Authentication and Authorization in Blazor Applications

GETTING STARTED WITH AUTHENTICATION IN BLAZOR

Kevin Dockx
ARCHITECT

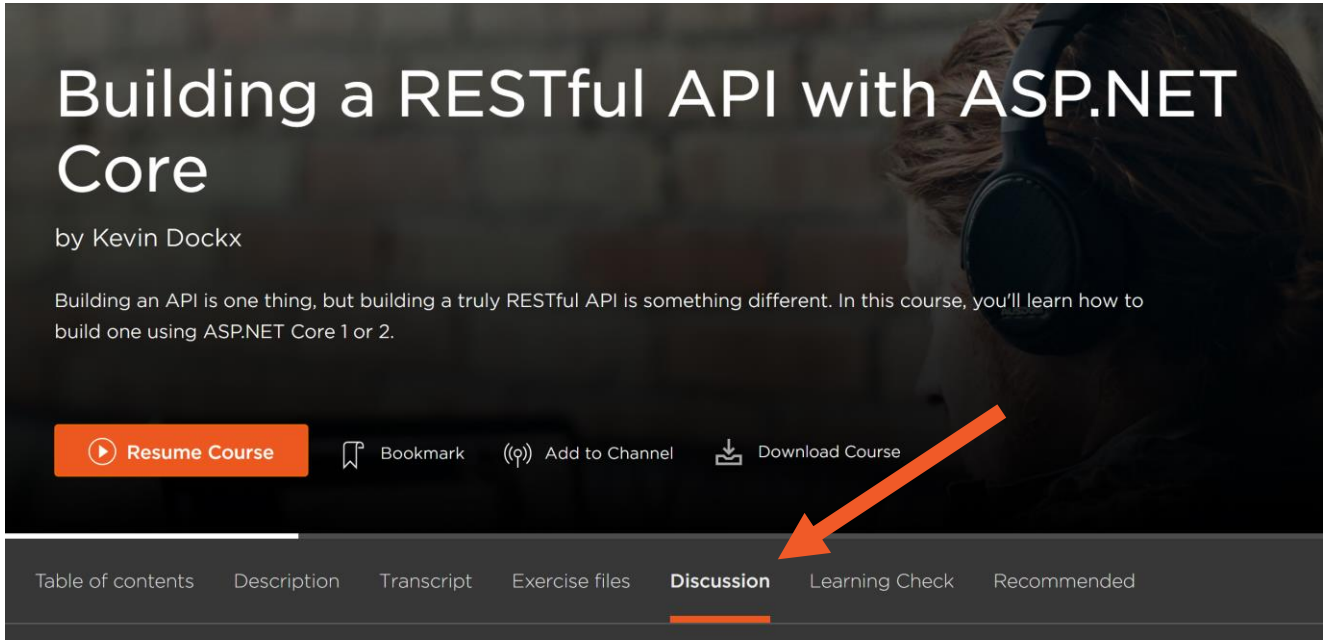
@KevinDockx https://www.kevindockx.com

Coming Up

- Course prerequisites, tooling and framework versions
- Blazor authentication scenarios
- Logging in and logging out with cookie authentication
- Working with authentication state
- Protecting the API
- Overview of authentication-related concepts

Discussion tab on the course page

Twitter: @KevinDockx



(course shown is one of my other courses, not this one)

Course Prerequisites

Good knowledge of C#

Some knowledge of Blazor (server)

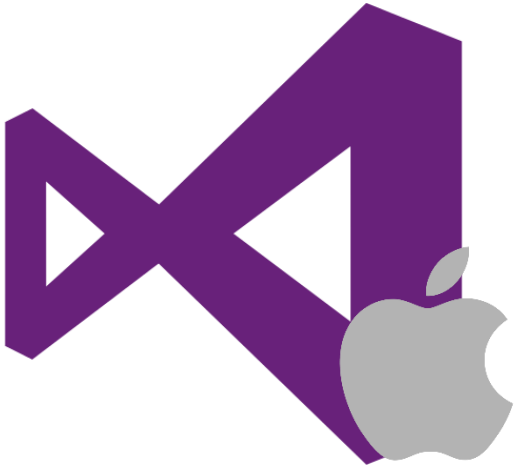
ASP.NET Core Fundamentals (Scott Allen)

Getting Started with Blazor (Gill Cleeren)

Tooling

Visual Studio 2019
v16.3 or better

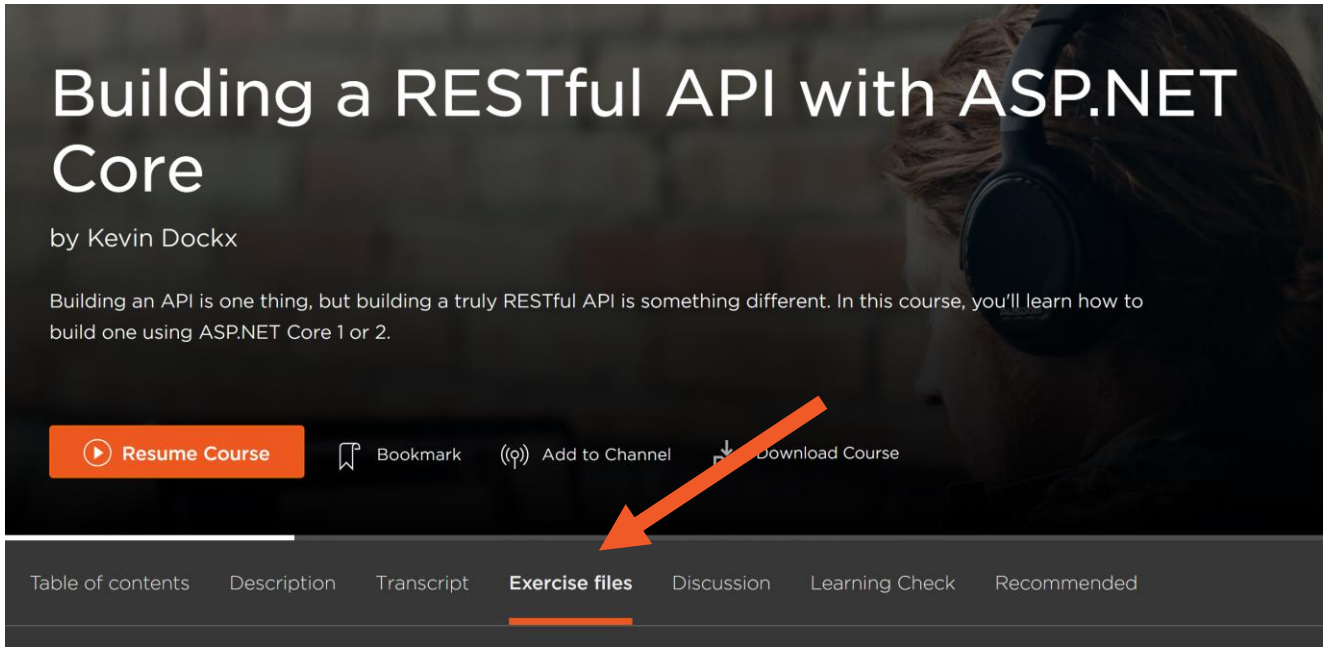
Visual Studio
Code



Visual Studio for
Mac

JetBrains Rider

Exercise files tab on the course page

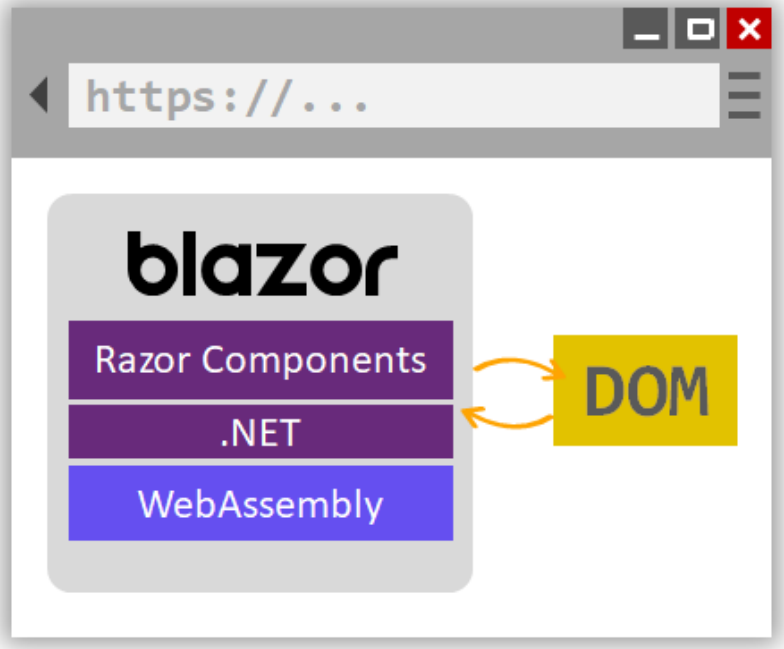


(course shown is one of my other courses, not this one)

Blazor Authentication Scenarios

Blazor WebAssembly

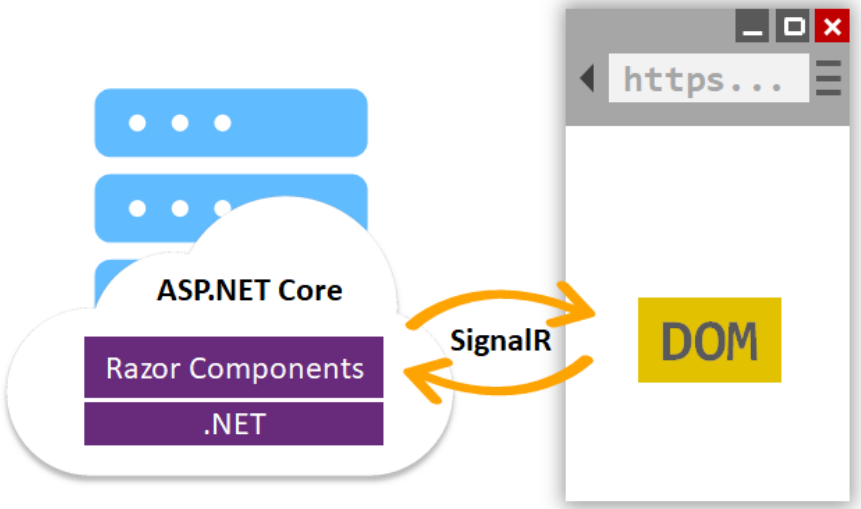
Blazor Server



(image by Microsoft, https://bit.ly/2NKq1U8)

Blazor WebAssembly

- Compiled .NET Core assemblies & runtime downloaded to browser
- WebAssembly bootstraps & configures runtime
- JavaScript interop to handle DOM manipulation & API Calls



(image by Microsoft, https://bit.ly/2NKq1U8)

Blazor Server

- Razor components hosted on the server in an ASP.NET Core application
- UI updates handled over SignalR connection (also used for JavaScript interop calls)
- Runtime handles sending UI events from the browser to the server and applies UI updates sent by the to the client

Blazor WebAssembly

Runs on the client thus cannot be trusted

Any authorization check can be bypassed

Focus is on securing the API

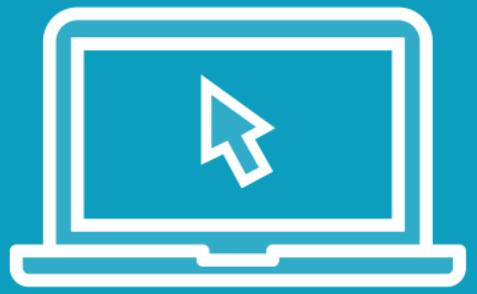
Blazor Server

Runs on the server thus can be trusted

Authorization checks can be enforced, access rules can be implemented

Securing the API is still a focus point

Demo



Introducing the demo application

Demo

Adding cookie authentication and logging in

Demo

Logging out

Cookie Authentication in Blazor

Client (browser)

Server

IPrincipal

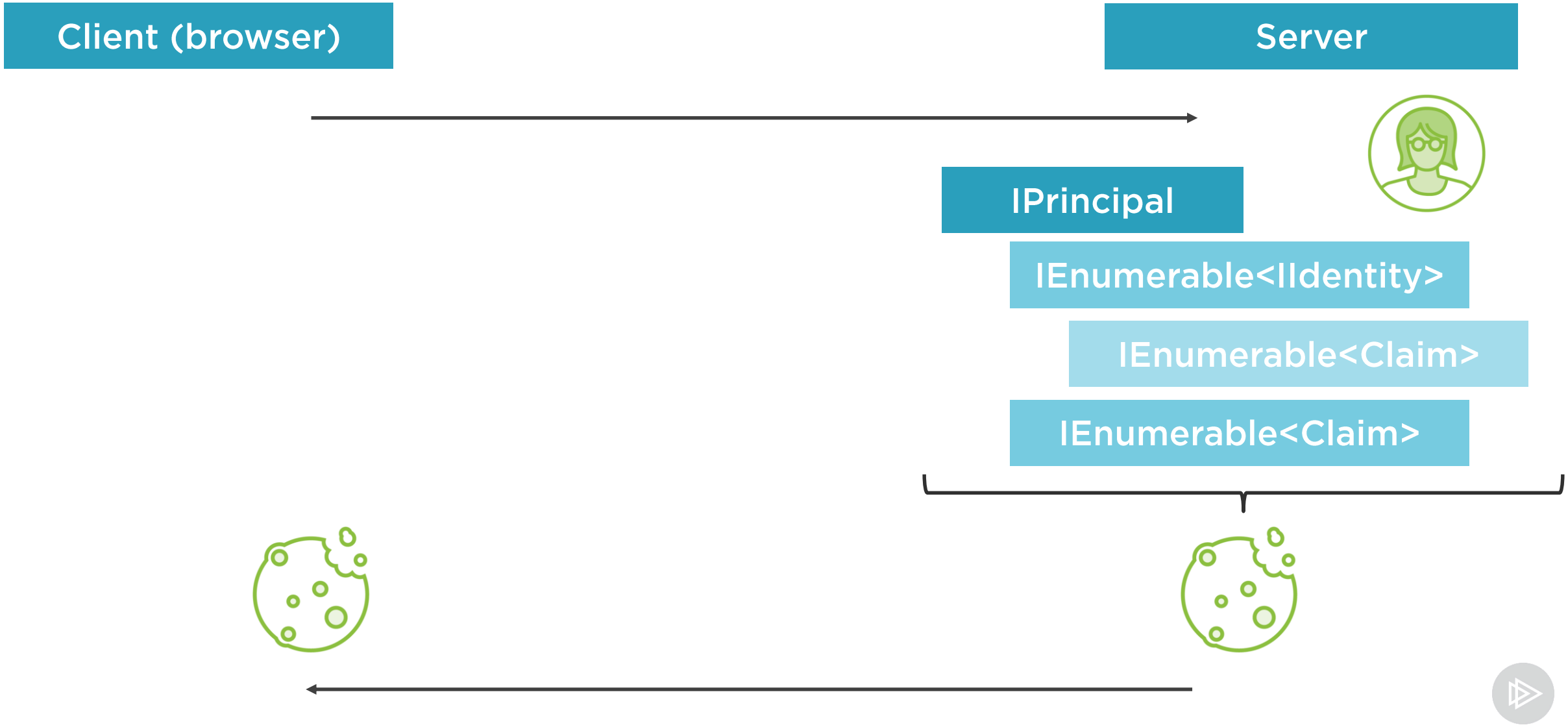
IPrincipal

Represents the security context of the user on whose behalf the code is running, and includes one or more user identities

IIdentity

Represents the user on whose behalf the code is running

Cookie Authentication in Blazor



Cookie Authentication in Blazor

Cookie Authentication in Blazor

Client (browser)

Server

IPrincipal

Cookie Authentication in Blazor

Blazor Server operates over SignalR

- User must be associated with each connection
- Cookie authentication allows your existing user credentials to automatically flow to SignalR connections

Demo

Getting the authenticated user

Demo

Hiding or showing parts of the UI depending on the authentication state

AuthenticationStateProvider

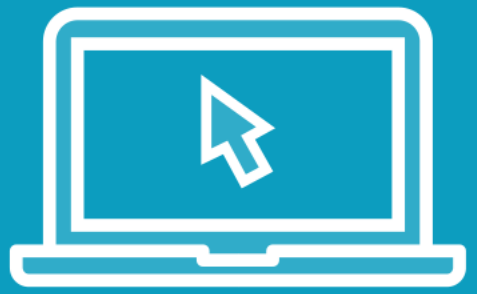
A built-in service that obtains authentication state data from ASP.NET Core's HttpContext.User and powers components like AuthorizeView and CascadingAuthenticationState

Explaining the Authentication StateProvider

Don’t directly use the AuthenticationStateProvider

- Component isn’t automatically notified if the underlying authentication state data changes
- User AuthorizeView and CascadingStateProvider components instead

Demo



Blocking unauthorized access to a page

Demo

Customizing unauthorized content

Demo

Using authentication state data in procedural logic

Protecting the API

Our API project can be deployed to another domain than the Blazor Server project

- Cookies, by default, cannot be shared across domains
- In that case, the API server cannot read the Blazor cookie

Protecting the API

Other approaches / workarounds exist

- Cookies can potentially be shared across subdomains
- Some old SSO solutions used “cross-domain” cookies (currently considered a bad practice)

Protecting the API

We shouldn't make assumptions regarding where our Blazor Server application & API will be deployed

- Cookie authentication is not a good fit for our API

Authentication and Authorization

Use cookies for web clients, use tokens for APIs

```
var authCookie = HttpContext.Request.Cookies[".AspNetCore.Cookies"];

if (authCookie != null) {
    httpClient.DefaultRequestHeaders.Add("Cookie",
        ".AspNetCore.Cookies=" + authCookie);
     }
```

Protecting the API

If you really need to use cookie authentication for the API, make sure you pass the cookie on each request

- HttpClient doesn't automatically add this when sending a request to the API

```
var authCookie = HttpContext.Request.Cookies[".AspNetCore.Cookies"];

if (authCookie != null) {
    httpClient.DefaultRequestHeaders.Add("Cookie",
        ".AspNetCore.Cookies=" + authCookie);
     }
```

Protecting the API

If you really need to use cookie authentication for the API, make sure you pass the cookie on each request

- HttpClient doesn't automatically add this when sending a request to the API

```
var authCookie = HttpContext.Request.Cookies[".AspNetCore.Cookies"];

if (authCookie != null) {
    httpClient.DefaultRequestHeaders.Add("Cookie",
        ".AspNetCore.Cookies=" + authCookie);
     }
```

Protecting the API

If you really need to use cookie authentication for the API, make sure you pass the cookie on each request

- HttpClient doesn't automatically add this when sending a request to the API

Overview of Authentication-related Concepts

Means of authentication

- Username/password, Windows credentials
- Certificates / USB tokens
- ...
- Potentially combined with 2FA

Proof that we are who we say we are

Overview of Authentication-related Concepts

Delivering this proof to different parts of the application

- Cookies
- Tokens
  - OAuth2, OpenID Connect
  - No domain restrictions

The network/system architecture of your application matters

Overview of Authentication-related Concepts

Dealing with this in code

- Write your own user-related classes and DB schema, implement custom membership-like functionality and/or user registration, ...
- Use ASP.NET Core Identity

Regardless of how you've proven who you are or how that proof has been delivered to different parts of your application

Summary

Write a cookie after successful authentication to log in, remove it to log out

Summary

Use the AuthorizeView component to selectively displays UI parts depending on whether the user is authorized to see them

Use the CascadingStateProvider to provide the current authentication state to its descendent components

- Router and AuthorizeView use this to control access to various parts of the UI

Summary

Use the Authorize attribute to control access to the page in full

- Combine with AuthorizeRouteView instead of RouteView

Get information on the user in your C# code by accessing the User object from the current AuthenticationState

Summary

Use cookies for securing web applications (like our Blazor application), and use tokens for securing APIs