Developing ASP.NET MVC Controllers

Exercise 1: Adding an MVC Controller and Writing the Actions

Task 1: Create a photo controller

1- Copy Folder MNS/Kursus/MVC-Controllers/Starter to your local machine
2- Open PhotoSharingApplication.sln from your “copied” Starter Folder
3- Create a new controller for handling photo objects by using the following information:
Controller name: PhotoController
Template: Empty MVC controller
4- 5. Add using statements to the controller for the following namespaces:

System.Collections.Generic
System.Globalization
PhotoSharingApplication.Models

5- In the PhotoController class, create a new private object by using the following information:

Scope: private
Class: PhotoSharingContext
Name: context
Instantiate the new object by calling the PhotoSharingContext constructor.

Task 2: Create the Index action.

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

Scope: **public**
Return Type: **ActionResult**
View name: Index
Return class: **View**
Model: context.Photos.ToList()

Task 3: Create the Display action.

1- Add a method for the **Display** action by using the following information:

Scope: **public**
Return Type: **ActionResult**
Name: **Display**
Parameters: One integer called **id**

2- Within the **Display** action code block, add code to find a single **photo** object from its **ID**.
3- If no photo with the right ID is found, return the **HttpNotFound** value.
4- If a photo with the right ID is found, pass it to a view called **Display**.

→ Photo photo = context.Photos.Find(id)
→ if (photo==null)
→ return HttpNotFound()
→ return View("Display", photo)

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: **ActionResult**
   Name: **Create**
2. Add code to the **Create** action that creates a new Photo and sets its **CreatedDate** property to today's date. → DateTime.Today
3. Pass the new **Photo** to a view called **Create**.
4. Add another method for the **Create** action by using the following information:
   HTTP verb:→ **[HTTP Post]**
   Scope: **public**
   Return type: **ActionResult**
   Name: **Create**
   First parameter: a **Photo** object called **photo**.
   Second parameter: an **HttpPostedFileBase** object called **image**.

5. Add code to the **Create** action that sets the **photo.CreatedDate** property to today's date.
6. If the **ModelState** is not valid, pass the **photo** object to the **Create** view.
   Else, if the image parameter is not null, set the **photo.ImageMimeType** property to the value of **image.ContentType**, set the **photo.PhotoFile** property to be a new byte array of length, **image.ContextLength**, and then save the file that the user posted to the **photo.PhotoFile** property by using the **image.InputStream.Read()** method.

7. Add the **photo** object to the context, save the changes, and then redirect to the **Index** action.

```
if (!ModelState.IsValid)→ return View("Create", photo)

else{ if (image != null){

photo.ImageMimeType = image.ContentType

photo.PhotoFile = new byte[image.ContentLength]

image.InputStream.Read(photo.PhotoFile, 0, image.ContentLength) }}

    context.Photos.Add(photo)

    context.SaveChanges()

    return RedirectToAction("Index")
```

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: ActionResult
   Name: Delete
   Parameter: an integer called id
2. In the Delete action, add code to find a single photo object from its id.
3. If no Photo with the right id is found, return the HttpNotFound 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 → [ActionName("Delete")]
Scope: public
Return type: ActionResult
Name: DeleteConfirmed
Parameter: an integer called id
6. Find the correct photo object from the context by using the id parameter.
7. Remove the photo object from the context, save your changes and redirect to the Index action.

→
```
Photo photo = context.Photos.Find(id)
context.Photos.Remove(photo)
```

Task 6: Create the GetImage action.

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

Scope: public
Return type: FileContentResult
Name: GetImage
Parameter: an integer called id

2- Find the correct photo object from the context by using the id parameter.

3- 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.

4. Save the file.

→
```
if (photo !=null) return File(photo.PhotoFile, photo.ImageMimeType)

else return null
```

Exercise 2: Writing the Action Filters in a Controller

Task 1: Add an action filter class.

1. Create a new class for the action filter by using the following information:
Name: **ValueReporter**
Folder: **Controllers**

2. Add **using** statements to the controller for the following namespaces:
System.Diagnostics
System.Web.Mvc
System.Web.Routing

3. Ensure that the **ValueReporter** class inherits from the **ActionFilterAttribute** class.

Task 2: 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**.

2. Within the **logValues** method, call the **Debug.WriteLine** method to send the name of the controller and action to the Visual Studio Output window. For the category, use the string, “Action Values”.

3. Within the **logValues** method, create a **foreach** loop that loops through the **var** items in **routeData.Values**.
4. In the **foreach** loop, call the **Debug.WriteLine** method to send the key name and value to Visual Studio Output window.

→
```
var controller = routeData.Values["controller"];
var action = routeData.Values["action"];
string message = string.Format($"Controller: {controller}; Action: {action}")
Debug.WriteLine(message, "Action Values")
foreach (var item in routeData.Values)
Debug.WriteLine(">>>> Key: {0}; Value: {1}", item.Key, item.Value)
```

Task 3: 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.
   Delete → base.OnActionExecuting(filterContext);

➔
```
public override void
OnActionExecuting(ActionExecutingContext filterContext)
logValues(filterContext.RouteData)
```

3. Save the file.

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

1. Open the PhotoController class and add the ValueReporter action filter to the PhotoController class. → [**ValueReporter**]

2. Save the file.

Exercise 3: Using the Photo Controller

Task 1: Create the Index and Display views.

1. 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**
   Name: **Index**
   Template: **List**
   Model class: **Photo**
   Data Context class: **PhotoSharingContext**
   Reference script libraries : Checked
   Layout: Checked

3. In the Index.cshtml Replace → @Html.ActionLink("Details", "Details",new { id=item.PhotoID })
   With → @Html.ActionLink("Display", "Display", new { id=item.PhotoID })

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

   Folder: **Views/Photo**
   Name: **Display**
   Scaffold template: **Details**
   Model class: **Photo**
   Data Context class: **PhotoSharingContext**

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

1. In the Display.cshtml code window, after the code that displays the model.Title property, add a code that runs if the Model.PhotoFile property is not null.

2. Within the if code block, render an <img> tag. Use the following information:
   Tag: <img>
   Width: 800px
   Source: Blank
3. In the src attribute of the <img> tag, add a call to the Url.Action helper by using the following information:

   Controller: Photo
   Action: GetImage
   Parameters: Model.PhotoID

→
```
@if (Model.PhotoFile !=null){
  <img width="800"
  src="@Url.Action("GetImage", "Photo", new {id=Model.PhotoID})" />}
```

4. Save the file.
5. Build the solution.

Task 3: Run the application and display a photo.

1. Run the application and access the following relative path:
   Path: /photo/index

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 Display and the GetImage actions.

3. Display an image.

4. 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 Display and the GetImage actions.