اتصال به شبکه نوآوری تهران

مقدمه

مستند زیر برای رفع ابهامات و مشکلات فنی در اتصال شرکت های سرویس دهنده به شبکه نوآوری تهران تهیه شده است. لطفا با مطالعه دقیق ما را در ارائه سرویس بهتر همراهی کنید.

برخی نکات که هر اتصال گیرنده باید به آن توجه داشته باشد:

۱. تمامی اتصالات باید پروتکل REST و بر بستر HTTP یا HTTPS باشد.
۲. امکان دریافت اطلاعات از RSS و feed های مشابه وجود ندارد.
۳. اتصال گیرنده باید امکان ورود به حساب کاربری تینت را مطابق با مستند راهنمای راه اندازی در وب سایت خود پیاده سازی کند.
٤. اتصال گیرنده باید قوانین مندرج در پیوست ۱ بخش محرمانگی اطلاعات را رعایت کند.
٥. اتصال گیرنده باید قوانین مندرج در پیوست ۲ بخش توافق‌نامه سطح خدمات را رعایت کند.

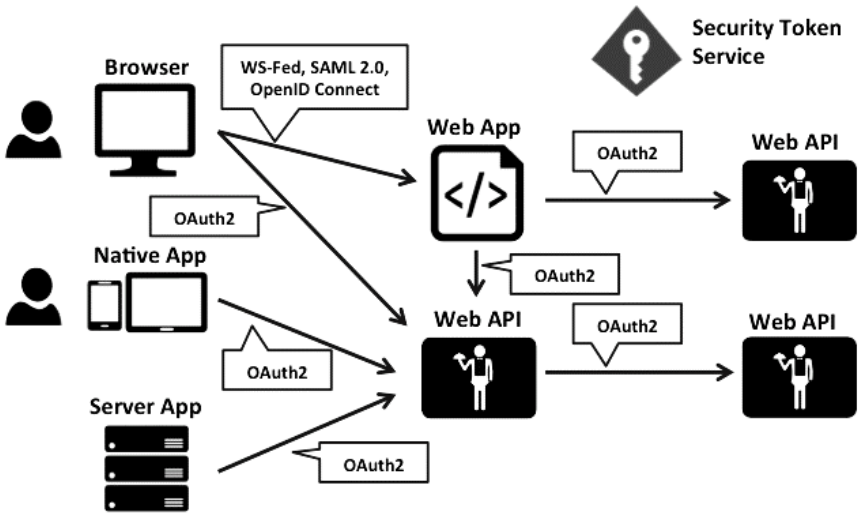
درصورتیکه که اتصال گیرنده هر یک از موارد را رعایت ننماید؛ ارائه خدمات الکترونیکی توسط شبکه نوآوری تهران و کلیه شرکت‌های وابسته به اینگونه پذیرندگان، امکانپذیر نمی‌باشد.

اتصال گیرنده گرامی، از اینکه با مطالعه این موارد از دستورالعمل ابلاغی شرکت سوران ارقام فناور پردیس اطلاع یافته و از بروز تخلف در اتصال به شبکه تینت، جلوگیری می‌نمایید و همچنین در فرهنگ‌سازی و اطلاع‌رسانی ما را یاری می‌فرمایید، سپاسگزاریم.

راهنمای استفاده از سرویس Authentication تینت

تصویر کلی

هدف کلی این سرویس ارائه تایید هویت و تامین دسترسی های لازم جهت کاربران برای استفاده از سرویس ها و برنامه ها می باشد. توسط این سرویس می توان به ارتباط های پیچیده ای مانند تصویر زیر دست یافت.



این نحوه طراحی مسائل امنیتی را به دو دسته تقسیم می کند:

Authentication:

اپلیکیشن ها جهت دسترسی به اطلاعات کاربر و تعیین دسترسی های لازم به بخش های مختلف برنامه، به احراز هویت کاربر نیازمند هستند. پروتکل های مختلفی برای احراز هویت وجود دارد که OpenIdConnect جدید ترین آنهاست.

Api Access:

سرویس ها و Api ها جهت پاسخ به درخواست ها نیازمند access token معتبر برای احراز دسترسی برنامه به آن سرویس ها هستند. در این زمینه معمولا از OAuth۲ استفاده می شود.

در این سرویس از OAuth۲ و OpenIdConnect تواما استفاده می شود.

ترمینولوژی:

Identity Server: این نامی است که به سرویس مورد بحث، داده شده و در حقیقت پروایدر OpenId Connect می باشد. این OpenId Connect و OAuth ۲,۰ را پیاده سازی کرده است.

User: شخصی است که در Identity Server ثبت نام کرده و با احراز هویت توسط Identity Server از کلاینت ها جهت دسترسی به منابع استفاده می کند.

Client: اپلیکیشن هایی که جهت احراز هویت کاربر و دریافت توکن جهت دسترسی به Api ها از سرویس Identity Server هستفاده می کند.

Resource: منابعی هستند که قصد حفاظت از آنها را توسط Identity Server داریم. منابع به دو دسته تقسیم بندی می شوند:

Identity Resources: شامل اطلاعات هویتی کاربر می باشد.

Api Resources: این منابع، داده هایی هستند که توسط سرویس ها و Api ها در اختیار کاربران و کلاینت های احراز هویت شده قرار داده می شود.

Identity Token: این توکن در حقیقت نتیجه فرایند تایید هویت می باشد. در ساده ترین حالت این توکن شامل شناسه کاربر (subject claim) می باشد. در این توکن ممکن است اطلاعات دیگری از کاربر نیز وجود داشته باشد.

Access Token: در این توکن اطلاعاتی از کلاینت و کاربر (در صورت وجود) وجود دارد. و از آن جهت دسترسی به اطلاعات Api ها استفاده می شود.

روش های استفاده برای کلاینت های دارای Authentication داخلی

در این حالت کلاینت دارای سیستم Authentication داخلی است ولی نیاز دارد که کاربران توسط این سرویس هم بتوانند وارد سیستم شوند. در این حالت کاربر در صفحه لاگین کلاینت بر روی دکمه ای مانند Login with Tinet کلیک کرده و به صفحه لاگین این سرویس هدایت می شود. پس از وارد کردن نام کاربری و رمز عبور صحیح یک Token برای کاربر ایجاد می شود و این توکن به همراه اطلاعاتی از کاربر که کلاینت درخواست کرده و کاربر آنها را تایید کرده برای کلاینت ارسال می شود. کلاینت می تواند از شناسه کاربر استفاده کرده و آن شخص را در صورت عدم وجود در سیستم Auth داخلی، ثبت کند. این بین می تواند اطلاعاتی که از کاربر نیاز دارد نیز از او دریافت کند و کاربر را ثبت نام کند. در صورت وجود کاربر نیز، او را لاگین کند و Token مناسب برای او صادر کند.

نکته ای که وجود دارد، این است که برای امنیت بیشتر و جهت تهیه access token براساس شناسه دریافت شده، می بایست به صحت اطلاعات دریافت شده اعتماد نکرده و access token دریافتی را به یک Endpoint خاص ارسال کرد و پس از بررسی صحت آن اطلاعات کاربر توسط بک اند کلاینت دریافت شود. سپس از این اطلاعات دریافتی برای لاگین یا ثبت نام استفاده کرد.

استفاده به این شکل را نمی توان برای grant type های ClientCredentials و ResourceOwnerPassword به کار برد و این حالت بیشتر برای Implicit و Hybrid به کار می رود.

روش استفاده بدون Authentication داخلی

کلاینت دارای سیستم Auth داخلی نیست و کلاینت فقط با استفاده از access token دریافت شده از سرویس تینت کاربر را احراز هویت می کند. البته به دلیل این که identity server یک سرویس عمومی است و امکان تعریف نقش و ... برای کاربر به ازای هر کلاینت وجود ندارد، خیلی قابل شخصی سازی نیست. لذا در صورت استفاده از سیستم ماکروسرویسی و استفاده از سرویس های مختلف و تعریف نقش برای کاربر یا سناریو های غیر عمومی، حالت اول توصیه می شود.

انواع اعطای دسترسی کلاینت ها

کلاینت های استفاده کننده از این سیستم می توانند به چند طریق تایید هویت و دریافت توکن را انجام دهند:

- ClientCredential:

این حالت ساده ترین نوع ارتباط می باشد که جهت ارتباطات بین اپلیکیشن و Api به صورت سروری استفاده می شود و کاربر در آن حضور ندرد. در این حالت کلاینت با ارسال Client Id و Client Secret خود به یک Endpoint مشخص، یک access token دریافت می کند. این توکن دسترسی های محدودتری دارد.

- ResourceOwnerPassword:

در این حالت کلاینت، نام کاربری و رمز عبور کاربر را از کاربر دریافت کرده و آن را همراه با Client Id خود به یک Endpoint مشخص ارسال می کن. به دلیل این که رمز عبور کاربر توسط کلاینت دریافت می شود این حالت پیشنهاد نمی گردد.

- Implicit:

این حالت برای مرورگر ها بهینه شده است و فقط برای احراز هویت کاربر (identity token) یا احراز هویت همراه با دریافت access token استفاده می شود. در این حالت توسط مرورگر انتقال داده می شود. و بنابراین امکانات پیشرفته ای مثل refresh token در آن غیر فعال است. البته در صورتی که از سیستم تایید هویت داخلی استفاده شود کماکان می توان از refresh token داخلی استفاده کرد.

- Hybrid:

این حالت کامل ترین روش ارتباط است. در این حالت Identity Token توسط مرورگر و access token توسط سرور دریافت میشود و برای کلاینت های دارای BackEnd کاربرد دارد.

تنظیمات کلاینت ها:

جهت راه اندازی کلاینت ها برای ارتباط با identity server، از کتابخانه های OpenIdConnect در آن محیط استفاده می شود. معمولا این کتابخانه ها تنظیماتی دارند که در ادامه به توضیح هر یک پرداخته می شود.

redirect url: آدرسی است که پس از لاگین، identity server به آنجا ریدایرکت می کند.

Post logout redirect url: آدرسی است که پس از لاگ اوت، identity server به آنجا ریدایرکت می کند.

Client Id: شناسه کلاینت

Client Secret: رمز عبور کلاینت

Authority: آدرس سرور Identity

Response type: نوع پاسخ بازگشتی از identity server است و می تواند یکی از حالات زیر باشد:

"Id_token": در این حالت فقط identity token بازگشت داده می شود و در این حالت access token بازگشت داده نمی شود.

"code id_token": در صورت استفاده از hybrid grant type از این حالت استفاده می شود.

"id_token token": در صورت استفاده از implicit grant type استفاده از این حالت باعث می شود access token هم همراه با identity token بازگشت داده شود.

Scope: شامل identity scope و api scope هایی می باشد که کلاینت قصد دسترسی به آنها را دارد. این موارد می بایست با space از هم جدا شوند (به عنوان مثال "openid profile email"). Identity scope های پیش فرض عبارتند از:

openid: شامل شناسه کاربر می باشد. این scope ضروری می باشد. این scope شامل claim های زیر است:

Subject, UserId

البته در حال حاضر Sub و user_id مقداری یکسان دارند. اما به دلایلی ممکن است بعدا تصمیم گرفته شود شناسه های متفاوتی برگردانده شود.

profile: شامل اطلاعاتی از کاربر می باشد. این scope شامل claims های زیر (در صورت وجود) می باشد:

Name, FamilyName, GivenName, MiddleName, NickName, PreferredUserName, Profile, Picture, WebSite, Gender, BirthDate, ZoneInfo, Locale, UpdatedAt

email: شامل ایمیل کاربر می باشد. این scope شامل claim های زیر است:

Email, EmailVerified

phone: شامل شماره تماس کاربر می باشد. این scope شامل claim های زیر است:

PhoneNumber, PhoneNumberVerified

DefaultScheme یا SignInAsAuthenticationType: نام کلیدی که برای تنظیم کوکی ها استفاده می شود و فعلا برابر "Cookies" می باشد.

DefaultChallengeScheme یا AuthenticationType: نامی است که توسط authentication middleware مورد استفاده قرار می گیرد و برابر "oidc" می باشد.

حفاظت از Api ها:

جهت حفاظت از سرویس های Api، کلاینت می بایست Access token به همراه درخواست خود به api ارسال نماید. سپس api این توکن را اعتبار سنجی کرده و در صورت معتبر بودن به کلاینت پاسخ می دهد. کلاینت جهت اعتبار سنجی توکن به کتابخانه های زیر نیازمند است:

.Net Framework:

```
id="IdentityModel" version="1.9.2"
id="IdentityServer3.AccessTokenValidation" version="2.15.1"
id="Microsoft.Owin.Security" version="3.0.1"
id="Microsoft.Owin.Security.Jwt" version="3.0.1"
id="Microsoft.Owin.Security.OAuth" version="3.0.1"
```

.Net core:

```
id=" IdentityServer4.AccessTokenValidation" version="2.6"
```

.Net Framework

```
public void Configuration(IAppBuilder appBuilder)
{
      var config = new HttpConfiguration();

      config.MapHttpAttributeRoutes();
      config.Routes.MapHttpRoute(
          name: "Default",
          routeTemplate: "api/{controller}/{id}",
          defaults: new { id = RouteParameter.Optional }
      );

      config.EnsureInitialized();

      appBuilder.UseIdentityServerBearerTokenAuthentication(new
IdentityServerBearerTokenAuthenticationOptions()
     {
         AuthenticationType = "Bearer",
```

```
        Authority = "https://auth.tinet.ir",
        RequiredScopes = new[] { "myapi" },
    });

    appBuilder.UseWebApi(config);
```

.Net Core:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvcCore()
        .AddAuthorization()
        .AddJsonFormatters();

    services.AddAuthentication("Bearer")
        .AddIdentityServerAuthentication(options =>
        {
            options.Authority = "https://auth.tinet.ir";
            options.RequireHttpsMetadata = false;

            options.ApiName = "myapi";
        });
}

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

    app.UseMvc();
}
```

در اینجا scope یا apiName به api scope ی اشاره می کنند که باید در توکن وجود داشته باشد و به عبارتی توکن برای استفاده در این api مجاز باشد. Authority هم همانگونه که قبلا اشاره شد آدرس identity server می باشد.

حال برای محافظت از api فقط کافی است از Authorize Attribute در کنترلر یا اکشن استفاده کنیم.

```
[Route("api/[controller]")]
[Authorize]
public class ValuesController : Controller
{
    ...
}
```

نمونه کد ها

۱- **SampleApi**

این پروژه، یک WebApi می باشد که توسط توکن های تولید شده توسط سرور Authentication به درخواست ها پاسخ می دهد. نکته ای که وجود دارد این است که در توکن دریافت شده باید Scope ذکر شده در این api وجود داشته باشد. برای اجرا باید Autority برابر با آدرس identity server باشد و RequiredScopes ها معتبر باشند و در identity server تعریف شده باشند.

۲- **SampleApiCore**

این پروژه نسخه dotnetCore پروژه قبل است

حفاظت از کلاینت ها:

کلاینت های ClientCredentials و ResourceOwnerPassword:

این کلاینت ها ساده ترین نوع کلاینت می باشند. چرا که ریدایرکت شدن به صفحه لاگین identity server در آنها وجود ندارد. بنابراین می توانند لزوما کلاینت های وبی نباشند و می توانند کنسولی یا ویندوزی نیز باشند. در این کلاینت ها فقط نیاز به نصب وابستگی IdentityModel می باشد. این کتابخانه تنظیمات endpoint ها را جهت دریافت توکن از آدرسی مشخص و ثابت به نام discovery endpoint می خواند.

در این دو حالت ابتدا discovery خوانده شده و کلاینت ساخته می شود. سپس توسط RequestClientCredentialsAsync و یا RequestResourceOwnerPasswordAsync توکن کلاینت یا کاربر دریافت شده و از آن برای درخواست به api استفاده می شود. نکته ای که وجود دارد دسترسی کلاینت به api scope ذکر شده باید تعریف شده باشد و هنگام درخواست توکن از identity server باید دسترسی به این اسکوپ ذکر شود.

دریافت توکن توسط اطلاعات کلاینت:

```
static async Task<TokenResponse> RequestTokenAsync()
{
    var disco = await DiscoveryClient.GetAsync(Authority);
    if (disco.IsError) throw new Exception(disco.Error);

    var client = new TokenClient(
        disco.TokenEndpoint,
        "myclient", "myclient_secret");

    return await client.RequestClientCredentialsAsync("myapi");
}
```

دریافت توکن توسط اطلاعات کاربر:

```
static async Task<TokenResponse> RequestTokenAsync()
{
    var disco = await DiscoveryClient.GetAsync(Authority);
    if (disco.IsError) throw new Exception(disco.Error);

    var client = new TokenClient(
        disco.TokenEndpoint,
        "myclient", "myclient_secret");

    Return    await    client.RequestResourceOwnerPasswordAsync("0080196632","P1@00099d",
"myapi");
}
```

نمونه کد ها

۱- **ClientCredentials**

در این پروژه کلاینت توسط clientId و clientSecret خود، توکن دریافت می کند و توسط آن از api استفاده می کند. برای اجرا باید Autority برابر با آدرس identity server باشد و clientId و clientSecret معتبر باشند و در identity server تعریف شده باشند. همچنین در grant type های کلاینت ClientCredentials وجود داشته باشد.

۲- **ClientCredentialCore**

این پروژه، نسخه dotnetCore پروژه قبل است.

۳- **ResourceOwner**

در این پروژه کلاینت با دریافت نام کاربری و رمز عبور کاربر و ارسال clientId و username و password، توکن دریافت می کند و توسط آن از api استفاده می کند. برای اجرا باید Autority برابر با آدرس identity server باشد و clientId و client secret معتبر باشد. همچنین کاربری با نام کاربری username تعریف شده و حساب کاربری وی فعال شده باشد. همچنین در grant type های کلاینت ResourceOwnerPassword وجود داشته باشد.

٤- **ResourceOwnerCore**

این پروژه، نسخه dotnetCore پروژه قبل است.

کلاینت های جاوا اسکریپتی:

در این کلاینت ها کاربر پس از ورود به اپلیکیشن جاوا اسکریپتی (انگولار و ...) به صفحه لاگین identity server هدایت شده و پس از ورود، کاربر به صفحه redirect uri باز می گردد. در backend پروژه می توان از api هایی مانند sampleApi استفاده کرد که از توکن های identity server استفاده می کند. همچنین در صورت استفاده از سرویس Auth داخلی، می توان از اطلاعات دریافتی استفاده کرد و کاربر را با سیستم Authentication داخلی سایت لاگین کرد. برای این منظور می توان از claim ی به نام user_id که در اطلاعات دریافتی موجود است و حاوی کد منحصر به فرد شناسه شخص است، استفاده کرد. نکته ای که در اینجا وجود دارد این است که در صورت استفاده از شناسه کاربر برای لاگین، به دلیل این که در این حالت بدون وارد کردن رمز عبور در اپلیکیشن مقصد کاربر لاگین می شود و هر شخصی می تواند با ارسال شناسه لاگین کند لازم است به جای ارسال شناسه به backend، access token دریافت شده به backend ارسال شده و کلاینت توسط آن توکن اطلاعات کاربر را از endpoint خاصی به نام userinfo بخواند. چون در اینجا معتبر بودن توکن بررسی می شود نگرانی بابت استفاده ناصحیح از بین می رود. پیاده سازی این سیستم البته می تواند متفاوت باشد و بستگی به طراحی کلاینت دارد.

نمونه کد ها

۱- **JavaScriptClient**

این پروژه توسط کتابخانه ای به نام oidc-client کار می کند و همانند کلاینت قبل آدرس endpoint ها را به طور اتوماتیک دریافت می کند. تنظیمات کلاینت در فایل app.js موجود است که در پیشتر توضیح داده شد. برای اجرای پروژه باید در فایل app.js موارد زیر معتبر باشند و در identity server تعریف شده باشند:

a. Authority
b. Client_id
c. Redirect_uri
d. Post_logout_redirect_uri
e. Response_type
f. Scope

همچنین در grant type های کلاینت Implicit وجود داشته باشد. در صورتی که response_type برابر "id_token" قرار داده شود، identity token و در صورتی که برابر با "id_token token" قرار داده شود علاوه بر Identity token، access token نیز دریافت می گردد.

```
var config = {
    authority: "https://auth.tinet.ir/",
    client_id: "myclient",
    redirect_uri: window.location.origin + "/callback.html",
    post_logout_redirect_uri: window.location.origin + "/index.html",
    response_type: "id_token token",
    scope: "openid profile email",
    loadUserInfo: true,
    silent_redirect_uri: window.location.origin + "/silent.html",
    automaticSilentRenew: true,
    revokeAccessTokenOnSignout: true,
    filterProtocolClaims: false
};
```

نکته ای که در اینجا وجود دارد این است که silent_redirect_uri جهت دریافت access token جدید هنگام expire شدن access token فعلی به کار می رود.

۲- AngularClient

این پروژه، نسخه انگولاری پروژه قبل است. تنظیمات کلاینت ها در فایل constants.ts موجود است. برای اجرای پروژه باید تنظیمات معتبر باشند.

```
export class Constants {
  public static stsAuthority = 'https://auth.tinet.ir/';
  public static clientId = 'ada47f94-4c65-480d-8bc9-d4a2b6f98688';
  public static redirectUri = 'http://localhost:4200/assets/signin-callback.html';
  public static silentRedirectUri = 'http://localhost:4200/assets/silent-callback.html';
  public static postLogoutRedirectUri = 'http://localhost:4200/'
  public static clientScope = 'openid profile email myapi';
  public static responseType = 'id_token token';
}
```

کلاینت های MVC:

پروژه های dotnet full framework

ابتدا وابستگی های زیر را نصب می کنیم.

```
id="IdentityModel" version="1.9.2"
id="IdentityServer3.AccessTokenValidation" version="2.15.1"
id="Microsoft.Owin.Security" version="4.0.0"
id="Microsoft.Owin.Security.Jwt" version="3.0.1"
id="Microsoft.Owin.Security.OAuth" version="4.0.0"
id="Microsoft.Owin.Security.Cookies" version="4.0.0"
id="Microsoft.Owin.Security.OpenIdConnect" version="4.0.0"
```

سپس کد زیر را به فایل startup پروژه اضافه می کنیم.

```
public void Configuration(IAppBuilder app)
{
      JwtSecurityTokenHandler.DefaultInboundClaimTypeMap = new Dictionary<string,
string>();

    app.UseCookieAuthentication(new CookieAuthenticationOptions
    {
        AuthenticationType = "Cookies"
    });

    app.UseOpenIdConnectAuthentication(new OpenIdConnectAuthenticationOptions
    {
        ClientId = "b9d1dd7b-b991-4e09-82fa-8e096b423de3",
        ClientSecret = "b9d1dd7b-b991-4e09-82fa-8e096b423de3",
        Authority = "https://auth.tinet.ir/",
        RedirectUri = "http://localhost:5002/signin-oidc",
        PostLogoutRedirectUri = "http://localhost:5002/signout-callback-oidc",
        ResponseType = "id_token token",
        Scope = "openid profile email phone",

        RequireHttpsMetadata = false,
        AuthenticationType = "oidc",
        SignInAsAuthenticationType = "Cookies",

        AuthenticationMode = AuthenticationMode.Active,
        Notifications = new OpenIdConnectAuthenticationNotifications()
        {
        },

    });
}
```

این تنظیمات قبلا بیان شده است. بخش Notifications شامل چند اکشن است که با در صورت ست کردن مقدار آنها، در شرایط مختلف اکشن ها invoke می پردد. در ادامه از این بخش استفاده خواهیم کرد. ضمنا باید به این نکته توجه داشت که آدرس RedirectUri به صورت درونی توسط کتابخانه OpenIdConnect هندل می شود و عمل پردازش اطلاعات حاصل از لاگین را انجام می دهد. اما PostLogoutRedirectUri به طور اتوماتیک هندل نمی شود و بنابراین نیاز به یک کنترلر / اکشن برای این کار است. در ساده ترین حالت پس از لاگ اوت، کاربر را به صفحه اصلی هدایت می کنیم.

```
[Route("signout-callback-oidc")]
public async Task<ActionResult> SignedOut()
{
    return RedirectToAction("Index");
}
```

پس از لاگین، نیاز به ذخیره access token و identity token برای عملیات بعدی داریم. برای این منظور زمانی که توکن ها دریافت شد، آنها را به claim های کاربر اضافه می کنیم تا در بخش های مختلف برنامه به آنها دسترسی داشته باشیم. برای این منظور به بخش Notifications، کد زیر را اضافه می کنیم.

```
SecurityTokenValidated = notification =>

   var accessToken = notification.ProtocolMessage.AccessToken;
   if (accessToken != null)
       notification.AuthenticationTicket.Identity.AddClaim(new
Claim("token.access_token", accessToken));

        var identityToken = notification.ProtocolMessage.IdToken;
        if (identityToken != null)
            notification.AuthenticationTicket.Identity.AddClaim(new
Claim("token.id_token", identityToken));

    var refreshToken = notification.ProtocolMessage.RefreshToken;
    if (refreshToken != null)
        notification.AuthenticationTicket.Identity.AddClaim(new
Claim("token.refresh_token", refreshToken));

   return Task.CompletedTask;
,
```

در صورتی که کاربر پس از redirect به صفحه لاگین، روی بدون لاگین کردن به بازگشت را انتخاب کرد، در redirect uri پیغام access denied دریافت شده و این به exception تبدیل می گردد و باعث نمایش پیغام خطا به کاربر می شود. برای جلوگیری از این موضوع، به Notifications کد زیر را اضافه می کنیم

```
AuthenticationFailed = async (notification) =>

   if (string.Equals(notification.ProtocolMessage.Error, "access_denied",
       StringComparison.OrdinalIgnoreCase))
   {
       notification.HandleResponse();
       notification.Response.Redirect("/");
   }
,
```

پس از خروج کاربر، identity server نیاز دارد تا توسط اطلاعات کلاینت، آدرس post logout را استخراج کند و کاربر را به این صفحه منتقل کند. Identity server این اطلاعات را از identity token استخراج می کند. کتابخانه جاری، این توکن را به صورت پیش فرض ارسال نمی کند و باعث می شود پس از لاگ اوت به صفحه post logout کلاینت باز نگردیم. جهت جلوگیری از این موضوع کد زیر را به Notification اضافه می کنیم.

```
RedirectToIdentityProvider = notification =>
    {
        if (notification.ProtocolMessage.RequestType == OpenIdConnectRequestType.Logout)
        {
            var idTokenHint =
notification.OwinContext.Authentication.User.FindFirst("token.id_token");
            if (idTokenHint != null)
            {
                notification.ProtocolMessage.IdTokenHint = idTokenHint.Value;
            }
        }
        return Task.FromResult(0);
    },
```

در این کد identity token که قبلا آن را ذخیره کردیم را استخراج کرده و به message اضافه می کنیم.

ویژگی UseTokenLifeTime باعث می شود مقدار عمر کوکی ذخیره شده برابر با مقدار expire time توکن دریافتی درنظر گرفته شود و پس از این زمان، در صورتی که کاربر در identity server هنوز لاگین باشد توکن جدید دریافت شده و در صورتی که لاگین نباشد با صفحه لاگین هدایت شود. در صورتی که این پارامتر false شود میزان عمر کوکی ها از CookieMiddleware دریافت خواهد شد. البته بهتر است این ویژگی فعال شود. زیرا در صورتی که عمر کوکی بیشتر از expire توکن باشد، زمانی که قصد کال کردن api داشته باشیم و توکن expire شده باشد خطای unauthorized دریافت می شود. در این پروژه عمر توکن ها به صورت پیش فرض برابر با هفت روز قرار داده شده است.

پروژه های dotnet Core

ابتدا وابستگی های زیر را نصب می کنیم.

```
id=" IdentityServer4.AccessTokenValidation" version="2.6"
```

سپس کد زیر را به فایل startup پروژه اضافه می کنیم.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();

            JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();

            services.AddAuthentication(options =>
                {
                    options.DefaultScheme = "Cookies";
                    options.DefaultChallengeScheme = "oidc";
                })
                .AddCookie("Cookies")
                .AddOpenIdConnect("oidc", options =>
                {
                    options.SignInScheme = "Cookies";

                    options.Authority = "https://auth.tinet.ir/";
                    options.RequireHttpsMetadata = false;

                    options.ClientId = "b9d1dd7b-b991-4e09-82fa-8e096b423de3";
                    options.ClientSecret = "b9d1dd7b-b991-4e09-82fa-8e096b423de3";
                    options.ResponseType = "id_token token";

                    options.SaveTokens = true;
                    options.GetClaimsFromUserInfoEndpoint = true;

                    options.Scope.Add("phone");
                    options.Scope.Add("email");
                    options.Scope.Add("offline_access");

                    options.Events = new OpenIdConnectEvents()
                    {
                    };
                });
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
```

```
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseAuthentication();

            app.UseStaticFiles();
            app.UseMvcWithDefaultRoute();
        }
```

نکته ای که وجود دارد این است که در scope مقدار opened و profile به صورت پیش فرض وجود دارند لذا بقیه scope های مورد نظر را به آن اضافه کرده ایم. همچنین SaveTokens باعث می شود توکن های دریافتی در AuthenticationProperties ذخیره شوند. بنابراین مانند کلاینت full framework نیازی به ذخیره دستی آنها در claim های کاربر نیست. آدرس های redirect uri و post logout uri هم به صورت پیش فرض برابر با "signin-oidc" و "signout-callback-oidc" می باشد و توسط کتابخانه هندل می شوند و نیازی به ذکر صریح آنها و هندل کردن دستی post logout مثل قبل نیست. البته می توان توسط پارامتر های "CallbackPath" و "SignedOutCallbackPath" این مقادیر پیش فرض را عوض کرد.

```
options.CallbackPath = "/signin-oidc";
options.SignedOutCallbackPath = "/signout-callback-oidc";
```

در این تنظیمات Events نقش Notification در قسمت قبل را دارد. بنابراین پس از لاگین در صورتی که کاربر نصد انصراف را داشت برای جلوگیری از نمایش خطای access_denied کد زیر را به events اضافه می کنیم.

```
OnRemoteFailure = context =>
{
    context.HandleResponse();
    context.Response.Redirect("/");
    return Task.CompletedTask;
},
```

نمونه کدها

۱. MvcClient

در این پروژه کاربر پس از هدایت به صفحه لاگین identity server و وارد کردن نام کاربری و رمز عبور و تایید دسترسی های لازم توکن های identity و access را دریافت می کند. در این حالت می توان از Hybrid grant type نیز استفاده کرد. در این حالت access token در سمت سرور دریافت می شود. البته در این مثال از implicit استفاده شده است. برای تغییر باید علاوه بر تغییر grant type این کلاینت در identity server، response type هم با توجه به توضیحات قبل تغییر داد.

۲. MvcClientCore

این پروژه، نسخه dotnetCore پروژه قبل است.

Endponit ها:

در سرور identity چند endpoint وجود دارد که در ادامه شرح داده می شود.

۱- discovery:

در اکثر کتابخانه های موجود برای OpenID Connect برای جلوگیری از این که کلاینت تمام آدرس های Endpoint ها و دیگر تنظیمات را به طور دستی وارد کند، به طور اتوماتیک آنها را از آدرسی مشخص و ثابت می خواند.

Request	https://<ServerAddress>/.well-known/openid-configuration
Method	GET
Headers	
Query	
Body	
Response	{ "issuer": "https://auth.tinet.ir", "jwks_uri": "https://auth.tinet.ir/.well-known/openid-configuration/jwks", "authorization_endpoint": "https://auth.tinet.ir/connect/authorize", "token_endpoint": "https://auth.tinet.ir/connect/token", "userinfo_endpoint": "https://auth.tinet.ir/connect/userinfo", "end_session_endpoint": "https://auth.tinet.ir/connect/endsession", "check_session_iframe": "https://auth.tinet.ir/connect/checksession", "revocation_endpoint": "https://auth.tinet.ir/connect/revocation", "introspection_endpoint": "https://auth.tinet.ir/connect/introspect", "frontchannel_logout_supported": true, "frontchannel_logout_session_supported": true, "backchannel_logout_supported": true, "backchannel_logout_session_supported": true, "scopes_supported": [

```
    "openid",
    "profile",
    "phone",
    "email",
    "myapi",
    "offline_access"
  ],
  "claims_supported": [
    "sub",
    "user_id",
    "name",
    "family_name",
    "given_name",
    "middle_name",
    "nickname",
    "preferred_username",
    "profile",
    "picture",
    "website",
    "gender",
    "birthdate",
    "zoneinfo",
    "locale",
    "updated_at",
    "phone_number",
    "phone_number_verified",
    "email",
    "email_verified"
  ],
  "grant_types_supported": [
    "authorization_code",
    "client_credentials",
    "refresh_token",
    "implicit",
    "password"
  ],
```

```
  "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"
  ],
  "token_endpoint_auth_methods_supported": [
    "client_secret_basic",
    "client_secret_post"
  ],
  "subject_types_supported": [
    "public"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "code_challenge_methods_supported": [
    "plain",
    "S256"
  ]
}
```

۲- Authorize:

این آدرس جهت درخواست توکن یا authorize code توسط مرورگر و تایید consent های کاربر استفاده می شود. این آدرس پس از لاگین در identity server صدا زده می شود و معمولا توسط خود identity server مورد استفاده قرار می گیرد.

Request	https://<ServerAddress>/connect/authorize	
Method	GET	
Headers		
Query	Client_Id	شناسه کلاینت
	scope	Scope های مورد نیاز که توسط space جدا شده اند.
	Response_type	نوع response که قبلا شرح داده شد.
	Redirect_uri	آدرس redirect_uri کلاینت
	state	مقداری دلخواه، سرور identity مقدار state را مجددا در پاسخ token باز می گرداند. این برای همبستگی request و response و جلوگیری از حملات CSRF/reply استفاده می شود.
	nonce	مقداری دلخواه، سرور identity مقدار state را مجددا در پاسخ identity token باز می گرداند. این برای جلوگیری از حملات reply استفاده می شود.
	Response_mode	در صورتی که برابر با form_post قرار داده شود، اطلاعات ارسالی به redirect_uri به صورت post انجام می شود و در غیر این صورت به صورت querystring ارسال می شود. در کلاینت های مختلف این متفاوت است.
	Prompt	در صورتی که برابر با login قرار داده شود، صفحه لاگین identity server نمایش داده می شود و در صورتی که برابر با none قرار داده شود و کاربر لاگین نباشد خطای login_required به redirect_uri ارسال می شود.
Body		
Response	/ redirect page Redirect to login	

اطلاعات ارسال شده می بایست urlencode شده باشند.

۳- token:

در صورت استفاده از client credentials و یا resource owner password توکن را بازگشت می دهد و در صورت استفاده از authorization key، توکن دریافتی را به redirect uri از طریق مرورگر ارسال می کند. این api به صورت درونی توسط کتابخانه های opened ممکن است استفاده شود.

Request	https://<ServerAddress>/connect/authorize
Method	POST

Headers		
Query		
Body	Client_Id	شناسه کلاینت
	Client_secret	رمز کلاینت
	Grant_type	یکی از مقادیر زیر: **authorization_code:** جهت دریافت authorization code. در این حالت پاسخ این endpoint ریدایرکت به redirect_uri می باشد. بنابراین redirect_uri باید وارد شود. این حالت توسط authorization code grant type مورد استفاده قرار می گیرد. **client_credentials:** در این حالت access token برای کلاینت صادر می شود. این حالت توسط client credentials grant type مورد استفاده قرار می گیرد. **password:** در این حالت access token برای کاربر صادر می شود. بنابراین usename و password برای کاربر الزامی است و باید وارد شود. این حالت توسط resource owner password grant type مورد استفاده قرار می گیرد. **refresh_token:** در این حالت access token جدید صادر می شود. وارد کردن refresh_token در این حالت الزامی است. این حالت توسط hybrid grant type مورد استفاده قرار می گیرد.
	Redirect_uri	آدرس redirect_uri که قبلا بحث شد. (در صورتی که grant_type برابر با authorization_code قرار داده شود مورد نیاز است)
	Code	مقدار authorization code (در صورتی که grant_type برابر با authorization_code قرار داده شود مورد نیاز است)
	scope	Scope های مورد نیاز که توسط space جدا شده اند.
	Code_verifier	کلید PKCE
	Username	نام کاربری (در صورتی که grant_type برابر با password قرار داده شود مورد نیاز است)
	password	رمز عبور (در صورتی که grant_type برابر با password قرار داده شود مورد نیاز است)

	Refresh_token	مقدار refresh token (در صورتی که grant_type برابر با refresh_token قرار داده شود مورد نیاز است)
Response	Redirect to redirect_uri / token response	

۴- userinfo:

این endpoint شاید بیشترین استفاده را در بین کلاینت ها داشته باشد. این endpoint با دریافت access token اطلاعات کاربر را بازگشت می دهد. در صورت استفاده از سیستم auth داخلی می توان هنگام دریافت access token، آن را به این آدرس ارسال کرد و اطلاعات کاربر را برای ثبت نام یا لاگین دریافت کرد. اطلاعاتی از این طریق دریافت می شود که scope آن در token وجود داشته باشد.

Request	https://<ServerAddress>/connect/userinfo
Method	GET
Headers	Bearer <AccessToken>
Query	
Body	

Response	{
	"sub": "c1dc3de5-a34d-40a6-886b-6fe54a9f9729",
	"name": "firstname lastname",
	"given_name": "firstname",
	"family_name": "lastname",
	"user_id": "c1dc3de5-a34d-40a6-886b-6fe54a9f9729",
	"preferred_username": "0080196632",
	"email": "email@example.ir",
	"email_verified": false,
	"phone_number": "09123456789",
	"phone_number_verified": true
	}

۵- introspection:

این endpoint جهت اعتبار سنجی توکن های reference token و (و یا jwt، در صورتی که api از توابع رمزنگاری jwt پشتیبانی نمی کند و نمی تواند صحت آن را تایید کند) توسط api ها مورد استفاده قرار می گیرد. این endpoint معمولا توسط کتابخانه ها به صورت داخلی مورد استفاده قرار می گیرد.

Request	https://<ServerAddress>/connect/introspect
Method	POST
Headers	Basic <apiAuth>
Query	
Body	Token=<token>

Response	{
	"active": false
	}
	Or
	{
	"active": true,
	"sub": "123"
	}

در این endpoint، هدر ارسال شده حاوی ApiAuth برابر با نام api و رمز آن می باشد که به صورت basic ارسال شده است. سرور زمانی پیغام active برمی گرداند که توکن معتبر باشد و در scope های آن ، scope این api درخواست دهنده نیز موجود باشد.

۶- revocation:

این endpoint جهت حذف reference token ها و refresh token ها استفاده می شود. از آنجایی که در این سرور از jwt استفاده می شود نیاز چندانی به این endpoint احساس نمی شود.

Request	https://<ServerAddress>/connect/revocation
Method	POST
Headers	Basic <clientAuth>
Query	
Body	Token=<token>&token_type_hint=<token_type>
Response	

در این endpoint، هدر ارسال شده حاوی clientAuth برابر با نام کلاینت و رمز آن می باشد که به صورت basic ارسال شده است. همچنین token_type می تواند برابر با یکی از مقادیر access_token یا refresh_token باشد.

۷- **endsession:**

این endpoint جهت پیاده سازی single sign out استفاده می شود. در صورتی که کلاینت نیاز دارد تا با خروج کاربر، کاربر را از identity server هم logout کند از این endpoint استفاده می کند. البته اکثر کتابخانه ها این مورد را پیاده سازی کرده اند و معمولا نیازی به فراخوانی دستی آن نیست.

Request	https://<ServerAddress>/connect/endsession	
Method	GET	
Headers		
Query	Id_token_hint	برابر با identity token می باشد
	Post_logout_redirect_uri	برابر با آدرس post logout می باشد.
Body		
Response	`Redirect to post_logout_redirect_uri`	

نکات تکمیلی

۱- نکته ای که باید هنگام تعریف کلاینت به آن توجه کرد این است که تعریف کلاینت با grant type ترکیبی Implicit و Hybrid همزمان امکان پذیر نمی باشد. بنابراین در صورتی که اپلیکیشن توسط یک کلاینت جاوااسکریپتی (اپلیکیشن های SPA) و به صورت mvc به صورت همزمان عمل ورود به سیستم را انجام می دهد تنها راه استفاده از Implicit grant type می باشد.

۲- در صورتی که کلاینت دارای سیستم Authentication داخلی باشد و در عین حال نیاز به تایید اعتبار توکن های تولید شده توسط این سرویس را داشته باشد می تواند از هر دو middleware تایید هویت OauthBearerAuthentication و IdentityServerBearerTokenAuthentication همزمان استفاده کند. البته باید ترتیب قرار گیری آنها به گونه ای باشد که ابتدا IdentityServer و سپس Oauth قرار بگیرد.

۳- لیست Scope هایی که پس از لاگین کاربر در اختیار اپلیکیشن قرار می گیرد از سه نقطه می تواند محدود شود:

a. هنگام تعریف کلاینت در سرویس Authentication : کلاینت مجاز است که فقط Scope هایی را درخواست کند که برای آن تعریف شده است. در غیر این صورت با خطای unautorized client مواجه خواهد شد.
b. هنگام لاگین کاربر، کلاینت می تواند Scope های کمتری از میزان تعریف شده مجاز خود را درخواست نماید.
c. پس از ورود کاربر، کاربر می تواند دسترسی به Scope هایی که مایل نیست در اختیار اپلیکیشن قرار بگیرد را حذف کند.

همچنین در صورتی که کلاینت بخواهد شخص را در سیستم Auth داخلی ثبت نام کند و با ارسال توکن قصد دریافت اطلاعات کاربر را اشته باشد، فقط اطلاعاتی بازگردانده می شوند که کاربر آنها را قبلا در هنگام لاگین تایید کرده باشد. لذا هنگام طراحی باید این مورد را مدنظر داشت.

پیوست ۱ : محرمانگی اطلاعات

اطلاعاتی که توسط اتصال گیرنده ذخیره می‌شود در هر مورد و به هر دلیل نباید مورد استفاده غیر، قرار بگیرد و استفادهٔ تمامی اپراتورها، راهبران، کاربران سرویس و مشتریان از سرویس های تینت به منزلهٔ قبول شرایط مندرج در این مستند می باشد.

۱- اتصال گیرنده متعهد می شود اطلاعات دریافتی از سرویس های شبکه را در اختیار شخص دیگری قرار ندهد.

۲- اتصال گیرنده متعهد می شود اطلاعات دریافتی از سرویس های شبکه را بدون کسب مجوز از شبکه نوآوری تهران، به منظور مقاصد آماری و تحلیلی استفاده نکند.

۳- اتصال گیرنده متعهد است از اطلاعات دریافتی به منظور مقاصد تبلیغاتی استفاده نکند.

٤- اتصال گیرنده متعهد است اطلاعات کاربران را در اختیار هیچ شخص حقوقی و حقیقی نگذارد.

پیوست ۲ : توافق سطح خدمات

بی‌تردید، ارایه سرویس با بهترین کیفیت ممکن و تامین رضایت کاربران باید همواره یکی از مهم‌ترین ارکان فعالیت شبکه نوآوری تهران و شرکت های اتصال گیرنده باشد و همواره تلاش همه جانبه‌ای برای تحقق بخشیدن به این مهم از اهداف اصلی تینت و اتصال گیرندگان است.

برای دست‌یابی به این هدف و اطمینان از تامین رضایت کاربران، ارایه خدمات به ایشان مطابق ضوابط مندرج در این مستند برای اتصال گیرندگان الزامی است. پارامترهای این موافقت‌نامه و توضیح کوتاهی درباره هر یک از آن‌ها در جدول و در سطرهای آتی آورده شده است:

شرح پارامتر	میزان
پارامتر a برای محاسبه حداکثر میزان مجازPLR	۳%
پارامتر b برای محاسبه حداکثر میزان تاخیر مجاز(Latency)	۱۳۰ میلی‌ثانیه
پارامتر c برای محاسبه پهنای باند تضمین‌شده(CIR)	۰.۱۲۵
میانگین زمان بازیابی یا تعمیر و برقراری مجدد (MTTR) خدمات	۷۲ ساعت
خطای اپلیکیشن در روز	۱۰ بار در روز

میزان تلفات بسته‌ها: PLR (Packet Loss Ratio)

به میانگین گم‌شدن و یا از دست‌رفتن بسته‌های IP در طول شبکه خدمت‌دهنده اطلاق می‌شود و به روش ارسال بسته‌های ICMP به اندازه ۱۰۰ بایت و به تعداد ۱۰۰۰ عدد از پورت دسترسی خدمت‌گیرنده تا نقطه انتهایی داخل شبکه خدمت‌دهنده و بر اساس میانگین نمونه‌برداری‌های متوالی Ping Test در طول یک ساعت و یا بر اساس سایر روش‌های استانداردی که سازمان اعلام خواهد کرد، محاسبه می‌شود.

تاخیر:(Latency)

متوسط زمانی که طول می‌کشد تا یک بسته IP در شبکه خدمت‌دهنده از پورت دسترسی خدمت‌گیرنده تا نقطه انتهایی شبکه خدمت-دهنده برسد. این پارامتر بر اساس میانگین نمونه‌برداری در دوره زمانی و یا بر اساس سایر روش‌های استانداری که سازمان اعلام خواهد کرد محاسبه می‌شود.

پهنای باند تضمین شده: Committed Information Rate

به حداقل پهنای باند اختصاص یافته به خدمت‌گیرنده در دوره زمانی اطلاق می‌شود که همان ضریب اشتراک یک به هشت است.

میانگین زمان بازیابی یا تعمیر : Mean Time To Restore or Repair

به میانگین زمان رفع خرابی و برقراری مجدد خدمت براساس توافق بین خدمت‌دهنده و خدمت‌گیرنده اطلاق می‌شود.

لازم به ذکر موارد زیر شامل این توافق نامه سطح خدمات نمی باشد:

- قطعی‌های ناشی از قوه قاهره (فورس ماژور) مانند حوادث طبیعی. در این حالت زمان کاهش سطح کیفیت خدمات
- قطعی‌هایی که به واسطه خرابی تجهیزات شبکه باشد.
- قطعی‌هایی که در زمان Down Time باشد.
- قطعی‌های ناشی از صدور احکام توسط مراجع قضایی و یا امنیتی کشور و یا سایر مراجع ذی‌صلاح .