



Module 12

Managing Security



Module Overview

- Authentication in ASP.NET Core
- Authorization in ASP.NET Core
- Defending from Attacks





Lesson 1: Authentication in ASP.NET Core

- The Need for Authentication
- Setting up ASP.NET Core Identity
- Interfacing with ASP.NET Core Identity
- ASP.NET Core Identity Configuration
- Demonstration: How to use ASP.NET Core Identity
- Customizing Providers with ASP.NET Core





The Need for Authentication

Why Authentication is Needed

- To identify the users that are connected to your application or website
- To ensure connected users use the correct credentials
- To enable you to block sensitive resources from unauthenticated users
- To help prevent malicious activities on your application or website





Authentication Methods

Authentication methods

- Single Factor
- Two Factor
- Multi Factor
- Mobile Authentication
- API Authentication
- Biometrics
- OAuth





Setting up ASP.NET Core Identity

ASP.NET Core Identity requires some configurations to be specified before it can be used:

- It requires an **Identity** class inheriting from **IdentityUser**
- It requires **IdentityDbContext** which is used for all database communications
- It requires a call to **AddDefaultIdentity** in **ConfigureServices**
- It requires a call to **UseAuthentication** in **Configure**





Startup.cs Configurations for Adding Authentication

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddIdentity<WebsiteUser, IdentityRole>()  
    .AddEntityFrameworkStores<AuthenticationContext>();  
}
```

```
public void Configure(IApplicationBuilder app,  
    AuthenticationContext authContext) {  
    app.UseAuthentication();  
}
```





Interfacing with ASP.NET Core Identity

ASP.NET Core MVC utilizes two main services for authentication

- **SignInManager** – Service designed to perform authentication operations. This includes logging in and out of the application.
- **UserManager** – Service designed to perform operations on users themselves. Can be used to add or remove, as well as find users.

In addition, the **Controller User** property exposes Identity which allows us to get information about the current user.





ASP.NET Core Identity Configuration

There are many things you can configure in ASP.NET Core Identity configuration:

- User settings – Allow to configure what constitutes a legal username and other options
- Lockout settings – Allow to customize lockout behavior if incorrect passwords are supplied
- Password settings – Allow setting password complexity rules
- Sign in settings – Allow requiring additional methods of authentication before creating users
- Cookie settings – Allow changing the behavior of cookies on the website



Using Multiple Configurations

```
public void ConfigureServices(IServiceCollection services) {  
    services.AddMvc();  
    services.AddDbContext<AuthenticationContext>(options => {  
        options.UseSqlite("Data Source=user.db")  
    });  
    services.AddDefaultIdentity<WebsiteUser>(options => {  
        options.Password.RequiredLength = 7;  
        options.SignIn.RequireConfirmedEmail = true;  
    }).AddEntityFrameworkStores<AuthenticationContext>();  
    services.ConfigureApplicationCookie(options => {  
        options.Cookie.Name = "AuthenticationCookie";  
    });  
}
```



Demonstration: How to use ASP.NET Core Identity

In this demonstration, you will see how to:

- Configure ASP.NET Core Identity
- Use SignInManager and UserManager





Customizing Providers with ASP.NET Core

- By default, ASP.NET Core Identity uses Entity Framework and the built-in authentication logic
- You can extend Identity to allow authentication from multiple external sources
- You are also able to use Windows Active Directory to handle authentication for you, to prevent external users from accessing your application
- Furthermore, the backend can be fully customized to use any form of storage you require, as long as it can be used in an ASP.NET Core application





Lesson 2: Authorization in ASP.NET Core

- Introduction to Authorization
- Authorization Options
- Demonstration: How to Authorize Access to Controller Actions





Introduction to Authorization

- The **AuthorizeAttribute** attribute:
 - Restricts user access to information
 - Mandates that users should be authorized to access the authorized controller or action
- The **AllowAnonymousAttribute** attribute:
 - Allows users access to an action in a controller with the **Authorize** attribute





Authorization Options

There are many different ways to block users from accessing resources.

- You may want certain resources or actions only available to administrative users and can implement this by using roles
- You might want users to provide specific information before allowing them access to certain resources and can implement this by using claims
- You might want a completely unique logic for directing access and can implement this by using custom policies





Demonstration: How to Authorize Access to Controller Actions

In this demonstration, you will see how to:

- Set up authorization
- Use simple authorization





Lesson 3: Defending from Attacks

- Cross-Site Scripting
- Cross-Site Request Forgery
- SQL Injection
- Cross-Origin Requests
- Secure Sockets Layer





Cross-Site Scripting

- Cross-site scripting involves:
 - Inserting malicious code in the session of a user
 - Posting information to other websites without the knowledge of the concerned users
- You can prevent cross-site scripting by:
 - Using the JavaScriptEncoder class and the Encode method when encoding inputs into JavaScript
 - Using the URLEncoder class and the Encode method when encoding inputs into URL





Cross-Site Request Forgery

Cross-Site Request Forgery

- Exploits existing cookies on a browser to perform actions on victim sites maliciously.

To prevent them in your ASP.NET Core MVC applications you should do the following:

- Utilize **ValidateAntiForgeryToken** or **AutoValidateAntiForgeryToken** to protect actions on your controllers
- Use tag helpers and forms with **PUT**, **POST** or **DELETE** methods in your views to ensure tokens are created correctly






SQL Injection

SQL Injection Attack

To prevent this attack, you can:

- Utilize Entity Framework or other ORMs – These libraries are designed with SQL injection in mind, and make it harder to perform
 - Use parameterized queries – These can allow you to create dynamic SQL while preventing external sources from affecting it
 - Use stored procedures – Stored procedures use parameters, and do not execute SQL that is not present inside the stored procedure
 - Use the lowest required permissions – Restricting permissions granted to your application can help prevent table modifications
 - Sanitize parameters – Ensure that any parameter added into SQL cannot be used to run SQL code
- 



Cross-Origin Requests

Cross-Origin Resource Sharing (CORS)

- By default your server will not accept any cross-origin requests from external sources.
- You can enable it and utilize CORS policies to create conditions under which your application content will be accessible to external applications.
- Enabling CORS can create a risk within applications and needs to be handled with care. It is important to create policies as specific as required.





Secure Sockets Layer

SSL:

- Encrypts content by using the public key declared by the certificate
- Decrypts content by utilizing the private key only available to the certificate owner
- Ensures that data sent across the web is encrypted, making it much harder to steal or change
- Certificate is determined on the initial request between the client and server





Lab: Managing Security

- Exercise 1: Use Identity
- Exercise 2: Add Authorization
- Exercise 3: Avoid the Cross-Site Request Forgery Attack

Estimated Time: 60 minutes





Lab Review

- A member of your team needs to configure the database context to work with identity dbContext, how should he do it?
- A member of your team changed the LibrarianController class, removed the following attribute from the class [Authorize(Roles = "Administrator")] what is the impact of his change?





Module Review and Takeaways

- Review Question
- Best Practice
- Common Issues and Troubleshooting Tips



