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

# Scenario

Your manager has highlighted that the performance of some pages in the application is too slow for a production site.

You want to return images in quick time, for which you decide to configure caching in your application.

The senior developer has also asked you to implement the following functionality in your Photo Sharing web application.

* Any visitor of the application, including anonymous users, should be able to mark a photograph as a favorite.
* If a user has marked a favorite, a link should be available to display the favorite photos.
* Favorite photos should be displayed in the slideshow view.

# Objectives

After completing this lab, you will be able to:

* Configure ASP.NET caches to quickly serve images.
* Implement a Favorites functionality using Sessions

Estimated Time: 60 minutes

# Exercise 1: Configuring the ASP.NET Cache

## Scenario

You have been asked to configure the ASP.NET cache 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 cache by using a Distributed Cache.
* Use the developer tools in the browser to examine the speed at which image files and pages render with and without caching.
* Store the images in the cache so that the GetImage action can execute faster.

The main tasks for this exercise are as follows:

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

### 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 browser 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 /Photos 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 /Photos 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. Open the Startup class and configure the services to add the DistributedMemoryCache

### Task3: Configure DI on the PhotoController

1. Add a second parameter to the PhotoController constructor of type IDistributedCache and save the parameter value into a private \_cache variable

### Task4: Create a CachedPhoto DTO

You’re going to serialize your images in the cache in a binary format using a BinaryFormatter. Seen the fact that you don’t need to save the whole Photo model but just the information necessary to create a File (the byte array and the mime type), you decide to create a CachedPhoto class that you will use to store the information on the cache.

1. In the root of the PhotoSharingApplication project, create a new Cache folder
2. In the newly created Cache folder, create a new class DistributedMemoryCacheExtensions
3. Make the class static
4. Create a private CachedPhoto class
5. Mark the CachedPhoto class as Serializable
6. Add two properties
   * byte[] PhotoFile
   * string ImageMimeType
7. Add a ToPhoto method that returns a new Photo instance initialized with the PhotoFile and ImageMimeType properties
8. Add a static FromPhoto method that accepts a Photo and creates and returns a new instance of a CachedPhoto object initialized with the PhotoFile and ImageMimeType values

### Task5: Create an extension method for the IDistributedCache interface to get a Photo

You decide to create two extension methods for the IDistributedCache interface to hide the complexity of the serialization and deserialization. This way the code of your controller will be much simpler and the responsibility of dealing with the cache will be of the cache itself.

1. In the DistributedMemoryCacheExtensions, add an async SetPhotoAsync extension method for the IDistributedCache interface, accepting a Photo and a CancellationToken
   * Create a string for a key, using the photo id
   * Create a CachedFoto from the photo
   * Create a BinaryFormatter
   * Create a MemoryStream
   * Use the BinaryFormatter to serialize the CachedPhoto into the MemoryStream
   * Set the memoryStream.ToArray() in the cache, binding the value with the key

### Task5: Create an extension method for the IDistributedCache interface to set a Photo

1. In the DistributedMemoryCacheExtensions, add an async GetPhotoAsync extension method for the IDistributedCache interface, accepting an id and a CancellationToken and returning a Photo
   * Create a string for a key, using the id (make sure to use the same format you used in the SetPhotoAsync)
   * Get an array of byte from the cache, using the key
   * If the array is not null
     + Create a BinaryFormatter
     + Create a MemoryStream initialized with the array of byte retrieved from cache
     + Use the BinaryFormatter to deserialize a CachedPhoto from the MemoryStream
     + Create a Photo from the CachedPhoto
   * Return the Photo

### Task 6: Use the extension methods from the PhotoController

1. In the PhotosController, locate the GetImage method
2. Try to get the value of the photo from the cache using your extension method
   * If the retrieved photo is null
   * Retrieve the photo from the repository
   * Set the photo in the cache by using your extension method
3. Return a File content with the File and MimeType of the Photo object
4. Save all your changes.

### Task 5: Retest the All Photos 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 browser 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 /Photos/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 /Photos/GetImage/1 request and return the page to the browser.

### Task 6: Update the Tests to pass the MemoryDistributedCache to the Controller constructor.

1. Open the PhotosControllerTests class in the PhotoSharingApplication.Tests project.
2. In both the tests, create an instance of a MemoryDistributedCache class passing an instance of a IOptions<MemoryDistributedCacheOptions> by writing the following code:
3. var cache = new MemoryDistributedCache(Options.Create(new MemoryDistributedCacheOptions()));
4. Pass the cache object as second parameter to the constructor of the PhotosController.
5. Verify that the tests still pass
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.

# Exercise 2 (if time permits): Using Session for Favorite Photos

## Scenario

The senior developer has asked you to implement the following functionality in your Photo Sharing web application.

* Any visitor of the application, including anonymous users, should be able to mark a photograph as a favorite.
* If a photo has already been marked as favorite, it should show it
* If a user has marked at least one favorite, a link in the navigation bar should be available to display the favorite photos.
* Favorite photos should be displayed in a slideshow view.

In this exercise, you will:

* Setup Session
* Create the Favorites Slideshow action and View.
* Create the Add Favorite action.
* Create a ViewComponent that checks if a given photo is already in Session and renders either an Add To Favorite Button or an indication that the photo is already a favorite
* Modify the Details View and the PhotoGallery to include the new ViewComponent
* Create a ViewComponent that checks if there is any favorite Photo in the Session and renders either nothing or a navigation item to the Favorite SlideShow
* Modify the Layout to include the new ViewComponent

The main tasks for this exercise are as follows:

1. Setup Session and extend Session
2. Create the Favorites Slideshow action and View.
3. Create the AddToFavorites action.
4. Create an AddToFavorites ViewComponent
5. Modify the Details View and the PhotoGallery to include the new ViewComponent
6. Create a FavoritesMenuItemViewComponent
7. Modify the Layout to include the new ViewComponent

### Task 1: Setup and extend Session.

You decide to store in the session the id of each photo that the user wants to add as favorite. You will store each id with a string key that will begin with a prefix such as “photoId”. You will use the prefix to retrieve the value of the id when necessary. This way you can be sure that you can retrieve only the information you need without accidentally read unrelated information you may have to add in the future. You decide to create four extension methods that will be used throughout the application:

1. An AddFavoritePhoto, to store the photo id together with a string key. You will use this method in an action of the Photos controller, invoked whenever a user wants to mark a photo as favorite.
2. A GetFavoritePhotos, to retrieve the list of ids whose keys start with the prefix used during the add phase. You will use this method in a PhotosController method for the slideshow.
3. An IsThereAnyFavorite, that will return true if the session has any key starting with the prefix used during the add phase. You will use this to understand if you have to render the navigation link to the slideshow.
4. An IsFavoritePhoto, returning true if a given photo id is present in the session. You will use this to understand if you have to render an “Add To Favorite” button for each photo of the photo gallery and details views.
5. In the Startup class, add Session as a service and use the Session middleware before the MVC middleware.
6. Create a Session folder
7. Create a public static class ISessionExtensions
8. Add an AddFavoritePhoto extension method for the ISession interface, accepting an int id
   1. Save the id in the session with a string key, using a “photoId” string prefix
9. Add a GetFavoritePhotos extension method for the ISession interface
   1. Create a List of int
   2. For each session key starting with the “photoId” prefix, extract the value and add it to the list
   3. Return the list
10. Add an IsThereAnyFavoritePhoto extension method for the ISession interface
    1. Return true if there is any session key starting with the “photoId” prefix
11. Add an IsFavoritePhoto extension method for the ISession interface, accepting an int id
    1. Get the item whose key is “photoId” followed by the input parameter
    2. Return true if the retrieved item has a value.

### Task 2: Create the Favorites Slideshow action and View.

1. Add a new method in your PhotosRepository to retrieve a List of Photo given a list of photo ids.
   1. Open the IPhotosRepository interface
   2. Add a GetPhotosByIdsAsync, accepting a list of int and returning a list of Photo
   3. Open the PhotosRepository class
   4. Implement the GetPhotosByIdsAsync method, using a Linq query that returns the Photos where the list of ids received as input contains the photo id range variable
   5. Open the PhotosController
   6. Create a FavoritesSlideShow action
   7. Retrieve the list of favorite ids from the session, by invoking the GetFavoritePhotos extension method you wrote previously
   8. Retrieve the list of photos from the repository, by invoking the GetPhotosByIdsAsync method you wrote previously
   9. Pass the list of photos to a SlideShow View as a model
   10. Create a SlideShow View
   11. Use the bootstrap Carousel component to display the List of Photos received as model
   12. Use indicators and controls. Display title and description as caption.

### Task 3: Create the AddToFavorites action.

1. Open the PhotosController
2. Add a new AddToFavorites action accepting an int id parameter
3. Invoke the AddFavoritePhoto ISession extension method you wrote previously.
4. Return the Index Action

### Task 4: Create an AddToFavorites ViewComponent.

1. In the ViewComponents folder, add a new AddToFavorites class
2. Let the class derive from ViewComponent
3. Implement an InvokeAsync method, accepting an int id parameter and returning a Task of IViewComponentResult
4. Invoke the IsFavoritePhoto ISession extension method you wrote previously
5. Depending on the result, either return a Default View passing the id as model, or return a “AlreadyFavorite” View.
6. Under the Views/Shared/Components, add a AddToFavorites folder
7. Inside the AddToFavorites folder, add a Default View
8. In the Default View, add a link to the AddToFavorites action of the Photos Controller, passing the Model as id parameter in the route data.
9. Inside the AddToFavorites folder, add a AlreadyFavorite View
10. In the AlreadyFavorite View, add paragraph with a “Photo Already Favorite” message in it

### Task 4: Modify the Details View and the PhotoGallery to include the new ViewComponent.

1. In the Views / Photos folder, open the Details View
2. Render the AddToFavorites ViewComponent, next to the Delete button. Don’t forget to pass the id parameter

### Task 5: Create a FavoritesMenuItemViewComponent.

1. In the ViewComponents folder, add a new FavoritesMenuItemViewComponent class
2. Let the class derive from ViewComponent
3. Implement an InvokeAsync method, returning a Task of IViewComponentResult
4. Invoke the IsThereAnyFavorite ISession extension method you wrote previously
5. Depending on the result, either return a Default View or return a “ThereAreFavorites” View.
6. Under the Views/Shared/Components, add a FavoritesMenuItemViewComponent folder
7. Inside the FavoritesMenuItemViewComponent folder, add a Default View
8. Leave the Default View empty
9. Inside the FavoritesMenuItemViewComponent folder, add a ThereAreFavorites View
10. In the ThereAreFavorites View, add navigation list item with a link to the FavoritesSlideShow action of the Photos Controller

### Task 6: Modify the Layout to include the new ViewComponent.

1. In the Views / Shared folder, open the \_Layout View
2. Locate the navigation bar
3. After the last list item, render the FavoritesMenuItemViewComponent

**Result: After completing this exercise, you will be able to use the Session object to store and retrieve the user favorite photos.**

**Question**: In Exercise 1, why was the Request timing for /Photos not reduced for the first request when you configured the output cache for the index action?