Artigo
Daniel Kutac · Nov. 19, 2022 14min de leitura

Implementação do Open Authorization Framework (OAuth 2.0) na InterSystems IRIS – parte 1

Este artigo e os próximos dois artigos da série são um guia do usuário para desenvolvedores ou administradores de sistema que precisam usar o framework OAuth 2.0 (chamado de OAUTH para simplificar) em suas aplicações baseadas no produto InterSystems.

Criado por Daniel Kutac, Engenheiro de vendas sênior, InterSystems

Histórico de correções e alterações após a publicação

- 3 de agosto de 2016 – Correção da captura de tela da configuração do Google Client; atualização da captura de tela das APIs do Google para refletir a nova versão das páginas.
- 28 de agosto de 2016 – Alterações do código JSON devido às mudanças no suporte ao JSON do Caché 2016.2.
- 3 de maio de 2017 – Atualizações do texto e imagens para refletir a nova interface gráfica e os novos recursos lançados no Caché 2017.1.
- 19 de fevereiro de 2018 – Alteração de Caché para InterSystems IRIS para refletir os desenvolvimentos mais recentes. Porém, é importante salientar que, apesar da alteração do nome do produto, o **artigo aborda todos os produtos da InterSystems**: InterSystems IRIS Data Platform, Ensemble e Caché.
- 17 de agosto de 2020 – Tudo muda, especialmente o software. Consulte o URL do OAuth2 do Google atualizado na resposta do Micholai Mitchko.

Parte 1. Cliente

Introdução

Esta é a primeira parte de uma série de três artigos sobre a implementação do Open Authorization Framework na InterSystems.

Nesta primeira parte, apresentamos uma breve introdução do tópico e mostramos um cenário simples em que a aplicação InterSystems IRIS atua como cliente de um servidor de autorização, solicitando alguns recursos protegidos.

A segunda parte descreverá um cenário mais complexo, em que a InterSystems IRIS atua como servidor de autorização e também como servidor de autenticação via OpenID Connect.

A última parte da série descreverá partes individuais das classes do framework OAUTH conforme implementadas pela InterSystems IRIS.

Sobre o Open Authorization Framework[1]

Muitos de vocês já ouviram falar do Open Authorization Framework e para que ele pode ser usado. Vamos resumir para quem ainda não tiver ouvido falar dele.

O Open Authorization Framework, OAUTH, atualmente na versão 2.0, é um protocolo que permite principalmente

que aplicações web troquem informações de forma segura estabelecendo uma confiança indireta entre um cliente (aplicação que solicita dados) e o proprietário dos recursos (aplicação que detém os dados solicitados). A confiança é fornecida por uma entidade que tanto o cliente quanto o servidor de recursos reconhecem e na qual confiam. Essa entidade é chamada de servidor de autorização.

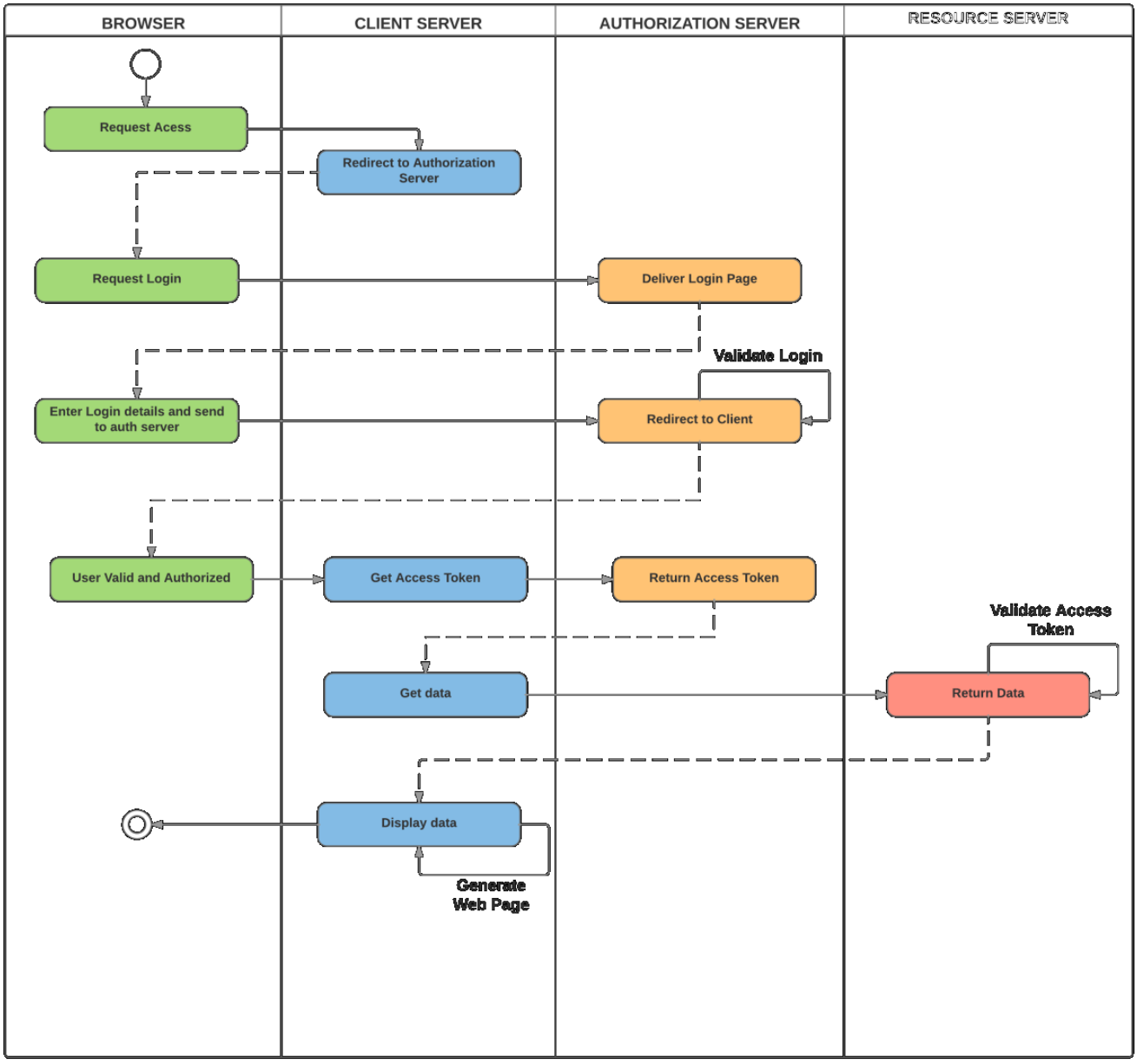
Veja um caso de uso simples:

Vamos supor que Jenny (na terminologia do OAUTH, é o proprietário dos recursos) esteja trabalhando em um projeto na empresa JennyCorp. Ela cria um plano de projeto para um possível negócio maior e convida seu parceiro comercial John (usuário cliente) da empresa JohnInc para revisar o documento. Mas ela não está contente de dar ao John acesso à VPN de sua empresa, então ela coloca o documento no Google Drive (o servidor de recursos) ou outra ferramenta de armazenamento em nuvem similar. Ao fazer isso, ela estabeleceu uma confiança entre ela e o Google (o servidor de autorização). Ela compartilha o documento com John (John já usa o serviço Google Drive, e Jenny sabe qual é o e-mail dele).

Quando John deseja ler o documento, ele faz a autenticação em sua conta do Google e, em seu dispositivo móvel (tablet, notebook, etc.), abre um editor de documentos (o servidor cliente) e carrega o arquivo do projeto da Jenny.

Embora pareça bem simples, há muita comunicação entre as duas pessoas e o Google. Todas as comunicações seguem a especificação do OAuth 2.0, então o cliente de John (o leitor de documentos) precisa primeiro fazer a autenticação no Google (essa etapa não é coberta pelo OAUTH) e, após John consentir autorização no formulário fornecido pelo Google, o Google autoriza que o leitor de documentos acesse o documento emitindo um token de acesso. O leitor de documentos usa o token de acesso para emitir uma solicitação ao serviço Google Drive para obter o arquivo de Jenny.

O diagrama abaixo mostra a comunicação entre todas as partes



Nota: embora todas as comunicações do OAUTH 2.0 sejam feitas por solicitações HTTP, os servidores não precisam ser aplicações web.

Vamos ilustrar esse cenário simples com a InterSystems IRIS.

Demonstração simples do Google Drive

Nesta demonstração, vamos criar uma aplicação de pequeno porte baseada em Cloud Solution Provider (CSP) que solicita recursos (lista de arquivos) armazenados no serviço Google Drive com nossa própria conta (e também uma lista de nossos calendários, como bônus).

Pré-requisitos

Antes de começarmos a programar a aplicação, precisamos preparar o ambiente. Precisaremos de um servidor web com SSL ativado e um perfil do Google.

Configuração do servidor web

Conforme informado acima, precisamos estabelecer comunicação com o servidor de autorização com SSL, pois isso é exigido pelo OAuth 2.0 por padrão. Queremos manter nossos dados seguros, certo?

Está fora do escopo deste artigo descrever como configurar um servidor web com suporte ao SSL, então consulte os manuais de usuário do servidor web de sua preferência. Usaremos o servidor IIS da Microsoft neste exemplo específico.

Configuração do Google

Para nos registrarmos no Google, precisamos usar o Google API Manager:
https://console.developers.google.com/apis/library?project=globalsummit2016demo

Para o propósito da demonstração, criamos uma conta GlobalSummit2016Demo. É preciso confirmar se a API do Drive está ativada

Agora, está na hora de definir as credenciais

Observe o seguinte:

Authorized JavaScript (JavaScript autorizado) – permitimos somente scripts originados localmente em relação à página chamadora

Authorized redirect URIs (URIs de redirecionamento autorizados) – teoricamente, podemos redirecionar nossa aplicação cliente para qualquer site, mas, ao usar a implementação do OAUTH da InterSystems IRIS, precisamos redirecioná-la para https://localhost/csp/sys/oauth2/OAuth2.Response.cls**.** É possível definir vários URIs de redirecionamento autorizados, conforme mostrado na captura de tela, mas, para esta demonstração, só precisamos da segunda entrada.

Por último, precisamos configurar a InterSystems IRIS como cliente do servidor de autorização do Google

Configuração do Caché

A configuração do cliente OAUTH2 da InterSystems IRIS é um processo de duas etapas. Primeiro, precisamos criar uma configuração de servidor.

No SMP, acesse **System Administration (Administração do Sistema) > Security (Segurança) > OAuth 2.0 > Client Configurations (Configurações do cliente)**.

Clique no botão **Create Server Configuration (Criar configuração de servidor)**, preencha o formulário e salve.

Todas as informações inseridas no formulário estão disponíveis no site do console de desenvolvedores do Google. A InterSystems IRIS tem suporte à descoberta automática do Open ID. Entretanto, não estamos usando esse recurso. Inserimos todas as informações manualmente

Agora, clique no link Client Configurations (Configurações do cliente) ao lado do Issuer Endpoint (Endpoint emissor) recém-criado
e clique no botão **Create Client Configuration (Criar configuração de cliente)**.

Deixe as abas Client Information (Informações do cliente) e JWT Settings (Configurações do JWT) em branco (com os valores padrão) e preencha a aba Client credentials (Credenciais do cliente).

Nota: estamos criando um Confidential Client (Cliente confidencial – é mais seguro que o público e significa que o segredo do cliente nunca deixa a aplicação do servidor cliente – nunca é transmitido ao navegador)

Além disso, confirme se **Use SSL/TLS (Usar SSL/TLS)** está marcado e forneça o nome do host (localhost, já que estamos redirecionando localmente para a aplicação cliente) e, posteriormente, a porta e o prefixo (útil quando há

várias instâncias da InterSystems IRIS na mesma máquina). Com base nas informações preenchidas, o URL de redirecionamento do cliente é computado e exibido na linha acima.

Na captura de tela acima, fornecemos uma configuração SSL chamada GOOGLE. O nome é usado somente para ajudar a determinar qual configuração SSL dentre várias é usada por esse canal de comunicação específico. O Caché está usando configurações SSL/TLS para armazenar todas as informações necessárias para receber/enviar tráfego seguro ao servidor (neste caso, os URIs OAuth 2.0 do Google).

Consulte mais detalhes na documentação.

Preencha os valores Client ID (ID do cliente) e Client Secret (Segredo do cliente) obtidos pelo formulário de definição das credenciais do Google (ao fazer a configuração manual).

Agora, concluímos todas as etapas de configuração e podemos prosseguir para a programação de uma aplicação CSP.

Aplicação cliente

A aplicação cliente é uma aplicação CSP web simples. Ela é composta por um código fonte no servidor, definido e executado pelo servidor web, e uma interface do usuário, exibida ao usuário por um navegador. O exemplo de código fornecido espera que a aplicação cliente seja executada no namespace GOOGLE. Modifique o caminho /csp/google/ para o seu namespace.

Servidor cliente

O servidor cliente é uma aplicação simples de duas páginas. Dentro da aplicação, nós vamos:

· Montar o URL de redirecionamento para o servidor de autorização do Google

· Fazer solicitações à API do Google Drive e à API do Google Agenda e exibir o resultado

Página 1

Esta é uma página da aplicação, na qual decidimos fazer uma chamada aos recursos do Google.

Veja abaixo um código minimalista, mas completamente funcional, que representa a página.

```
Class Web.OAUTH2.Google1N Extends %CSP.Page
{
Parameter OAUTH2CLIENTREDIRECTURI = "https://localhost/csp/google/Web.OAUTH2.Google2N
.cls";
Parameter OAUTH2APPNAME = "Google";
ClassMethod OnPage() As %Status
{
  &html<<html>
<head>
</head>
<body style="text-align: center;">
        <!-- insert the page content here -->
        <h1>Google OAuth2 API</h1>

<p>This page demo shows how to call Google API functions using OAuth2 authorization.

<p>We are going to retrieve information about user and his/her Google Drive files as
well as calendar entries.
```

```
        >

  // we need to supply openid scope to authenticate to Google
  set scope="openid https://www.googleapis.com/auth/userinfo.email "_
  "https://www.googleapis.com/auth/userinfo.profile "_
  "https://www.googleapis.com/auth/drive.metadata.readonly "_
  "https://www.googleapis.com/auth/calendar.readonly"
  set properties("approval_prompt")="force"
  set properties("include_granted_scopes")="true"

set url=##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(..#OAUTH2APPN
AME,scope,
    ..#OAUTH2CLIENTREDIRECTURI,.properties,.isAuthorized,.sc)
  w !,"<p><a href='"_url_"'><img border='0' alt='Google Sign In' src='images/google-
signin-button.png' ></a>"
      &html<</body>
</html>>
  Quit $$$OK
}
ClassMethod OnPreHTTP() As %Boolean [ ServerOnly = 1 ]
{
  #dim %response as %CSP.Response
  set scope="openid https://www.googleapis.com/auth/userinfo.email "_
    "https://www.googleapis.com/auth/userinfo.profile "_
    "https://www.googleapis.com/auth/drive.metadata.readonly "_
    "https://www.googleapis.com/auth/calendar.readonly"

if ##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,scope,.accessToken
,.idtoken,.responseProperties,.error) {
    set %response.ServerSideRedirect="Web.OAUTH2.Google2N.cls"
  }
  quit 1
  }
}
```

Veja abaixo uma breve explicação do código:

1. Método OnPreHTTP: primeiro, verificamos se, por acaso, já obtivemos um token de acesso válido como resultado de uma autorização do Google. Isso pode acontecer, por exemplo, quando simplesmente atualizamos a página. Caso não tenhamos, precisamos fazer a autorização. Se já tivermos o token, apenas redirecionamos para a página que mostra os resultados

2. Método OnPage: só chegamos a este método se não tivermos um token de acesso válido disponível. Então, precisamos iniciar a comunicação: fazer a autenticação e autorização no Google para que ele nos conceda o token de acesso.

3. Definimos uma string de escopo e uma array de propriedades que modificam o comportamento da janela de autenticação do Google (precisamos fazer a autenticação no Google antes que ele possa nos autorizar com base em nossa identidade).

4. Por último, recebemos o URL de uma página de login do Google e a mostramos ao usuário, seguida pela página de consentimento.

Mais uma nota:

Especificamos a verdadeira página de redirecionamento em https://www.localhost/csp/google/Web.OAUTH2.Google2N.cls no parâmetro OAUTH2CLIENTREDIRECTURI. Entretanto, usamos a página de sistema do framework OAUTH da InterSystems IRIS na definição das credenciais do Google! O redirecionamento é tratado internamente por nossa classe manipuladora OAUTH.

Página 2

Esta página mostra os resultados da autorização do Google e, em caso de êxito, fazemos chamadas à API do Google para obter os dados. Novamente, o código abaixo é minimalista, mas completamente funcional. Deixamos a exibição dos dados recebidos de uma maneira mais estruturada para a imaginação dos leitores.

```
Include %occInclude
Class Web.OAUTH2.Google2N Extends %CSP.Page
{
Parameter OAUTH2APPNAME = "Google";
Parameter OAUTH2ROOT = "https://www.googleapis.com";
ClassMethod OnPage() As %Status
{
  &html<<html>
   <head>
   </head>
   <body>>
  // Check if we have an access token
  set scope="openid https://www.googleapis.com/auth/userinfo.email "_
    "https://www.googleapis.com/auth/userinfo.profile "_
    "https://www.googleapis.com/auth/drive.metadata.readonly "_
    "https://www.googleapis.com/auth/calendar.readonly"

set isAuthorized=##class(%SYS.OAuth2.AccessToken).IsAuthorized(..#OAUTH2APPNAME,,scop
e,.accessToken,.idtoken,.responseProperties,.error)
  if isAuthorized {

// Google has no introspection endpoint - nothing to call - the introspection endpoin
t and display result -- see RFC 7662.
    w "<h3>Data from <span style='color:red;'>GetUserInfo API</span></h3>"

// userinfo has special API, but could be also retrieved by just calling Get() method
 with appropriate url
    try {
    set tHttpRequest=##class(%Net.HttpRequest).%New()

$$$THROWONERROR(sc,##class(%SYS.OAuth2.AccessToken).AddAccessToken(tHttpRequest,"quer
y","GOOGLE",..#OAUTH2APPNAME))

$$$THROWONERROR(sc,##class(%SYS.OAuth2.AccessToken).GetUserinfo(..#OAUTH2APPNAME,acce
ssToken,,.jsonObject))
      w jsonObject.%ToJSON()
    } catch (e) {

w "<h3><span style='color: red;'>ERROR: ",$zcvt(e.DisplayString(),"O","HTML")_"</span
></h3>"
    }
    /*****************************************
    *                                       *
    *      Retrieve info from other APIs    *
    *                                       *
    *****************************************/
    w "<hr>"
    do ..RetrieveAPIInfo("/drive/v3/files")

    do ..RetrieveAPIInfo("/calendar/v3/users/me/calendarList")
```

```
  } else {
    w "<h1>Not authorized!</h1>"
  }
  &html<</body>
  </html>>
  Quit $$$OK
}


ClassMethod RetrieveAPIInfo(api As %String)
{
  w "<h3>Data from <span style='color:red;'>"_api_"</span></h3><p>"
  try {
    set tHttpRequest=##class(%Net.HttpRequest).%New()

$$$THROWONERROR(sc,##class(%SYS.OAuth2.AccessToken).AddAccessToken(tHttpRequest,"quer
y","GOOGLE",..#OAUTH2APPNAME))
    $$$THROWONERROR(sc,tHttpRequest.Get(..#OAUTH2ROOT_api))
    set tHttpResponse=tHttpRequest.HttpResponse
    s tJSONString=tHttpResponse.Data.Read()
    if $e(tJSONString)'="{" {
      // not a JSON
      d tHttpResponse.OutputToDevice()
    } else {
      w tJSONString
      w "<hr/>"
      /*
      // new JSON API
      &html<<table border=1 style='border-collapse: collapse'>>
      s tJSONObject={}.%FromJSON(tJSONString)
      set iterator=tJSONObject.%GetIterator()
        while iterator.%GetNext(.key,.value) {
          if $isobject(value) {
            set iterator1=value.%GetIterator()
            w "<tr><td>",key,"</td><td><table border=1 style='border-
collapse: collapse'>"
            while iterator1.%GetNext(.key1,.value1) {
            if $isobject(value1) {
                set iterator2=value1.%GetIterator()
                w "<tr><td>",key1,"</td><td><table border=0 style='border-
collapse: collapse'>"
                while iterator2.%GetNext(.key2,.value2) {

write !, "<tr><td>",key2, "</td><td>",value2,"</td></tr>"
                }
                // this way we can go on and on into the embedded objects/arrays
              w "</table></td></tr>"
            } else {
                 write !, "<tr><td>",key1, "</td><td>",value1,"</td></tr>"
            }
            }
          w "</table></td></tr>"
          } else {
              write !, "<tr><td>",key, "</td><td>",value,"</td></tr>"
          }
        }
    &html<</table><hr/>
    >
    */
```

```
    }
  } catch (e) {

w "<h3><span style='color: red;'>ERROR: ",$zcvt(e.DisplayString(),"O","HTML")_"</span
></h3>"
  }
}
}
```

Vamos dar uma olhada no código:

1. Antes de tudo, precisamos verificar se temos um token de acesso válido (para verificarmos se fomos autorizados)

2. Caso afirmativo, podemos enviar solicitações às APIs oferecidas pelo Google usando o token de acesso emitido

3. Para isso, usamos a classe padrão %Net.HttpRequest, mas adicionamos o token de acesso ao método GET ou POST de acordo com a especificação da API chamada

4. Como é possível ver, o framework OAUTH implementou o método GetUserInfo() para sua comodidade, mas você pode obter as informações do usuário diretamente usando a especificação da API do Google da mesma maneira como feito no método auxiliar RetrieveAPIInfo()

5. Como é comum no mundo do OAUTH trocar dados no formato JSON, apenas lemos os dados recebidos e os colocamos no navegador. Cabe ao desenvolvedor da aplicação analisar e formatar os dados recebidos para apresentá-los ao usuário de alguma forma que faça sentido. Mas isso está além do escopo desta demonstração. (Embora tenhamos colocado um código comentado que mostra como a análise pode ser feita.)

Veja abaixo uma captura de tela da saída exibindo os dados JSON não tratados.

Prossiga para a parte 2, que descreve como a InterSystems IRIS atua como servidor de autorização e provedor do OpenID Connect.

[1] https://tools.ietf.org/html/rfc6749, https://tools.ietf.org/html/rfc6750

#Autenticação #Controle de acesso #OAuth2 #Segurança #Caché #InterSystems IRIS

URL de origem:https://pt.community.intersystems.com/post/implementa%C3%A7%C3%A3o-do-open-authorization-framework-oauth-20-na-intersystems-iris-%E2%80%93-parte-1