Jose Andres Matarrita Miranda C04668

A) En el segundo bloque de código, explique brevemente ¿Por qué los casos de prueba unitarios tienen la etiqueta [Fact] antes de la firma del método de prueba?

La etiqueta `[Fact]` se utiliza en xUnit.net para marcar los métodos de prueba unitarios. En el código proporcionado, el método está marcado con `[Fact]` para indicar que es una prueba unitaria que se debe ejecutar.

En este caso de prueba, se está probando el comportamiento del método `Index` en el controlador de inicio. La prueba verifica si el método devuelve un resultado de `ViewResult` que contiene una lista de sesiones de tormenta (`BrainstormSessions`).

Al utilizar la etiqueta `[Fact]`, se garantiza que este método sea ejecutado como una prueba unitaria durante la ejecución de las pruebas, y las verificaciones se realizan para confirmar que el comportamiento del método `Index` sea el esperado.

B) En el segundo bloque de código donde aparece el caso de prueba

```
Index_ReturnsAViewResult_WithAListOfBrainstormSessions()
```

Explique brevemente el efecto que tienen las siguientes líneas de código sobre el objeto **`controller`**:

```
// Arrange
var mockRepo = new Mock<IBrainstormSessionRepository>();
mockRepo.Setup(repo => repo.ListAsync())
    .ReturnsAsync(GetTestSessions());
var controller = new HomeController(mockRepo.Object);
```

Estas líneas de código en la sección "Arrange" preparan el entorno de la prueba al configurar el objeto simulado del repositorio de sesiones y crear una instancia del controlador utilizando dicho objeto simulado. De esta manera, se asegura que el controlador utilice el comportamiento esperado durante la ejecución de la prueba.

El efecto de estas líneas de código es garantizar que el controlador `HomeController` utilice el objeto simulado del repositorio de sesiones, y que el método `ListAsync()` en el repositorio simulado devuelva una lista de sesiones de prueba específica al ser invocado desde el método `Index()` del controlador. Esto permite realizar la prueba de forma controlada y predecible, verificando el resultado obtenido en el método `Index()` con las aserciones realizadas en la sección "Assert".

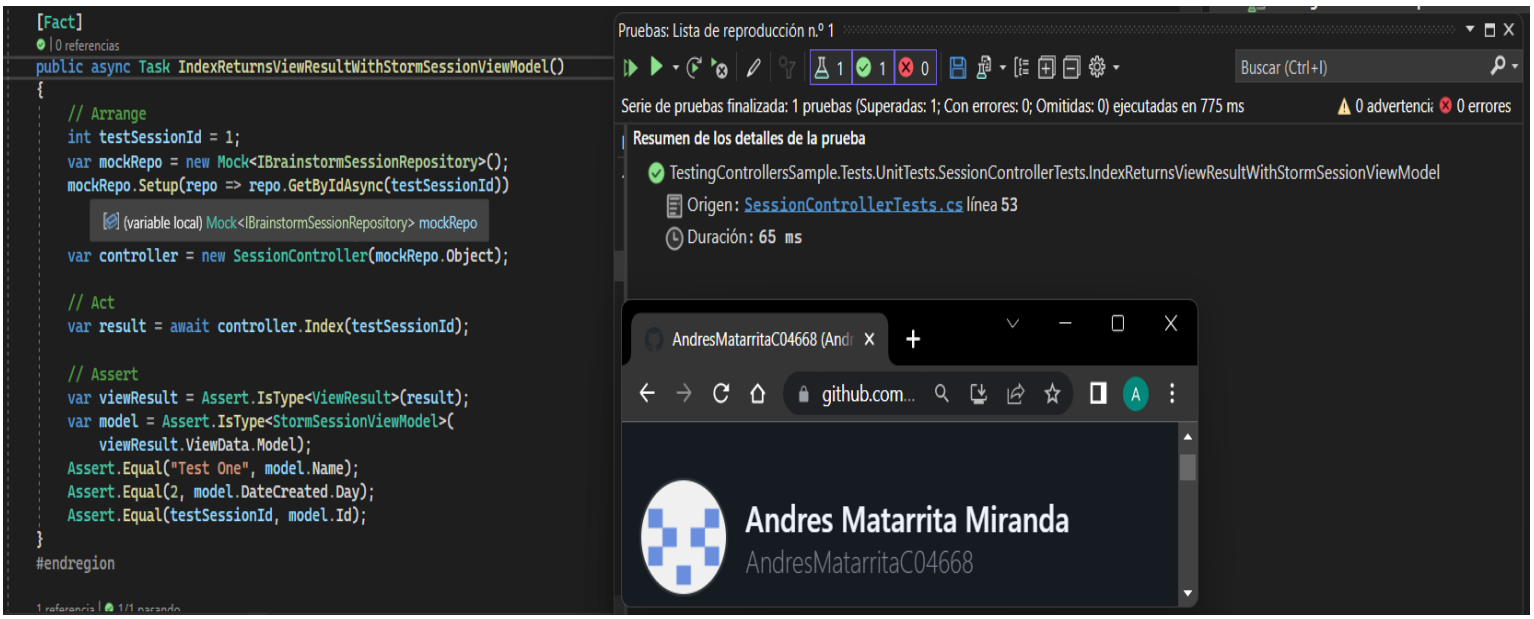
C) En el sexto bloque de código, explique brevemente ¿Por qué se utilizan tres casos de prueba para el método Index de la clase SessionController?

Los tres casos de prueba para el método `Index` de `SessionController` se utilizan para:
1. Verificar que se redireccione correctamente a la acción "Index" del controlador "Home" cuando el ID es nulo.
2. Asegurar que se retorne un contenido específico indicando "Session not found" cuando el ID de sesión no existe en el repositorio.
3. Confirmar que se obtenga un `ViewResult` que contiene un objeto `StormSessionViewModel` válido cuando se proporciona un ID de sesión válido.

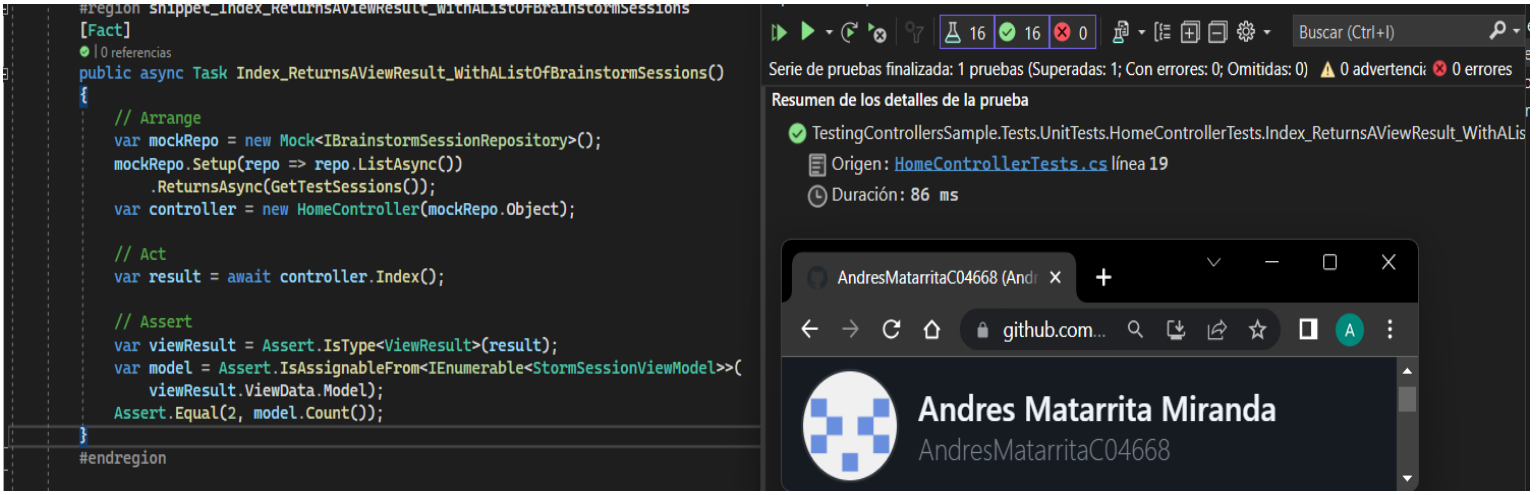
Al utilizar estos tres casos de prueba, se cubren diferentes escenarios y se verifican diferentes aspectos del método `Index`, lo que ayuda a garantizar que el comportamiento del método sea correcto y que se manejen adecuadamente diferentes situaciones, como IDs nulos, sesiones no encontradas y visualización de datos en la vista. Esto aumenta la confiabilidad y la calidad del código.

Screenshots

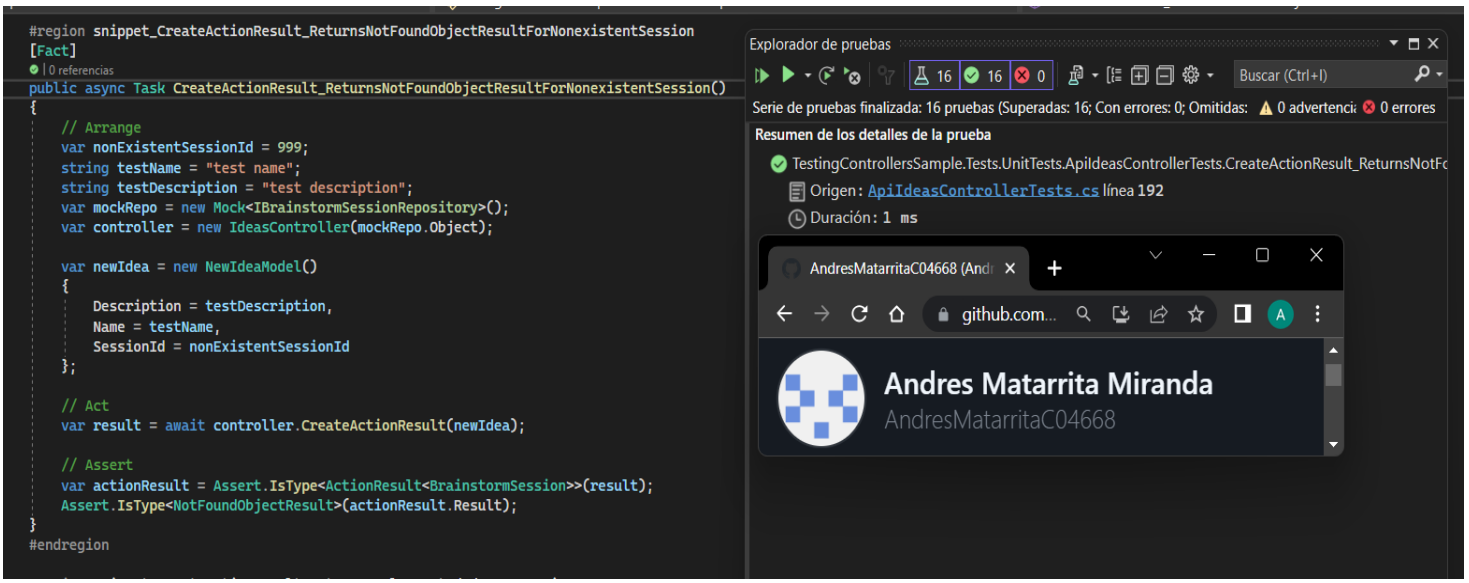
Pantallazo 1. Ejecución de la prueba de IndexReturnsViewResultWithStormSessionViewModel



Pantallazo 2. Ejecución de la prueba de Index_ReturnsAViewResult_WithAListOfBrainstormSessions



Pantallazo 3. Ejecución de la prueba de CreateActionResult_ReturnsNotFoundObjectResultForNonexistentSession



En su documento de reporte, y con base en lo aprendido en este laboratorio, liste y describa cuatro casos de prueba unitarios que va a realizar en el programa en que usted está trabajando en el Proyecto Integrador. La descripción de cada caso de prueba debe incluir:

- (2%) Método que va a probar y nombre de la clase a la que pertenece
- (2%) Nombre del caso de prueba
- (3%) Objetivo de la prueba
- (3%) Nombre de la clase y método que podría reemplazar con un mock.

Prueba unitaria 1:

- Método que va a probar: `ReservasPorIdentificador` (reservas por identificador)
- Nombre de la clase a la que pertenece: `AdministrarReservasController`
- Nombre del caso de prueba:
`ReservasPorIdentificador_WithValidIdentifier_ReturnsReservations`
- Objetivo de la prueba: Verificar que el método `ReservasPorIdentificador` del controlador `AdministrarReservasController` devuelve la lista correcta de reservaciones según el identificador proporcionado.
- Nombre de la clase y método que podría reemplazar con un mock: Se podría reemplazar la llamada al método `ObtenerReservas` de la clase `AdministrarReservasHandler` con un mock.

Prueba unitaria 2:

- Método que va a probar: `EliminarReserva` (eliminar reserva)
- Nombre de la clase a la que pertenece: `AdministrarReservasController`
- Nombre del caso de prueba:
`EliminarReserva_WithValidIdentifier_RedirectsToReservas`
- Objetivo de la prueba: Verificar que el método `EliminarReserva` del controlador `AdministrarReservasController` redirige correctamente a la acción "Reservas" después de eliminar una reserva según el identificador proporcionado.
- Nombre de la clase y método que podría reemplazar con un mock: Se podría reemplazar la llamada al método `EliminarReservacion` de la clase `AdministrarReservasHandler` con un mock.

Prueba unitaria 3:

- Método que va a probar: `Login` (método POST)
- Nombre de la clase a la que pertenece: `LoginController`
- Nombre del caso de prueba:
`Login_WithValidCredentials_RedirectsToAdminHome`
- Objetivo de la prueba: Verificar que el método `Login` del controlador `LoginController` redirige correctamente a la página de inicio del área de administración cuando se proporcionan credenciales válidas.
- Nombre de la clase y método que podría reemplazar con un mock: Se podría reemplazar la llamada al método `ObtenerCredencialesTrabajador` de la clase `LoginHandler` con un mock.

Prueba unitaria 4:

- Método que va a probar: `EditarTarifa` (método POST)
- Nombre de la clase a la que pertenece: `TarifasController`
- Nombre del caso de prueba:
`EditarTarifa_WithValidTarifaModel_RedirectsToTarifas`
- Objetivo de la prueba: Verificar que el método `EditarTarifa` del controlador `TarifasController` redirige correctamente a la página de tarifas después de actualizar una tarifa válida.
- Nombre de la clase y método que podría reemplazar con un mock: Se podría reemplazar la instancia de `TarifasHandler` con un mock y simular el comportamiento de los métodos `obtenerTarifasActuales` y `actualizarPrecioTarifas`.

Enlace del repositorio de github:

https://github.com/AndresMatarritaC04668/c04668_ci0126_23a.git

Agregue un pequeño resumen, no más de 200 palabras indicando claramente

1. Tres cosas que no sabía y aprendió en el laboratorio
2. Una cosa que se le hizo difícil de realizar y explique por qué fue difícil.
3. Una cosa que se le hizo fácil de realizar y explique por qué fue fácil.
4. Indique cuánto tiempo tardó en realizar el laboratorio.

En el laboratorio de pruebas unitarias de controladores en ASP.NET Core, aprendí tres cosas nuevas:

a. Las pruebas unitarias se centran en probar una parte específica de una aplicación de forma aislada, sin considerar su infraestructura ni dependencias. En el caso de las pruebas unitarias de la lógica del controlador, se prueba únicamente el contenido de una acción individual, sin tener en cuenta el comportamiento de las dependencias o del propio framework.

b. Se puede utilizar Moq, un framework de objetos simulados, para crear objetos simulados con comportamientos predefinidos y utilizarlos en las pruebas unitarias. Esto permite simular el comportamiento de las dependencias del controlador y verificar las interacciones entre el controlador y esas dependencias.

c. Es importante evitar devolver entidades de dominio directamente a través de llamadas API, ya que estas entidades pueden contener más información de la necesaria para el cliente y acoplar innecesariamente el modelo de dominio interno de la aplicación con la API pública. En su lugar, se recomienda realizar una asignación manual o automática entre las entidades de dominio y los tipos devueltos al cliente.

Una cosa que me resultó difícil de realizar fue probar el método de acción "Create" cuando el ModelState no es válido. Fue difícil porque tuve que simular un ModelState no válido y verificar que el controlador respondiera correctamente devolviendo un BadRequestObjectResult.

Por otro lado, una cosa que me resultó fácil de realizar fue probar el método de acción "Index" cuando el ModelState es válido. Esto fue fácil porque solo tuve que configurar el comportamiento esperado de las dependencias simuladas y luego verificar que el controlador devolviera un RedirectToActionResult con los argumentos correctos.

El tiempo que tardé en realizar el laboratorio de pruebas unitarias de controladores en ASP.NET Core fue de aproximadamente 5 horas y 30 minutos.