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 and protecting the API

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

Additional Information

Course: Securing ASP.NET Core 2 with OAuth2 and OpenID Connect (yours truly)

- Covers securing ASP.NET Core in depth

We're focusing on specifics related to Blazor Server applications

Introducing the Upcoming Demo

We're going to inspect a preconfigured IdentityServer4 instance

We learned that ASP.NET Core Identity doesn't belong in the client application

- We're starting from the solution we ended up with after the "Getting Started with Authentication in Blazor" module

Demo

Inspecting IdentityServer

Authentication with an Identity Token

A Blazor Server application can safely store secrets

Advised flows:

- Authorization Code + PKCE
- Hybrid

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

The Hybrid Flow

Client application
(Blazor Server app)

IDP

authentication request

authorization endpoint

user authenticates

(user gives consent)

code id_token

code id_token

token is validated

token request (code, clientid, clientsecret)

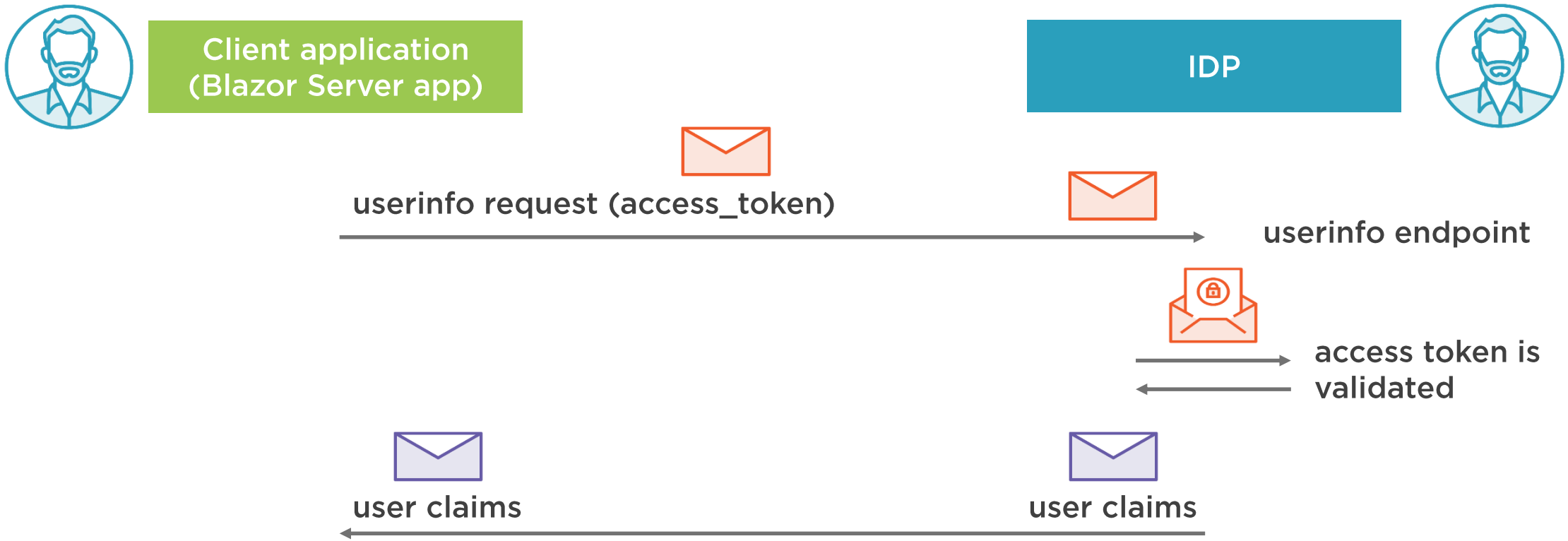
token endpoint

id_token, access_token

id_token, access_token

tokens are validated

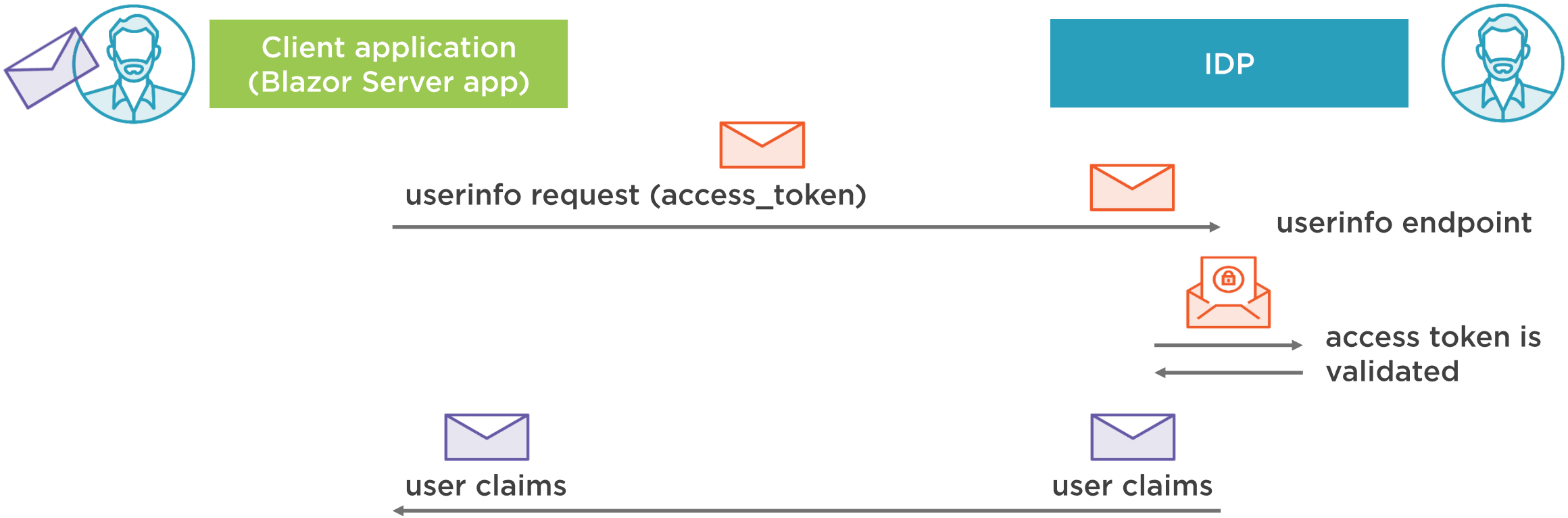
The Hybrid Flow



The UserInfo Endpoint

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

The Hybrid Flow



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)

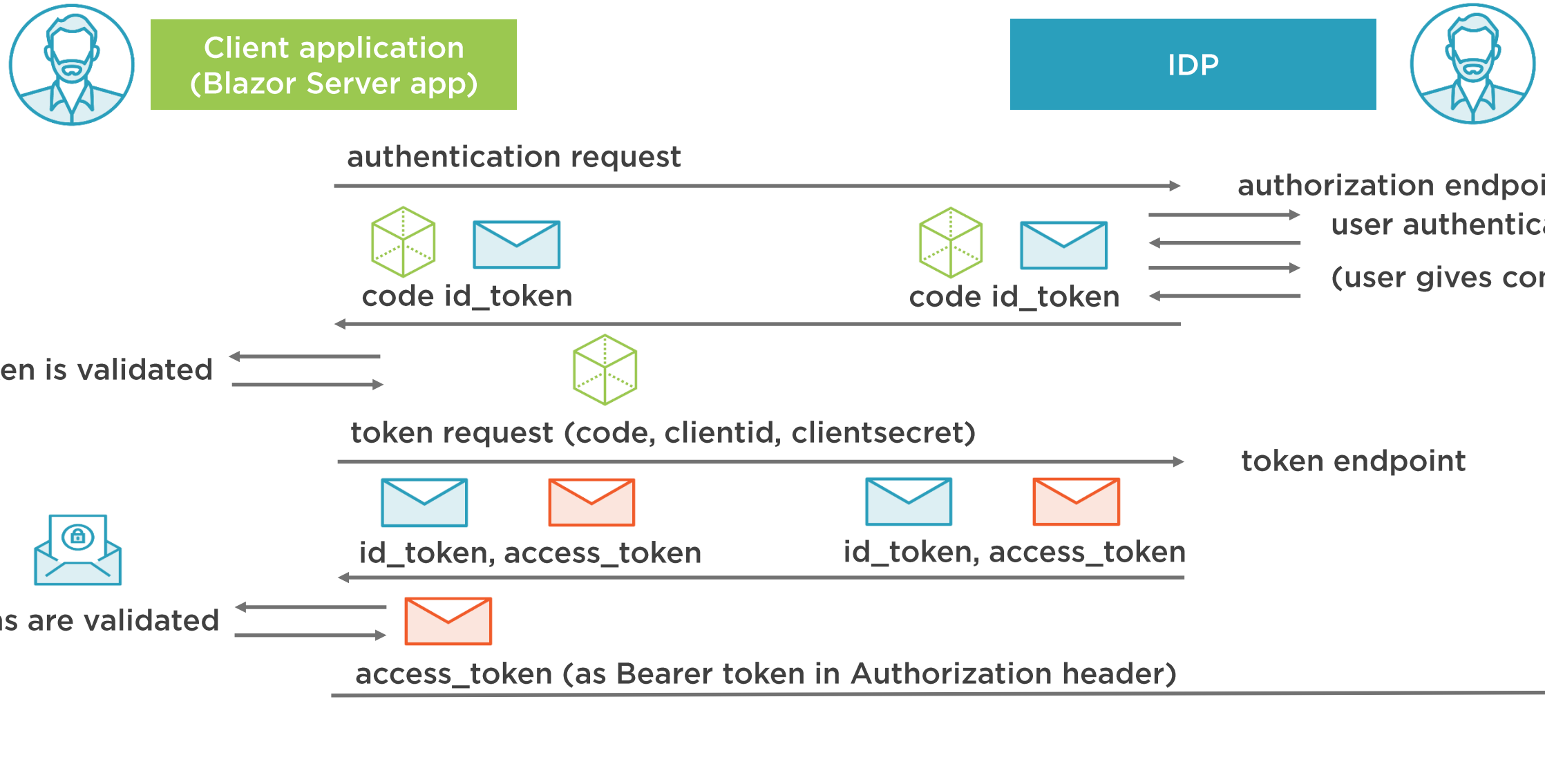
Demo

Logging in

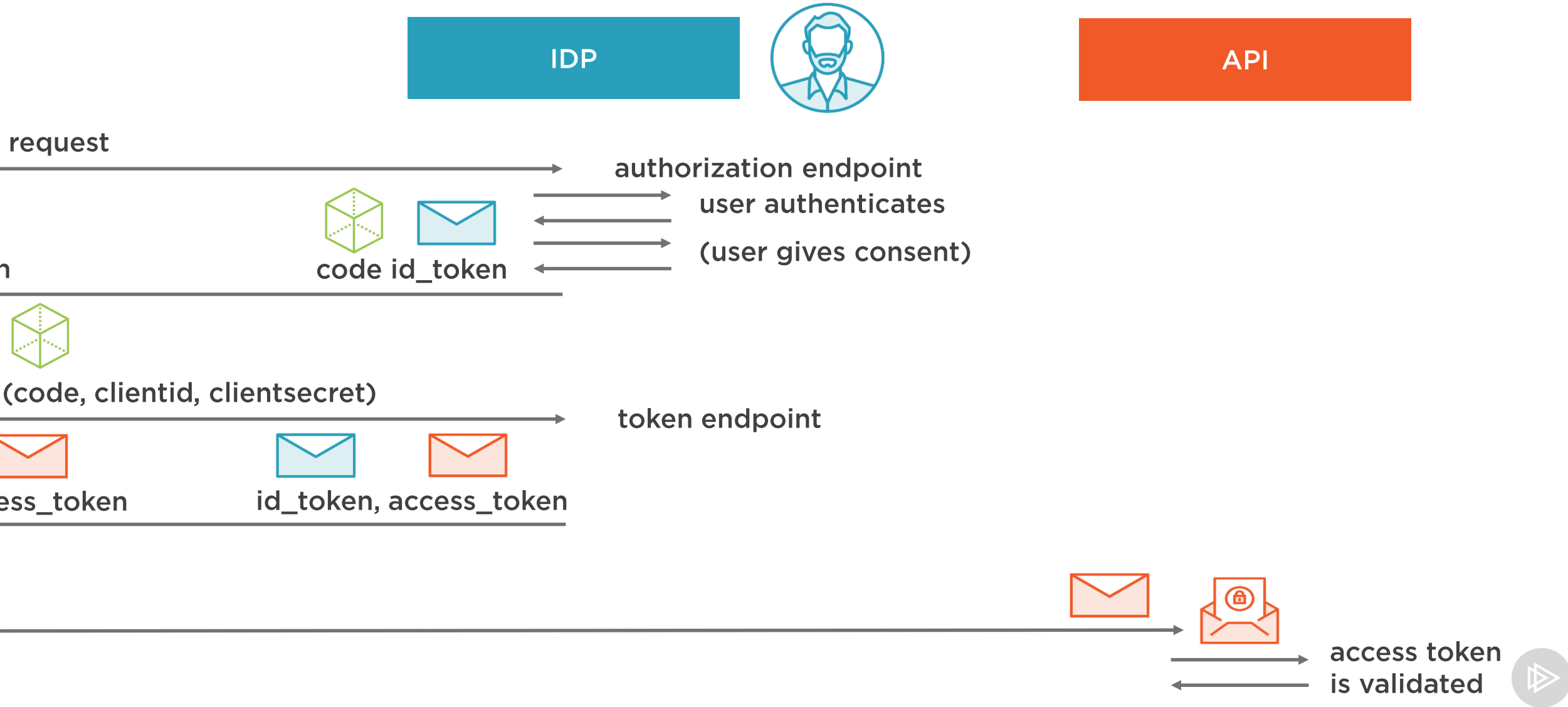
Demo

Logging out

Authorization with an Access Token



Authorization with an Access Token



Demo

Protecting the API

Demo

Passing an access token to our API

What's Next?

Additional concerns are implemented just like they are in "regular" ASP.NET Core

- Check out ASP.NET Core security-related courses to learn more

Summary

Use the Hybrid flow for Blazor Server applications

- Implement at client-level using Microsoft's OpenID Connect middleware

Summary

Response type “code id_token” allows the middleware to verify the identity token before calling the token endpoint

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 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