**HRM CODE TUTORIAL COLLECTION**

**Collection By:** R&D Team

Contents

[Identiyserver4 2](#_Toc63148972)

[The Big Picture 2](#_Toc63148973)

[Authentication 3](#_Toc63148974)

[API Access 3](#_Toc63148975)

[OpenID Connect và OAuth 2.0 – Better together 3](#_Toc63148976)

[How IdentityServer4 can help 4](#_Toc63148977)

[How to use 4](#_Toc63148978)

[Protecting an API using Client Credentials 4](#_Toc63148979)

[Interactive Applications with ASP.NET Core 6](#_Toc63148980)

[ASP.NET Core and API Access 6](#_Toc63148981)

[Topics 7](#_Toc63148982)

[Sign-in 7](#_Toc63148983)

[Cookie Authentication 7](#_Toc63148984)

[Overriding Cookie Handler Configuration 7](#_Toc63148985)

[Login Workflow 8](#_Toc63148986)

[Login Context 8](#_Toc63148987)

[Issuing a cookie and Claims 9](#_Toc63148988)

[Sign-out 10](#_Toc63148989)

[Removing the Authentication Cookie 10](#_Toc63148990)

[Notifying Clients that the User has signed-out 10](#_Toc63148991)

[Sign-out initiated by a Client Application 12](#_Toc63148992)

[Sign-out of External Identity Providers 13](#_Toc63148993)

[Federated Sign-out 13](#_Toc63148994)

[Protecting APIs 13](#_Toc63148995)

[Note 14](#_Toc63148996)

[Validating Reference Tokens 14](#_Toc63148997)

[Support both JWTs and Reference Tokens 14](#_Toc63148998)

[Endpoints 16](#_Toc63148999)

[End Session Endpoint 16](#_Toc63149000)

[Parameters 16](#_Toc63149001)

[Example 16](#_Toc63149002)

[Note 16](#_Toc63149003)

[Reference 17](#_Toc63149004)

[Client 17](#_Toc63149005)

[IdentityServer Interaction Service 17](#_Toc63149006)

[Deep investigation 17](#_Toc63149007)

[SignOutAsync and IdentityServer Cookies 17](#_Toc63149008)

[Identity Provider (IdP) Migration 17](#_Toc63149009)

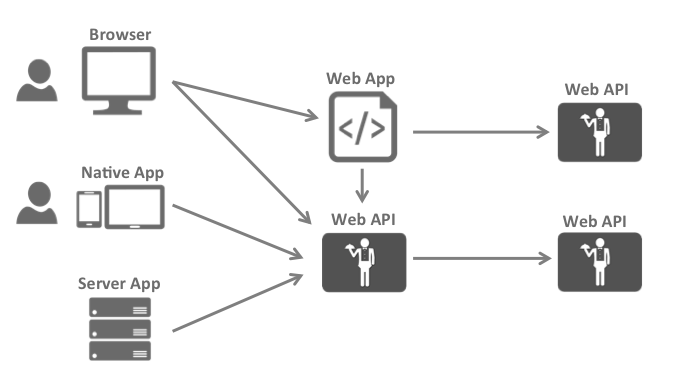
[Token validation 17](#_Toc63149010)

[Validate JWT 18](#_Toc63149011)

# Identiyserver4

## The Big Picture

Đa phần các Modern Application sẽ có kiểu



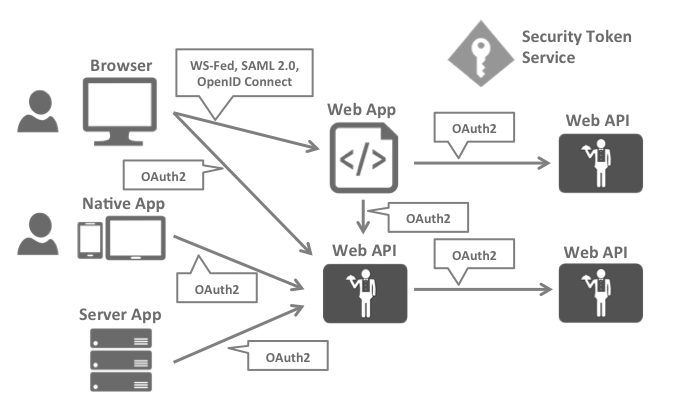
Các tương tác phổ biến nhất thường là

* Browser giao tiếp với Web Applications
* Web Applications giao tiếp với Web APIs
* Browser-Based Applications giao tiếp với Web APIs
* Native Applications giao tiếp với Web APIs
* Server-based Applications giao tiếp với Web APIs
* Web APIs giao tiếp với Web APIs

Thông thường với mỗi Layer (Front-end, Middle-tier và Back-end) sẽ phải bảo vệ Resources và triển khai xác thực và/hoặc phân quyền với cùng User

Việc Outsourcing những chức năng bảo mật cơ bản cho một Security Token Service sẽ ngăn ngừa việc Duplicate những chức năng đó qua những Applications và Endpoint.

Xây dựng lại Application hỗ trợ Security Token sẽ có mô hình như sau



Mô hình mà vấn đề Security được chia làm hai phần

### Authentication

Authentication (xác thực) cần thiết khi một Application cần biết Idenitty (định danh) của User hiện tại. Thông thường những Applications này sẽ quản lý Data thay cho người dùng, và đảm bảo việc người dùng chỉ có thể truy cập vào Data họ được cho phép. Ví dụ phổ biến nhất cho vấn đề này là Web Applications – nhưng Native và JS-based Applications hiện nay cũng cần phải được xác thực.

Các Authentication Protocols phổ biến nhất hiện nay là SAML2p, WS-Federation và OpenId Connect – SAML2p hiện nay đang phổ biến nhất và được xử dụng một cách rộng rãi

### API Access

Application có hai hướng cơ bản để có thể giao tiếp với APIs – sử dụng Application Identity, hoặc ủy quyền User’s Identity. Đôi khi cần kết hợp cả hai phương thức.

#### OAuth2

OAuth2 là một Protocol cho phép Applications yêu cầu Access Tokens từ một Security Token Service và sử dụng chúng để giao tiếp với APIs. Sự ủy quyền này làm giảm sự phức tạp của cả Client Applications và APIs bởi vì Authentication và Authorization có thể được Centralized.

### OpenID Connect và OAuth 2.0 – Better together

OpenID Connect và OAuth 2.0 khá giống nhau – trên thực tế OpenID Connect là một Extension dựa trên OAuth 2.0. Hai vấn đề bảo mật cơ bản cần quan tâm ở đây là Authentication và API Access được kết hợp thành một Protocol.

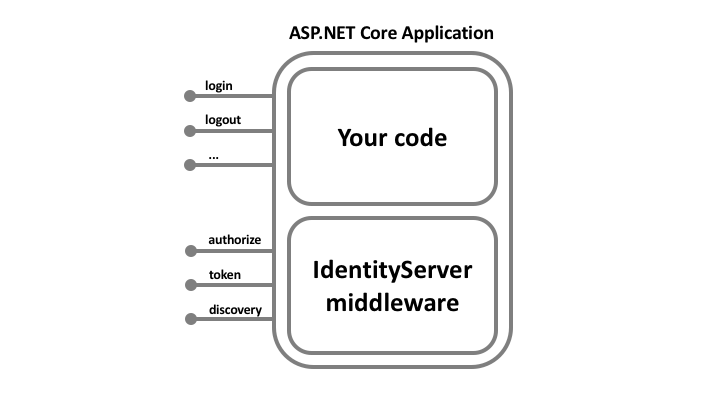
Hiện tại việc kết hợp OpenID Connect và OAuth 2.0 được coi là phương pháp tốt nhất để bảo vệ Applications trong một tương lai có thể thấy trước.

IdentityServer4 là một sự triển khai của hai Protocols này và được tối ưu cao để xử lý các vấn đề bảo mật cơ bản hiện nay trên Mobile, Native và Web Applications.

### How IdentityServer4 can help

IdentityServer là một Middleware tuân theo OpenID Connect và OAuth 2.0 bổ sung Endpoints vào một ASP.NET Core Application.

Thông thường, chúng ta build (hoặc sử dụng lại) một Application chứa các Page Login và Logout (đôi khi bao gồm cả Consent – Consent thể hiện một cách rõ ràng User’s Permission để cho phép một Application truy cập vào Resouces được bảo vệ bởi một Scope cụ thể), và IdentityServer Middleware thêm các Protocol cần thiết để Client Applications có thể giao tiếp với chúng sử dụng Standard Protocols



## How to use

### Protecting an API using Client Credentials

#### Setup ASP.NET Core Application

##### Identity Server

##### API

##### Client

#### Defining an API Scope

API là một Resouce trong hệ thống của chúng ta và cần được bảo vệ….

public static IEnumerable<ApiScope> ApiScopes =>

new List<ApiScope>

{

new ApiScope("api1", "My API")

};

#### Defining the Client

Define một Client Application sẽ truy cập vào API mới tạo

public static IEnumerable<Client> Clients =>

new List<Client>

{

new Client

{

ClientId = "client",

// no interactive user, use the clientid/secret for authentication

AllowedGrantTypes = GrantTypes.ClientCredentials,

// secret for authentication

ClientSecrets =

{

new Secret("secret".Sha256())

},

// scopes that client has access to

AllowedScopes = { "api1" }

}

};

#### Configure IdentityServer

Loading Resouce và Client Definitions diễn ra tại Startup.cs (IdentityServer)

public void ConfigureServices(IServiceCollection services)

{

var builder = services.AddIdentityServer()

.AddInMemoryApiScopes(Config.ApiScopes)

.AddInMemoryClients(Config.Clients);

builder.AddDeveloperSigningCredential();

}

### Interactive Applications with ASP.NET Core

### ASP.NET Core and API Access

# Topics

## Sign-in

Để IdentityServer có thể issue Token đại diện cho một User, User phải sign-in tại IdentityServer

### Cookie Authentication

Authentication được theo dõi với Cookie được quản lý bởi Cookie Authentication Handler của ASP.NET Core <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-5.0>

Identity Server đăng ký hai Cookie Handlers (một cho Authentication Session và một cho các Temporary External Cookies). Những thứ trên được sử dụng như Default và chúng ta có thể get Names từ IdentityServerConstants Class (DefaultCookieAuthenticationScheme và ExternalCookieAuthenticationScheme) ì chúng ta muốn reference tới chúng bằng tay.

Chỉ các Basic Settings được exposed thông qua những Cookies đó (expiration và sliding), nhưng chúng ta có thể Register Cookie Handler của chúng ta nếu cần sự quản lý cao hơn.

IdentityServer sử dụng bất kỳ Cookie Handler nào match với DefaultAuthenticationScheme như cấu hình tại AuthenticationOptions khi sử dụng AddAuthentication của ASP.NET Core

#### Note

Khi sử dụng Authentication Cookie, IdentityServer sẽ issue thêm một Cookie với tên mặc định là “idsrv.session”. Cookie này được lấy từ Main Authentication Cookie, và nó được sử dụng để kiểm tra Session Endpoint cho Browser-based JavaScript Clients tại lúc Signout. Nó được giữ đồng bộ với Authentication Cookie và sẽ được xóa khi User thực hiện Signout

### Overriding Cookie Handler Configuration

Nếu chúng ta sử dụng Cookie Authentication Handler của mình, chúng ta cần phải tự cấu hình. Việc này phải được hoàn thành bởi ConfigureServices sau khi đăng ký IdentityServer DI (AddIdentityServer). Ví dụ

services.AddIdentityServer()

.AddInMemoryClients(Clients.Get())

.AddInMemoryIdentityResources(Resources.GetIdentityResources())

.AddInMemoryApiResources(Resources.GetApiResources())

.AddDeveloperSigningCredential()

.AddTestUsers(TestUsers.Users);

services.AddAuthentication("MyCookie")

.AddCookie("MyCookie", options =>

{

options.ExpireTimeSpan = ...;

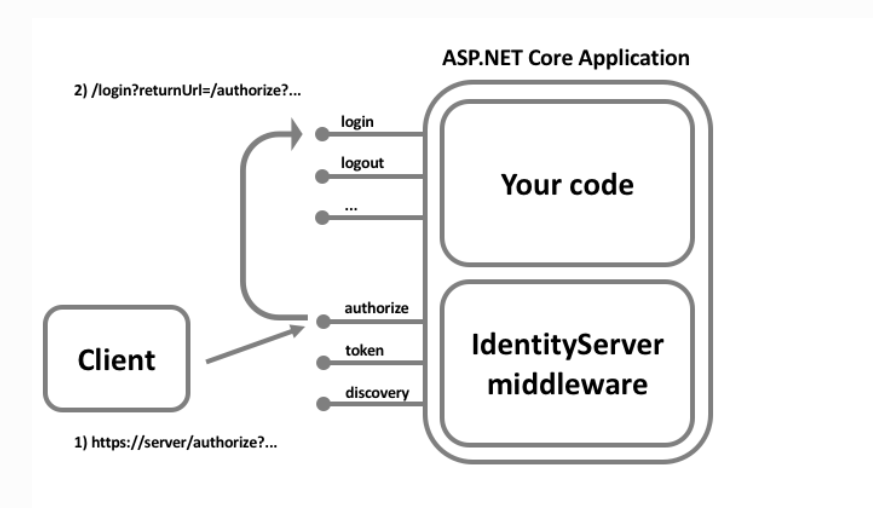
});

#### Note

IdentityServer sẽ gọi cả hai AddAuthentication và AddCookie internally với một Custom Scheme (thông qua Constant IdentityServerConstants.DefaultCookieAuthenticationScheme), vì vậy để Override chúng chúng ta phải tạo cùng một lần gọi sau AddIdentityServer.

### Login Workflow

Khi IdentityServer nhận một Request tại Authorization Endpoint và user chưa được xác thực (authenticated), User sẽ bị redirect tới trang Login đã được cấu hình. Chúng ta phải thông báo cho IdentityServer đường dẫn tới trang Login của chúng ta thông qua UserInteraction được cài đặt tại options (mặc định là /account/login). Một returnUrl Parameter được truyền vào để thông báo nơi mà User cần được redirect tới sau khi đã Login thành công.



#### Note

Phải nhận biết được open-redirect attacks thông qua returnUrl Parameter. Chúng ta nên xác thực returnUrl chỉ tới well-know location. Xem thông tin tại Interaction Service để APIs có thể xác thực returnUrl Parameter.

### Login Context

Tại Login Page chúng ta có thể cần yêu cầu thông tin về Context của Request để có thể Customize Login Page Experience (như Client, Prompt, Parameter, IdP hint, hoặc những thứ khác nữa). Điều này có thể thực hiện thông qua GetAuthorizationContextAsync API tại Interaction Service.

### Issuing a cookie and Claims

Có những Authentication-related Extension Methods tại HttpContext từ ASP.NET Core thực hiện việc issue Authentication Cookie và signin một User. Authentication Scheme được sử dụng phải match Cookie Handler chúng ta sử dụng (xem lại phía trên)

Khi chúng ta cho một User signin chúng ta phải issue ít nhất một “sub” Claim và một “name” Claim. IdentityServer cũng cung cấp một vài SignInAsync Extension Methods tại HttpContext để sử dụng một cách thuận tiện hơn.

Chúng ta cũng có thể issue một “idp” Claim (cho tên của Identity Provider) và một “arm” Claim (cho Authentication Method được sử dụng), và/hoặc một “auth\_time” Claim (cho Epoch time một User được xác thực). Nếu chúng ta không cung cấp những thứ trên, IdentityServer sẽ cugn cấp những giá trị mặc định

## Sign-out

Để đăng xuất khỏi IdentityServer chỉ đơn giản là xóa đi Authentication Cookie, nhưng vì chúng ta sử dụng một IdentityServer cho nhiều Client Applications, nên việc đăng xuất cần thực hiện trên toàn bộ các Client được liên kết

### Removing the Authentication Cookie

Để xóa Authentication Cookie, chúng ta chỉ cần sử dụng SignOutAsync Extension Method của HttpContext. Chúng ta sẽ cần thêm Scheme được sử dụng (được cung cấp bởi IdentityServerConstants.DefaultAuthenticationScheme ngoại trừ việc chúng ta thay đổi nó)

**await** HttpContext.SignOutAsync(IdentityServerConstants.DefaultCookieAuthenticationScheme);

hoặc chúng ta có thể sử dụng Extension Method được cung cấp bởi IdentityServer

**await** HttpContext.SignOutAsync();

### Notifying Clients that the User has signed-out

Như là một bộ phận của quá trình signout, chúng ta sẽ mong muốn đảm bảo việc Client Applications được thông báo rằng User đã signed out. IdentityServer hỗ trợ front-channel Specification cho Server-side Clients (e.g. MVC), back-channel Specification cho Server-side Clients (e.g. MVC) và Session Management Specification cho Browser-based Javascript Clients (e.g SPA, React, Angular, etc.).

#### Front-channel Server-side Clients

Để sign out User từ Server-side Client Applications thông qua Front-channel Spec, “logged out” Page tại IdentityServer phải render một <iframe> để thông báo Clients việc User đã signed out. Clients có mong muốn được thông báo phải có FrontChannelLogoutUri Configuration Value set. IdentityServer sẽ theo dõi Clients mà User đã signed into, và cung cấp một API gọi là GetLogoutContextAsync tại IIdentityServerInteractionService (chi tiết tại Reference). API này trả về một LogoutRequest Object với một SignOutIFrameUrl Property mà Logged out Page phải render trong một <iframe>

##### LogoutRequest Model

namespace IdentityServer4.Models

{

//

// Summary:

// Models the request from a client to sign the user out.

public class LogoutRequest

{

//

// Summary:

// Initializes a new instance of the IdentityServer4.Models.LogoutRequest class.

//

// Parameters:

// iframeUrl:

// The iframe URL.

//

// message:

// The message.

public LogoutRequest(string iframeUrl, LogoutMessage message);

//

// Summary:

// Gets or sets the client identifier.

public string ClientId { get; set; }

//

// Summary:

// Gets or sets the client name.

public string ClientName { get; set; }

//

// Summary:

// Gets or sets the post logout redirect URI.

public string PostLogoutRedirectUri { get; set; }

//

// Summary:

// Gets or sets the subject identifier for the user at logout time.

public string SubjectId { get; set; }

//

// Summary:

// Gets or sets the session identifier for the user at logout time.

public string SessionId { get; set; }

//

// Summary:

// Ids of clients known to have an authentication session for user at end session

// time

public IEnumerable<string> ClientIds { get; set; }

//

// Summary:

// Gets the entire parameter collection.

public NameValueCollection Parameters { get; }

//

// Summary:

// Gets or sets the sign out iframe URL.

//

// Value:

// The sign out iframe URL.

public string SignOutIFrameUrl { get; set; }

//

// Summary:

// Gets or sets a value indicating whether the user should be prompted for signout.

//

// Value:

// true if the signout prompt should be shown; otherwise, false.

public bool ShowSignoutPrompt { get; }

}

}

#### Back-channel Server-side Clients

Để signout User từ Server-side Client Applications thông qua Back-channel Spec chúng ta sử dụng IBackChannelLogoutService. IdentityServer sẽ tự động sử dụng Server này Logout Page của chúng ta xóa Authentication Cookie của User thông qua việc gọi HttpContext.SignOutAsync. Clients mong muốn được thông báo việc Signout phải có BackChannelLogoutUri Configuration Value Set.

#### Browser-based Javascript Clients

Dựa theo Session Management Specification, chúng ta không cần thực hiện điều gì đặc biệt tại IdentityServer để thông báo cho Clients việc User đã signed out. Thay vào đó Clients phải thực hiện việc theo dõi check\_session\_iframe, và việc này được triển khai bằng oidc-client JavaScript Library

### Sign-out initiated by a Client Application

Nếu việc sign-out được phát động bởi một Client Application, việc đầu tiên Client cần làm là điều hướng User tới End Session Endpoint (Chi tiết tại Endpoints). Quá trình diễn ra tại End Session Endpoint có thể yêu cầu một vài Temporary State để lưu giữ (e.g. Post logout redirect uri của Client) across (trải qua) redirect to the logout page (Processing at the end session endpoint might require some temporary state to be maintained across the redirect to the logout page). State này có thể được sử dụng tại Logout Page, và Id cho State được truyền thông qua một logoutId Parameter tới Logout Page.

GetLogoutContextAsync API tại Interaction Service có thể được sử dụng để load State. LogoutRequest Model Context Class là ShowSignoutPrompt chỉ ra việc Request cho sign-out đã được xác thực hay chưa, và từ đó nó có an toàn để không prompt User cho việc sign-out.

Mặc định State này được quản lý như một Protected Data Structure được truyền thông qua logoutId Value. Nếu chúng ta muốn sử dụng một vài Persistence khác giữa End Session Endpoint và Logout Page, chúng ta có thể implement IMessageStore<LogoutMessage> và sử dụng DI để register.

## Sign-out of External Identity Providers

## Federated Sign-out

Federated Sign-out là một trường hợp nơi người dùng sử dụng một External Identity Provider để log into Identity Server, và sau đó người dùng logs out External Identity Provider thông qua một Workflow mà IdentityServer không thể nhận diện. Khi người dùng sign out, Identity Server cần được thông báo để có thể sign out người dùng khỏi IdentityServer và tất cả Applications sử dụng IdentityServer.

Không phải tất cả External Identity Server hỗ trợ Federated sign-out, nhưng những Provider có hỗ trợ sẽ cung cấp một cơ chế để thông báo cho Clients rằng User đã thực hiện signed out. Thông tin này thường tới từ một Form Request tỏng một <iframe> từ External Identity Provider “logged out” Page. IdentityServer sau đó phải thông báo tới toàn bộ Clients (Xem tại Sign-out), và thông thường cũng bằng một <iframe> chứa Form Request từ bên trong <iframe> của External Identity Provider.

## Protecting APIs

IdentityServer mặc định issues Access Token với JWT Format.

Gần như tất cả Platform ngày nay đề hỗ trợ Validating với JWT Tokens

Các JWT Libraries phổ biến

* JWT bearer authentication handler – ASP.NET Core
* JWT bearer authentication middleware – Katana

Để bảo về một ASP.NET Core API chỉ cần quan tâm tới việc thêm một JWT Bearer Authentication Handler

**public** **class** **Startup**

{

**public** **void** ConfigureServices(IServiceCollection services)

{

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

*// base-address of your identityserver*

options.Authority = "https://demo.identityserver.io";

*// if you are using API resources, you can specify the name here*

options.Audience = "resource1";

*// IdentityServer emits a typ header by default, recommended extra check*

options.TokenValidationParameters.ValidTypes = **new**[] { "at+jwt" };

});

}

}

### Note

Nếu chúng ta không sử dụng Audience Claim, chúng ta có thể turn off kiểm tra Audience thông qua options.TokenvalidationParameters.ValidateAudience = false. Xem thêm tại Resource Topic để thêm thông tin về Resources, Scopes, Audiences và Authorization.

### Validating Reference Tokens

Nếu chúng ta sử dụng Reference Tokens, chúng ta cần một Authentication Handler có triển khai OAuth 2.0 token Introspection

Ví dụ

services.AddAuthentication("token")

.AddOAuth2Introspection("token", options =>

{

options.Authority = Constants.Authority;

*// this maps to the API resource name and secret*

options.ClientId = "resource1";

options.ClientSecret = "secret";

});

### Support both JWTs and Reference Tokens

Chúng ta có thể setup ASP.NET Core để dispatch tới một Handler dựa trên Incomming Token (chi tiết tại <https://leastprivilege.com/2020/07/06/flexible-access-token-validation-in-asp-net-core/>). Trong trường hợp này chúng ta có thể setup một Handler mặc định và một vài Forwarding Logic

services.AddAuthentication("token")

*// JWT tokens*

.AddJwtBearer("token", options =>

{

options.Authority = Constants.Authority;

options.Audience = "resource1";

options.TokenValidationParameters.ValidTypes = **new**[] { "at+jwt" };

*// if token does not contain a dot, it is a reference token*

options.ForwardDefaultSelector = Selector.ForwardReferenceToken("introspection");

})

*// reference tokens*

.AddOAuth2Introspection("introspection", options =>

{

options.Authority = Constants.Authority;

options.ClientId = "resource1";

options.ClientSecret = "secret";

});

# Endpoints

## End Session Endpoint

End Session Endpoint có thể được sử dụng để trigger Single Sign-out.

Để sử dụng End Session Endpoint một Client Application sẽ điều hướng Browser của người dùng tới End Session URL. Tất cả Applications mà người dùng sử dụng để logged into thông qua Browser trong User’s Session có thể tham gia quá trình sign-out

### Parameters

#### id\_token\_hint

khi người dùng được điều hướng tới Endpoint, họ sẽ được nhắc rằng có thực sự muốn thực hiện sign-out hay không. Prompt (lời nhắc) này có thể bỏ qua bởi một Client gửi đi Original id\_token nhận được từ việc Authentication. Nó được gửi đi dưới dạng một Query String Parameter gọi là id\_token\_hint

#### post\_logout\_redirect\_uri

nếu một id\_token\_hint hợp lệ được gửi, Client có thể cũng gửi một post\_logout\_redirect\_uri Parameter. Nó có thể được sử dụng để cho phép người dùng đượic điều hướng trở lại Client sau khi thực hiện sign-out. Value phải match với một trong những Pre-configured của Client là PostLogoutRedirectUris

#### state

Nếu một post\_logout\_redirect\_uri được truyền đi, Client có thể cũng sẽ gửi một state Parameter. Nó sẽ được trả ngược lại cho Client như là một Query String Parameter sau khi người dùng được điều hướng trở lại Client. Điều này thông thường được sử dụng bởi Client để thực hiện một Round-trip State across the Redirect.

### Example

GET

/connect/endsession?id\_token\_hint=eyJhbGciOiJSUzI1NiIsImtpZCI6IjdlOGFkZmMzMjU1OTEyNzI0ZDY4NWZmYmIwOThjNDEyIiwidHlwIjoiSldUIn0.eyJuYmYiOjE0OTE3NjUzMjEsImV4cCI6MTQ5MTc2NTYyMSwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo1MDAwIiwiYXVkIjoianNfb2lkYyIsIm5vbmNlIjoiYTQwNGFjN2NjYWEwNGFmNzkzNmJjYTkyNTJkYTRhODUiLCJpYXQiOjE0OTE3NjUzMjEsInNpZCI6IjI2YTYzNWVmOTQ2ZjRiZGU3ZWUzMzQ2ZjFmMWY1NTZjIiwic3ViIjoiODg0MjExMTMiLCJhdXRoX3RpbWUiOjE0OTE3NjUzMTksImlkcCI6ImxvY2FsIiwiYW1yIjpbInB3ZCJdfQ.STzOWoeVYMtZdRAeRT95cMYEmClixWkmGwVH2Yyiks9BETotbSZiSfgE5kRh72kghN78N3-RgCTUmM2edB3bZx4H5ut3wWsBnZtQ2JLfhTwJAjaLE9Ykt68ovNJySbm8hjZhHzPWKh55jzshivQvTX0GdtlbcDoEA1oNONxHkpDIcr3pRoGi6YveEAFsGOeSQwzT76aId-rAALhFPkyKnVc-uB8IHtGNSyRWLFhwVqAdS3fRNO7iIs5hYRxeFSU7a5ZuUqZ6RRi-bcDhI-djKO5uAwiyhfpbpYcaY\_TxXWoCmq8N8uAw9zqFsQUwcXymfOAi2UF3eFZt02hBu-shKA&post\_logout\_redirect\_uri=http%3A%2F%2Flocalhost%3A7017%2Findex.html

### Note

Chúng ta có thể sử dụng IdentityModel Client Library để tạo end\_session Requests .NET code. (<https://identitymodel.readthedocs.io/en/latest/client/end_session.html>)

# Reference

## Client

## IdentityServer Interaction Service

IIdentityServerInteractionService Interface có mục đích cung cấp Services có thể được sử dụng bởi User Interface để giao tiếp với IdentityServer, chủ yếu liên quan tới sự tương tác của người dùng.

# Deep investigation

## SignOutAsync and IdentityServer Cookies

Handle quá trình sign-out một cách chính xác thực sự là một việc khá phức tạp, và Behavior của ASP.NET Core Method thì không phải dễ nhận thấy. Chúng ta sẽ cần tìm hiểu một vài kịch bản phổ biến mà có thể sẽ rất khó để phân tích.

### Identity Provider (IdP) Migration

#### IdP-migration problem

System của chúng ta có một Database lưu trữ thông tin người dùng. Để có sử dụng OIDC Authentication, chúng ta dựa vào IdP Claims để định danh người dùng. Trong thực tế, Subject Id (thường gọi là “Subject”) là một Unique Identifier cho một User trong một IdP System. Chúng ta có thể sử dụng IdP Claims để nhận các thông tin về User Name và Email Address. Một Typical Model với Identity Server là phát sinh ra một locally-assigned Subject Id – Unique Identifier của chúng ta. Chúng ta không thể dựa vào IdP Subjects qua các System khác nhau (across System) bởi vì có khả năng xung đột giữa các IdPs khác nhau. Identity Server cũng đỏi hỏi Persisting Grant vào Database, và trong System chúng ta xây dựng, người dùng cũng lưu IdP Claims vào Database bởi vì chúng có thể được sử dụng để hỗ trợ tìm lỗi và các mục đích hỗ trợ khác. Tất cả những vấn đề trên gặp phải khi migrating một Account tới một IdP mới

Trong hệ thống của chúng ta, có một khía cạnh phức tạp nữa cần đối mặt là việc người dùng có thể Login với một Local Account (username và password login) thay vì sử dụng một Third-party IdP. Vì vậy chúng ta cần hỗ trợ việc Login giữa Third-party và Local Account IdPs, cũng như External-to-External,

## Token validation

Parameter nhận vào một token và một expectedScope

public async Task<TokenValidationResult> ValidateAccessTokenAsync(string token, string expectedScope = null)

{

TokenValidationResult result;

if (token.Contains("."))

{

}

else

{

}

}

1. Kiểm tra Token có phải một JWT hay là một Reference (một JWT sẽ có ký tự . để phân cách header, payload và signature, reference thì không)
2. Kiểm tra độ dài Token (jwt <= 51200, reference <= 100)
3. Validate Token dựa vào Token Type

### Validate JWT

Parameter bao gồm

* string jwt
* IEnumerable<SecurityKeyInfo> validationKeys
* bool validateLifetime = true
* string audience = null

Sử dụng một handler là một JwtSecurityHandler

Token Validation Parameter

* ValidIssuer
* IssuerSigningKeys
* ValidDateLifetime
* ValidAudience (if !IsNullOrWhiteSpace(aud))

Try

Sử dụng handler để validate token

var id = handler.ValidateToken(jwt, parameters, out var securityToken);

var jwtSecurityToken = securityToken as JwtSecurityToken;

* Kiểm tra audience tồn tại hay không
* Nếu không có audience và Access Token Type tồn tại (!IsNullOrWhiteSpace)
* Kiểm tra Access Token Type có phải “jwt” hay không
* Nếu không phải “jwt” báo Invalid Token
* Kiểm tra Access Token tồn tại Id hay không, nếu có thì log lại
* Load clientId từ jwt
* Kiểm tra việc clientId có tồn tại trong Store không
* Kiểm tra Scopes Format
* Convert Scopes string sang một List<Claim> (scopes dưới dạng string “scope1 scope2” => scopes{scope1, scope2})

Trả về một TokenValidationResult

* IsError = false
* Claims
* Client
* Jwt