Lab: Developing ASP.NET MVC Core Views And View Components

# Scenario

You have been asked to add the following views to the photo sharing application:

* Views for the Photo model objects. These views will enable users to upload a new photo to the gallery, view the details of existing photos and deleting photos.
* A Photo Gallery view component. This view component will display many photos in thumbnail sizes, with the title, owner, and created date properties. This view component will be used on the All Photos webpage to display all the photos in the application. In addition, this view component will also be used on the home page to display the three most recent photos.

After adding these views to the photo sharing application, you will also test the working of the web application.

# Objectives

After completing this lab, you will be able to:

* Add Razor views to an MVC application.
* Write both HTML markup and C# code in a view by using Razor syntax.
* Create a view component and use it to display re-usable markup.

# Exercise 1: Add the Views for Photos

## Scenario

In this exercise, you will

* Add the Create View to upload new photos.
* Add a Display View to show the properties of a photo, such as title, description, and created date.
* Add a Delete View to show the properties of a photo and ask for confirmation.

The main tasks for this exercise are as follows:

1. Add the Create View
2. Add the Display View
3. Add the Delete View

### Task 1: Add the Create View

1. Add a new Photos folder to the Views folder
2. In the Views / Photos folder, add a new View
   1. Name the view Create
   2. Use an empty template
   3. Use no layout page

* Modify the title in Create New Photo

1. Set the model type to Photo
2. Add a form element
   * Set an asp-action attribute to “Create”

* Set an enctype attribute to multipart/form-data

1. In the form, group a label, an input field and an eventual validation into a div for each of the following properties:
   1. Title
   2. Description
   3. CreatedDate
   4. UserName
   5. PhotoFile (use an input of type file with a name of Image, not bound to any property, no validation necessary)
2. Save the Create.cshtml file.

### Task 2: Add the Details View

1. In the Views / Photos folder, add a new View
   1. Name the view Details
   2. Use an empty template
   3. Use no layout page

* Modify the title in Photo Details

1. Set the model type to Photo
2. Create a description list
3. Set a definition and a description for each of the following properties:
   1. Title
   2. Description
   3. CreatedDate
   4. UserName
   5. PhotoFile (first check if the PhotoFile is not null, then use an image element whose source is the url of the GetImage action of the Photos controller)
4. Save the Details.cshtml file.

### Task 3: Add the Delete View

1. In the Views / Photos folder, add a new View
   1. Name the view Delete
   2. Use an empty template
   3. Use no layout page

* Modify the title in Delete Photo

1. Set the model type to Photo
2. Create a description list
3. Set a definition and a description for each of the following properties:
4. Title
5. Description
6. CreatedDate
7. UserName
8. PhotoFile (first check if the PhotoFile is not null, then use an image element whose source is the url of the GetImage action of the Photos controller)
9. **TIP: you may want to implement a Partial View and reuse the code you wrote in the Details View**
10. Insert a form element for the Delete action with a submit button
11. Add a link to the Index action
12. Save the Delete.cshtml file

**Results**: After completing this exercise, you will be able to create a web application with a Razor view to create new photos.

# Exercise 2: Creating and Using a View Component

## Scenario

In this exercise, you will:

* Add a Photo Gallery View Component.
* Add a Default View for the Photo Gallery View Component.
* Complete the Default View.
* Use the Photo Gallery View Component.

The main tasks for this exercise are as follows:

1. Add a Photo Gallery View Component.
2. Add a Default View for the Photo Gallery View Component.
3. Complete the Default View.
4. Use the Photo Gallery View Component.

### Task 1: Add Photo Gallery View Component.

1. Create a new folder ViewComponents.
2. In the new folder, add a new class named PhotoGalleryViewComponent.
3. Derive the class from Microsoft.AspNetCore.Mvc.ViewComponent.
4. Declare a private \_context field of type PhotoSharingApplication.Data.PhotoSharingApplicationContext.
5. Create a constructor with a PhotoSharingApplication.Data.PhotoSharingApplicationContext parameter.
6. In the constructor, initialize the \_context field with the value of the constructor parameter.
7. Add a new action by using the following information:

* **Scope**: public
* **Return** **Type**: async Task<IViewComponentResult>
* **Name**: InvokeAsync
* **Parameter**: an Integer called number with a default value of 0

1. Create a new List of Photo objects named photos. Add an if statement, to set photos to include all the Photos in the context object, if the number parameter is zero.
2. If the number parameter is not zero, set photos to list the most recent Photo objects. The number of Photo objects in the list should be the number attribute.
3. Pass the photos object to view and return the view.
4. Save the file.

### Task 2: Add a default view for the photo gallery view component.

1. Under the Views/Shared folder, create a new folder named Components
2. Under the Views/Shared/Components folder, create a new folder named PhotoGallery
3. Create a new view named Default by using the following information:

* **View** **Name**: Default
* **Template**: Empty
* **Create** **as** **partial** **view**: checked

### Task 3: Complete the photo gallery partial view.

1. In the Default.cshtml partial view file
   1. Set the model type to an enumerable list of Photo
   2. Add a For Each statement that loops through all the items in the Model.
   3. In the For Each statement, add an H3 element that renders the item.Title property.
   4. After the H3 element, add an if statement that checks that the item.PhotoFile value is not null.
   5. If the item.PhotoFile value is not null, render an <img> tag with width 200. Call the Url.Action helper to set the src attribute by using the following information:

* **Action**: GetImage
* **Controller**: Photos
* **Parameters**: for the id parameter, pass item.Id
  1. After the if statement, add a P element, and call the @Html.DisplayFor helper to render the words Created By: followed by the value of the item.UserName property.
  2. After the UserName display controls, add a P element, and call the @Html.DisplayFor helper to render the words Created On: followed by the value of the item.CreatedDate property.
  3. After the CreatedDate display controls, add a link to the Details action of the Photos controller with an id set to item.Id and a text set to Details:
* **Link** **text**: Details
* **Controller name: Photos**
* **Action name**: Details
* **Parameters**: pass the item.Id value as the id parameter

1. Save the Default.cshtml file.

### Task 4: Use the photo gallery view component.

1. Register your assembly as a TagHelper source
   1. Open the \_ViewImports.cshtml under the Views folder
   2. At the end of the file, add   
      @addTagHelper \*, PhotoSharingApplication
2. Modify the Index action in the PhotosController.cs so that no model class is passed to the Index view.
3. Add a View for the Index action (empty template, no layout, no partial view)
   1. Change the title to All Photos.
   2. Add an H2 element to the page body to display the heading as All Photos
   3. Add a P element to add a link to the Create action in the Photo controller by using the following information:

* **Tag**: a
* **Link text**: Add a Photo
* **Action name**: Create
* **Controller name**: Photos

1. Insert the PhotoGallery view component by using the following information:

* **Prefix**: vc
* **Tag Helper**: photo-gallery
* **Parameters**: for the number parameter, pass the value 0

1. Save the Index.cshtml file.

**Results**: After completing this exercise, you will be able to create a web application with a partial view to display multiple photos.

# Exercise 4: Modify the Home View and Test the Views

## Scenario

In this exercise, you will modify the home page to reuse the photo gallery view component, displaying only the three most recent photos.

The main tasks for this exercise are as follows:

1. Modify the View for the home page.
2. Use the web application.

### Task 1: Modify the View for the home page.

1. Open the Index view under the Views/Home folder
2. Change the title of the page to Welcome to Adventure Works Photo Sharing.
3. Add the following text to the home page:

* Welcome to Adventure Works Photo Sharing! Use this site to share your adventures.

1. Add an H2 element to display the heading as Latest Photos.
2. Insert the PhotoGallery view component by using the following information:

* **Prefix**: vc
* **Tag Helper**: photo-gallery
* **Parameters**: for the number parameter, pass the value 3

1. Save the Index.cshtml file.

### Task 2: Use the web application.

1. Start the Photo Sharing web application with debugging.
2. Verify the number of photos displayed on the home page.
3. Display a photo of your choice to verify whether the display shows the required information.
4. Verify the number of photos displayed on the All Photos page.
5. Add a new photo of your choice to the application by using the following information:

* **Title**: My First Photo
* **Description**: This is the first test of the Create photo view.
* **File path**: Allfiles (C):\LabfilesMVC5\Mod05\SamplePhotos

1. Close your browser and Microsoft Visual Studio.

**Results**: After completing this exercise, you will be able to create a web application in which users can upload and view the photos.

**Question**: How can you improve the accessibility of the HTML that your photo views render?

**Question**: In the lab, how did you ensure that the Create view for Photo model objects could upload photo files when the user clicked the Create button?