Reporte Lab 9
Daniel Lizano Morales
C04285

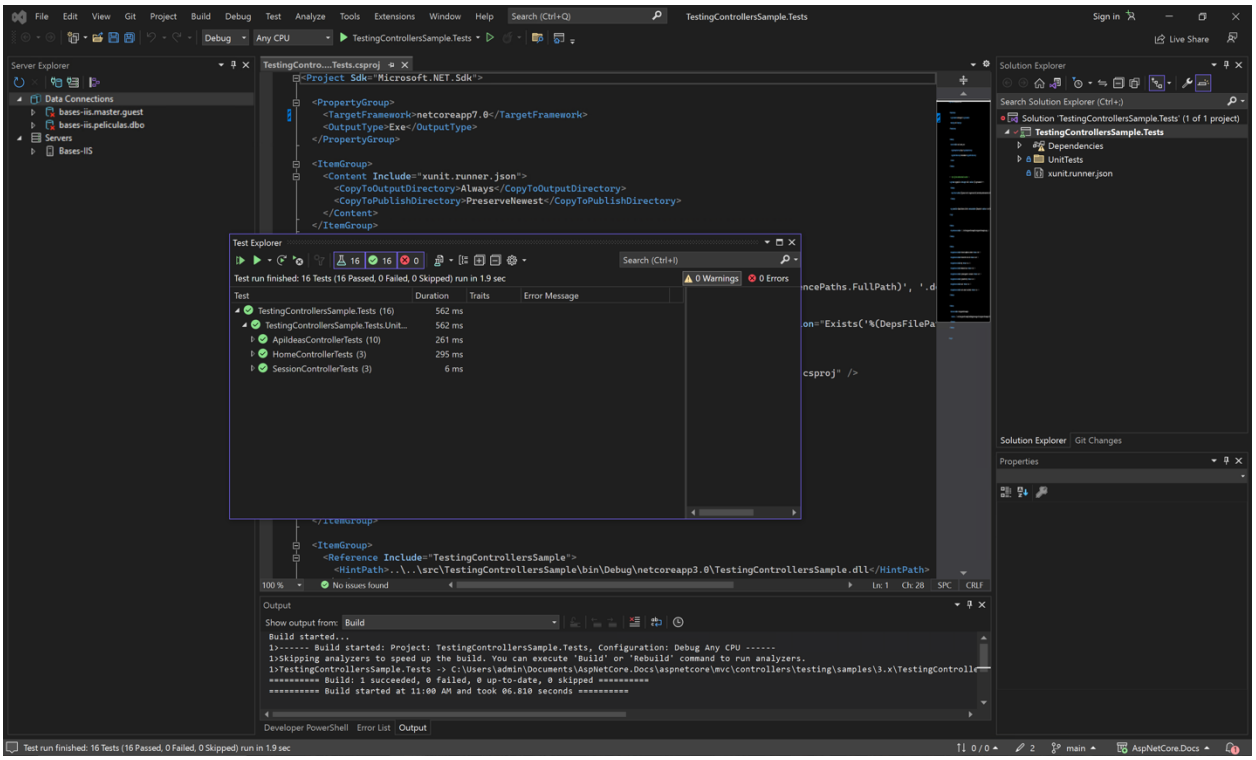
Preguntas de controller

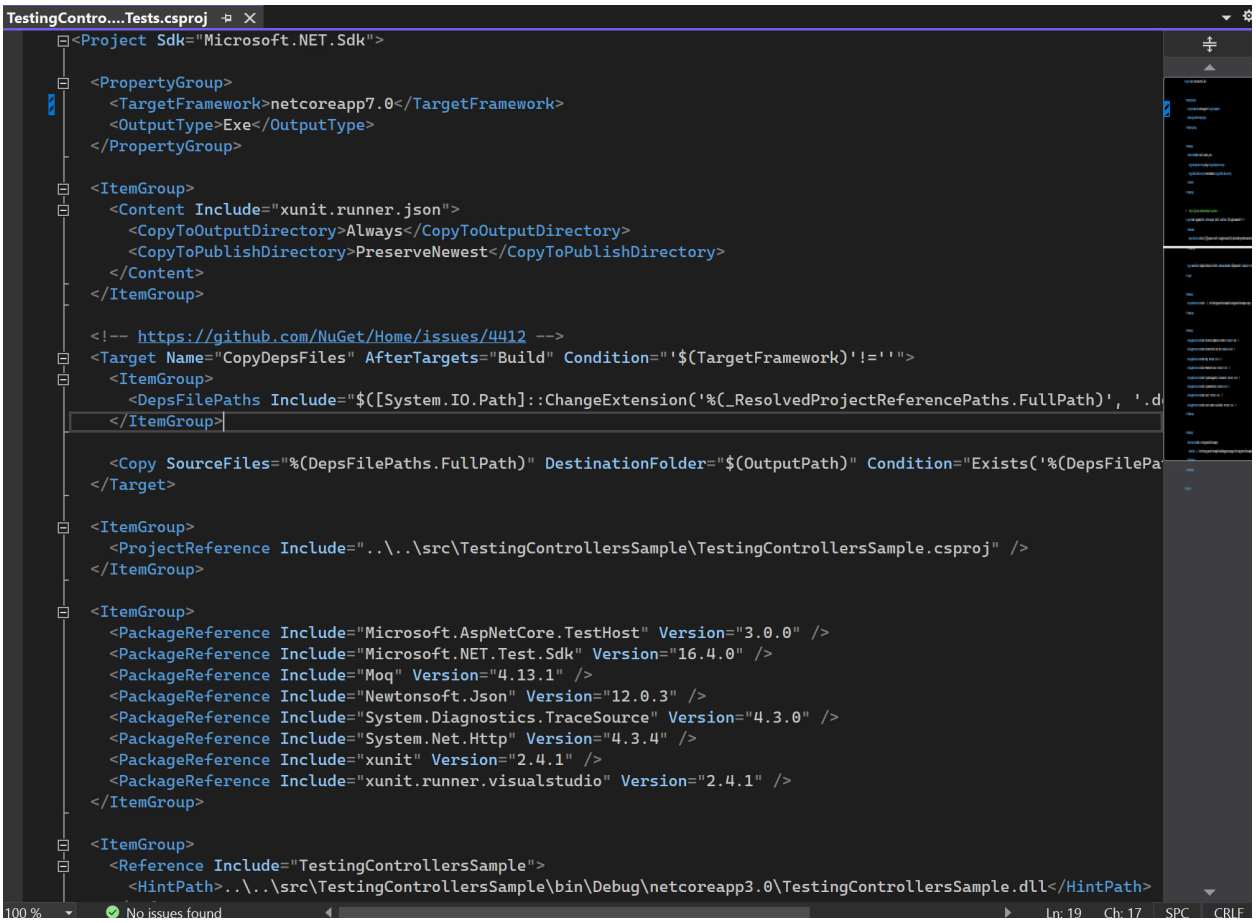
Respuestas:

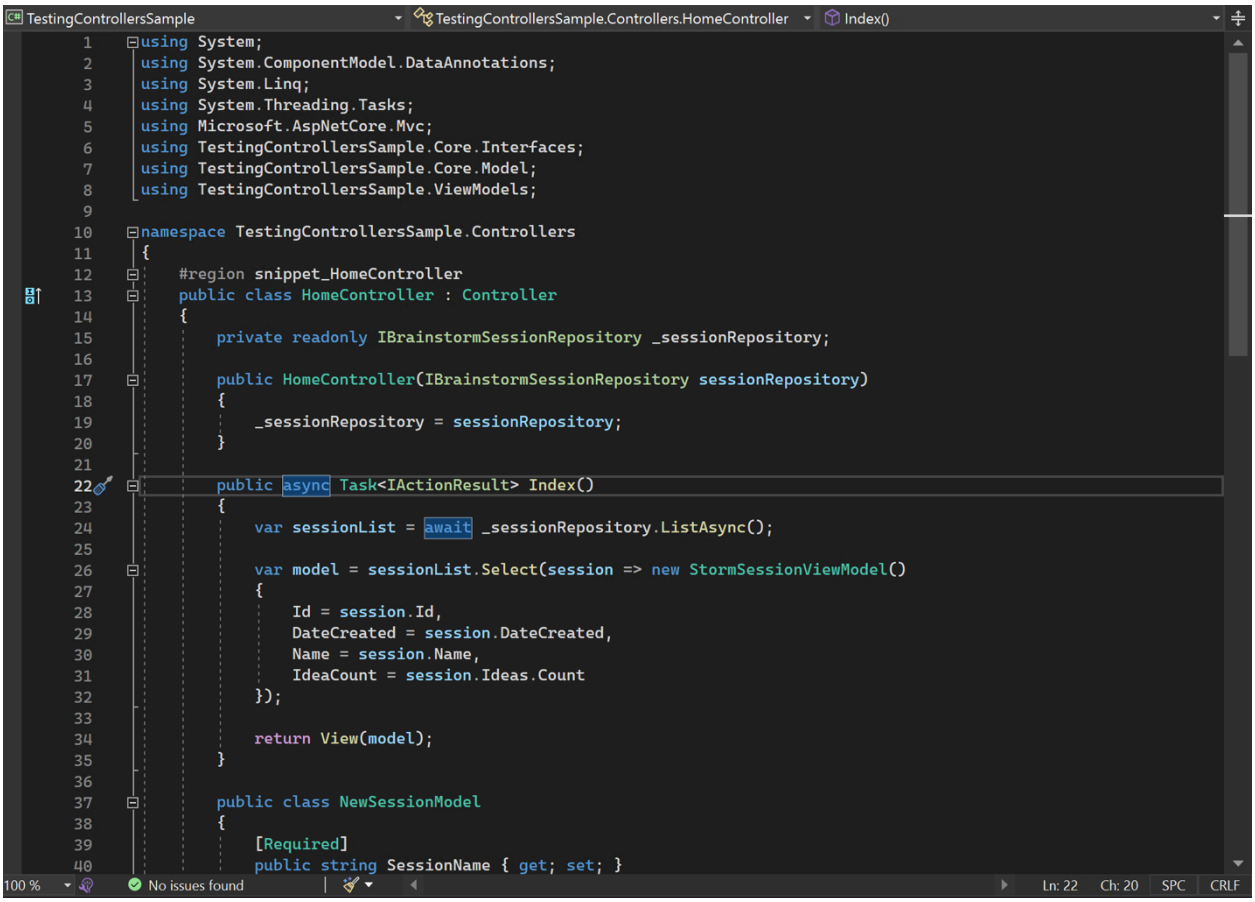
A) Los bloques de pruebas unitarias cuentan con la etiqueta [FACT] para que el sistema sepa que es una prueba unitaria en NUnit, indica que se debe de ejecutar de manera independiente e insolada.

B) Se crea un objeto simulado llamado mockrepo que puede devolver datos simulados en el contexto de esa prueba cuando se hace el llamado a listAsync(). Mas adelante este objeto tiene la capacidad de poder generar un controlador con HomeController()

C) Se usan tres métodos distintos para poder evaluar distintos resultados que pueden tener las pruebas. Como la verificación de resultados esperados, validación de límites y cobertura de que tipo de camino escogimos que hiciera la prueba.




TestingContro....Tests.csproj
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<TargetFramework>netcoreapp7.0</TargetFramework>
<OutputType>Exe</OutputType>
</PropertyGroup>
<ItemGroup>
<Content Include="xunit.runner.json">
<CopyToOutputDirectory>Always</CopyToOutputDirectory>
<CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
</Content>
</ItemGroup>
<!-- https://github.com/NuGet/Home/issues/4412 -->
<Target Name="CopyDepsFiles" AfterTargets="Build" Condition="'$(TargetFramework)'!=''">
<ItemGroup>
<DepsFilePaths Include="$([System.IO.Path]::ChangeExtension('%(_ResolvedProjectReferencePaths.FullPath)', '.d
</ItemGroup>
<Copy SourceFiles="%(DepsFilePaths.FullPath)" DestinationFolder="$(OutputPath)" Condition="Exists('%(DepsFilePa
</Target>
<ItemGroup>
<ProjectReference Include="..\..\src\TestingControllersSample\TestingControllersSample.csproj" />
</ItemGroup>
<ItemGroup>
<PackageReference Include="Microsoft.AspNetCore.TestHost" Version="3.0.0" />
<PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.4.0" />
<PackageReference Include="Moq" Version="4.13.1" />
<PackageReference Include="Newtonsoft.Json" Version="12.0.3" />
<PackageReference Include="System.Diagnostics.TraceSource" Version="4.3.0" />
<PackageReference Include="System.Net.Http" Version="4.3.4" />
<PackageReference Include="xunit" Version="2.4.1" />
<PackageReference Include="xunit.runner.visualstudio" Version="2.4.1" />
</ItemGroup>
<ItemGroup>
<Reference Include="TestingControllersSample">
<HintPath>..\..\src\TestingControllersSample\bin\Debug\netcoreapp3.0\TestingControllersSample.dll</HintPath>
100 %
No issues found
Ln: 19
Ch: 17
SPC
CRLF


TestingControllersSample
TestingControllersSample.Controllers.HomeController
Index()
using System;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using TestingControllersSample.Core.Interfaces;
using TestingControllersSample.Core.Model;
using TestingControllersSample.ViewModels;
namespace TestingControllersSample.Controllers
{
#region snippet_HomeController
public class HomeController : Controller
{
private readonly IBrainstormSessionRepository _sessionRepository;
public HomeController(IBrainstormSessionRepository sessionRepository)
{
_sessionRepository = sessionRepository;
}
public async Task<IActionResult> Index()
{
var sessionList = await _sessionRepository.ListAsync();
var model = sessionList.Select(session => new StormSessionViewModel()
{
Id = session.Id,
DateCreated = session.DateCreated,
Name = session.Name,
IdeaCount = session.Ideas.Count
});
return View(model);
}
public class NewSessionModel
{
[Required]
public string SessionName { get; set; }
100 %
No issues found
Ln: 22
Ch: 20
SPC
CRLF

Pruebas para realizar en el PI

En el Hook que creamos para el proyecto useExchange.jsx
En la función flecha useExchange podemos hacer una prueba que realice el happy path de que tanto la data como el setter de la función
Como mock usaríamos los token de axios para la conexión con el API y el cliente que se comunica con el backend
La prueba sería la siguiente:

```
it('test_return_exchange_rate_data_and_setter_function', async () => {
    const mockResult = {
        data: [
            {
                Value: 10
            }
        ]
    };
    const mockAuthToken = jest.fn();
    const mockAxiosClient = jest.fn().mockResolvedValue(mockResult);
    jest.mock('../config/AuthToken', () => mockAuthToken);
    jest.mock('../config/AxiosClient', () => mockAxiosClient);
    const { exchange, setExchange } = useExchange();
    await expect(exchange).toEqual({
        USD: 10,
        CRC: 0.1
    });
    await expect(setExchange).toBeInstanceOf(Function);
});
```

Tambien podemos plantear un Edge case en este mismo hook. Este Edge case revisaría si un token es nulo o tiene un valor indefinido.
Para ello como mock usaríamos nuevamente el token y el cliente.

```
  it('test_token_is_null_or_undefined', async () => {
      const mockAuthToken = jest.fn();
      const mockAxiosClient = jest.fn();
      jest.mock('../config/AuthToken', () => mockAuthToken);
      jest.mock('../config/AxiosClient', () => mockAxiosClient);
      const { exchange } = useExchange();
      await expect(mockAuthToken).toThrow();
  });
```

En el hook useServices propongo dos test
La prueba happy path que el fetch de servicesNames revise que los servicesNames estén correctamente
Usando un mock que sustituye los nombres de los servicios

```
it('test_fetch_services_names', async () => {
    const mockSetServicesNames = jest.fn();
    jest.spyOn(React, 'useContext').mockReturnValue({ token: 'mockToken' });
    jest.spyOn(global, 'fetch').mockResolvedValueOnce({
        json: async () => ({ data: [{ Name: 'mockName' }] })
    });
    await useServices().fetchServicesNames(mockSetServicesNames);
    expect(mockSetServicesNames).toHaveBeenCalledWith(['mockName']);
});
```

Y otra prueba happy path pero que esta vez se encarga de revisar si devuelve los precios correctamente usando como mock un servicio x

```
it('test_search_service_price', () => {
    const servicesPrices = [{ Name_Service: 'mockName', Currency: 'USD', Price: 10 }];
    const result = useServices().searchServicePrice('mockName', 'USD', servicesPrices);
    expect(result).toEqual(10);
});
```

Link al repo: https://github.com/DanielLM2002/c04285_ci0126_23a.git

Resumen:

A) Realmente no aprendí nada nuevo con el laboratorio ya que todo esto lo habíamos realizado en la entrega pasada del PI.
B) Se me dificultó hacer que las pruebas corrieran debido a un problema interno de .NET
C) Se me hizo muy fácil de realizar el tutorial debido a que Microsoft explica bien que es cada cosa y cuál es su función.
D) Tarde alrededor de 4 horas .