Unit Testing

Goals

In this lab we're going to explore how to unit test Razor Pages, Api Controllers and Minimal Apis, using `xUnit`, `Moq` and `WebApplicationFactory`.
For Razor pages, we're going to test: - That the Index sets the Photos property to the list of photos from the service. - That the Upload return a Page when the input is valid. - That the Upload return a Redirect when the input is invalid. For the Api Controller, we're going to test: - That the Get returns the list of comments from the service. - That the GetById returns a not found when the id is not found. For the Minimal API we're going to test: - That the Get returns a file when the photo exists. - That the Get returns a not found when the photo does not exist.

xUnit

- Add a new project of type `xUnit` to the solution. Name the project `PhotoSharingApplication.Web.UnitTests`.
- Add a reference to the `PhotoSharingApplication.Web` project.
- Add a reference to the `Moq` NuGet package.

Unit Testing Razor Pages

- Add a new folder to the `PhotoSharingApplication.Web.UnitTests` project called `Pages`.
- Add a new folder to the `Pages` folder called `Photos`.
- Add a new file to the `PhotoSharingApplication.Web.UnitTests/Pages/Photos` folder called `IndexTests.cs`.
- Add a method `OnGet_Sets_Photos` to the `IndexTests.cs` file.
    - Create a Mock of the `IPhotosService` interface.
    - Setup the mock so that its `GetAllPhotosAsync` returns a `List<Photo>` with three photos.
    - Create an instance of the `IndexModel` class, passing the mock as a parameter to the constructor
    - Call the `OnGetAsync` method.
    - Assert that the `Photos` property on the `IndexModel` is the same as the `List<Photo>` that was returned from the `GetAllPhotosAsync` method on the mock.

```
 using Moq;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Shared.Entities;
using PhotoSharingApplication.Web.Pages.Photos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Xunit;

namespace PhotoSharingApplication.Web.UnitTests.Pages.Photos;

public class IndexTests {
    [Fact]
    public async Task OnGet_Sets_Photos() {
        Mock<IPhotosService> photosServiceMock = new Mock<IPhotosService>();
        List<Photo> expected = new List<Photo>() {
            new Photo(){Id = 1, Title = "Title1", Description = "Description1" },
            new Photo(){Id = 2, Title = "Title2", Description = "Description2" },
            new Photo(){Id = 3, Title = "Title3", Description = "Description3" },
        };
        photosServiceMock.Setup(ps => ps.GetAllPhotosAsync()).ReturnsAsync(expected);

        IndexModel index = new IndexModel(photosServiceMock.Object);

        await index.OnGetAsync();

        var actual = Assert.IsAssignableFrom<List<Photo>>(index.Photos);

        Assert.True(expected.SequenceEqual(actual));
    }
}
```

- Add a new file to the `PhotoSharingApplication.Web.UnitTests/Pages/Photos` folder called `UploadTests.cs`.
- In the constructor of the `UploadTests.cs` file, setup all the services necessary to create an instance of the page and tp set the `ModelState`.
    - `Mock<IPhotosService> photosServiceMock`
    - `DefaultHttpContext? httpContext`

    - `ModelStateDictionary modelState`
    - `ActionContext actionContext`
    - `EmptyModelMetadataProvider modelMetadataProvider`
    - `ViewDataDictionary viewData`
    - `TempDataDictionary tempData`
    - `PageContext pageContext`
    - `UploadModel pageModel`

```
public class UploadTests {
    private Mock<IPhotosService> photosServiceMock;
    private DefaultHttpContext? httpContext;
    private ModelStateDictionary modelState;
    private ActionContext actionContext;
    private EmptyModelMetadataProvider modelMetadataProvider;
    private ViewDataDictionary viewData;
    private TempDataDictionary tempData;
    private PageContext pageContext;
    private UploadModel pageModel;
    public UploadTests() {
        photosServiceMock = new Mock<IPhotosService>();
        httpContext = new DefaultHttpContext();
        modelState = new ModelStateDictionary();
        actionContext = new ActionContext(httpContext, new RouteData(), new PageActionDescriptor(), modelState);
        modelMetadataProvider = new EmptyModelMetadataProvider();
        viewData = new ViewDataDictionary(modelMetadataProvider, modelState);
        tempData = new TempDataDictionary(httpContext, Mock.Of<ITempDataProvider>());
        pageContext = new PageContext(actionContext) {
            ViewData = viewData
        };
        pageModel = new UploadModel(photosServiceMock.Object) {
            PageContext = pageContext,
            TempData = tempData,
            Url = new UrlHelper(actionContext)
        };
    }
}
```

- Add a method `OnPostAsync_ReturnsARedirectToPageResult_WhenModelStateIsValid` to the `UploadTests.cs` file.
    - Set the `Photo` property of the `PageModel` to a new `Photo` instance.
    - Set teh `FormFile` property of the `PageModel` to a new `Mock` of an `IFormFile`.
    - Call the `OnPostAsync` method.
    - Assert that the return value of the `OnPostAsync` method is of type `RedirectToPageResult`.

```
[Fact]
public async Task OnPostAsync_ReturnsARedirectToPageResult_WhenModelStateIsValid() {
    // Arrange
    pageModel.Photo = new Shared.Entities.Photo();
    pageModel.FormFile = new Mock<IFormFile>().Object;

    // Act
    // A new ModelStateDictionary is valid by default.
    var result = await pageModel.OnPostAsync();

    // Assert
    Assert.IsType<RedirectToPageResult>(result);
}
```

- Add a method `OnPost_ReturnsPageResult_WhenModelStateIsInvalid` to the `UploadTests.cs` file.
    - Add a Model Error to the `ModelState` property of the `PageModel`
    - Call the `OnPostAsync` method.
    - Assert that the return value of the `OnPostAsync` method is of type `PageResult`.

```
[Fact]
public async Task OnPost_ReturnsPageResult_WhenModelStateIsInvalid() {
    // Arrange
    pageModel.ModelState.AddModelError("Photo.Title", "The Title field is required.");

    // Act
    var result = await pageModel.OnPostAsync();

    // Assert
    Assert.IsType<PageResult>(result);
}
```

Unit Testing Api Controllers

- Add a new folder to the `PhotoSharingApplication.Web.UnitTests` project called `Controllers`.
- Add a new file to the `PhotoSharingApplication.Web.UnitTests/Controllers` folder called `CommentsControllerTests.cs`.
- Add a constructor and initialize the `CommentsController` with a `Mock` of the `ICommentsService` interface.

```
public class CommentsControllerTests {
    private Mock<ICommentsService> commentsServiceMock;
    private CommentsController sut;
    public CommentsControllerTests() {
        commentsServiceMock = new Mock<ICommentsService>();
        sut = new CommentsController(commentsServiceMock.Object);
    }
}
```

- Add a method `GetCommentsForPhoto_ShouldReturnComments` to the `CommentsControllerTests.cs` file.
    - Setup the `GetCommentsForPhotoAsync` method of the `ICommentsService` interface to return a `List<Comment>` of 3 comments.
    - Call the `GetCommentsForPhoto` method.
    - Assert that the result of the `GetCommentsForPhoto` method is of type `ActionResult<IEnumerable<Comment>>`.
    - Assert that the result value is of type `List<Comment>`
    - Assert that the result value is the same as the `List<Comment>` that was returned from the `GetCommentsForPhotoAsync` method on the mock.

```
[Fact]
public async Task GetCommentsForPhoto_ShouldReturnComments() {
    //Arrange
    List<Comment> expected = new List<Comment>() {
        new Comment(){Id = 1, Title = "Title1", Body = "Body1" },
        new Comment(){Id = 2, Title = "Title2", Body = "Body2" },
        new Comment(){Id = 3, Title = "Title3", Body = "Body3" },
    };
    commentsServiceMock.Setup(ps => ps.GetCommentsForPhotoAsync(1)).ReturnsAsync(expected);

    //Act
    var result = await sut.GetCommentsForPhoto(1);

    // Assert
    ActionResult<IEnumerable<Comment>> actionResult =
        Assert.IsType<ActionResult<IEnumerable<Comment>>>(result);
    List<Comment> returnValue = Assert.IsType<List<Comment>>(actionResult.Value);
    Assert.True(returnValue.SequenceEqual(expected));
}
```

- Add a method `GetCommentById_ShouldReturnNotFound_WhenCommentIdDoesNotExist` to the `CommentsControllerTests.cs` file.
    - Setup the `GetCommentByIdAsync` method of the `ICommentsService` interface to return null.
    - Call the `GetCommentById` method.
    - Assert that the result of the `GetCommentById` method is of type `ActionResult<Comment>`.
    - Assert that the result value is of type `NotFoundResult`.

```
[Fact]
public async Task GetCommentById_ShouldReturnNotFound_WhenCommentIdDoesNotExist() {
    //Arrange
    commentsServiceMock.Setup(ps => ps.GetCommentByIdAsync(1)).ReturnsAsync((Comment)null);

    //Act
    var result = await sut.GetCommentById(1);

    // Assert
    ActionResult<Comment> actionResult = Assert.IsType<ActionResult<Comment>>(result);
    Assert.IsType<NotFoundResult>(actionResult.Result);
}
```

Unit Testing a Minimal Api

- Add a new folder to the `PhotoSharingApplication.Web.UnitTests` project called `MinimalApi`.
- Add a new file to the `PhotoSharingApplication.Web.UnitTests/MinimalApi` folder called `PhotoSharingApplicationApp.cs`.
- Let the class derive from `WebApplicationFactory<CommentsController>`
- In the constructor, accept and initialize a `Mock` of the `IPhotosService` interface.
- Override the `CreateHost`, accepting an `IHostBuilder` and returning an `IHost`.
- In the `CreateHost` method, invoke the `ConfigureServices` method of the `IHostBuilder` parameter. Add the `Mock` of the `IPhotosService` interface to the `IServiceCollection` parameter. Return the result of the `base.CreateHost` method.

```
using Microsoft.AspNetCore.Mvc.Testing;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Moq;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Web.Controllers;

namespace PhotoSharingApplication.Web.UnitTests.MinimalApi;

class PhotoSharingApplicationApp : WebApplicationFactory<CommentsController> {
    private readonly Mock<IPhotosService> mock;

    public PhotoSharingApplicationApp(Mock<IPhotosService> mock) {
        this.mock = mock;
    }
    protected override IHost CreateHost(IHostBuilder builder) {
        builder.ConfigureServices(services => {
            services.AddScoped<IPhotosService>(s => mock.Object);
        });

        return base.CreateHost(builder);
    }
}
```

- Add a new file to the `PhotoSharingApplication.Web.UnitTests/MinimalApi` folder called `MinimalApiTests.cs`.
- In the constructor, initialize a new instance of a `Mock<IPhotosService>`

```
using Moq;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Shared.Entities;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using Xunit;

namespace PhotoSharingApplication.Web.UnitTests.MinimalApi;

public class MinimalApiTests {
    private Mock<IPhotosService> photosServiceMock;
    public MinimalApiTests() {
        photosServiceMock = new();
    }
}
```

- Add a method `GetPhotoImage_ShouldReturnNotFound_WhenPhotoDoesNotExist`
- Setup the `GetPhotoByIdAsync` of the mock to return null.
- Create an instance of the `PhotoSharingApplicationApp` passing the mock as a parameter.
- Invoke the `CreateClient` method of the application factory and save the result in a `client` variable.
- Invoke the `GetAsync` method of the `client` variable passing the `/photos/image/{id}` route and save the result in a `response` variable.
- Assert that the `StatusCode` property of the `response` variable is equal to `HttpStatusCode.NotFound`.

```
[Fact]
public async Task GetPhotoImage_ShouldReturnNotFound_WhenPhotoDoesNotExist() {
    //Arrange
    int id = 1;
    photosServiceMock.Setup(s => s.GetPhotoByIdAsync(id)).ReturnsAsync(default(Photo));
    await using var application = new PhotoSharingApplicationApp(photosServiceMock);

    var client = application.CreateClient();

    //Act
    using HttpResponseMessage response = await client.GetAsync($"/photos/image/{id}");

    //Assert
    Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);
}
```

- Add a method `GetPhotoImage_ShouldReturnFile_WhenPhotoExists` to the `MinimalApiTests.cs` file.
- Setup the `GetPhotoByIdAsync` of the mock to return a new `Photo` instance having a `ContentYpe` property set to `jpg` and a `PhotoFile` property set to an array of byte.
- Create an instance of the `PhotoSharingApplicationApp` passing the mock as a parameter.
- Invoke the `CreateClient` method of the application factory and save the result in a `client` variable.
- Invoke the `GetAsync` method of the `client` variable passing the `/photos/image/{id}` route and save the result in a `response` variable.
- Retrieve the content type of the response by looking for the first value equal to `jpg` of the first header with a key equal to `Content-Type`.
- Assert that the value is not null
- Read the `Content` property of the `response` variable as an array of byte.
- Assert that the result is equal to the array of the PhotoFile property of the `Photo` instance.

```
 [Fact]
public async Task GetPhotoImage_ShouldReturnFile_WhenPhotoExists() {
    //Arrange
    string expectedContentType = "jpg";
    byte[] expectedContent = new byte[] { 1, 2, 3, 4 };
    int id = 1;
    Photo photo = new() { ContentType = expectedContentType, PhotoFile = expectedContent };
    photosServiceMock.Setup(s => s.GetPhotoByIdAsync(id)).ReturnsAsync(photo);
    await using var application = new PhotoSharingApplicationApp(photosServiceMock);

    var client = application.CreateClient();

    //Act
    using HttpResponseMessage res = await client.GetAsync($"/photos/image/{id}");

    //Assert
    string? actualContentType = res.Content.Headers.First(kv => kv.Key == "Content-Type").Value.FirstOrDefault(v => v == expectedCon
    Assert.NotNull(actualContentType);
    byte[] actualContent = await res.Content.ReadAsByteArrayAsync();
    Assert.Equal(expectedContent, actualContent);
}
```

Your tests should all pass.

Resources

- https://docs.microsoft.com/en-us/aspnet/core/test/razor-pages-tests?view=aspnetcore-6.0
- https://github.com/dotnet/AspNetCore.Docs/tree/main/aspnetcore/test/razor-pages-tests/samples
- https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-6.0#test-actionresultt
- https://www.hanselman.com/blog/minimal-apis-in-net-6-but-where-are-the-unit-tests
- https://github.com/DamianEdwards/MinimalApiPlayground/blob/main/tests/MinimalApiPlayground.Tests/Examples.cs