Securing Blazor Server-side Applications

GETTING STARTED WITH AUTHENTICATION IN BLAZOR SERVER

Kevin Dockx
ARCHITECT

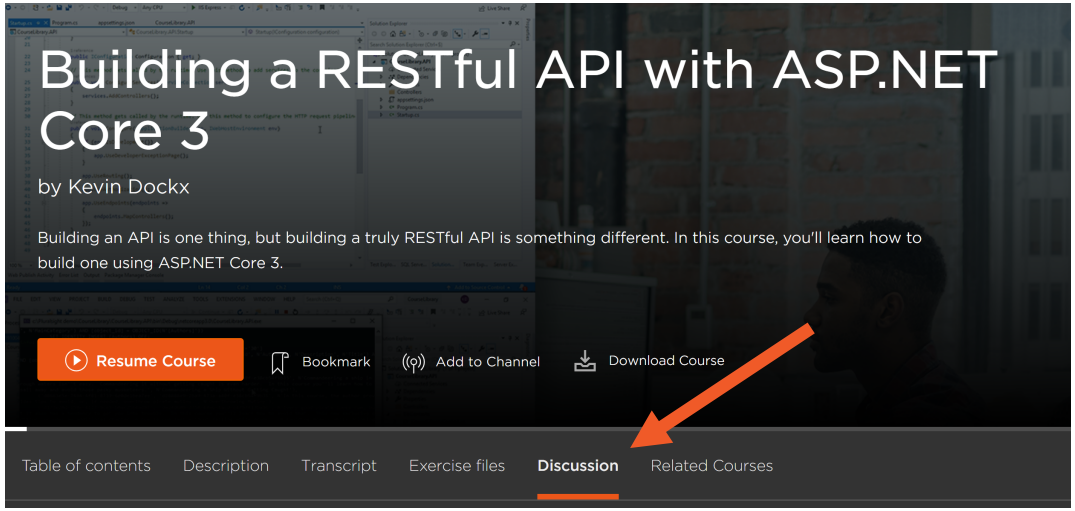
@KevinDockx https://www.kevindockx.com

Coming Up

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

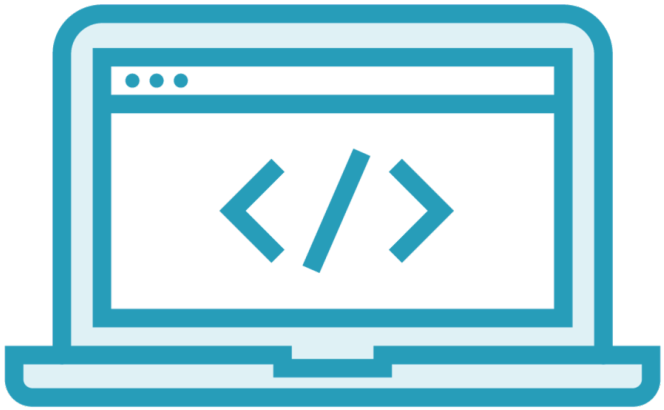
Discussion tab on the course page

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

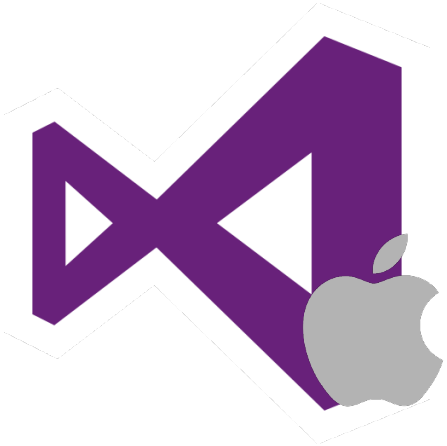
Visual Studio 2019
v16.4 or better

.NET Core 3.1

Frameworks and Tooling

Visual Studio 2019
v16.4 or better

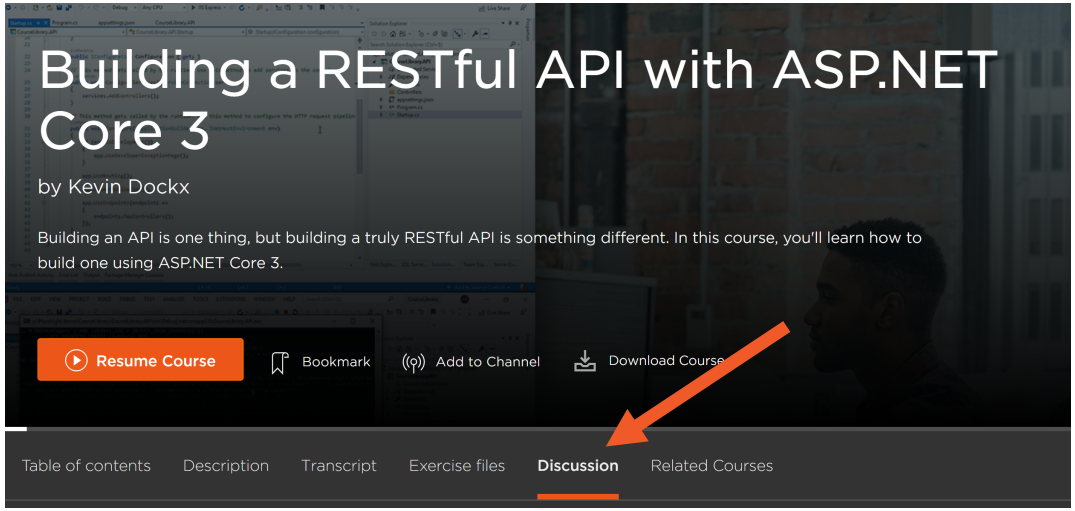
Visual Studio
Code



Visual Studio for
Mac

JetBrains Rider

Exercise files tab on the course page



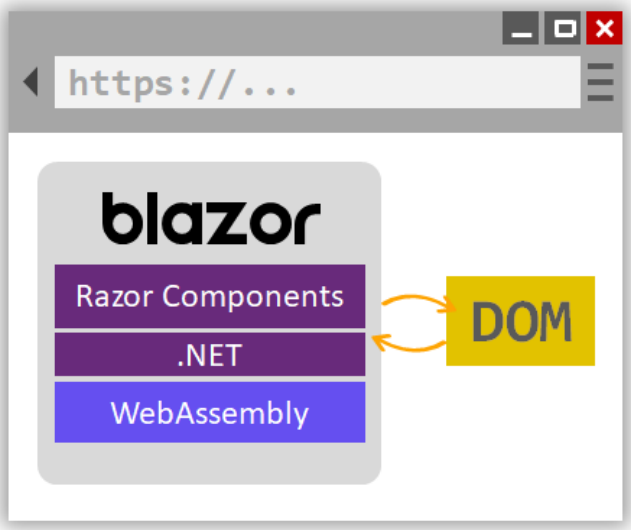
(course shown is one of my other courses, not this one)

Blazor Authentication Scenarios

Blazor WASM (WebAssembly)

Blazor Server

Blazor WASM



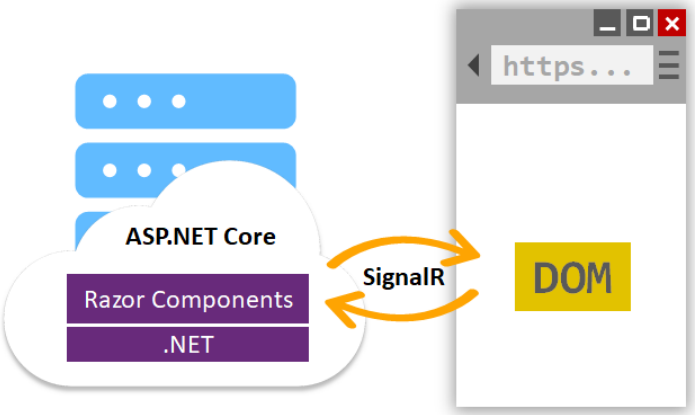
(image by Microsoft, https://bit.ly/2NKq1U8)

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

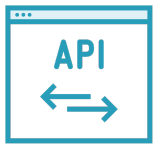
UI updates handled over SignalR connection (also used for JavaScript interop calls)

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

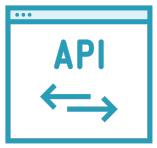


Focus is on securing the API

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

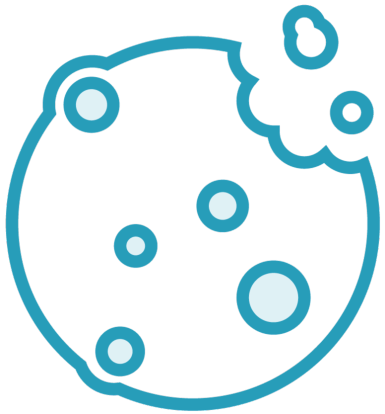
Introducing the demo application

Authentication Models, Cookies, and Tokens

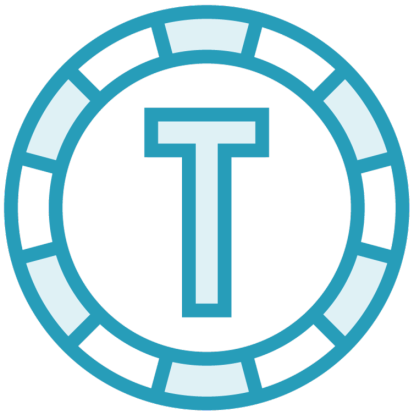
ASP.NET Core security concepts apply to Blazor Server

- 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

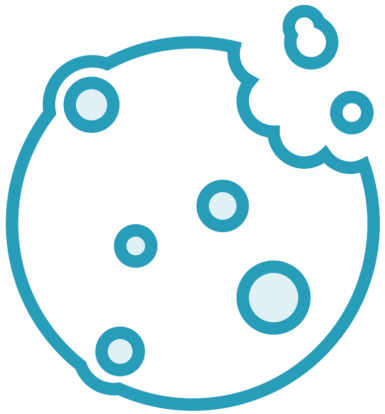
Short lifetime (limited attack window)

Can be revoked

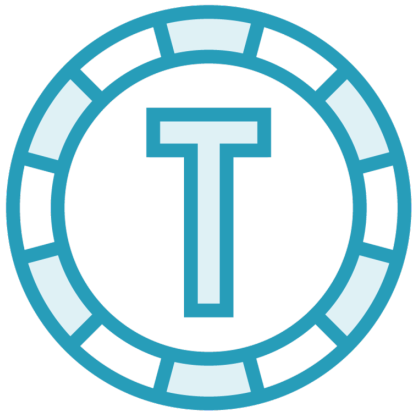
No CSRF protection for APIs needed

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

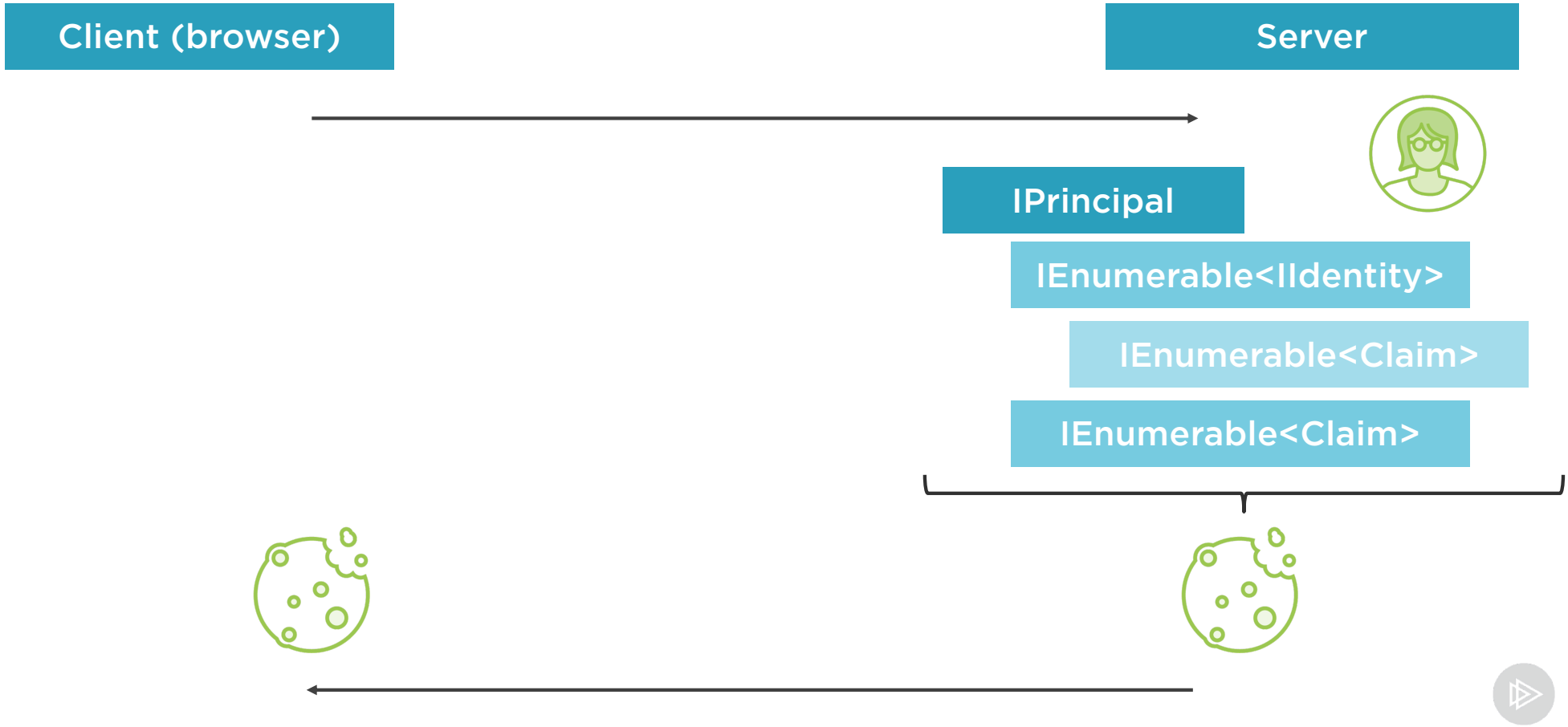
IPrincipal

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

Cookie-based Authentication with ASP.NET Core Identity

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Positioning ASP.NET Core Identity

Replacing our current approach with ASP.NET Core Identity

Potential issues

Positioning ASP.NET Core Identity

Membership system that support user interface login functionality

- Provides views, classes and methods we can use to login, logout, create and edit users and store them in a database

We're focusing on how to integrate it with Blazor

- ASP.NET Core Identity Deep Dive (Scott Brady)

Demo

Scaffolding ASP.NET Core Identity into a Blazor application

Demo

Authenticating with ASP.NET Core Identity

From Cookies to… Something Else

Our standalone approach is up & running

- Yet… cookie authentication isn't the best fit for APIs

In an enterprise landscape, you mostly don't want standalone applications

- Shared APIs, shared users, shared user management, …

Summary

ASP.NET Core Identity is a membership system that support user interface login functionality

- Select “authentication with individual user accounts” when creating a new project
- Scaffold it into existing projects

Token-based Authentication with OAuth2/OIDC

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Token-based authentication with Blazor

OAuth2 and OpenID Connect

- Logging in
- Logging out
- Protecting the API
- Gaining long-lived access

Token-based Authentication with Blazor

The identity provider (IDP) will be responsible for providing

- Proof of authentication
- Proof of authorization

to the Blazor application

Users will prove who they are at IDP level

Token-based Authentication with Blazor

Identity token represents proof of identity

Used at client level, transformed into a cookie

Access token represents consent

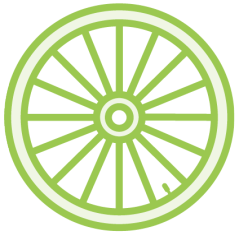
Passed from the client to the API, used at API level

Common Token Concerns

Expiration

Token signing and validation

Token format



Authentication and authorization

Securely delivering tokens to different application types

...

OAuth2

OAuth2 is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications

OAuth2 for Blazor Applications

OAuth2 defines how our Blazor application can securely achieve authorization

To that avail, our Blazor application can request an access token

OAuth2 for Blazor Applications

Not all applications are created equal

- For example: not all application types can safely store secrets

OAuth2 defines how different types of applications can securely get such a token from the IDP through different flows

OpenID Connect

OpenID Connect is a simple identity layer on top of the OAuth2 protocol

OpenID Connect for Blazor Applications

A client application can request an identity token (next to an access token)

That identity token is used to sign in to the client application

OpenID Connect for Blazor Applications

OpenID Connect is the superior protocol: it extends and supersedes OAuth2

Even if the client application only requires authorization to access an API, we should use OIDC instead of plain OAuth2

IdentityServer4

- http://docs.identityserver.io/

IdentityServer4 is an OpenID Connect and OAuth2 framework for ASP.NET Core

- Part of the .NET Foundation

What About Other Identity Providers?

Okta, Auth0, Azure AD, Azure AD B2C, … all of these are OIDC-implementing IDPs

- Integrating with them follows the same principles as integrating with IdentityServer4

Additional Information

Course: Securing ASP.NET Core 3 with OAuth2 and OpenID Connect (yours truly)

- Covers securing ASP.NET Core in depth

We're focusing on specifics related to Blazor Server applications

Demo

Inspecting IdentityServer

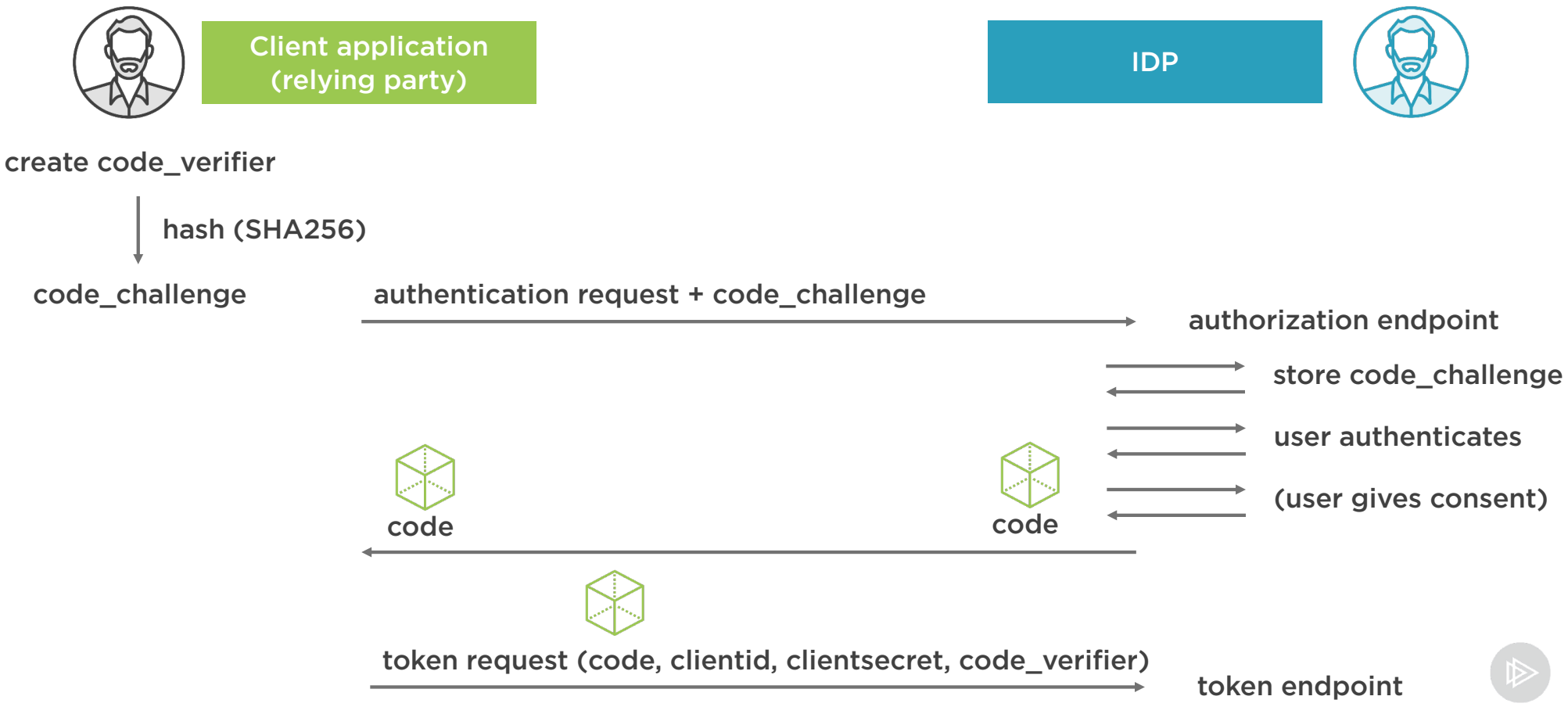
Authentication with an Identity Token

A Blazor Server application can safely store secrets

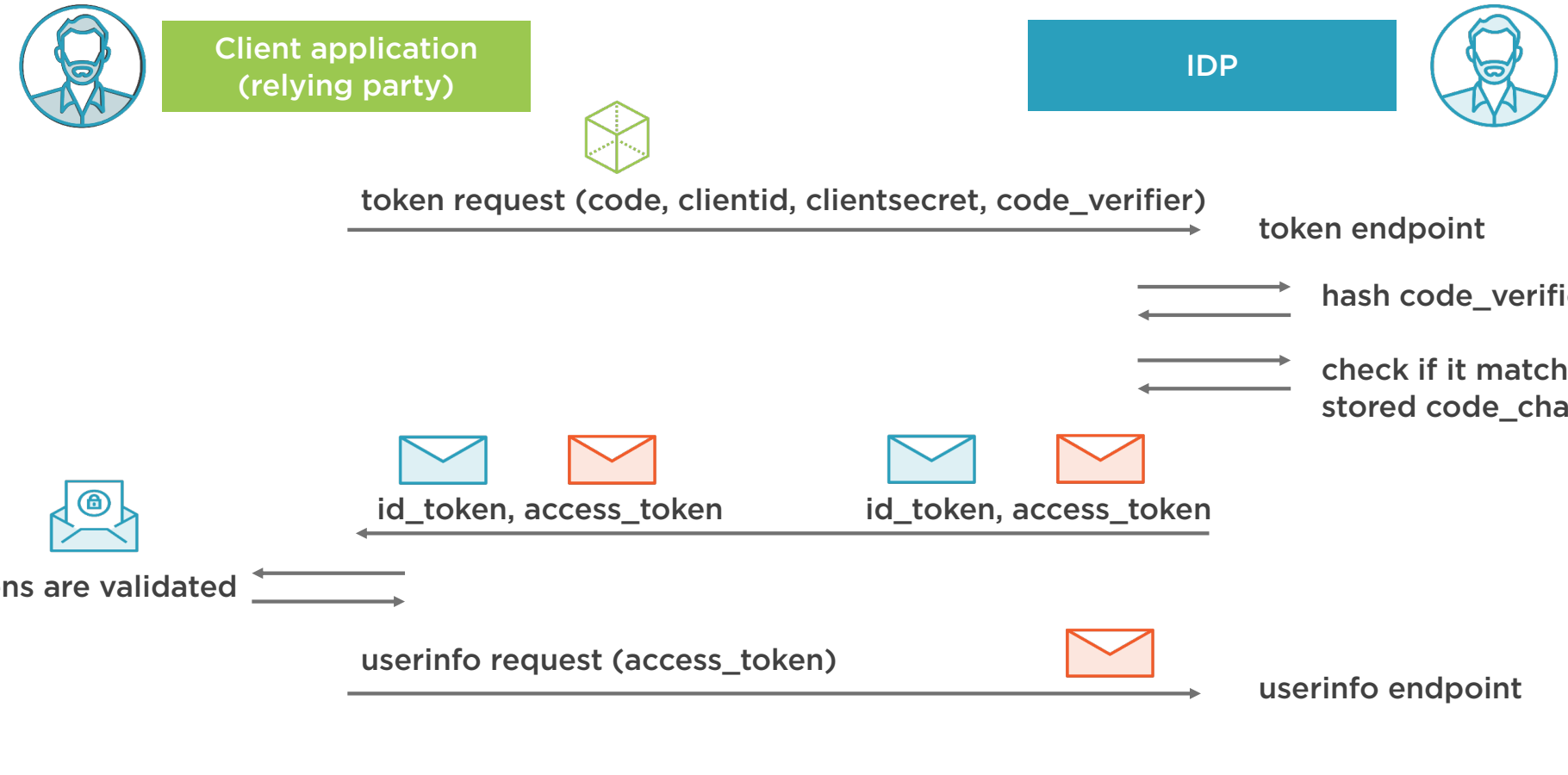
Advised flow:

- Authorization Code + PKCE
- Microsoft's OpenID Connect middleware implements the client-level parts of this flow in our Blazor app

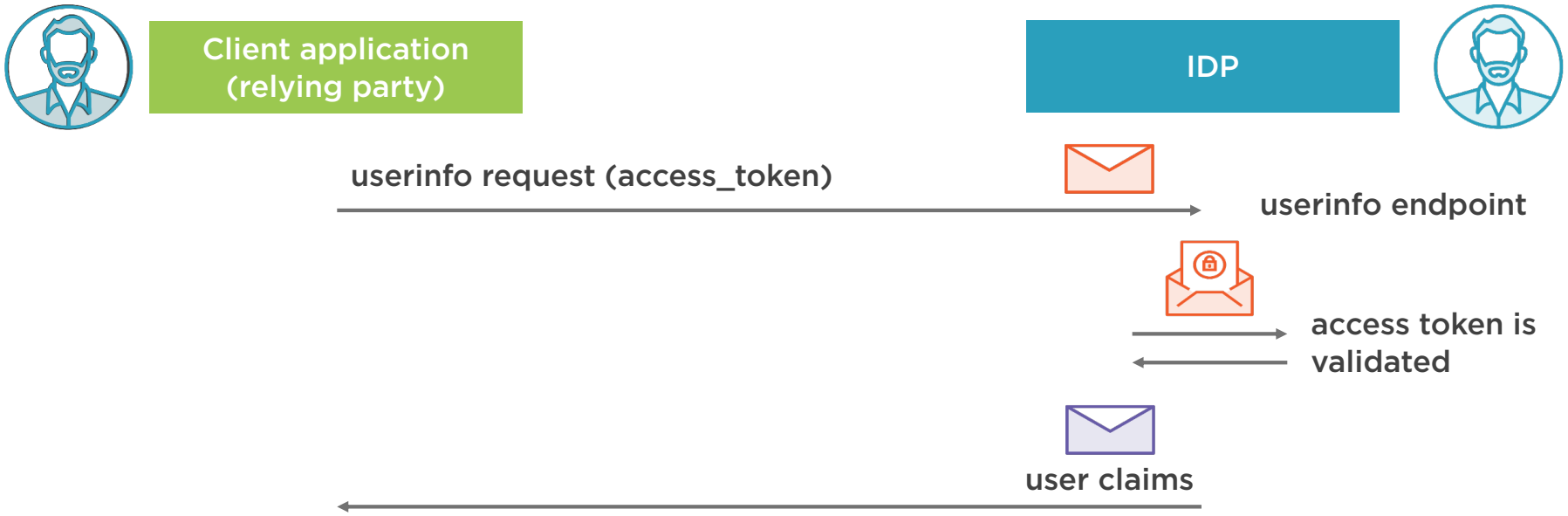
Authentication with an Identity Token



Authentication with an Identity Token



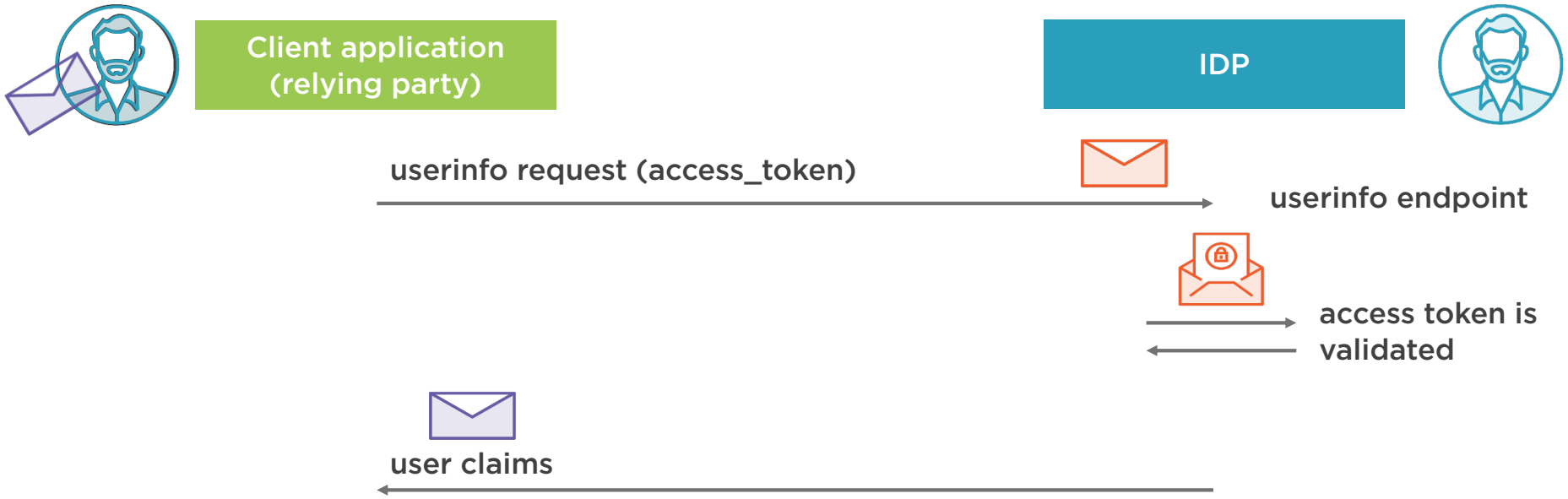
Authentication with an Identity Token



The UserInfo Endpoint

Not including the claims in the id_token keeps the token smaller, avoiding URI length restrictions

Authentication with an Identity Token



Tokens and Cookie Authentication

Our Blazor application still uses cookie authentication (as it should)

- The identity token is used to transfer proof of identity from the IDP to the client application (= our Blazor application)

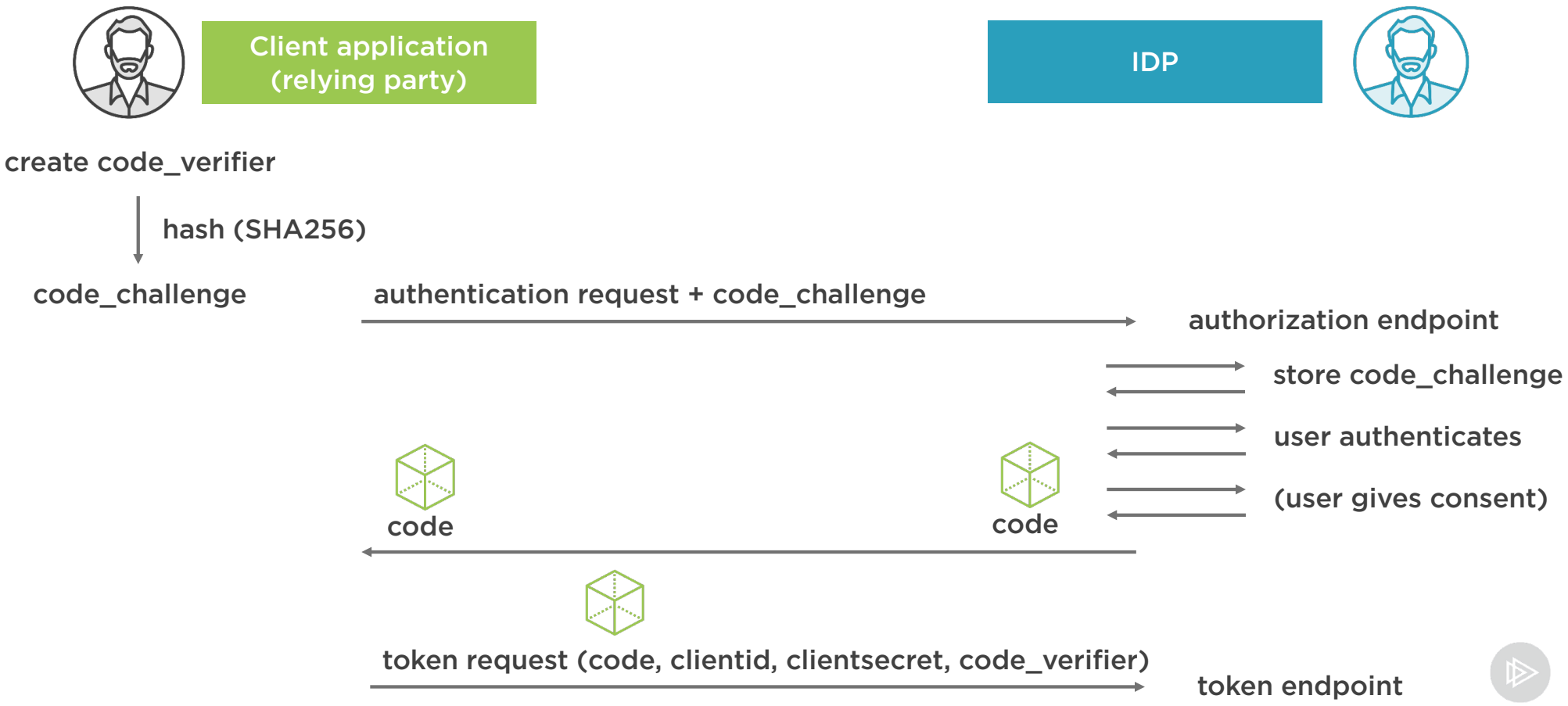
Demo

Logging in

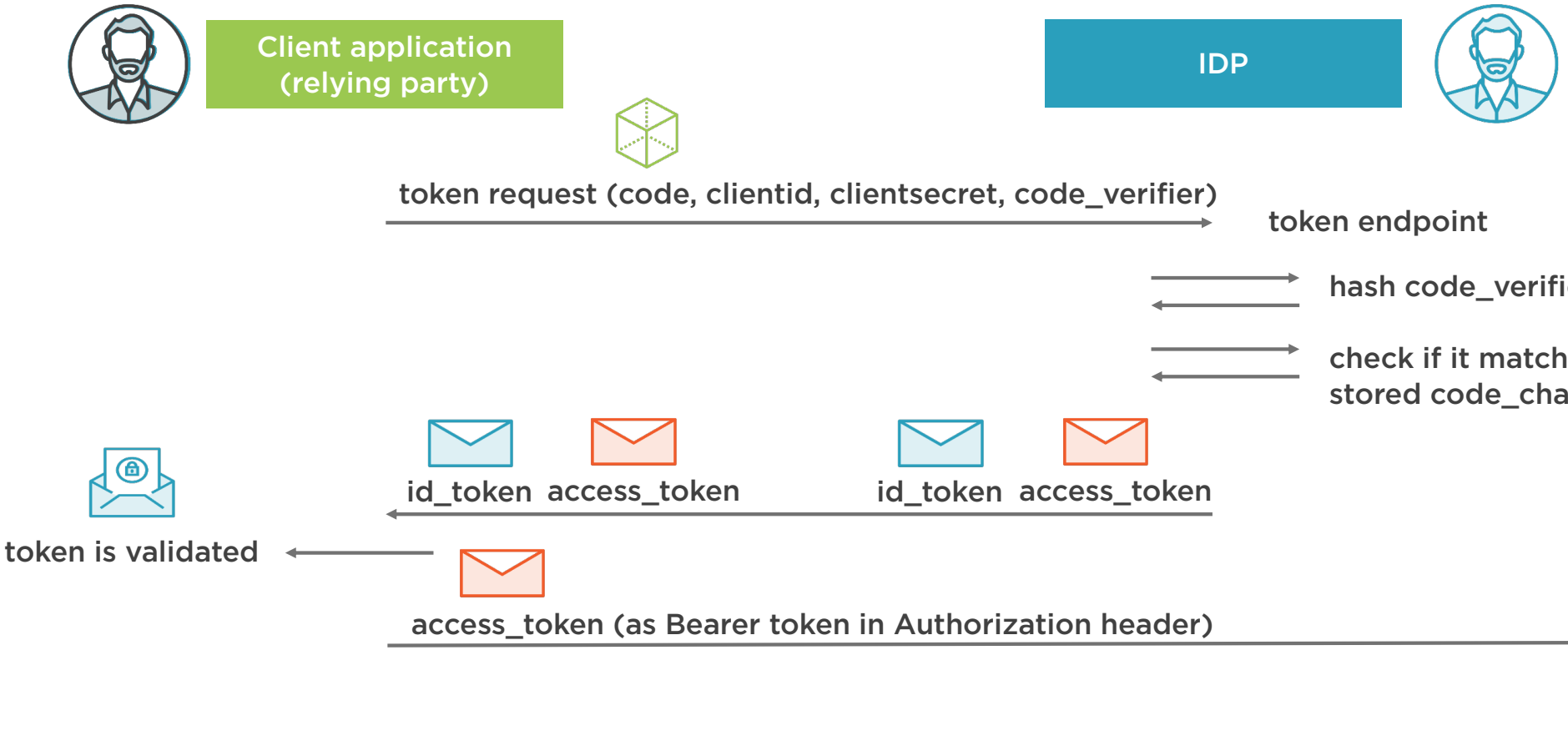
Demo

Logging out

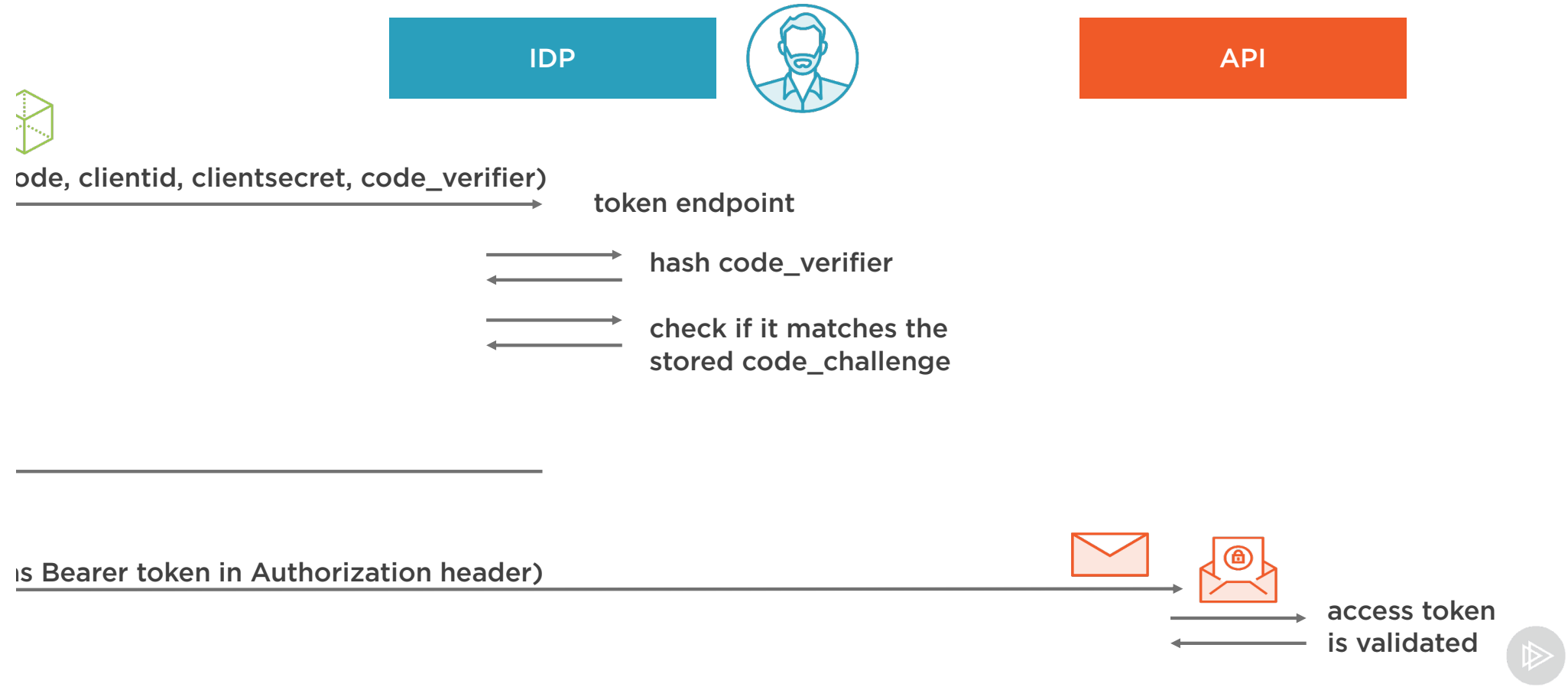
Authorization with an Access Token



Authorization with an Access Token



Authorization with an Access Token



Demo

Protecting the API

Demo

Passing an access token to our API

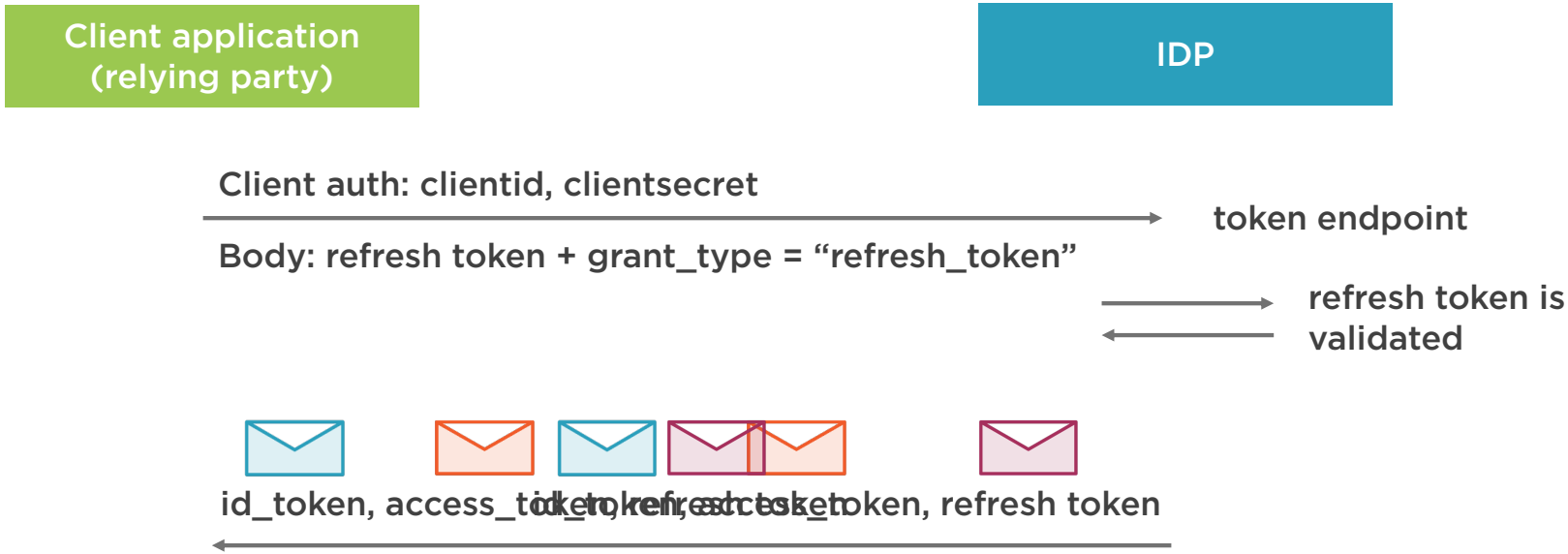
Gaining Long-lived Access with Refresh Tokens

When a token expires, the flow can be triggered again to get a new one

Confidential clients can use refresh tokens to get new tokens via the back channel

- A refresh token is a credential to get new tokens

Refresh Token Flow



Gaining Long-lived Access with Refresh Tokens

Scope: “offline_access”

- "Access to your applications and resources, even when you are offline“
- Offline in this context means the user is not logged in to the IDP

Demo

Gaining long-lived access with refresh tokens

What's Next?

Additional concerns

- Integrating ASP.NET Core Identity at level of the IDP
- Working with Windows Authentication
- Dealing with Authorization and Authorization Policies

Course: Securing Blazor Client-side Applications

Summary

Use the code flow with PKCE protection for Blazor Server applications

- Implement at client-level using Microsoft's OpenID Connect middleware
- The validated identity token is converted to a ClaimsIdentity, which is in turn stored in a cookie

Summary

The claims are returned from the UserInfo endpoint, as this keeps the identity token smaller which avoids URL length restrictions

Summary

Call ChallengeAsync on the HttpContext to challenge a scheme and start a flow

- Authentication-related calls are handled via Razor pages

Don't forget to sign out of the IDP when signing out of the Blazor application

Summary

Use IdentityServer's AccessTokenValidation middleware to validate access tokens at API level

Pass access tokens from client to API as Bearer tokens on each request to the API

Use refresh tokens for long-lived access


you are ready
to be Awesome !!!
@kevinDockx