EXPERT NETWORK

Web Development with ASP.NET Core 7

S4

FII Practic Project Presentation (1st of April)

- In our last session you will be invited to present your own project for this course.
- You may choose any subject for your project, other than cars, and it needs to be your own creation.
- The best 3 projects will receive Prizes and points on the Internship Technical Test.

EXPERT NETWORK

The Sessions

1. Data Access
2. Concepts and Techniques
3. ASP.NET Core Introduction
4. **ASP.NET Core Advanced**
5. Deploy in the Cloud

*Note that each session builds upon the previous one.

For this session

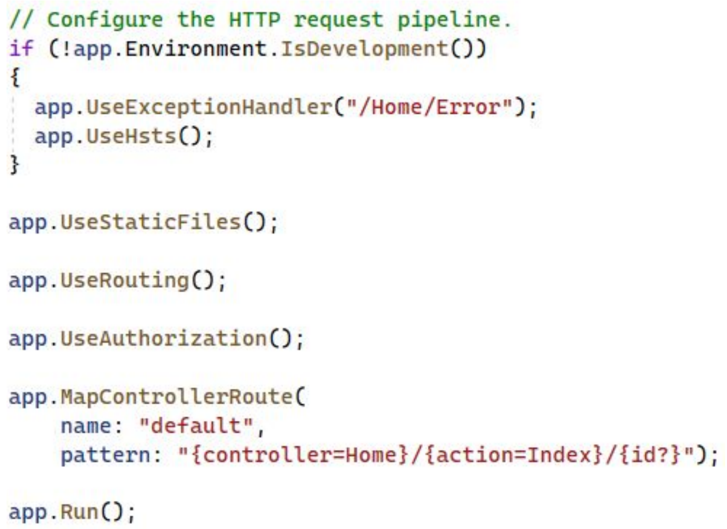
- HTTP Request Pipeline (middlewares)
- Authentication and Authorisation
- Security

*We will start from https://github.com/AlexandruCristianStan/FII-Practic-2023.

EXPERT NETWORK

HTTP Request Pipeline

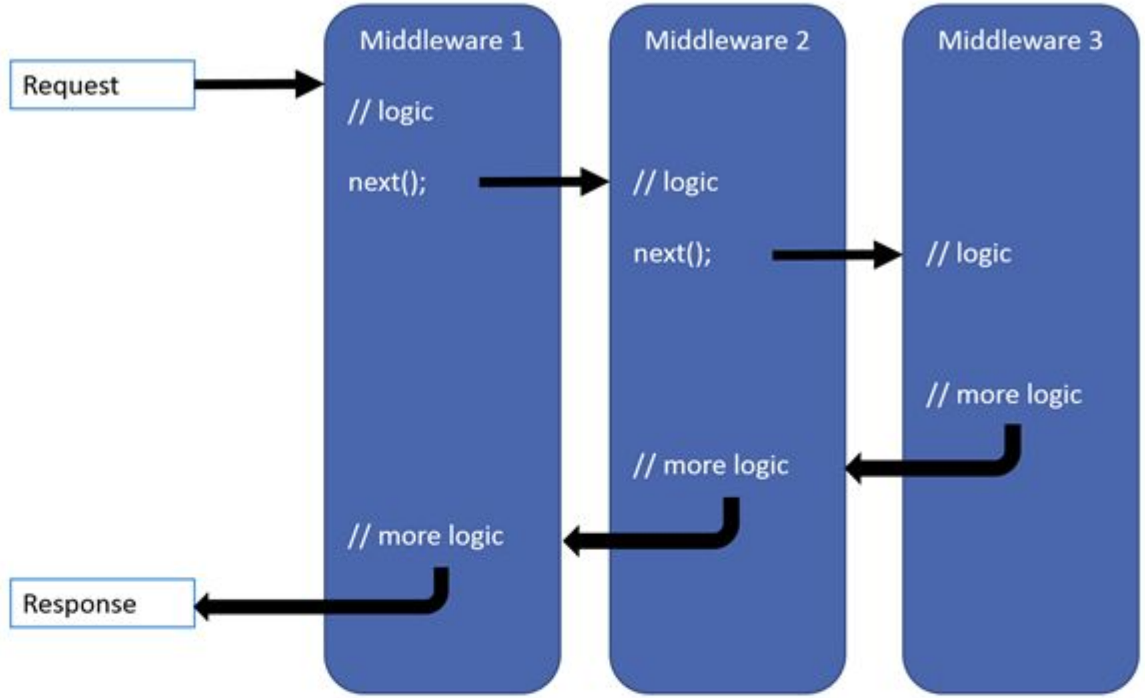
Configuration



```
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

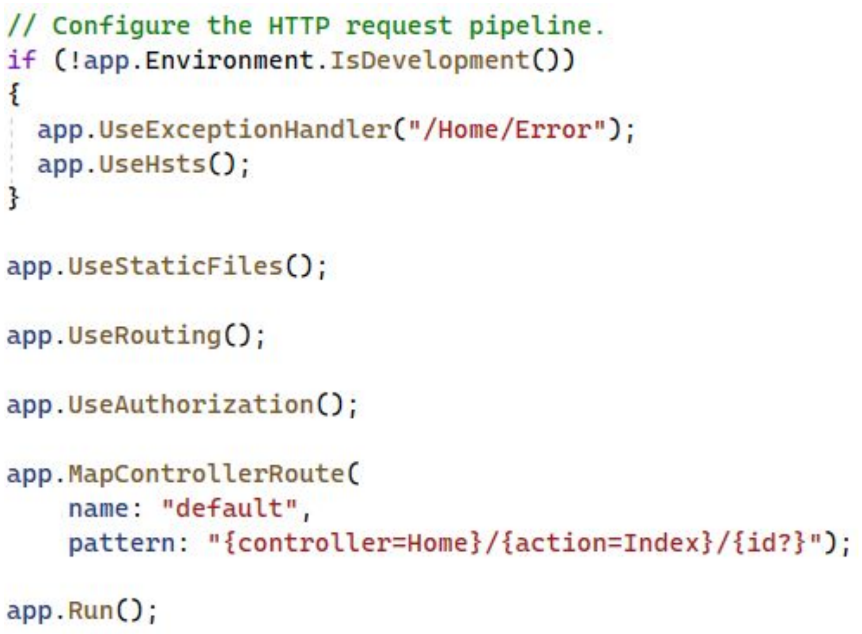
EXPERT NETWORK

HTTP Request Pipeline

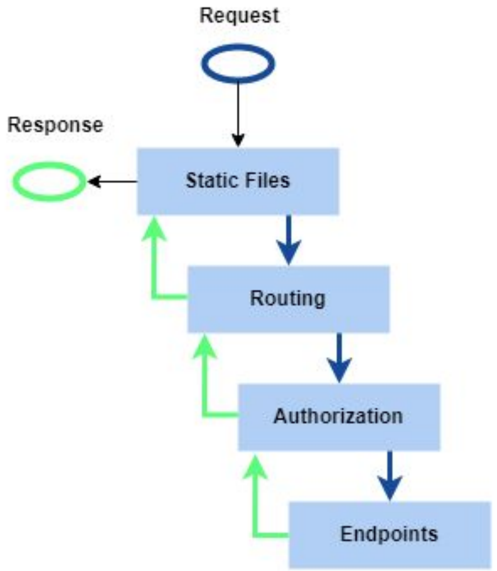


EXPERT NETWORK

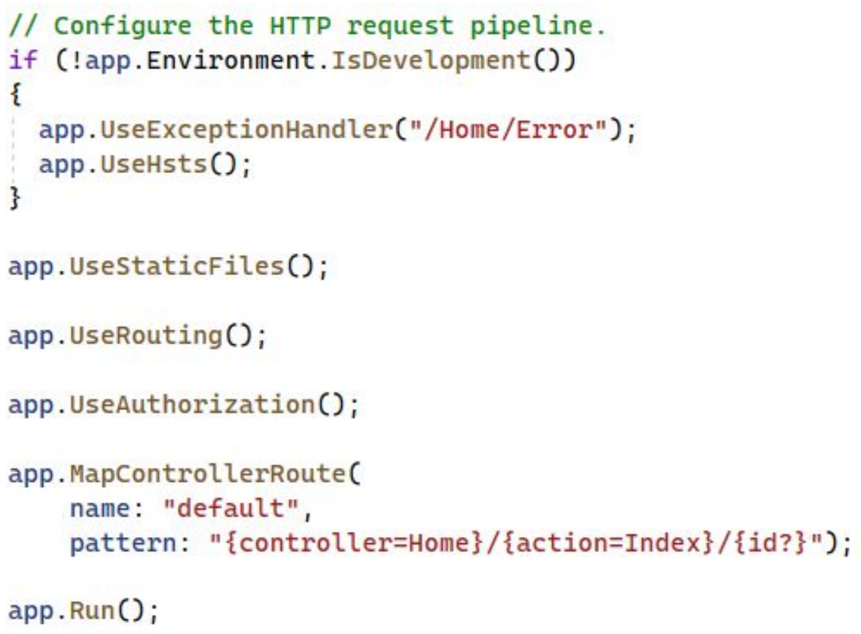
HTTP Request Pipeline (DEV)



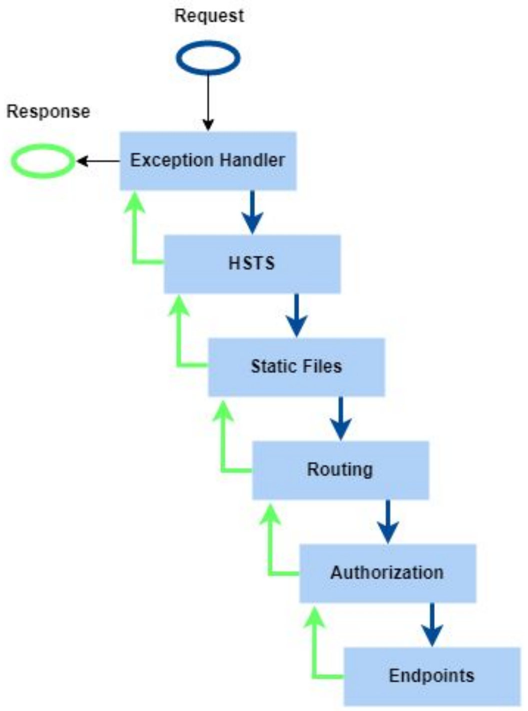
```
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```



HTTP Request Pipeline (PROD)

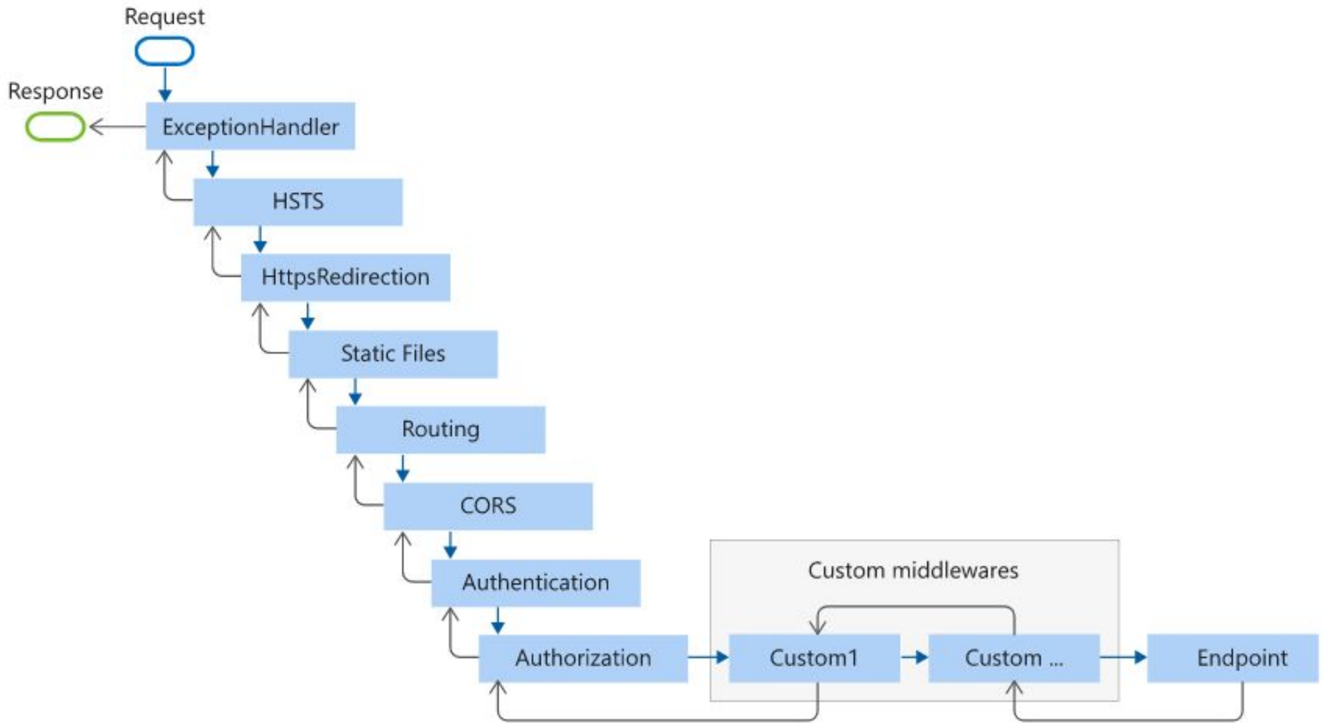


```
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```



EXPERT NETWORK

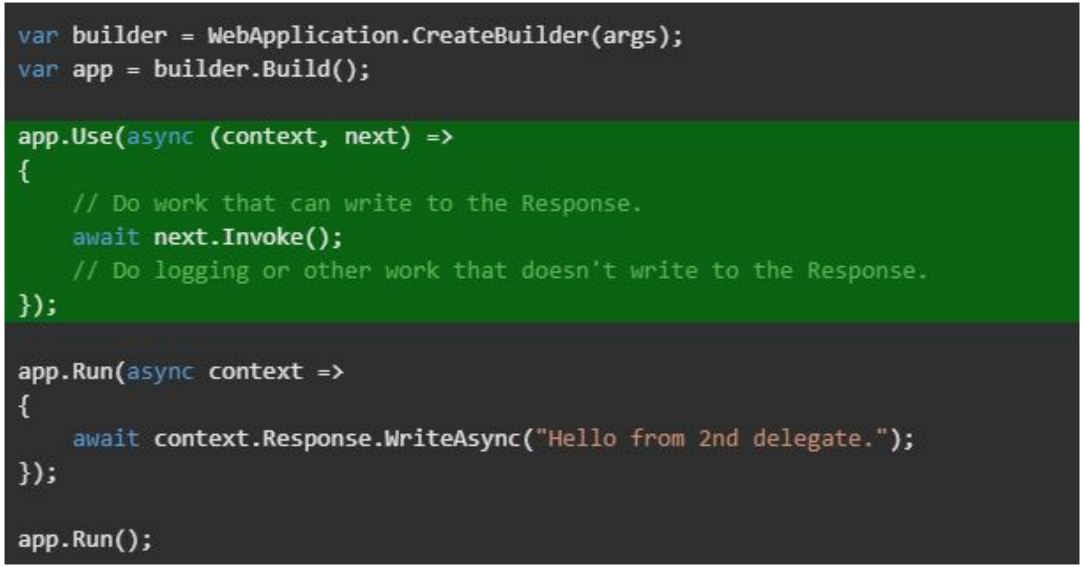
Most used pipeline



EXPERT NETWORK

Adding a middleware

Chain multiple request delegates together with **Use**. The next parameter represents the next delegate in the pipeline. You can short-circuit the pipeline by not calling the next parameter. You can typically perform actions both before and after the next delegate, as the following example demonstrates:



```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    // Do work that can write to the Response.
    await next.Invoke();
    // Do logging or other work that doesn't write to the Response.
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello from 2nd delegate.");
});

app.Run();
```

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-7.0

EXPERT NETWORK

Authentication and Authorization

In simple terms, authentication is the process of verifying who a user is, while authorization is the process of verifying what they have access to.

Authentication
user name
******
LOGIN
Who are you?
Validate a system is accessing by the right person
Authorization
Are you allowed to do that?
Check users' permissions to access data

EXPERT NETWORK

Authentication and Authorization

Here's a quick overview of the differences between authentication and authorization:

Authentication	Authorization
Determines whether users are who they claim to be	Determines what users can and cannot access
Challenges the user to validate credentials (for example, through passwords)	Verifies whether access is allowed through policies and rules
Usually done before authorization	Usually done after successful authentication
Example: when you want to cross the border you are required to 'authenticate' (show passport)	Example: after successfully authenticated, whether you have the right visa or not will determine if you are allowed to enter the other country (if you are authorized)

EXPERT NETWORK

Authentication

Types of authentication:

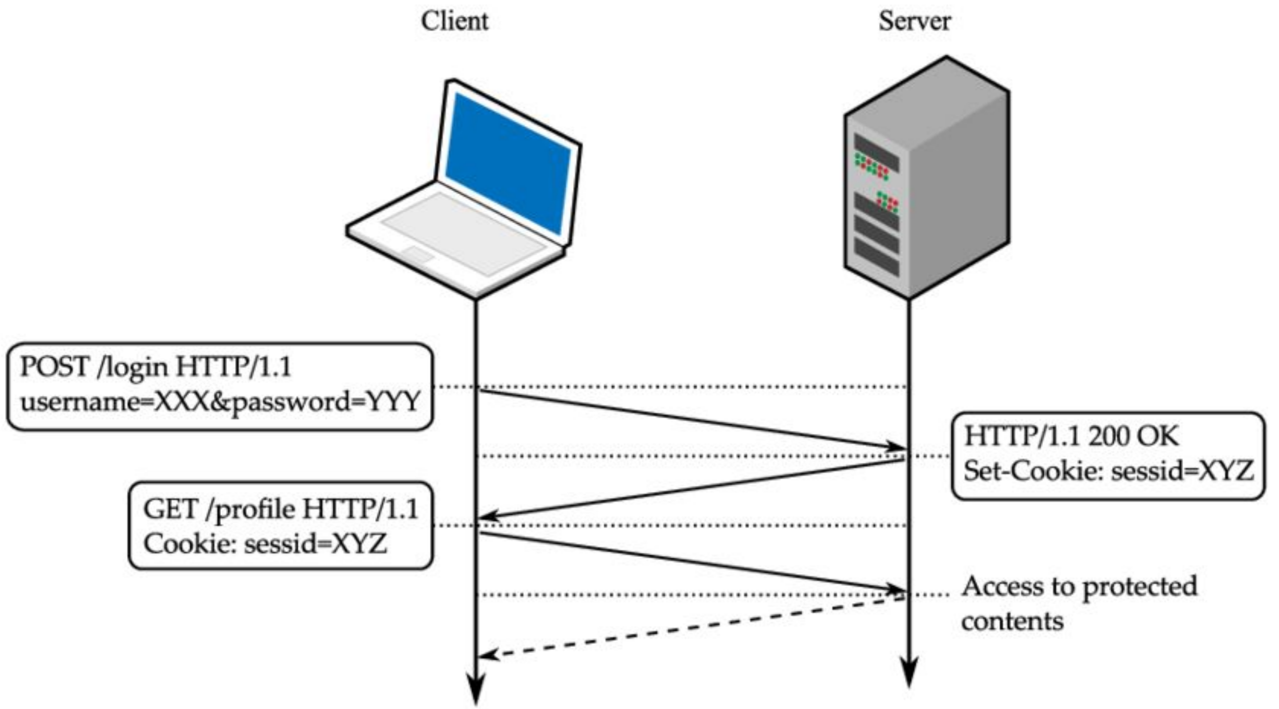
- Basic authentication
- **Cookie authentication (we’ll use this one)**
- Token authentication
- API Key Based authentication
- OAuth (Open authorization)

Cookie Authentication

We will use this type of authentication because of its simplicity and because it fits our simple application.

An advantage of using cookies is that the browser will **automatically send cookies**. This feature makes cookies a good way to secure websites, where a user logs in and navigates between pages using links.

Cookie Authentication



EXPERT NETWORK

Cookie Authentication

The browser automatically sending cookies also has a big downside, which is CSRF attacks. In a CSRF attack, a malicious website takes advantage of the fact that your browser will automatically attach authentication cookies to requests to that domain and tricks your browser into executing a request.
We can protect by using CSRF tokens / Cookie SameSite attribute (which modern browsers now protects you by default) / SameOrigin policy (same as previous).

Also, cookies make it more difficult for non-browser based applications (like mobile to tablet apps) to consume your API.

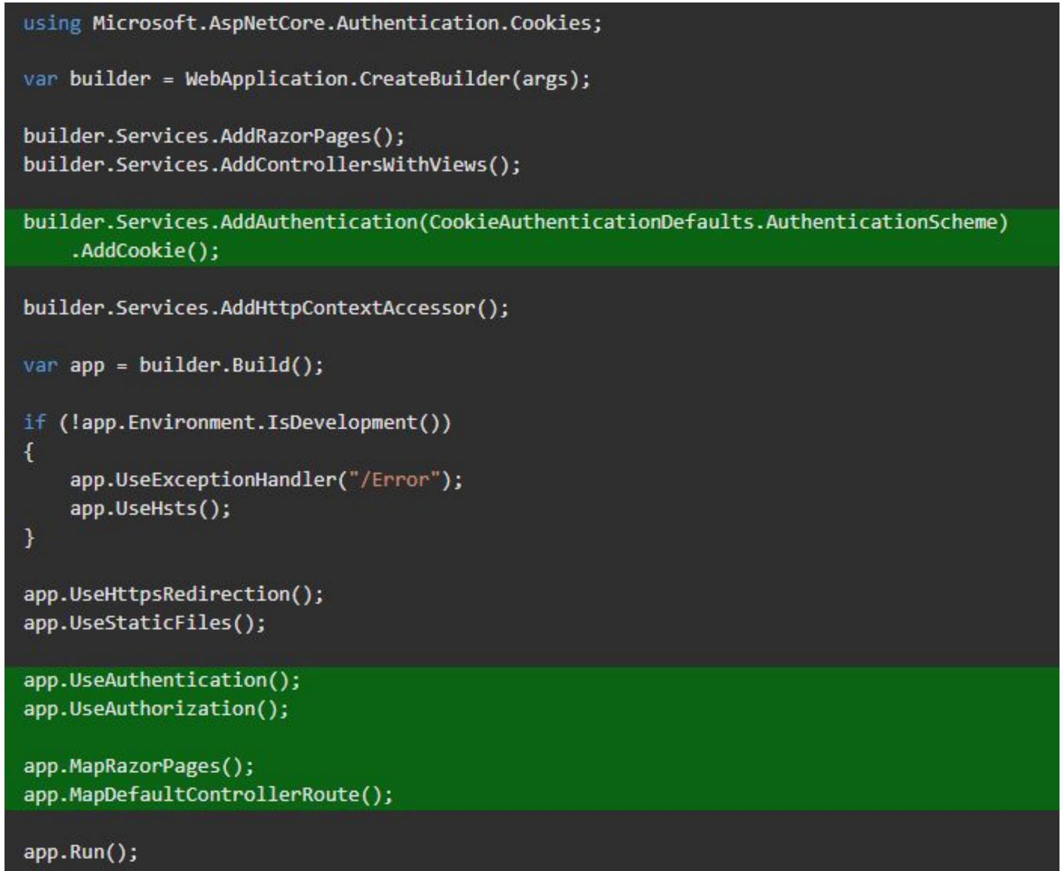
Cookie Authentication

Add the Authentication Middleware services with the **AddAuthentication** and **AddCookie** methods.
Call **UseAuthentication** and **UseAuthorization** to set the **HttpContext.User** property and run the Authorization Middleware for requests. **UseAuthentication** and **UseAuthorization** must be called before Map methods such as **MapRazorPages** and **MapDefaultControllerRoute**.

https://learn.microsoft.com/en-us/aspnet/core/security/authentication/cookie?source=recommendations&view=aspnetcore-7.0

EXPERT NETWORK

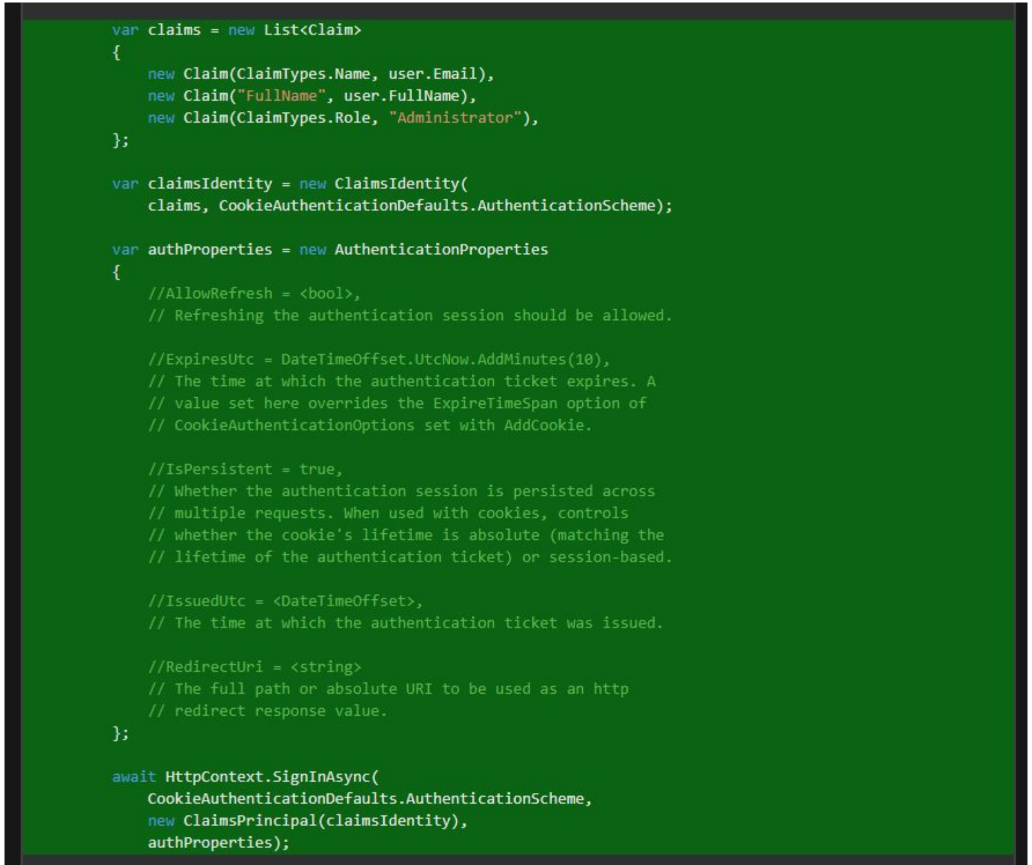
Cookie Authentication



```
using Microsoft.AspNetCore.Authentication.Cookies;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();
builder.Services.AddControllersWithViews();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie();

builder.Services.AddHttpContextAccessor();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseAuthentication();
app.UseAuthorization();

app.MapRazorPages();
app.MapDefaultControllerRoute();

app.Run();
```

EXPERT NETWORK

Cookie Authentication



```
var claims = new List<Claim>
{
    new Claim(ClaimTypes.Name, user.Email),
    new Claim("FullName", user.FullName),
    new Claim(ClaimTypes.Role, "Administrator"),
};

var claimsIdentity = new ClaimsIdentity(
    claims, CookieAuthenticationDefaults.AuthenticationScheme);

var authProperties = new AuthenticationProperties
{
    //AllowRefresh = <bool>,
    // Refreshing the authentication session should be allowed.

    //ExpiresUtc = DateTimeOffset.UtcNow.AddMinutes(10),
    // The time at which the authentication ticket expires. A
    // value set here overrides the ExpireTimeSpan option of
    // CookieAuthenticationOptions set with AddCookie.

    //IsPersistent = true,
    // Whether the authentication session is persisted across
    // multiple requests. When used with cookies, controls
    // whether the cookie's lifetime is absolute (matching the
    // lifetime of the authentication ticket) or session-based.

    //IssuedUtc = <DateTimeOffset>,
    // The time at which the authentication ticket was issued.

    //RedirectUri = <string>
    // The full path or absolute URI to be used as an http
    // redirect response value.
};

await HttpContext.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    new ClaimsPrincipal(claimsIdentity),
    authProperties);
```

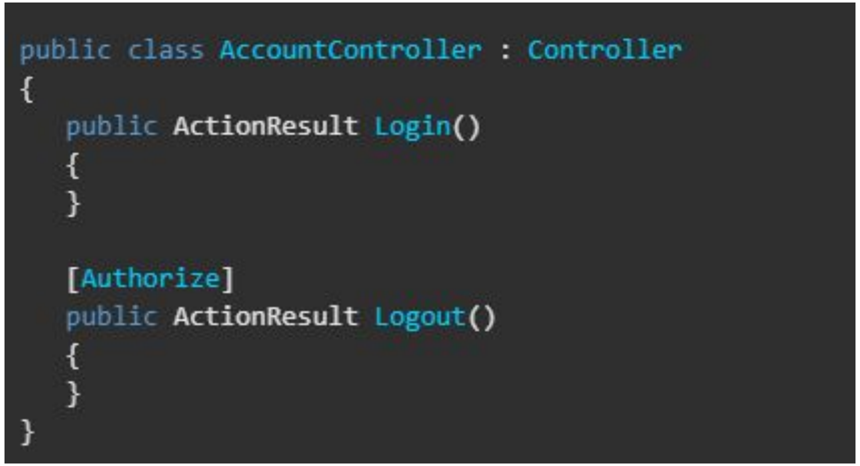
EXPERT NETWORK

Exercise 1: Implement cookie authentication

EXPERT NETWORK

Authorization

Authorization in ASP.NET Core is controlled with **AuthorizeAttribute** and its various parameters. In its most basic form, applying the **[Authorize]** attribute to a controller, action, or Razor Page, limits access to that component to authenticated users.



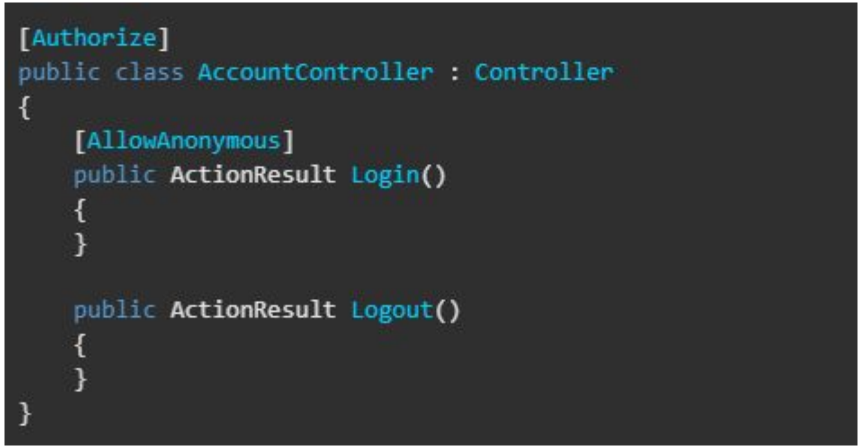
```
public class AccountController : Controller
{
    public ActionResult Login()
    {
    }

    [Authorize]
    public ActionResult Logout()
    {
    }
}
```

https://learn.microsoft.com/en-us/aspnet/core/security/authorization/simple?view=aspnetcore-7.0

EXPERT NETWORK

Authorization

You can also use the **AllowAnonymous** attribute to allow access by non-authenticated users to individual actions. For example:



```
[Authorize]
public class AccountController : Controller
{
    [AllowAnonymous]
    public ActionResult Login()
    {
    }

    public ActionResult Logout()
    {
    }
}
```

EXPERT NETWORK

Authorization

If you want to make more custom authorization, you can implement claim based authorization.

https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-7.0

Exercise 2: Add authorized pages

EXPERT NETWORK

Password storage

It is essential to store passwords in a way that prevents them from being obtained by an attacker even if the application or database is compromised.

After an attacker has acquired stored password hashes, they are always able to brute force hashes offline. As a defender, it is only possible to slow down offline attacks by selecting hash algorithms that are as resource intensive as possible.

Hashing vs Encryption

EXPERT NETWORK

Hashing vs Encryption

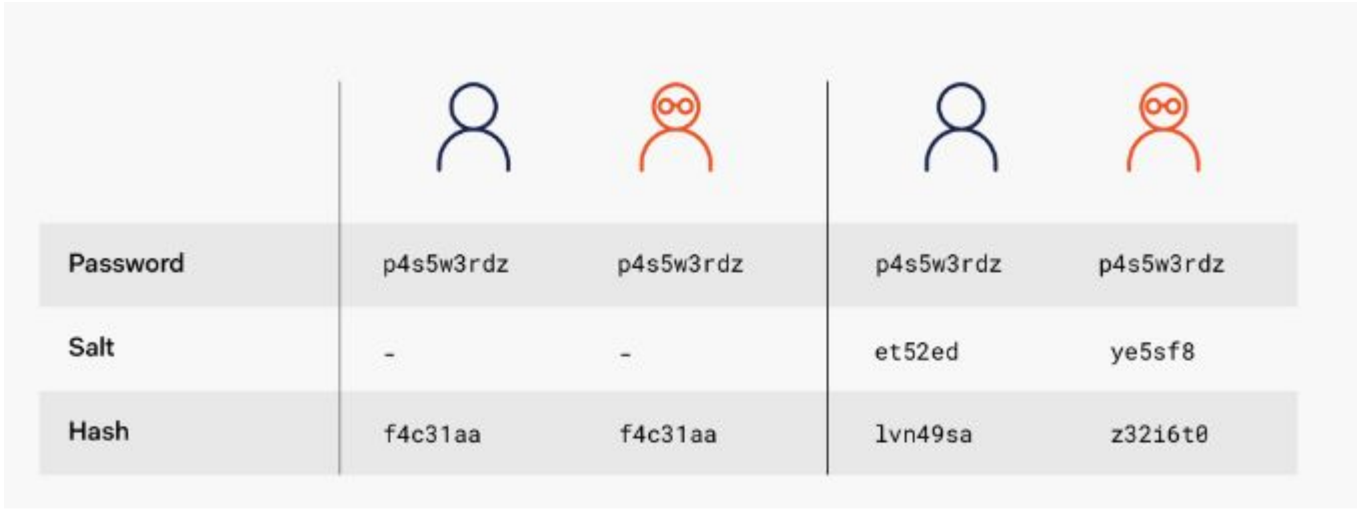
Hashing and encryption both provide ways to keep sensitive data safe. However, in almost all circumstances, passwords should be hashed, **NOT encrypted**.

Hashing is a one-way function (i.e., it is impossible to "decrypt" a hash and obtain the original plaintext value). Hashing is appropriate for password validation. Even if an attacker obtains the hashed password, they cannot enter it into an application's password field and log in as the victim.

Encryption is a two-way function, meaning that the original plaintext can be retrieved. Encryption is appropriate for storing data such as a user's address since this data is displayed in plaintext on the user's profile. Hashing their address would result in a garbled mess.

Salting

A salt is a unique, randomly generated string that is added to each password as part of the hashing process.



Password	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz
Salt	-	-	et52ed	ye5sf8
Hash	f4c31aa	f4c31aa	lvn49sa	z32i6t0

Salting

Salting also protects against an attacker pre-computing hashes using rainbow tables or database-based lookups. Finally, salting means that it is impossible to determine whether two users have the same password without cracking the hashes, as the different salts will result in different hashes even if the passwords are the same.

Password storage in our app

More details about password storage here (including pepper, which is an additional layer of security): https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

We will use Argon2id in our application as hashing algorithm as it is one of the most recommended algorithms today.

Is also has salting mechanism out of the box.

Exercise 3: Password hashing

EXPERT NETWORK

Next week

- Application deployment in cloud.
  Please make an account on https://portal.azure.com/ (card details might be required, but don’t worry, we’ll use free stuff)
- Final project presentation

*Don’t forget the feedback form!

EXPERT NETWORK