Lab: Developing ASP.NET MVC Core Controllers

# Scenario

You have been asked to add a controller in the photo sharing application that corresponds to the Photo model class that you have created in an earlier module. The controller should include actions that respond when users upload photos, list all photos, display a single photo, and delete photos from the application. You should also add an action that returns the photo as a .jpg file to show on a webpage.

The members of your development team are new to ASP.NET MVC and they find the use of controller actions confusing. Therefore, you need to help them by adding a component that displays action parameters in the Visual Studio Output window whenever an action runs. You will add an action filter to achieve this.

# Objectives

* After completing this lab, you will be able to:
* Add an MVC controller.
* Write action filters that run code for multiple actions.

**Estimated Time**: 60 minutes

# Exercise 1: Add an MVC Controller

## Scenario

In this exercise, you will add an MVC controller that handles photo operations. You will implement the following actions:

* Index. This action gets a list of all the Photo objects and passes the list to the Index view for display.
* Display. This action takes an ID to find a single Photo object. It passes the Photo to the Display view for display.
* Create (GET). This action creates a new Photo object and passes it to the Create view, which displays a form that the visitor can use to upload a photo and describe it.
* Create (POST). This action receives a Photo object from the Create view and saves the details to the database.
* Delete (GET). This action displays a Photo object and requests confirmation from the user to delete the Photo object.
* DeleteConfirmed (POST). This action deletes a Photo object after confirmation.
* GetImage: This action returns the photo image from the database as a JPEG file. This method is called by multiple views to display the image.

The main tasks for this exercise are as follows:

1. Add a Photos controller.
2. Implement the Index action.
3. Implement the Details action.
4. Implement the Create actions for GET and POST HTTP verbs.
5. Implement the Delete actions for GET and POST HTTP verbs.
6. Implement the GetImage action.

### Task 1: Add a Photos controller.

1. In the Solution Explorer, right click on the Controllers folder, select Add -> Controller.
2. Select MVC Controller Empty
3. Name the controller PhotosController
4. Create a private field of type PhotoSharingApplicationContext
5. Create a constructor with a PhotoSharingApplicationContext parameter
6. Save the parameter in the private field you created on step 3

### Task 2: Implement the Index action.

1. Create an Index action
2. Use the PhotoSharingApplication context to asynchronously retrieve the list of Photos
3. Pass it as model to the View

### Task 3: Implement the Details action.

1. Create a Details action that receives an id parameter of type int
2. Use the PhotoSharingApplicationContext to asynchronously look for the Photo by Id.
3. Return a NotFound if it does not exists
4. Otherwise, return the Details View and pass the found photo as model

### Task 4: Implement the Create action for GET verb

1. Create a Create action
2. Instance a new Photo and set its CreatedDate property to today’s date.
3. Pass the new Photo to a view called Create.

### Task 5: Implement the Create action for POST HTTP verb

1. Add a Create action and constrain it with an HttpPost attribute
2. Make sure that the action accepts two parameters, one of type Photo and one of type IFormFile
3. Set the photo.CreatedDate property to today’s date.
4. If the ModelState is not valid, pass the photo object back to the Create view.
5. Else, if the IFormFile image parameter is not null
   1. set the photo.ImageMimeType property to the value of image.ContentType,
   2. set the photo.PhotoFile property to be a new byte array of length image. Length
   3. save the file into the photo.PhotoFile property by using the image.OpenReadStream().ReadAsync() method.
6. Add the photo object to the PhotoSharingApplicationContext
7. Save the changes
8. Redirect the browser to the Index action.

### Task 6: Create the Delete action for GET HTTP verb

1. Create a Delete action that receives an id parameter of type int
2. Use the PhotoSharingApplicationContext to asynchronously look for the Photo by Id.
3. Return a NotFound if it does not exists
4. Otherwise, return the Details View and pass the found photo as model

### Task 7: Create the Delete action for POST HTTP verb

1. Create a Delete action that receives an id parameter of type int
   1. Name the method DeleteConfirmed
   2. Name the action Delete
   3. Constrain it with an HttpPost attribute
2. Use the PhotoSharingApplicationContext to asynchronously look for the Photo by Id.
3. Remove the photo object from the PhotoSharingApplicationContext
4. Save the changes
5. Redirect the browser to the Index action

### Task 8: Create the GetImage action.

1. Add a method for the GetImage action by using the following information:

* **Scope**: public
* **Return type**: async Task<IActionResult>
* **Name**: GetImage
* **Parameter**: an integer called id

1. Find the correct photo object from the context by using the id parameter.
2. If the photo object is not null, return a File result constructed from the photo.PhotoFile and photo.ImageMimeType properties, else return the null value.

**Results**: After completing this exercise, you will be able to create an MVC controller that implements common actions for the Photo model class in the Photo Sharing application.

# Exercise 2: Optional—Writing the Action Filters in a Controller

## Scenario

Your development team is new to MVC and is having difficulty in passing the right parameters to controllers and actions. You need to implement a component that displays the controller names, action names, parameter names, and values in the Visual Studio Output window to help with this problem. In this exercise, you will create an action filter for this purpose.

Complete this exercise if time permits.

The main tasks for this exercise are as follows:

1. Add an action filter class.
2. Initialize a logger
3. Add a logValues method to the action filter class.
4. Add a handler for the OnActionExecuting event.
5. Register the Action Filter as a service.
6. Register the Action Filter with the Photo Controller.

### Task 1: Add an action filter class.

1. Create a new folder and a new class for the action filter by using the following information:

* **Name**: ValueReporter
* **New Folder Name**: ActionFilters

1. Add using statements to the controller for the following namespaces:
   1. Microsoft.AspNetCore.Mvc.Filters
   2. Microsoft.AspNetCore.Routing
   3. Microsoft.Extensions.Logging
2. Ensure that the ValueReporter class inherits from the ActionFilterAttribute class.

### Task 2: Initialize a logger.

1. Add a private field \_logger of type ILogger
2. Create a constructor with an ILoggerFactory parameter named loggerFactory
3. Initialize the \_logger field with the CreateLogger method of the loggerFactory parameter, passing a “ValueReporter ActionFilter” parameter

### Task 3: Add a logValues method to the action filter class.

1. Add a method to the ValueReporter class by using the following information:

* **Scope**: private
* **Return type**: void
* **Name**: logValues
* **Parameter**: a RouteData object called routeData.

1. Within the logValues method
   1. Initialize a controller variable with the value of routeData.Values[“controller”]
   2. Initialize an action variable with the value of routeData.Values[“action”]
   3. Call the \_logger.LogInformation method and send the name of the controller and action.
   4. Create a foreach loop that loops through the items in routeData.Values.
   5. In the foreach loop, call the \_logger.LogInformation method and send the key name and value.

### Task 4: Add a handler for the OnActionExecuting event.

1. In the ValueReporter action filter, override the OnActionExecuting event handler.
2. In the OnActionExecuting event handler, call the logValues method, and pass the filterContext.RouteData object.
3. Save the file.

### Task 5: Register the Action Filter as a service.

1. In the Startup.cs file, in the ConfigureServices method, between the registration of the DbContext and Mvc, add a Scoped service of type ValueReporter.

### Task 4: Register the Action Filter with the Photo Controller.

1. Open the PhotoController class and a ServiceFilter of type ValueReporter action filter to the PhotoController class.
2. Add the PhotoSharingApplication.ActionFilters namespace
3. Save the file.

**Results**: After completing this exercise, you will be able to create an action filter class that logs the details of actions, controllers, and parameters to the Visual Studio Output window, whenever an action is called.

**Question**: Why did you use the ActionName annotation for the DeleteConfirmed action in the PhotoController class?

**Question**: In the lab, you added two actions with the name, Create. Why is it possible to add these actions without using the ActionName annotation?