

User Authentication in ASP.NET Core MVC

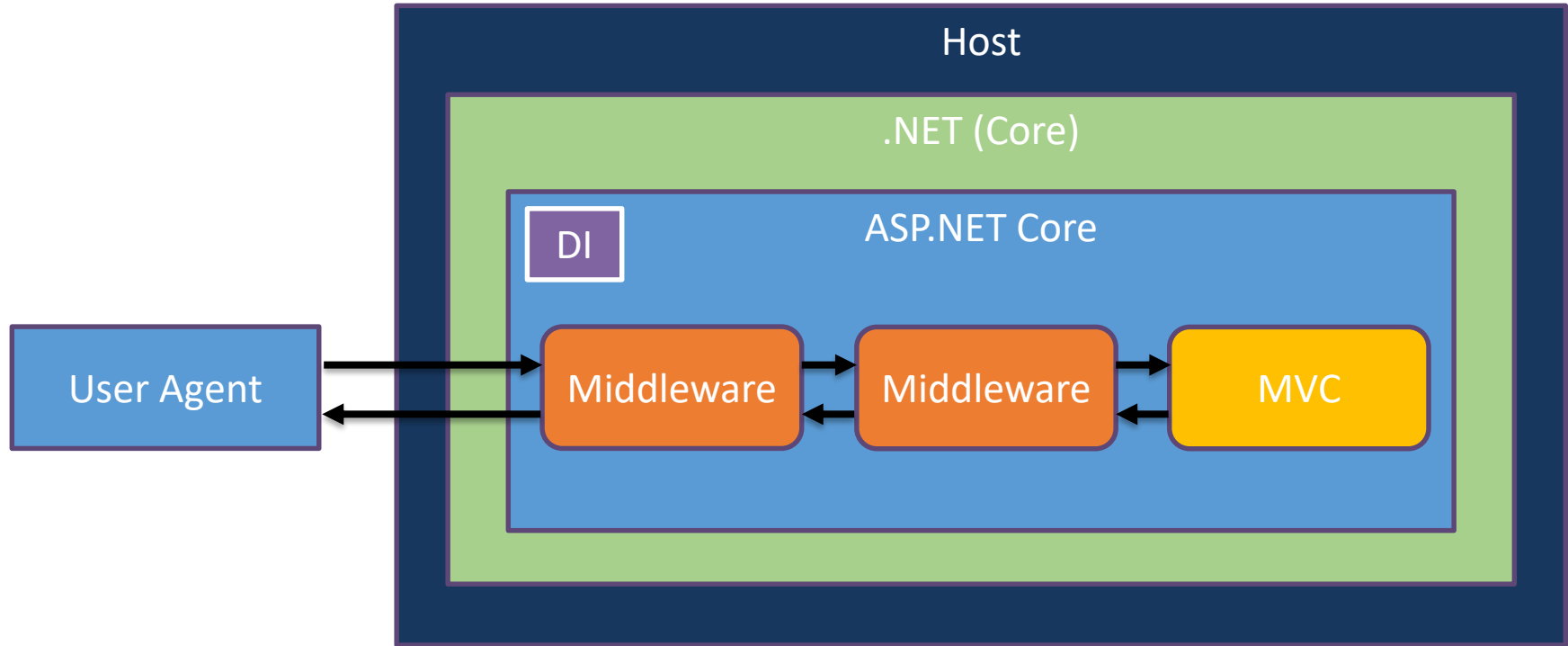
Brock Allen
@BrockLAllen
Solliance, Inc



Overview

- ASP.NET Core Security Architecture
- Cookie authentication
- Social authentication
- Centralized authentication and SSO

ASP.NET Core Architecture



Authentication in ASP.NET Core

- Combination of middleware and services and handlers in DI
 - Middleware invokes handlers for request related processing
 - Handlers can be also invoked manually
- Handlers implement specific authentication methods
 - Cookies for browser based authentication
 - Google, Facebook, and other social authentication
 - OpenId Connect for external authentication
 - JSON web token (JWT) for token-based authentication

Authentication Service

- Extension methods on *HttpContext* call the *IAuthenticationService* in DI

```
public static class AuthenticationHttpContextExtensions
{
    public static Task SignInAsync(this HttpContext context,
                                   string scheme, ClaimsPrincipal user)
    public static Task SignOutAsync(this HttpContext context, string scheme)
    public static Task ChallengeAsync(this HttpContext context, string scheme)
    public static Task ForbidAsync(this HttpContext context, string scheme)
    public static Task<AuthenticateResult> AuthenticateAsync(this HttpContext context,
                                                             string scheme)
}
```

Setting up authentication

- Services and global settings go into DI
- Authentication middleware invokes handlers

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(options =>
    {
        ...
    });
}

public void Configure(IApplicationBuilder app)
{
    app.UseAuthentication();
}
```

Setting up default schemes

- Can set defaults for schemes to use

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "...";

        options.DefaultSignInScheme = "...";
        options.DefaultSignOutScheme = "...";

        options.DefaultChallengeScheme = "...";
        options.DefaultForbidScheme = "...";

    });
}
```

Cookie authentication

- Use AddCookie extension method

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(defaultScheme: "Cookies")
        .AddCookie("Cookies", options =>
        {
            options.LoginPath = "/account/login";
            options.AccessDeniedPath = "/account/denied";

            options.Cookie.Name = "myapp";
            options.ExpireTimeSpan = TimeSpan.FromHours(8);
            options.SlidingExpiration = false;
        });
}
```


Cookies: Logging in

- SignInAsync issues cookie
 - Scheme parameter indicates which handler to use

```
var claims = new Claim[]
{
    new Claim("sub", "37734"),
    new Claim("name", "Brock Allen")
};

var ci = new ClaimsIdentity(claims, "password");
var cp = new ClaimsPrincipal(ci);

await HttpContext.SignInAsync("Cookies", cp);
```

Cookies: Logging out

- SignOutAsync removes cookie
 - Scheme parameter indicates which handler to use

```
await HttpContext.Authentication.SignOutAsync("Cookies");
```

Data Protection

- Used to protect cookies and other secrets
 - IDataProtectionProvider in DI
- Uses a key container file
 - Stored outside of application directory*
 - Uses a key ring with automatic rotation
 - Needs to be synchronized between nodes in a farm

* <https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/configuration/default-settings>

Social authentication middleware

- AddXxx adds social media authentication handlers
 - Relies upon cookie middleware after user is returned from provider

```
services.AddAuthentication("Cookies")
    .AddCookie("Cookies")
    .AddGoogle("Google", google =>
    {
        google.ClientId = "5383...";
        google.ClientSecret = "8TbR...";
    });
```

Social: Logging in

- ChallengeAsync triggers redirect
 - RedirectUri is where to send user once login is complete

```
var props = new AuthenticationProperties
{
    RedirectUri = "/Home/Secure"
};
await HttpContext.ChallengeAsync("Google", props);

// or if using MVC:

return new ChallengeResult("Google", props);
```

Controlling social logins

- Use separate “Temp” cookie for controlling social logins
 - Allows for custom logic to handle mapping and/or provisioning

```
services.AddAuthentication("Cookies")
    .AddCookie("Cookies")
    .AddCookie("Temp")
    .AddGoogle("Google", google =>
    {
        google.ClientId = "53830...";
        google.ClientSecret = "8TbR...";
        google.SignInScheme = "Temp";
    });
```

Reading “Temp” cookie

- AuthenticateAsync explicitly reads cookie

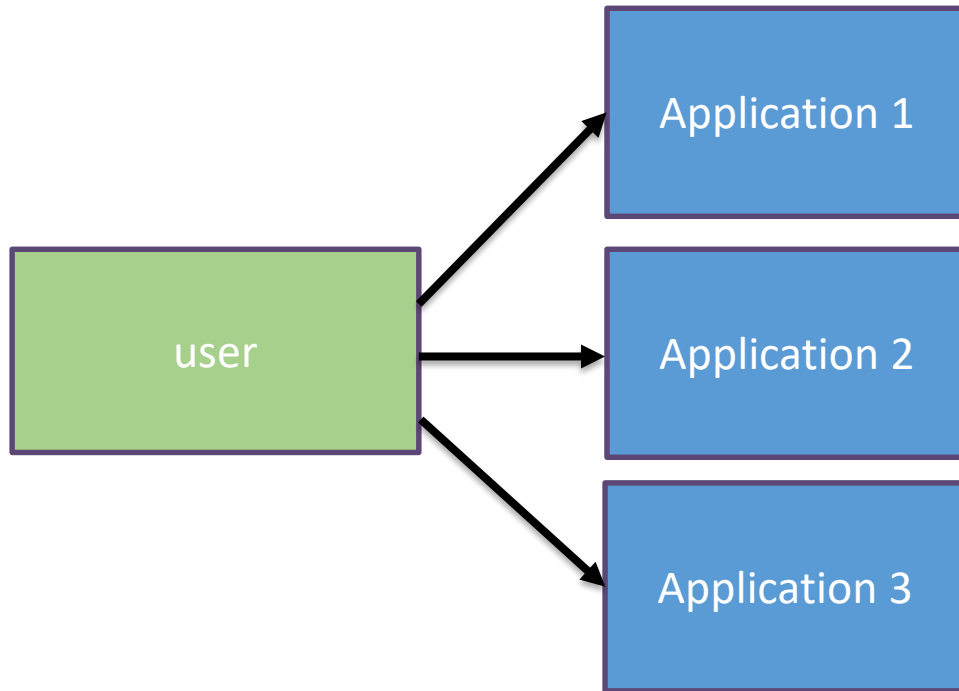
```
var tempUser = (await HttpContext.AuthenticateAsync("Temp")).Principal;
var userIdClaim = tempUser.FindFirst(ClaimTypes.NameIdentifier);
var provider = userIdClaim.Issuer;
var userId = userIdClaim.Value;

// create local account if new, or load existing local account

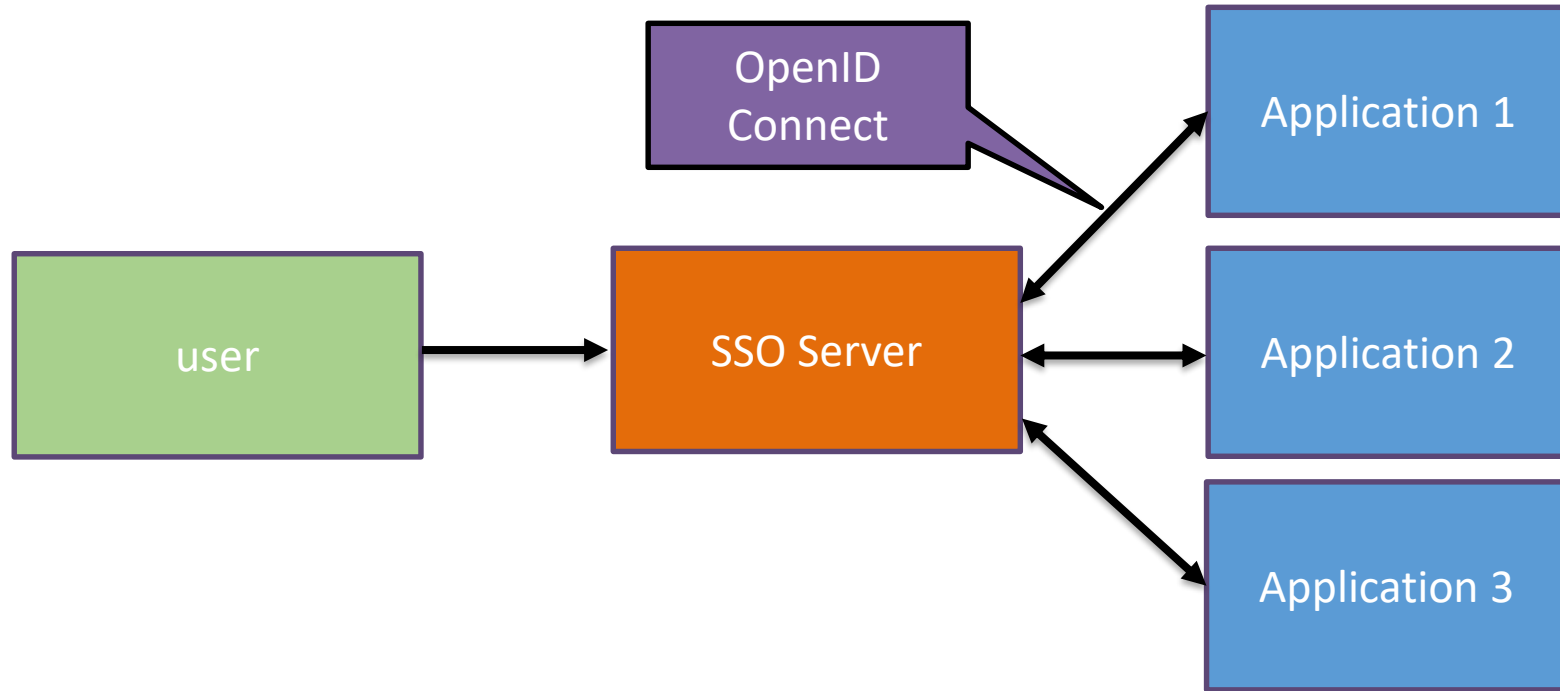
var user = new ClaimsPrincipal(...);
await HttpContext.SignInAsync("Cookies", user);
await HttpContext.SignOutAsync("Temp");
```

Application Architecture without SSO

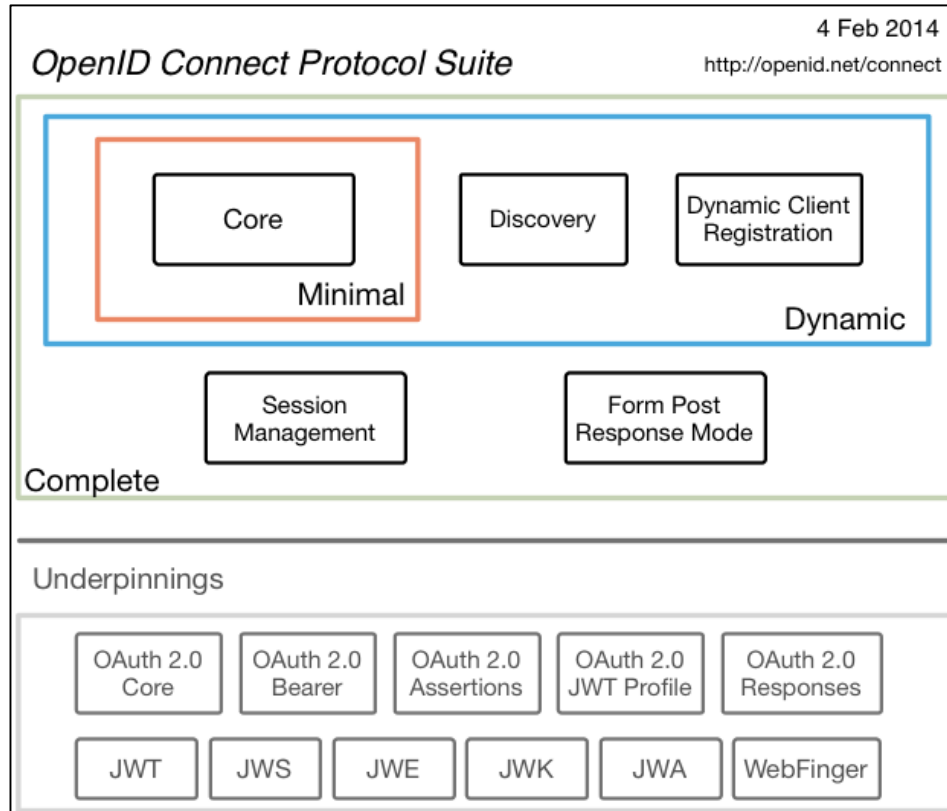
- Multiple user sign-ins
- Duplicated code for login, register, password reset, account linking, etc.



Application Architecture with SSO



<http://openid.net/connect/>



Libraries & Implementations

The image displays two overlapping browser windows showing the OpenID Connect libraries page. The left window shows the JavaScript section, and the right window shows the C# section.

JavaScript

passport-openidconnect

- OpenID Connect authentication strategy for Passport
- License: MIT
- Relying Party: Yes
- Identity Provider: No
- Target Environment: node.js

Lua

NGINX lua-resty-openidc

- NGINX Relying Party module for OpenID Connect
- License: Apache 2.0
- Relying Party: Yes
- Identity Provider: No
- Target Environment: NGINX Web Server

PHP

phpOIDC

- phpOIDC is a PHP implementation of OpenID Connect, developed by *Nomura Research Institute*. It also includes the JWT, JWS, and JWE support.
- License: Apache 2.0
- Relying Party: Yes
- Identity Provider: Yes
- Target Environment: Apache, nginx

OpenID-Connect-PHP

- A minimalist library supporting basic client authentication. Aims to make it simple enough for a developer with little knowledge of the OpenID Connect protocol to setup authentication.
- License: Apache License, Version 2.0
- Relying Party: Yes
- Identity Provider: No
- Target Environment: PHP, Apache, Nginx, etc.

oauth2-server-php

- A library for implementing an OAuth2 Server in PHP. Has been extended to support OpenID Connect.
- Identity provider functionality.
- License: MIT License
- Relying Party: No
- Identity Provider: Yes
- Target Environment: PHP

C#

JsonWebToken DelegatingHandler for ASP.NET WebAPI

- description:
- License: MIT
- Supports: JWS, JWT
- Target Environment: ASP.NET WebAPI

JSON Web Token Handler For the Microsoft .NET Framework 4.5

- This package provides an assembly containing classes which extend the .NET Framework 4.5 with the necessary logic to process the JSON Web Token (JWT) format.
- License: Microsoft Software License
- Supports: JWS, JWT
- Target Environment: .NET Framework 4.5

JWT (JSON Web Token) implementation for .NET 3.5+

- This library supports generating and decoding JSON Web Tokens.
- License: Creative Commons Public Domain 1.0
- Supports: JWS, JWT
- Target Environment: .NET Framework 3.5+

Microsoft.Owin.Security.Jwt

- Middleware that enables an application to protect and validate JSON Web Tokens.
- License: Microsoft Software License
- Supports: JWS, JWT
- Target Environment: OWIN

OWIN Authentication Middleware for Auth0 JWT Bearer Token

- License:
- Supports: JWS, JWT
- Target Environment: OWIN

Haskell

Haskell jose-jwt package

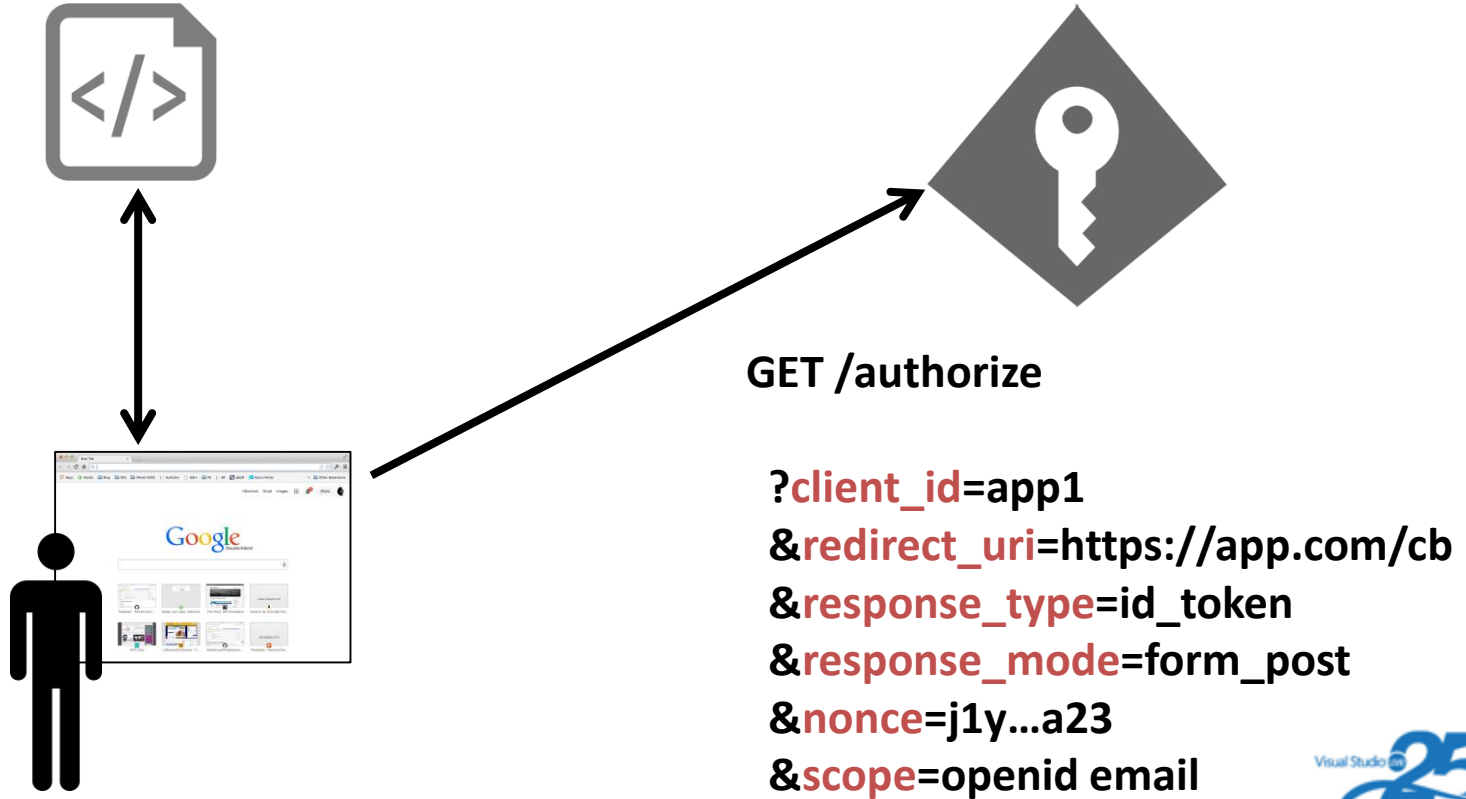
- Haskell jose-jwt package. Also see <http://hackage.haskell.org/package/jose-jwt-0.1/docs/Jose-Jwe.html>.
- License: BSD3
- Supports: JWT, JWS, JWE and JWK.
- Target Environment: Haskell

IdentityServer

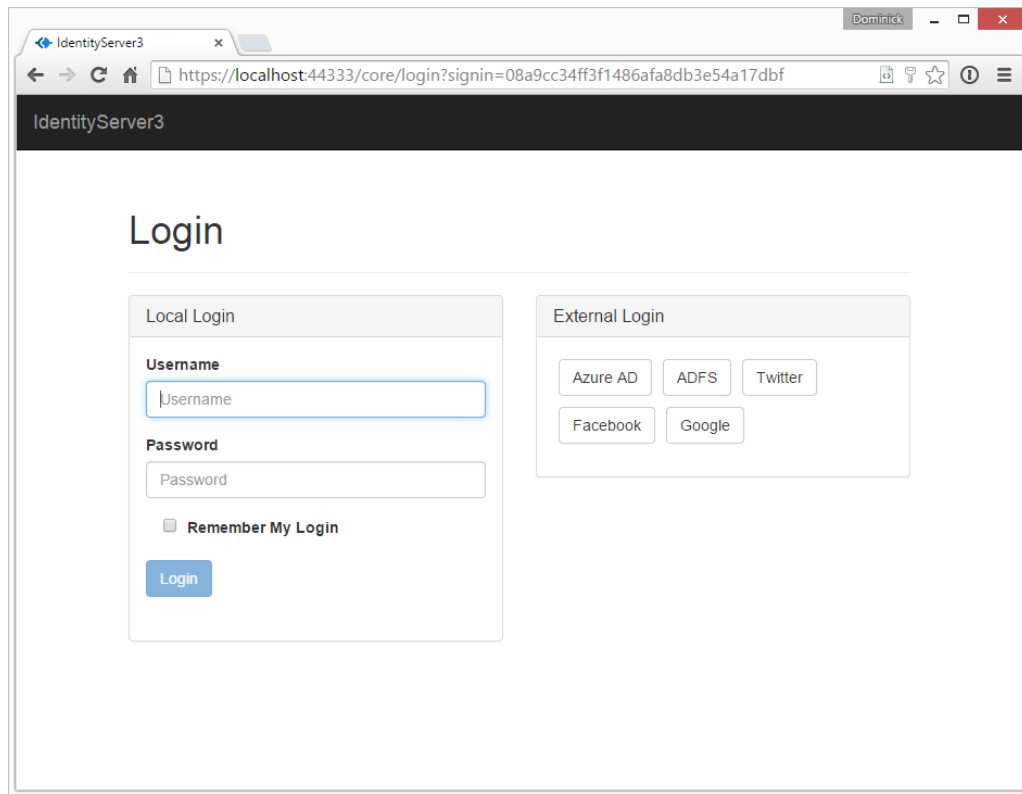
- OpenID Connect framework
 - FOSS, Apache 2.0
 - OpenID Certified
 - ASP.NET Core based
 - Part of .NET Foundation
 - <https://identityserver.io>



Authentication for web apps



Authentication



The screenshot shows a web browser window with the title 'IdentityServer3'. The address bar displays the URL 'https://localhost:44333/core/login?signin=08a9cc34ff3f1486afa8db3e54a17dbf'. The page content includes a dark header with the text 'IdentityServer3'. Below the header, the word 'Login' is centered. There are two main login sections: 'Local Login' and 'External Login'. The 'Local Login' section contains fields for 'Username' and 'Password', a 'Remember My Login' checkbox, and a 'Login' button. The 'External Login' section contains buttons for 'Azure AD', 'ADFS', 'Twitter', 'Facebook', and 'Google'.

IdentityServer3

Login

Local Login
Username


Password

☐ Remember My Login

External Login

Consent

IdentityServer3 Dominick Baier

 Implicit Client Demo is requesting your permission

Uncheck the permissions you do not wish to grant.

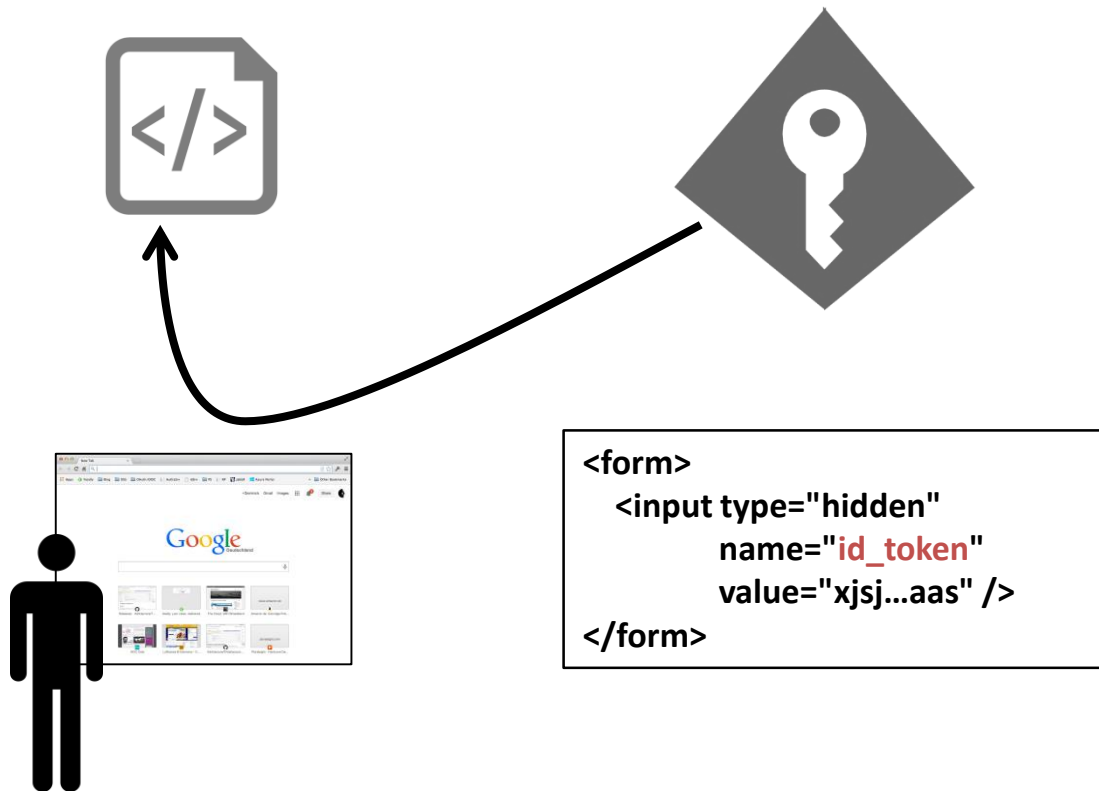
☒ Personal Information

☒ Your user identifier *(required)*

☒ Your email address ⓘ

☒ Remember My Decision

Response



Identity Token

Header

```
{  
  "typ": "JWT",  
  "alg": "RS256",  
  "kid": "mj399j..."  
}
```

Payload

```
{  
  "iss": "https://idsrv3",  
  "exp": 1340819380,  
  "aud": "app1",  
  "nonce": "j1y...a23",  
  
  "sub": "182jmm199",  
  "email": "alice@alice.com",  
  "email_verified": true,  
  "amr": [ "password" ],  
  "auth_time": 12340819300  
}
```

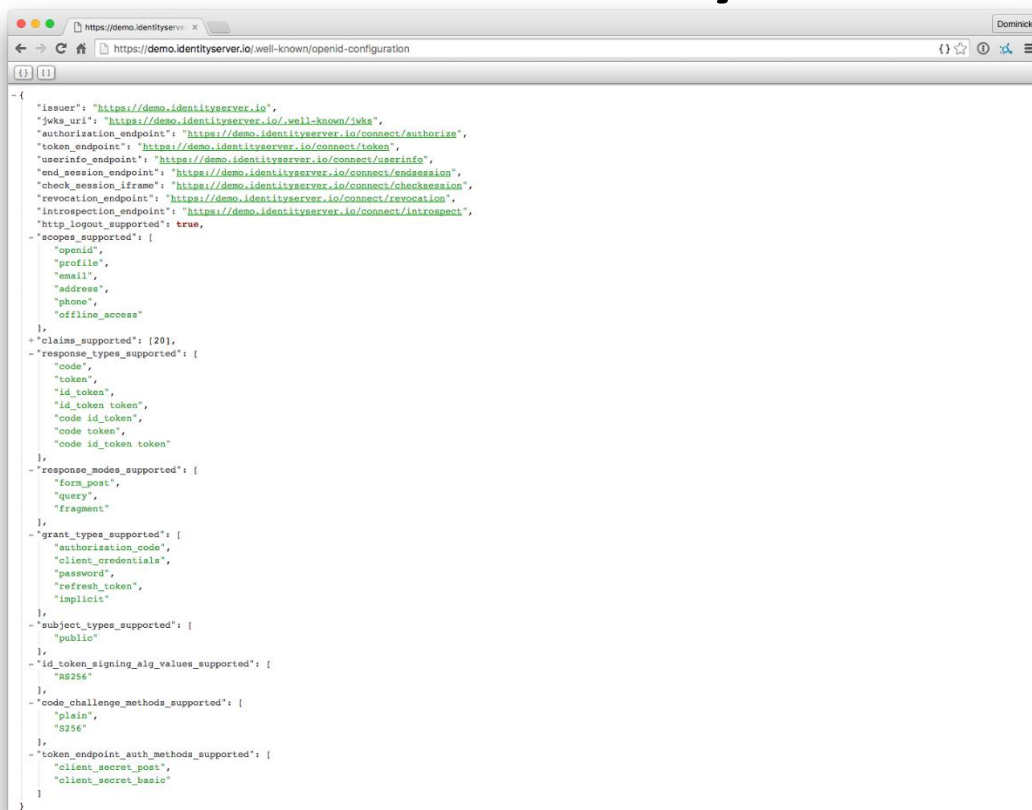
eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header

Payload

Signature

Discovery



```
{
  "issuer": "https://demo.identityserver.io",
  "jwks_uri": "https://demo.identityserver.io/.well-known/jwks",
  "authorization_endpoint": "https://demo.identityserver.io/connect/authorize",
  "token_endpoint": "https://demo.identityserver.io/connect/token",
  "userinfo_endpoint": "https://demo.identityserver.io/connect/userinfo",
  "end_session_endpoint": "https://demo.identityserver.io/connect/endsession",
  "check_session_iframe": "https://demo.identityserver.io/connect/checksession",
  "revocation_endpoint": "https://demo.identityserver.io/connect/revocation",
  "introspection_endpoint": "https://demo.identityserver.io/connect/introspect",
  "http_logout_supported": true,
  "scopes_supported": [
    "openid",
    "profile",
    "email",
    "address",
    "phone",
    "offline_access"
  ],
  "claims_supported": [20],
  "response_types_supported": [
    "code",
    "token",
    "id_token",
    "id_token token",
    "code id_token",
    "code token",
    "code id_token token"
  ],
  "response_modes_supported": [
    "form_post",
    "query",
    "fragment"
  ],
  "grant_types_supported": [
    "authorization_code",
    "client_credentials",
    "password",
    "refresh_token",
    "implicit"
  ],
  "subject_types_supported": [
    "public"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "code_challenge_methods_supported": [
    "plain",
    "S256"
  ],
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "client_secret_basic"
  ]
}
```

Handler for OpenID Connect

- Use AddOpenIdConnect extension method

```
services.AddAuthentication("Cookies")
    .AddCookie("Cookies")
    .AddOpenIdConnect("oidc", options =>
    {
        options.Authority = "https://url-to-openid-provider";
        options.ClientId = "your_app_client_id";
    });
```

Summary

- Services and middleware provide authentication
- Cookies are foundation for browser apps
- Social logins supported but requires work
- OpenID Connect protocol for SSO
- IdentityServer FOSS OIDC implementation