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.
* A comment should only be deleted its owner.
* 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 and comments. 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: Test Security

### 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.
* A comment can only be deleted its owner.
* Views only show links to actions that the user is authorized to perform.

The main tasks for this exercise are as follows:

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

### Task 1: 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. Create a constructor with a parameter of type UserManager<ApplicationUser> and save the value of the parameter in a private field \_userManager
4. Override the Task HandleRequirementAsync(AuthorizationHandlerContext context, OperationAuthorizationRequirement requirement, Photo photo)
5. Check if the context.User is authenticated
6. If it is, find the ApplicationUser through the userManager
7. If the requirement name is “Delete” and the user is the owner of the photo, invoke the context.Succeed(requirement)
8. If the requirement name is “Delete” but the user is not the owner of the photo, invoke the context.Succeed(requirement)
9. If the user is anonymous invoke the context.Fail()
10. Return Task.CompletedTask.
11. Repeat steps 2 / 10 for the CommentAuthorizationHandler

### Task 2: 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. Add a “CommentDelete” policy with one requirement: a new OperationAuthorizationRequirement with a name of “Delete”
5. Register a Singleton service of type IAuthorizationHandler with an implementation of type PhotoDeleteAuthorizationHandler
6. Register a Singleton service of type IAuthorizationHandler with an implementation of type CommentDeleteAuthorizationHandler

### Task 3: Restrict access to Photo actions.

1. In the PhotosController.cs file, use Dependency Injection to get hold of the UserManager<ApplicationUser> and the IAuthorizationService by receiving them in the constructor and saving them into two private fields
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. Add the [Authorize] annotation to ensure that only authenticated users can access the Delete action.
5. Add the [Authorize] annotation to ensure that only authenticated users can access the DeleteConfirmed action for the HTTP POST verb.
6. 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
7. If the user is not authorized, return a Challenge result
8. 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
9. If the user is not authorized, return a Challenge result
10. Save all the changes.

### Task 4: Restrict access to the Comment actions.

1. In the CommentsController.cs file, use Dependency Injection to get hold of the UserManager<ApplicationUser> and the IAuthorizationService by receiving them in the constructor and saving them into two private fields
2. Add the [Authorize] annotation to ensure that only authenticated users can access the Create action.
3. Add the [Authorize] annotation to ensure that only authenticated users can access the Delete action.
4. Add the [Authorize] annotation to ensure that only authenticated users can access the DeleteConfirmed action for the HTTP POST verb.
5. 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 comment and the “CommentDelete” policy name
6. If the user is not authorized, return a Challenge result
7. In the DeleteConfirmed action, before removing the Comment, check if the user is authorized to delete the photo by invoking the AuthorizeAsync method of the authorizationService and passing the User, the comment and the “CommentDelete” policy name
8. If the user is not authorized, return a Challenge result
9. Save all the changes.

### 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 in debugging mode and then attempt to add a new photo to the web application, without logging on to the application.
2. Without logging on to the application, view any photo in the application and attempt to add a comment.
3. Log on to the web application by using the following credentials:

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

1. Add a comment of your choice to the photo by using the following information:

* Subject: Authenticated Test Comment

1. 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**: In Exercise 3, when you tried to add a photo before logging on to the application, why did ASP.NET display the Login view?