Lab: Controlling Access to ASP.NET MVC Core Web Applications

# Scenario

A large part of the functionality for your proposed Photo Sharing application is in place. However, stakeholders are concerned about security because there are no restrictions on the tasks that users can complete. The following restrictions are required:

* Only site members should be able to add photos.
* A photo should only be deleted by its owner.
* Only site members should be able to add comments.
* Views should only show links to actions that the user is authorized to perform.

You have been asked to resolve these concerns by creating a policies for the Photo Sharing application. Visitors should be able to register as users of the web application and create user accounts for themselves. After registration, when the users log on to the application, they will have access to actions such as adding and deleting photos. Anonymous users will not have access to perform these actions. Additionally, registered users should also be able to reset their own password.

# Objectives

After completing this lab, you will be able to:

* Provide access to resources in a web application.

Estimated Time: 45 minutes

### Exercise 1: Add and Test Identity

### Task 1: Scaffold Identity

1. From Solution Explorer, right-click on the project > Add > New Scaffolded Item.
2. From the left pane of the Add Scaffold dialog, select Identity > ADD.
3. In the ADD Identity dialog, select the options you want.
4. Select your existing layout page ~/Views/Shared/\_Layout.cshtml
5. Select the + button to create a new Data context class. Name the class AuthContext.
6. Select the + button to create a new User class. Name the class PhotoSharingApplicationUser
7. Select ADD.
8. In the Startup file, locate the Configure method and add the following code **between UseStaticFiles and UseMvc**

app.UseAuthentication();

1. Create a migration and update the database. In the Visual Studio Package Manager Console:

Add-Migration -c PhotoSharingApplication.Models.AuthContext IdentitySchema

Update-Database -c PhotoSharingApplication.Models.AuthContext

1. In your Views / Shared / \_Layout.cshtm, add the following code in your navigation bar:

<partial name="\_LoginPartial" />

1. Save and verify that the application compiles correctly
2. **NOTE: If you have updated your project to use bootstrap 4, you will have to copy bootstrap 4 back, because the scaffolding command has most probably overwritten the bootstrap folder with version 3.3.7**

### Task 1: Test registration, log on, and log off.

1. Start the web application in debugging mode and register a user account by using the following information:

* Email: david@photosharing.com
* Password: Pa$$w0rd

1. Log off and then log on with the credentials you just created.
2. Stop debugging.

**Results**: At the end of this exercise, you will create a Photo Sharing application in which users can register for an account, log on, and log off.

# Exercise 2: Authorizing Access to Resources

## Scenario

You should ensure that:

* Only site members can add photos.
* A photo can only be deleted by its owner.
* Only site members can add comments.
* Views only show links to actions that the user is authorized to perform.

The main tasks for this exercise are as follows:

1. Restrict Access to the Create actions
2. Create the Authorization Handlers.
3. Add the Authorization Policies.
4. Restrict access to Photo actions.
5. Restrict access to the Comment actions.
6. Check authorization in the views.
7. Test authorization.

### Task 1: Restrict Access to the Create Actions and save the UserName in the PhotosController.

1. Open the PhotosController and locate the Create actions
2. Add the [Authorize] annotation to ensure that only authenticated users can access the Create action for the GET requests.
3. Add the [Authorize] annotation to ensure that only authenticated users can access the Create action for the HTTP POST verb.
4. In the Create in POST, set the photo.UserName property to the User.Identity.Name value
5. Open the Create View and remove the input field for the UserName

### Task 2: Restrict Access to the Create Actions and save the UserName in the CommentsController.

1. Open the CommentsController and locate the Create action
2. Add the [Authorize] annotation to ensure that only authenticated users can access the Create action for the HTTP POST verb.
3. In the Create action, set the comment.UserName property to the User.Identity.Name value

### Task 3: Test the Create actions.

1. Run the application and verify that you’re not logged in (if not, log out)
2. Click on the Create a Photo link in the navigation bar
3. Verify that you’re redirected to the Login page
4. Logon and create a photo
5. Verify that the photo is create successfully and that the user name is correct

### Task 4: Create the Authorization Handlers.

1. Create a new folder AuthorizationHandlers
2. In the AuthorizationHandlers folder, create a new class PhotoDeleteAuthorizationHandler and let it derive from AuthorizationHandler<OperationAuthorizationRequirement, Photo>
3. Override the Task HandleRequirementAsync(AuthorizationHandlerContext context, OperationAuthorizationRequirement requirement, Photo photo)
4. Check if the context.User is authenticated
5. If it is
   1. If the requirement name is “Delete” and the user is the owner of the photo, invoke the context.Succeed(requirement)
   2. If the requirement name is “Delete” but the user is not the owner of the photo, invoke the context.Succeed(requirement)
6. If the user is anonymous invoke the context.Fail()
7. Return Task.CompletedTask.

### Task 5: Add the Authorization Policies.

1. In the Startup.cs file, locate the ConfigureServices method
2. Invoke the AddAuthorization method of the services collection.
3. Add a “PhotoDelete” policy with one requirement: a new OperationAuthorizationRequirement with a name of “Delete”
4. Register a Singleton service of type IAuthorizationHandler with an implementation of type PhotoDeleteAuthorizationHandler

### Task 6: Restrict access to Photo actions.

1. In the PhotosController.cs file, use Dependency Injection to get hold of the IAuthorizationService by receiving it in the constructor and saving it into a private field
2. Add the [Authorize] annotation to ensure that only authenticated users can access the Delete action.
3. Add the [Authorize] annotation to ensure that only authenticated users can access the DeleteConfirmed action for the HTTP POST verb.
4. In the Delete action, before returning the View, check if the user is authorized to delete the photo by invoking the AuthorizeAsync method of the authorizationService and passing the User, the photo and the “PhotoDelete” policy name
5. If the user is not authorized, return a Forbid result
6. In the DeleteConfirmed action, before removing the Photo, check if the user is authorized to delete the photo by invoking the AuthorizeAsync method of the authorizationService and passing the User, the photo and the “PhotoDelete” policy name
7. If the user is not authorized, return a Forbid result
8. Save all the changes.
9. Run the application
   1. Try to delete a photo when not yet logged in. Verify that you’re redirected to the logon page
   2. Log on and try to delete a photo of another user. Verify that you’re redirected to AccessDenied
   3. Delete a photo

### Task 5: Check authentication status in a view.

1. Open the \_Layout.cshtml
2. Inject the SignInManager of type ApplicationUser
3. In the navigation bar, locate the link to the Create Action of the Photos Controller and embed that code in an if that checks whether the user is logged in by invoking the IsSignedIn method of the SignInManager
4. Save the file
5. Open the Details view of the Photos Controller
6. Inject the IAuthorizationService
7. Locate the link to the Delete action of the Photos controller and embed that code in an if that checks whether the user is authorized to delete the photo by using the AuthorizeAsync method of the authorizationService
8. Save the file
9. Open the Default view of the CommentsForPhoto View Component.
10. Inject the SignInManager of type ApplicationUser
11. Inject the IAuthorizationService
12. Locate the link to the Delete Action of the Comments Controller and embed that code in an if that checks whether the user is authorized to delete the comment by using the AuthorizeAsync method of the authorizationService
13. Locate the addComment div and embed that code in an if that checks whether the user is logged in by invoking the IsSignedIn method of the SignInManager
14. Save all the changes.

### Task 6: Test authorization.

1. Start the web application and verify that if you’re not logged on you cannot see any Create nor Delete button for photos.
2. Log on to the web application
3. Create a Photo
4. Add a comment of your choice to a photo
5. Delete a photo created by you
6. Stop debugging.

**Results**: At the end of this exercise, you will have authorized anonymous and authenticated users to access resources in your web application.

**Question**: When you tried to add a photo before logging on to the application, why did ASP.NET display the Login view?