Securing Blazor Server-side Applications

GETTING STARTED WITH AUTHENTICATION IN BLAZOR SERVER

Kevin Dockx
ARCHITECT

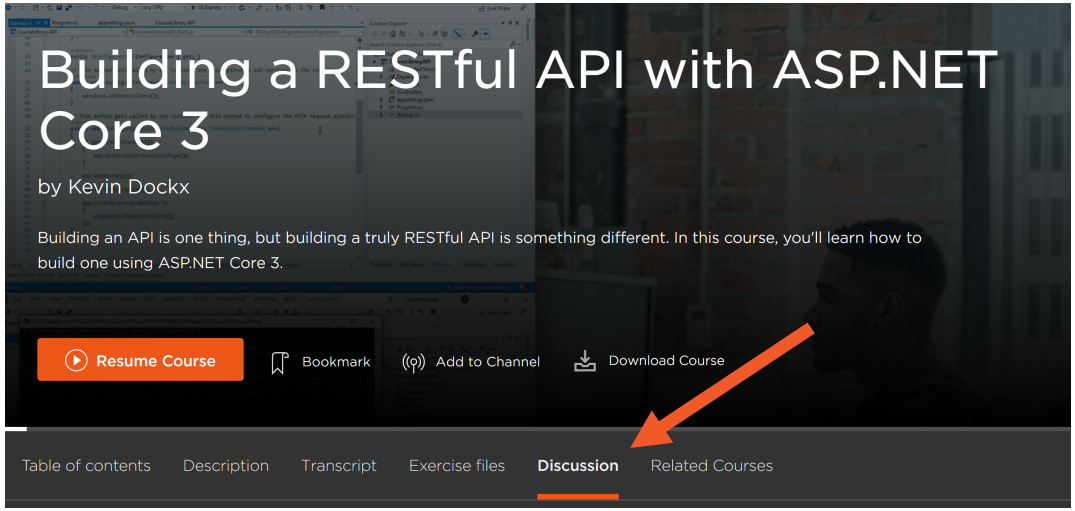
@KevinDockx https://www.kevindockx.com

Coming Up

- Course prerequisites and tooling
- Blazor authentication scenarios
- Logging in and logging out with cookie authentication
- Working with authentication state
- Protecting the API

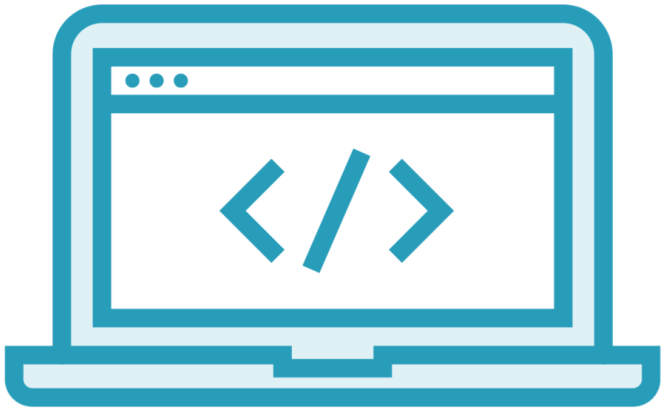
Discussion tab on the course page

Twitter: @KevinDockx



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

Course Prerequisites



Good knowledge of C#

Knowledge of Blazor Server

Course tip:
Blazor: Getting Started (Gill Cleeren)

Frameworks and Tooling

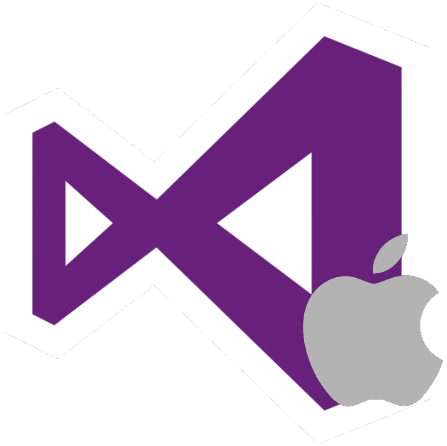
Visual Studio 2019
v16.4 or better

.NET Core 3.1

Frameworks and Tooling

Visual Studio 2019
v16.4 or better

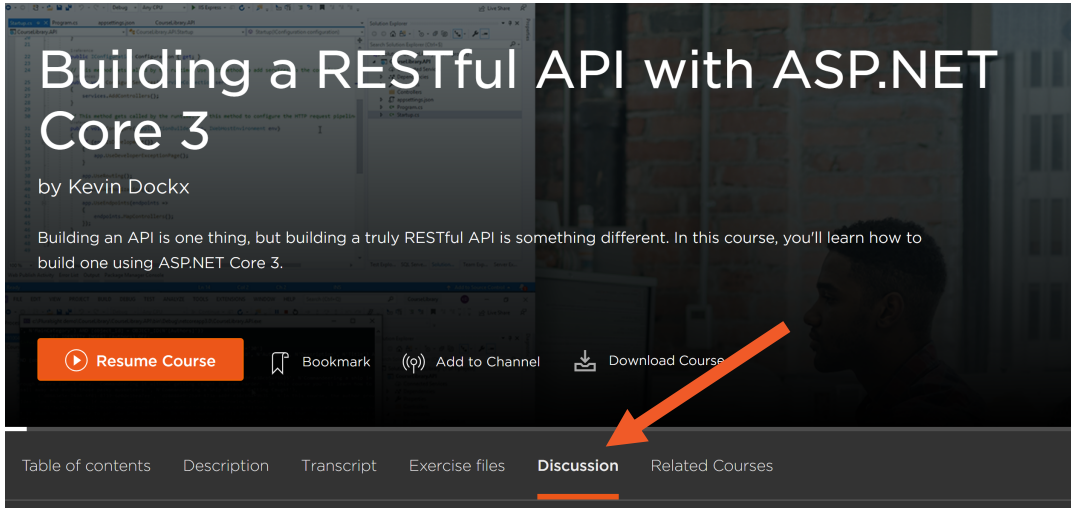
Visual Studio
Code



Visual Studio for
Mac

JetBrains Rider

Exercise files tab on the course page



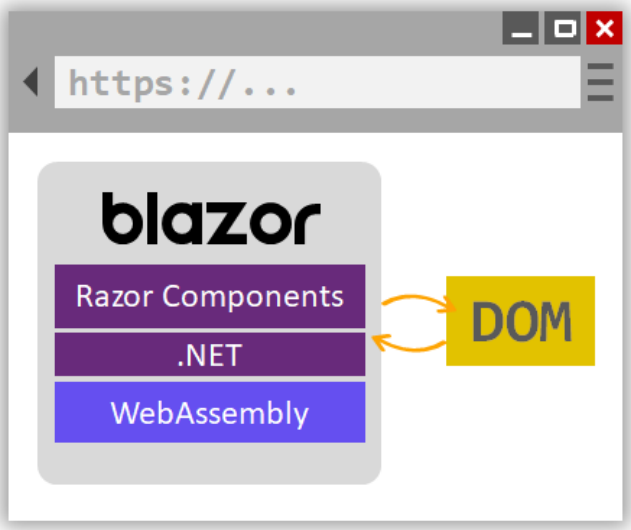
(course shown is one of my other courses, not this one)

Blazor Authentication Scenarios

Blazor WASM (WebAssembly)

Blazor Server

Blazor WASM



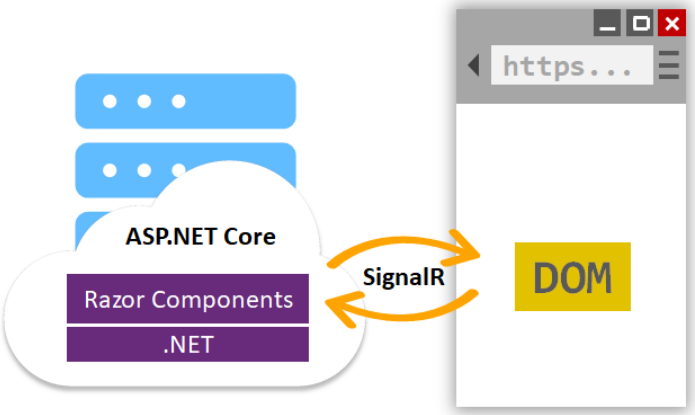
(image by Microsoft, https://bit.ly/2NKq1U8)

Compiled .NET Core assemblies & runtime downloaded to browser

WebAssembly bootstraps & configures runtime

JavaScript interop to handle DOM manipulation & API Calls

Blazor Server



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

Razor components hosted on the server in an ASP.NET Core application

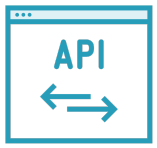
UI updates handled over SignalR connection (also used for JavaScript interop calls)

Runtime handles sending UI events from browser to server and applies UI updates sent by server to client

Blazor WASM

Runs on the client thus cannot be trusted

Any authorization check can be bypassed

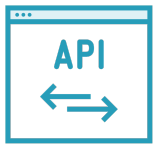


Focus is on securing the API

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

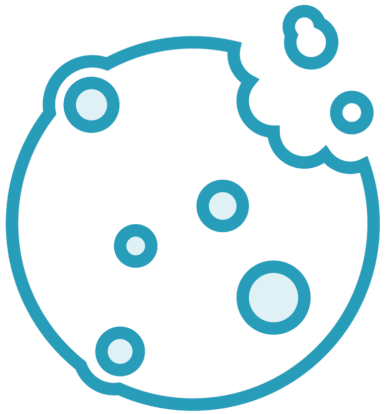
Introducing the demo application

Authentication Models, Cookies, and Tokens

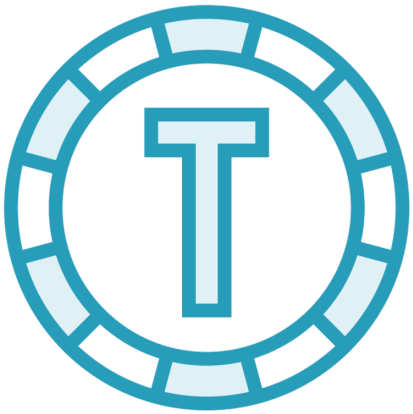
ASP.NET Core security concepts apply to Blazor Server

- Big contrast with Blazor WASM applications

Authentication Models, Cookies, and Tokens



SameSite cookies



Token-based security with OAuth2 and OpenID Connect

WASM Hosting Modes

Cookies

(Sub)domain restrictions

Browsers are adopting restrictive cookie policies

Simple to implement, less complex than token-based security

Tokens (OAuth2 / OpenID Connect)

Narrower permissions

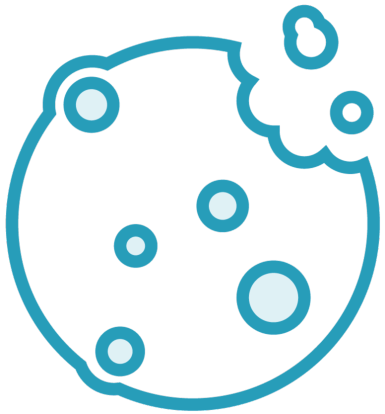
Short lifetime (limited attack window)

Can be revoked

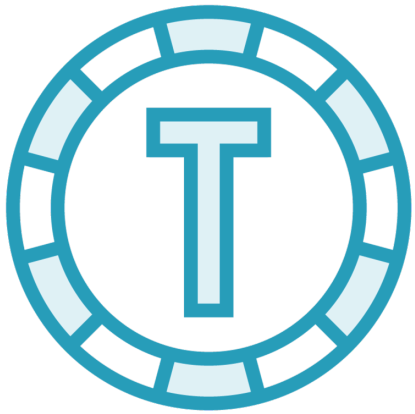
No CSRF protection for APIs needed

No (sub)domain restrictions

Authentication Models, Cookies, and Tokens



Useful if you want a self-contained solution with the users at application level



Useful in a multi-application landscape that requires a centralized user store, SSOn/SSOut

Demo

Adding cookie authentication and logging in

Demo

Logging out

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult AddItemPostBack([FromBody] Item itemToAdd)
{ …
}
```

Switching to POST for Logging Out

POST methods require an antiforgery token for XSRF protection

Works out of the box in ASP.NET Core

- Has access to HttpContext

Switching to POST for Logging Out

HttpContextAccessor (& HttpContext) should not be used in Blazor Server

- Depends on SignalR, which means the availability of the HttpContext depends on the underlying transport
- Not reliable

Solution: provide a XSRF token from the application host to the Blazor app

Demo

Providing Initial State Data

Cookie Authentication in Blazor

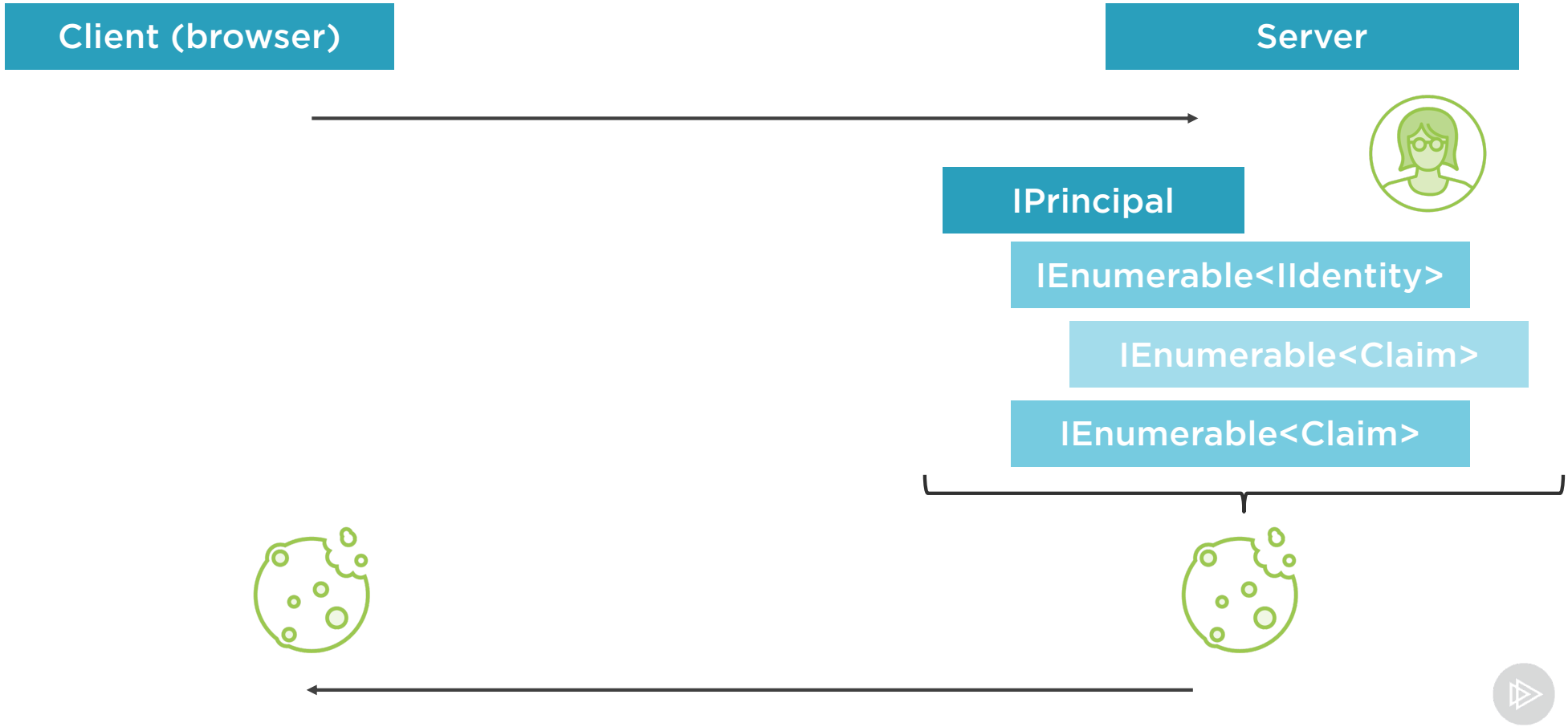
IPrincipal

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

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

AuthenticationStateProvider

A built-in service that obtains current authentication state data

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

Cookie authentication for APIs demo

- Add a dummy controller at level of the Server project
- Get the cookie to the Blazor Server app and send it on each request to the API

Demo

Protecting the API

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

Cookies can be used for securing an API on the same domain as the application host

- Pass the cookie through on each request