Task<HttpResponseMessage>>("SendAsync", ItExpr.IsAny<HttpRequestMessage>(), ItExpr.IsAny<CancellationToken>
                .ReturnsAsync(mockResponse);

            serviceCollection.AddTransient<IApiService>(_ => new ApiService(new HttpClient(mockHttpMessageHandler.Object)));
            var serviceProvider = serviceCollection.BuildServiceProvider();

            var apiService = serviceProvider.GetRequiredService<IApiService>();

            // Act
            var result = await apiService.GetMyModelsAsync();

            // Assert
            Assert.IsNotNull(result);
            Assert.AreEqual(1, result.Count());
            Assert.AreEqual("Test Title", result.FirstOrDefault().Title);
        }
    }
}
```

Este código representa una prueba unitaria en C# utilizando el framework NUnit y la biblioteca Moq para probar la clase `ApiService`. La prueba se asegura de que el método `GetMyModelsAsync` de `ApiService` funcione correctamente al realizar solicitudes HTTP simuladas utilizando Moq en lugar de una solicitud HTTP real. Aquí está el desglose del código:

1. Se incluyen los espacios de nombres necesarios para trabajar con pruebas unitarias, Moq, HTTP, JSON y la inyección de dependencias.
2. Se define la clase `ApiServiceTests`, que contiene la prueba unitaria.
3. Dentro del método `[Test]` llamado `GetMyModelsAsync_ReturnsDataFromHttpClient`, se realiza lo siguiente:
   - **Arrange** (Configuración): En esta sección, se prepara el entorno para la prueba.
     - Se crea una instancia de `ServiceCollection` que se utilizará para configurar la inyección de dependencias.
     - Se crea un objeto `mockResponse` de tipo `HttpResponseMessage` que simula una respuesta HTTP exitosa (código de estado 200) con un cuerpo JSON que contiene un modelo de datos ficticio.
     - Se crea un objeto `mockHttpMessageHandler` de tipo `Mock<HttpMessageHandler>`. Esto se utiliza para interceptar y controlar las solicitudes HTTP simuladas.
     - Se configura el comportamiento del objeto `mockHttpMessageHandler` para que devuelva `mockResponse` cuando se llame al método `SendAsync` de `HttpMessageHandler`. Esto simula una solicitud HTTP exitosa.
     - Se registra una implementación de la interfaz `IApiService` como `ApiService` en la colección de servicios. Se pasa un `HttpClient` que utiliza el `mockHttpMessageHandler` como dependencia. Esto garantiza que las solicitudes HTTP realizadas por `ApiService` sean interceptadas por el `mockHttpMessageHandler` y devuelvan la respuesta simulada.
     - Se construye el proveedor de servicios llamando a `BuildServiceProvider()`.
     - Se obtiene una instancia de `IApiService` del proveedor de servicios.
   - **Act** (Acción): En esta sección, se ejecuta el código que se va a probar.

- Se llama al método `GetMyModelsAsync` en `apiService` para obtener una colección de modelos desde una API web simulada.

- **Assert** (Afirmaciones): En esta sección, se realizan afirmaciones para verificar si el resultado de la acción es el esperado.
  - `Assert.IsNotNull(result)`: Se verifica que el resultado no sea nulo, lo que significa que se espera una respuesta válida.
  - `Assert.AreEqual(1, result.Count())`: Se verifica que la colección de resultados contenga exactamente un elemento.
  - `Assert.AreEqual("Test Title", result.FirstOrDefault().Title)`: Se verifica que el título del primer elemento de la colección sea igual a "Test Title".




```
re III\Practico\ing-software-3\MiNotSoSimpleApp\MiNotSoSimpleApp.csproj]
  MiNotSoSimpleApp -> D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleApp
  \bin\Debug\net7.0\MiNotSoSimpleApp.dll
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests\UnitTest1.cs(42,
43): warning CS8602: Desreferencia de una referencia posiblemente NULL. [D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de So
ftware III\Practico\ing-software-3\MiNotSoSimpleAppTests\MiNotSoSimpleAppTests.csproj]
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests\UnitTest1.cs(41,
13): warning NUnit2005: Consider using the constraint model, Assert.That(actual, Is.EqualTo(expected)), instead of the classic model, Assert.AreE
qual(expected, actual) (https://github.com/nunit/nunit.analyzers/tree/master/documentation/NUnit2005.md) [D:\Santiago\Ingenieria en Sistemas\4to
Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests\MiNotSoSimpleAppTests.csproj]
D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests\UnitTest1.cs(42,
13): warning NUnit2005: Consider using the constraint model, Assert.That(actual, Is.EqualTo(expected)), instead of the classic model, Assert.AreE
qual(expected, actual) (https://github.com/nunit/nunit.analyzers/tree/master/documentation/NUnit2005.md) [D:\Santiago\Ingenieria en Sistemas\4to
Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests\MiNotSoSimpleAppTests.csproj]
  MiNotSoSimpleAppTests -> D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimp
  leAppTests\bin\Debug\net7.0\MiNotSoSimpleAppTests.dll
Serie de pruebas para D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppT
ests\bin\Debug\net7.0\MiNotSoSimpleAppTests.dll (.NETCoreApp,Version=v7.0)
Herramienta de línea de comandos de ejecución de pruebas de Microsoft(R), versión 17.7.1 (x64)
Copyright (c) Microsoft Corporation. Todos los derechos reservados.

Iniciando la ejecución de pruebas, espere...
1 archivos de prueba en total coincidieron con el patrón especificado.

Correctas! - Con error:     0, Superado:     1, Omitido:     0, Total:     1, Duración: 325 ms - MiNotSoSimpleAppTests.dll (net7.0)
PS D:\Santiago\Ingenieria en Sistemas\4to Año\2do Semestre\Ingenieria de Software III\Practico\ing-software-3\MiNotSoSimpleAppTests>
```