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 a page is not found, users get a friendly page in return.

# Objectives

After completing this lab, you will be able to:

* Refactor the controller by using a Repository Pattern
* Perform unit tests of the components of an MVC web application.
* Configure a strategy for HTTP status code errors for an MVC web application.
* Use Visual Studio debugging tools against a web application.

**Estimated Time**: 90 minutes

# Exercise 1: Refactor the Photos Controller and the PhotoGalleryViewComponent by using a Repository Pattern

## Scenario

The PhotosController and the PhotoGalleryViewComponent make direct use of the PhotoSharingApplicationContext. To better separate the Application concerns from the Infrastructure concerns, you decide to use a Repository Pattern. This will help to better understand what to test at a controller level.

In this exercise, you will:

* Create a repository for the Photos model
* Move the infrastructure code from the controller / view component to the repository class
* Use the repository from the controller and the view component

The main tasks for this exercise are as follows:

1. Create an IPhotosRepository Interface
2. Create a PhotosRepository class that implements the IPhotosRepository Interface
3. Modify the controller to use the repository class
4. Modify the view component to use the repository class
5. Register the repository class as a service

# Exercise 2: 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.
  + DetailsReturnsNotFound\_WhenIdIsNull: This test checks that the Details action returns a NotFoundResult if the id parameter is null
  + DetailsReturnsNotFound\_WhenPhotoNotFound: This test checks that the Details action returns a NotFoundResult when the Photo is not found
  + DetailsReturnsView\_WithAPhoto: This test checks that the Details action returns a View with a Photo in the model
  + GetImageReturnsFile: This test checks that the GetImage action returns a file and not a view.
  + CreateReturnsCreateView\_WhenModelStateIsInvalid: This test checks that the Create action returns the Create View when the model state is invalid
  + CreateReturnsRedirect\_WhenModelStateIsValid: This test checks that the Create action returns a Redirect to the Index action when the model state is valid

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. Execute the tests.

**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 HTTP Status Code Error Handling

## Scenario

You want to display a custom message for when a page is not found. You also want to test what happens when an exception is not handled. You decide to implement a SlideShow action in the Photos Controller. This action will be completed during a later iteration of the project, so for now you will throw a NotImplementedException.

Note: this feature is a nice to have. Complete this exercise only if time permits.

The main tasks for this exercise are as follows:

1. Edit the Startup configuration to use Http Status Code Error Redirects.
2. Create a custom error page for the Not Found HTTP Status code.
3. Change your development environment to Production
4. Raise errors and test what happens.

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