Lab: Building Responsive Pages in ASP.NET MVC Core Web Applications

# Scenario

Your manager has asked you to include comments for photos in the Photo Sharing application. Your manager has also highlighted that the performance of some pages in the application is too slow for a production site.

You have already implemented the functionalities to display the comments for a photo, add a new comment and delete a comment, but you want to ensure that comments for photos take minimal loading time.

You decide to use partial page updates. You also want to return pages in quick time, while updated information is displayed, for which you decide to configure caching in your application.

# Objectives

After completing this lab, you will be able to:

* Use AJAX jQuery plugins to call asynchronous controller actions and insert the results into Razor views.
* Configure ASP.NET caches to serve pages in quick time.

Estimated Time: 60 minutes

# Exercise 1: Using Partial Page Updates

## Scenario

You have been asked to include a comment functionality on the photo display view of the Photo Sharing application. You want to ensure high performance by using AJAX partial page updates.

In this exercise, you will

* Add code to the views for partial page update.

The main tasks for this exercise are as follows:

1. Add the bower package for the Ajax jQuery plugin.
2. Complete the Photos/Details View
3. Complete the Default view of the CommentsForPhoto View Component

### Task 1: Add the Bower Package for the Ajax jQuery Plugin.

1. In the Solution Explorer, under the PhotoSharingApplication project, go to the Dependencies/Bower folder, right click and select “Open bower.json”
2. In the “dependencies” node, add "Microsoft.jQuery.Unobtrusive.Ajax": "3.2.3"
3. Save the file

### Task 2: Complete the Photos/Details View.

1. Open the Photos/Details view
2. At the end of the file, add a @section scripts section
3. If the environment is Development, add a script whose source is "~/lib/Microsoft.jQuery.Unobtrusive.Ajax/jquery.unobtrusive-ajax.js"
4. If the environment is Staging or Production, add a script whose source is "http://ajax.aspnetcdn.com/ajax/mvc/3.0/jquery.unobtrusive-ajax.min.js" and fallback to "~/lib/Microsoft.jQuery.Unobtrusive.Ajax/jquery.unobtrusive-ajax.min.js"

### Task 3: Complete the Default view of the CommentsForPhoto View Component.

1. Open the Default.cshtml view file under the Views/Shared/CommentsForPhoto folder
2. Locate the form at the bottom of the page that submits to the Create action of the Comments controller
3. Add HTML attributes to the form by using the following information:

* data-ajax="true"
* data-ajax-mode="replace"
* data-ajax-update="#comments-tool"

1. Start the web application in debugging mode, browse to Sample Photo 1, and observe the comments displayed.
2. Add a new comment to Sample Photo 1.

* Subject: Test Comment
* Body content: This comment is to test AJAX-based partial page updates.

1. Stop debugging.

**Results**: At the end of this exercise, you will have ensured that new comments can be added and displayed on the pages of the application without a complete page reload. You will create a Photo Sharing application with a comments tool, implemented by using partial page updates.

# Exercise 2: Configuring the ASP.NET Caches

## Scenario

You have been asked to configure the ASP.NET caches in the Photo Sharing application to ensure optimal performance. Senior developers are particularly concerned that the All Photos gallery might render slowly because it will fetch and display many photos from the database at a time.

In this exercise, you will:

* Configure the output cache.
* Use the developer tools in Internet Explorer to examine the speed at which image files and pages render with and without caching.
* Configure the output cache to store the results of the GetImage action so that image files can be returned from the cache.

Complete this exercise if time permits.

The main tasks for this exercise are as follows:

1. Test the All Photos page with no caching.
2. Configure caching.
3. Configure caching for the GetImage action.
4. Retest the All Photo page with GetImage caching.

### Task 1: Test the All Photos page with no caching.

1. Start the application in debugging mode and configure the browser to always refresh the page from the server by using the Internet Explorer developer tools.
2. Capture traffic between the browser and the server when the All Photos page is loaded, by using the Network tools.
3. Record the time taken by the server to render the /Photo page and return the page to the browser. This value is the Request duration, which you can find on the Timings tab.
4. Clear the first network capture, and capture a second request to the All Photos page.
5. Record the second instance of time taken by the server to render the /Photo page and return the page to the browser. Observe if the duration is more or less than the first instance.
6. Stop debugging.

### Task 2: Configure caching.

1. Install the NuGetPackage Microsoft.Extensions.Caching.Memory
2. Open the Startup class and configure the services to add the Memory Cache

### Task 3: Configure caching for the GetImage action.

1. In the root of the PhotoSharingApplication project, create a new Cache folder
2. In the newly created Cache folder, create a new class PhotoFile with the following properties: In the PhotoController, configure the GetImage action to use the output cache, by using the following information:

* public byte[] File {get; set;}
* public string ImageMimeType { get; set; }

1. In the PhotosController, locate the GetImage method
2. Start by creating a string key for the photo to retrieve from (or set to) the cache
3. Try to get the value of the photo associated with the key from the cache
   * If the retrieved photo is null
   * Select the PhotoFile and ImageMimeType from the database into a new Cache.PhotoFile object
   * Save the object into the cache associating it with the key and making sure it expires after 10 minutes
4. Return a File content with the File and MimeType of the PhotoFile object
5. Save all your changes.

### Task 4: Retest the All Photo page with GetImage caching.

1. Start the application in debugging mode and configure the browser to always refresh the page from the server, by using the Internet Explorer developer tools.
2. Capture the traffic between the browser and the server when the All Photos page is loaded, by using the Network tools.
3. Record the time taken by the server to render the /Photo/GetImage/1 request.
4. Clear the first network capture, and capture a second request to the All Photos page.
5. Record the second instance of the time taken by the server to render the /Photo/GetImage/1 request and return the page to the browser.
6. Close the developer tools, stop debugging, and close Visual Studio.

**Results**: At the end of this exercise, you will create a Photo Sharing application with the Output Cache configured for caching photos.

**Question**: In Exercise 2, why was the Request timing for /Photo not reduced for the first request when you configured the output cache for the index action?