Lab: Developing ASP.NET MVC Core Controllers

# Scenario

You have been asked to add a controller to 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.

Note: The controllers and views that you have added in Lab 3 were to test your new model classes. They have been removed from the project to create the actual controllers. You will create temporary views to test these controllers at the end of this lab.

# Objectives

* After completing this lab, you will be able to:
* Add an MVC controller to a web application.
* Write actions in an MVC controller that respond to user operations such as create, index, display, and delete.
* Write action filters that run code for multiple actions.

**Estimated Time**: 60 minutes

# Exercise 1: Adding an MVC Controller and Writing the Actions

## Scenario

In this exercise, you will create the MVC controller that handles photo operations. You will also add 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. Create a photo controller.
2. Create the Index action.
3. Create the Details action.
4. Write the Create actions for GET and POST HTTP verbs.
5. Create the Delete actions for GET and POST HTTP verbs.
6. Create the GetImage action.

### Task 1: Create a photo controller.

1. Create a new controller for handling photo objects by using the following information:

* Controller name: PhotosController
* Template: MVC Controller Empty

1. Add using statements to the controller for the following namespaces:

* PhotoSharingApplication.Data
* PhotoSharingApplication.Models
* Microsoft.EntityFrameworkCore

1. In the PhotosController class, create a new private object by using the following information:

* **Scope**: private
* **Class**: PhotoSharingContext
* **Name**: \_context

1. Create a constructor that accepts a parameter of type PhotoSharingContext
2. Initialize the \_context field in the PhotoSharingContext constructor with the value of the parameter.

### Task 2: Create the Index action.

1. Edit the code in the Index action by using the following information:

* **Return Type**: async Task<IActionResult>
* **View name**: Index
* **Model**: context.Photos.ToListAsync()

### Task 3: Create the Details action.

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

* **Scope**: public
* **Return Type**: async Task<IActionResult>
* **Name**: Details
* **Parameters**: One nullable integer called id

1. Within the Details action code block, return NotFound if the id parameter is null
2. Find a single photo object from its ID.
3. If no photo with the right ID is found, return the NotFound value.
4. If a photo with the right ID is found, pass it to a view called Details.

### Task 4: Write the Create actions for GET and POST HTTP verbs.

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

* **Scope**: public
* **Return type**: IActionResult
* **Name**: Create

1. Add code to the Create action that creates a new Photo and sets its CreatedDate property to today’s date.
2. Pass the new Photo to a view called Create.
3. Add another method for the Create action by using the following information:

* **HTTP verb**: HTTP Post
* **Scope**: public
* **Return type**: async Task<IActionResult>
* **Name**: Create
* **First parameter**: a Photo object called photo.
* **Second parameter**: an IFormFile object called image.

1. Add code to the Create action that sets the photo.CreatedDate property to today’s date.
2. If the ModelState is not valid, pass the photo object to the Create view.
3. Else, if the 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 that the user posted to the photo.PhotoFile property by using the image.OpenReadStream().ReadAsync() method.
4. Add the photo object to the context, save the changes, and then redirect to the Index action.

### Task 5: Create the Delete actions for GET and POST HTTP verbs.

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

* **Scope**: public
* **Return type**: async Task<IActionResult>
* **Name**: Delete
* **Parameter**: a nullable integer called id

1. In the Delete action, return a NotFound if the id parameter is null
2. Add code to find a single photo object from its id.
3. If no Photo with the right id is found, return the NotFound value
4. If a Photo with the right id is found, pass it to a view called Delete.
5. Add a method called DeleteConfirmed by using the following information:

* **HTTP verb**: HTTP Post
* **ActionName**: Delete
* **Scope**: public
* **Return type**: async Task<IActionResult>
* **Name**: DeleteConfirmed
* **Parameter**: an integer called id

1. Find the correct photo object from the context by using the id parameter.
2. Remove the photo object from the context, save your changes and redirect to the Index action.

### Task 6: 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.
3. Save the file.

**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.

# Exercise 3: Using the Photo Controller

## Scenario

In this exercise, you will:

* Create a temporary index and display views by using the scaffold code that is built into the Visual Studio MVC application template.
* Use the views to test controllers, actions, and action filters, and run the Photo Sharing application.

The main tasks for this exercise are as follows:

1. Create the Index and Details views.
2. Use the GetImage action in the Display view.
3. Run the application and display a photo.

### Task 1: Create the Index and Details views.

1. Compile the PhotoSharingApplication project to build the solution.
2. Add a new view to the Index action method of the PhotoController class by using the following information:

* **Folder**: Views/Photo
* **View Name**: Index
* **Template**: List
* **Model class**: Photo
* **Data Context class**: PhotoSharingContext
* **Create as partial view**: false
* **Reference Script libraries**: true
* **Use a layout page**: true

1. Add a new view to the Display action method of the PhotoController class by using the following information:

* **Folder**: Views/Photo
* **View Name**: Details
* **Template**: Details
* **Model class**: Photo
* **Data Context class**: PhotoSharingContext
* **Create as partial view**: false
* **Reference Script libraries**: true
* **Use a layout page**: true

### Task 2: Use the GetImage action in the Display view.

1. In the Details.cshtml code window, embed the lines to display the PhotoFile property into a code that runs if the Model.PhotoFile property is not null.
2. Within the if code block, replace the code to display the value of the PhotoFile property with an <img> tag. Use the following information:

* **Tag**: <img>
* **Class**: col-md-12 img-thumbnail

1. In the src attribute of the <img> tag, add a call to the Url.Action helper by using the following information:

* **Controller**: Photos
* **Action**: GetImage
* **Parameters**: Model.Id

1. Save the file.
2. Build the solution.

### Task 3: Run the application and display a photo.

1. Start debugging the application and access the following relative path:

* Path: /photo/index

1. If you completed Exercise 2, in the Output pane of the PhotoSharingApplication - Microsoft Visual Studio window, locate the last entry in the Action Values category to verify whether there are any calls to the Details and the GetImage actions.
2. Display an image.

* Path: /photo/details/1

1. If you completed Exercise 2, in the Output pane of the PhotoSharingApplication - Microsoft Visual Studio window, locate the last entry in the Action Values category to verify whether there are any calls to the Details and the GetImage actions.
2. Stop debugging and close Microsoft Visual Studio.

**Results**: After completing this exercise, you will be able to create an MVC application with views that you can use to test controllers, actions, and action filters.

**Question**: What will happen if you click the Edit or Delete links in the Index view in the Lab?

**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?