Lab: Testing and Debugging ASP.NET MVC Core Web Applications

# Scenario

The Photo Sharing application is in the early stages of development. However, frequent errors are hindering the productivity of the development team. The senior developer advises that you intercept exceptions and other flaws as early as possible. You have been asked to perform unit tests of the PhotoController to ensure that all scenarios work as expected and to avoid problems later in the web application development life cycle. You have also been asked to ensure that when critical errors occur, developers can obtain helpful technical information.

# Objectives

After completing this lab, you will be able to:

* Perform unit tests of the components of an MVC web application.
* Configure an exception handling strategy for an MVC web application.
* Use Visual Studio debugging tools against a web application.

**Estimated Time**: 90 minutes

# Exercise 1: Performing Unit Tests

## Scenario

In this exercise, you will:

* Create a test project and write the following tests.
  + IndexReturnsView: This test checks that the Index action returns a view named Index.
  + GetImageReturnsFile: This test checks that the GetImage action returns a file and not a view.

Note: The tests you add to the solution in this exercise will improve the quality of code and prevent bugs as development proceeds. However, this exercise does not conform to the principles of TDD because the PhotoController class already exists. In TDD, you would create these and other tests first, and then create a PhotoController class that passes the tests.

The main tasks for this exercise are as follows:

1. Create a test project.
2. Write the tests.
3. Add further tests.

### Task 1: Create a test project.

1. Under your solution folder ( (C):\LabFilesMvcCore\Mod06\Starter\PhotoSharingApplication) create a new folder named “test”
2. Under the test folder, create a PhotoSharingApplication.Tests folder
3. Open a command prompt
4. Navigate to the newly created PhotoSharingApplication.Tests folder
5. Type  
   dotnet new -t xunittest
6. In Visual Studio
   1. Add a new solution folder “test”
   2. Under the test solution folder, add the existing test project created in step 5
   3. Update all the NuGet packages
   4. Add a reference to the PhotoSharingApplication project
   5. Add a dependency to "Microsoft.DotNet.InternalAbstractions": "1.0.0" (NOTE: this is a workaround for a bug of version 1.1. It may not be necessary with the version of dotnet core you are using)
   6. Add a dependency to "Microsoft.EntityFrameworkCore.InMemory"

### Task 2: Write the tests.

* Rename the Test class as PhotoControllerTests.
* Create a static method using the following information:
  + Accessibility: private
  + Return type: static DbContext<PhotoSharingContext>
  + Name: CreateNewContextOptions
  + Input Parameters: none
* In the CreateNewContextOptions method:
  + Declare a variable serviceProvider and initialize it with:  
    new ServiceCollection().AddEntityFrameworkInMemoryDatabase().BuildServiceProvider()
  + Create a variable builder and initialize it with a new instance of a DbContextOptionsBuilder of type PhotoSharingContext
  + On the builder variable, invoke the UseInMemoryDatabase().UseInternalServiceProvider() method
  + Return the Options property of the builder object
* Rename the Test method as IndexReturnsView.
* Add using statements for the following namespaces:
* using Microsoft.AspNetCore.Mvc;
* using Microsoft.EntityFrameworkCore;
* using Microsoft.Extensions.DependencyInjection;
* using PhotoSharingApplication.Controllers;
* using PhotoSharingApplication.Data;
* using PhotoSharingApplication.Models;
* using System;
* using System.Collections.Generic;
* using Xunit;
* In the IndexReturnsView test
  + Create a variable options and initialize it by invoking the CreateNewContextOptions method
  + Create a new context variable and initialize it with an instance of a new PhotoSharingContext passing the options in the constructor
  + Create a new PhotoController passing the context in the constructor
  + Call the Index action
  + Assert that the name of the result view is Index.
* Add a new test method by using the following information:
* Annotation: [Theory], [InlineData(1)], [InlineData(2)]
* Scope: public
* Return type: async void
* Name: GetImageReturnsFile
* Parameters: an integer named id
* In the GetImageReturnsFile test
  + Create a variable options and initialize it by invoking the CreateNewContextOptions method
  + Create a new context variable and initialize it with an instance of a new PhotoSharingContext passing the options in the constructor
  + Create a new PhotoController passing the context in the constructor
  + Create two objects of type Photo and initialize them with some data
  + Add the 2 objects to the Photos dbset of the context object
  + Save the changes on the context
  + Call the GetImage action passing the id parameter
  + Assert that the result type is of type FileContentResult.
* Run all the tests in the PhotoSharingTests project and examine the results.

**Results**: After completing this exercise, you will be able to add a set of PhotoController tests defined in the PhotoSharingTests project of the Photo Sharing application.

# Exercise 2: Optional—Configuring Exception Handling

## Scenario

Now that you have developed unit tests for the Photo Sharing application, you need to configure an exception handling strategy for the MVC web application. This would ensure that when exceptions occur, the exception message is displayed in a custom MVC error view. You also want to display a custom message for when a page is not found. You also need to implement a placeholder action for the SlideShow action in the PhotoController view. This action will be completed during a later iteration of the project.

Complete this exercise if time permits.

The main tasks for this exercise are as follows:

1. Edit Startup.cs for exception handling.
2. Create a custom error view.
3. Raise errors.

### Task 1: Edit Startup.cs to redirect the status code pages.

1. Open the Startup.cs file in the root level folder of the PhotoSharingApplication project.
2. Add the UseStatusCodePagesWithRedirects by using the following path:

* "~/errors/{0}.html"

1. Add a new HTML page to the PhotoSharingApplication project by using the following information:

* Folder: ~/wwwroot/errors
* Template: HTML Page
* Name: 404.html

1. In the 404.html file, set the contents of the TITLE element to Error.
2. Add content to the 404.html file to explain to users that the entered address does not exists.

### Task 2: Modify the Error action of the Home Controller to pass the Exception to the View.

1. In the Error Action of the Home Controller:

* Declare a variable feature and initialize it with the result of HttpContext.Features.Get<IExceptionHandlerFeature>();
* Return the View method passing the value of the Error property of the feature object

### Task 3: Create a custom error view.

1. Add a new view to the Views/Home folder by using the following information:

* Name of the view: Error

1. In the Error.cshtml file, set the content of the TITLE element to Custom Error.
2. Set the @model for the Error.cshtml to Exception.
3. Render an H1 element by using the Exception.Message property of the Model object.
4. Save all the changes made to the Error.cshtml file.

### Task 4: Configure errors in the PhotoController class.

1. Add a new action to the PhotoController class by using the following information:

* Scope: public
* Return type: ActionResult
* Name: SlideShow
* Parameters: None

1. In the new action, throw an exception by using the following information:

* Type: NotImplmentedException
* Message: The SlideShow action is not yet ready

### Task 4: Raise errors.

1. In Visual Studio, in the Solution Explorer, right click the PhotoSharingApplication project and select Properties
2. In the debug tab, select the Environment Variable ASPNETCORE\_ENVIRONMENT and change the value from Development to Production
3. Start debugging and display Sample Photo 5.
4. In the Internet Explorer window, request the relative URL and view the error details.

* URL: /Photo/Display/malformedID

1. In the Internet Explorer window, request the relative URL.

* URL: /Photo/SlideShow

1. Stop debugging and close Visual Studio.

**Results**: After completing this exercise, you will be able to: Configure a custom error handling strategy for an MVC application.

**Question**: When you ran the tests for the first time in Exercise 1, why did IndexReturnsView pass, while GetImageReturnsFile failed?

**Question**: In Exercise 1, why did all the tests pass during the second run?