Artículo
Ricardo Paiva · 6 feb, 2020 Lectura de 15 min

Implementación de la estructura Open Authorization (OAuth 2.0) en InterSystems IRIS - Parte 1

Este artículo, y los dos siguientes de la serie, se conciben como una guía para los usuarios, los desarrolladores o para los administradores de sistemas que necesiten trabajar con la estructura de OAuth 2.0 (conocido también por simplicidad como OAUTH) en aplicaciones que estén basadas en los productos de InterSystems.

Correcciones y cambios después de la publicación

- 3 de agosto, 2016 - se corrigió la configuración para capturas de pantalla en la API Client de Google y se actualizaron las capturas de pantalla en las API de Google para incorporar la nueva versión en las páginas
- 28 de agosto, 2016 - se realizaron cambios en el código que se relaciona con JSON, dichos cambios se ven reflejados en la compatibilidad entre JSON y Cache 2016.2
- 3 de mayo, 2017 - se realizaron actualizaciones en el texto y en las pantallas para incluir la nueva IU y las funcionalidades disponibles con el lanzamiento de Cache 2017.1
- 19 de febrero, 2018 - Caché se cambió a InterSystems IRIS para mostrar los últimos desarrollos. Sin embargo, hay que tener en cuenta que a pesar de los cambios en el nombre del producto, el artículo cubre **todos los productos de InterSystems** - la plataforma de datos InterSystems IRIS, Ensemble y Caché.

Parte 1. Cliente

Introducción

Esta es la primera parte de una serie de 3 artículos sobre la implementación de la estructura Open Authorization en InterSystems.

En esta primera parte, veremos una breve introducción al tema y mostraremos un escenario sencillo en el que la aplicación InterSystems IRIS actúa como cliente para un Servidor de autorización, solicitando algunos recursos que están protegidos.

En la segunda parte se describirá un escenario más complejo, donde el propio InterSystems IRIS actúa como un servidor de autorización y también como un servidor de autenticación mediante OpenID Connect.

En la última parte de esta serie se describirán los distintos componentes de las clases en la estructura OAUTH, tal y como son implementados por InterSystems IRIS.

En qué consiste la estructura Open Authorization[1]

Muchos de ustedes ya han oído hablar sobre la estructura Open Authorization y para qué se puede utilizar. Así que solamente haré un breve resumen para aquellos que aún no la conocen.

La estructura Open Authorization, OAUTH (actualmente en su versión 2.0), es un protocolo que básicamente permite que las aplicaciones basadas en la web intercambien información de forma segura al establecer confianza, de manera indirecta, entre un cliente (la aplicación que solicita los datos) y el propietario de los recursos (la aplicación que posee los datos que se solicitaron). La confianza como tal se proporciona mediante un agente que tanto el cliente como el servidor de recursos reconocen y en el que confían. Este agente se denomina el *servidor de autorización*.

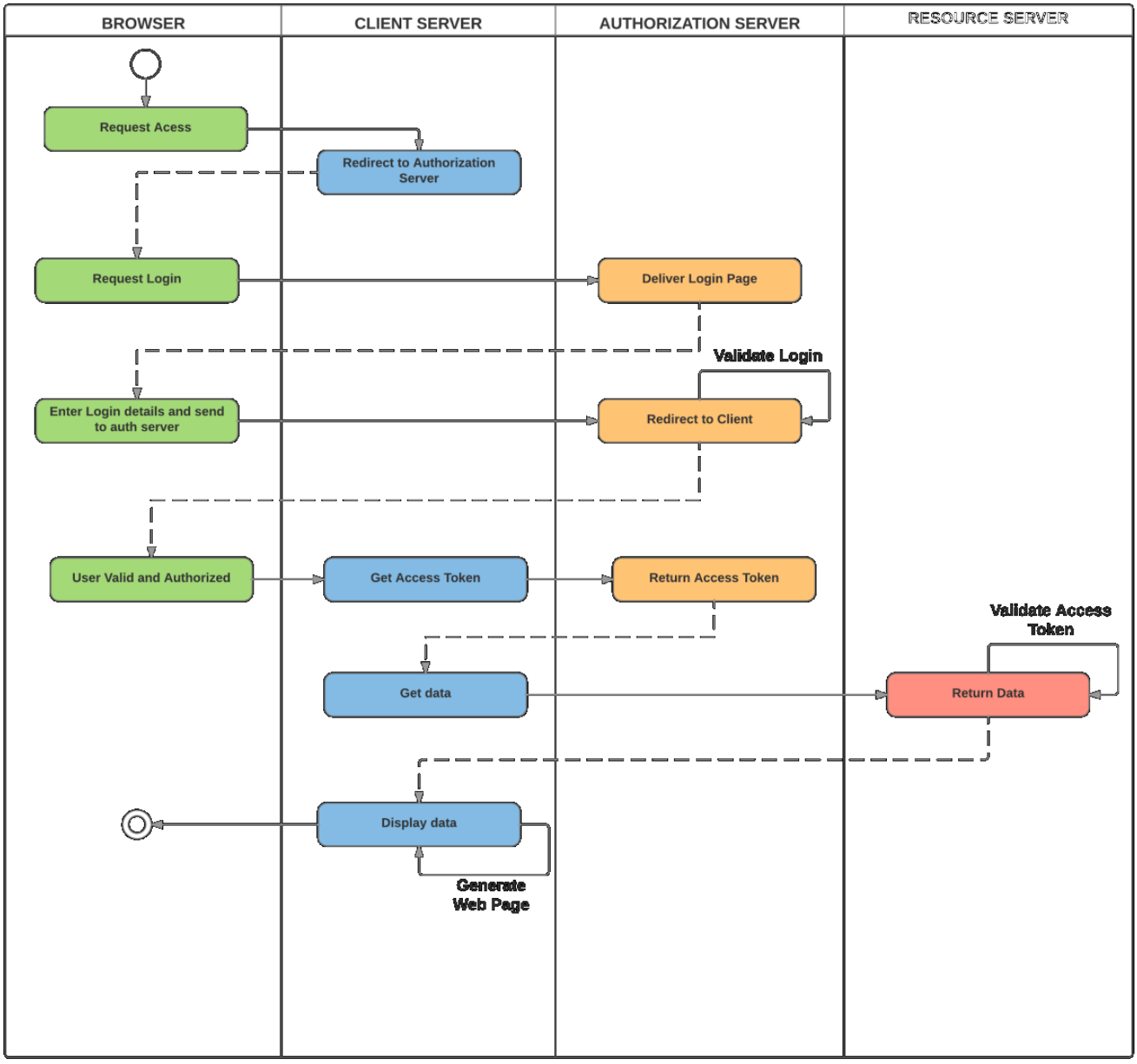
A continuación, se muestra un caso sencillo en el que podría utilizarse:

Supongamos que Jenny (en la terminología de OAUTH sería la propietaria del recurso) trabaja en un proyecto para la empresa JennyCorp. Ella crea un plan para realizar un proyecto sobre un negocio que será potencialmente más grande, e invita a su socio John (el usuario cliente) de JohnInc para que revise el documento. Sin embargo, a ella no le gusta que John entre en la VPN de su empresa, por lo que coloca el documento en una unidad de Google Drive (el Servidor de recursos) u otro tipo de almacenamiento similar en la nube. Cuando hizo esto, estableció una confianza digital entre ella y Google (el Servidor de autorización). Ella etiqueta el documento para ser compartido con John (John ya utiliza Google Drive y Jenny conoce su email).

Cuando John quiere leer el documento, primero lo autentica con su cuenta de Google y desde su dispositivo móvil (tableta, bloc de notas, etc.) abre un editor de documentos (el servidor del cliente) y carga el archivo del proyecto de Jenny.

Aunque esto parezca bastante sencillo, existe mucha comunicación entre las dos personas y Google. Toda la comunicación sigue la especificación de OAuth 2.0, por lo que el cliente de John (la aplicación de lectura), tiene que autenticarse primero con Google (este paso no está cubierto por OAUTH) y una vez que John otorgue su consentimiento en el formulario que proporciona Google, Google dará su autorización para que la aplicación de lectura tenga acceso al documento mediante la emisión de un token de acceso. La aplicación de lectura utiliza el token de acceso para emitir una solicitud hacia el servicio de Google Drive, con el fin de que sea posible recuperar el archivo de Jenny.

En el siguiente diagrama se muestra cómo ocurre la comunicación entre cada una de las partes:



Tenga en cuenta que aunque todas las comunicaciones de OAUTH 2.0 utilizan solicitudes HTTP, no es necesario que los servidores sean aplicaciones web.

Vamos a mostrar este sencillo escenario con InterSystems IRIS.

Demo sencilla de Google Drive

En esta demostración crearemos una pequeña aplicación con base en CSP, que solicitará los recursos (la lista de archivos) guardados en Google Drive con nuestra propia cuenta (y también una lista de nuestros eventos en el calendario como una ventaja adicional).

Requisitos previos

Antes de que comenzar a programar la aplicación, necesitamos preparar el entorno. Esto incluye un servidor web que tenga habilitado el protocolo SSL y un perfil de Google.

Configuración del servidor web

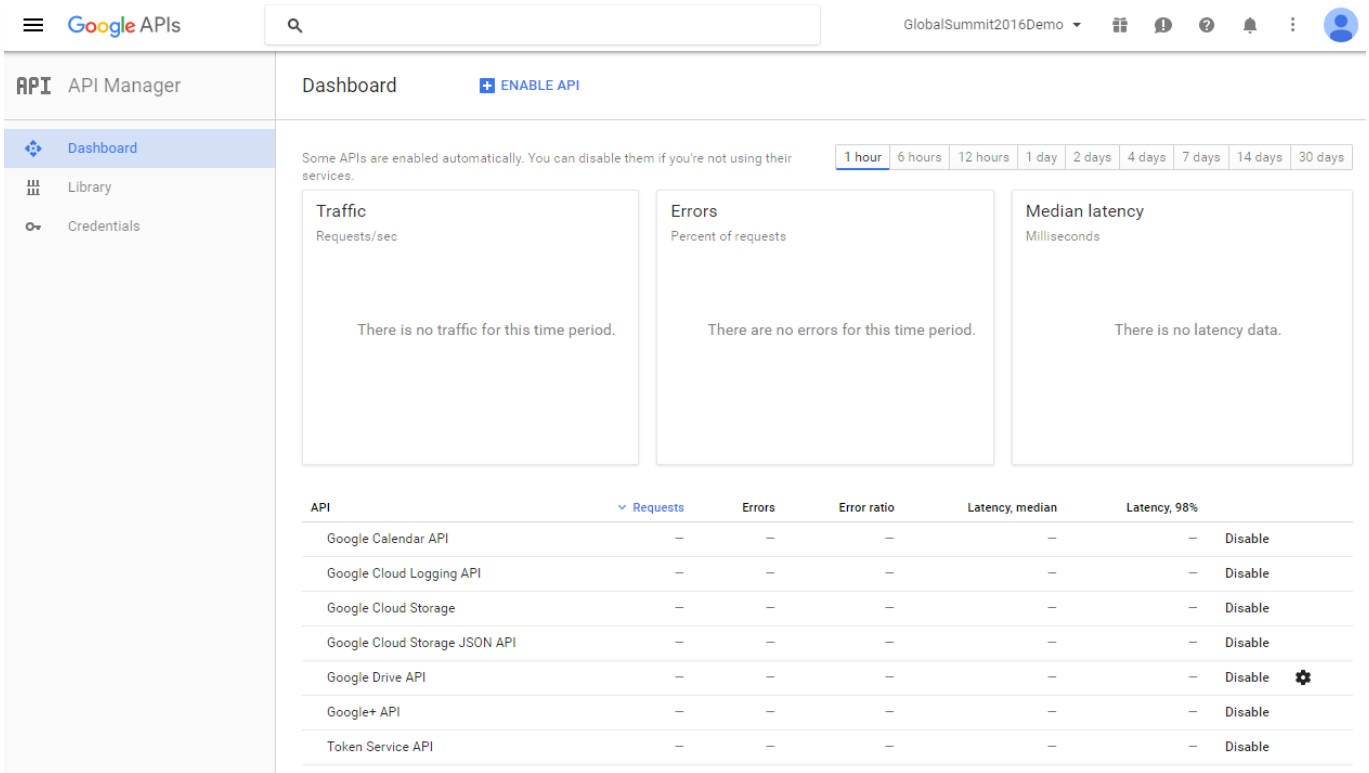
Como se indicó anteriormente, necesitamos que el Servidor de autorización se comunique con SSL, ya que esto es un requisito establecido por OAuth 2.0 de forma predeterminada. Queremos que nuestros datos se mantengan seguros, ¿verdad?

La explicación de cómo configurar un servidor web para que sea compatible con SSL está fuera del alcance de este artículo, por lo que debe consultar los manuales de usuario correspondientes al servidor web de su preferencia. Para su conocimiento (como veremos más adelante con algunas capturas de pantalla), en este ejemplo en particular utilizaremos el servidor Microsoft IIS.

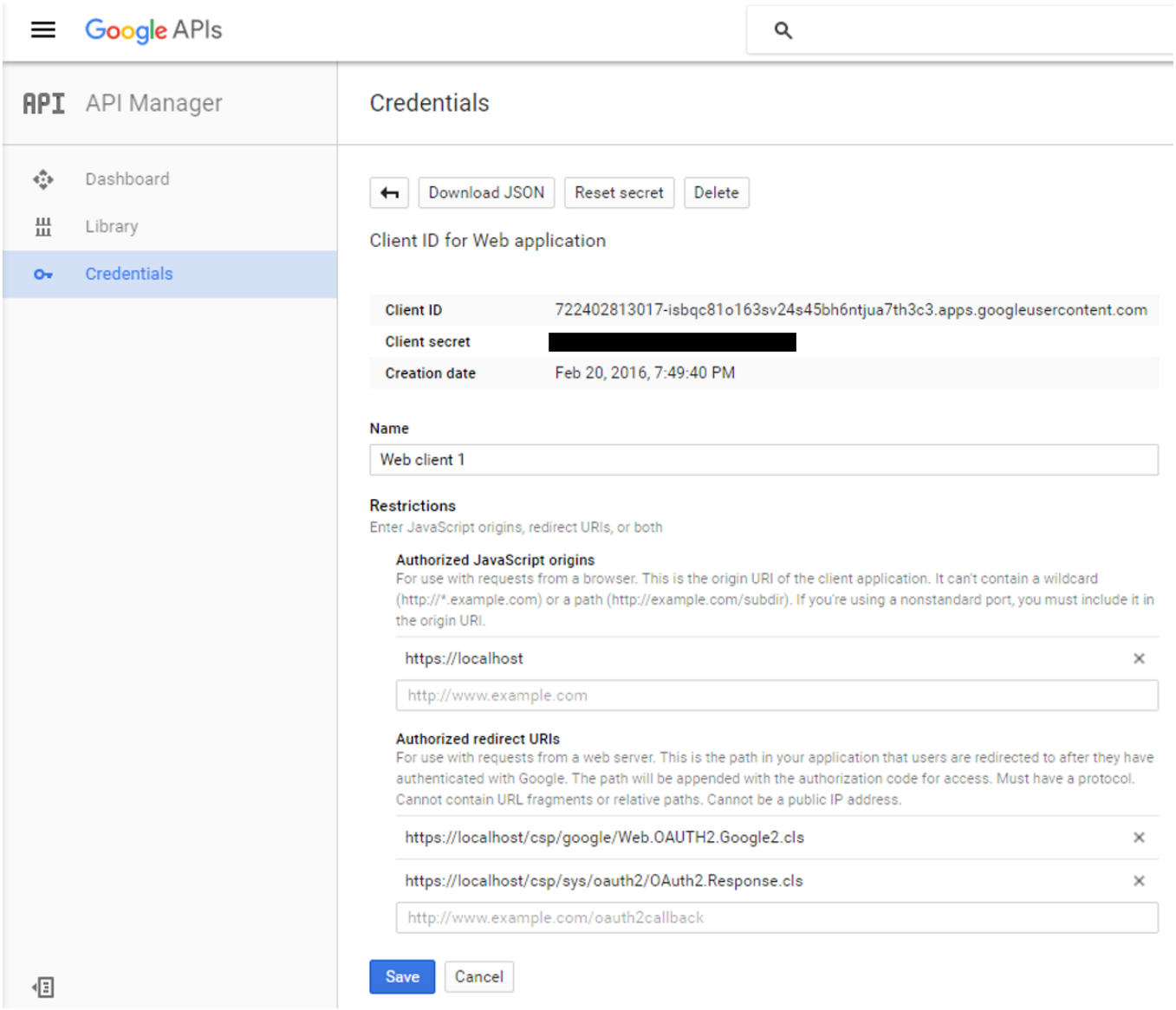
Configuración de Google

Para registrarnos en Google, necesitamos usar el administrador de APIs de Google (Google API Manager) - https://console.developers.google.com/apis/library?project=globalsummit2016demo

Para esta demo, creamos la cuenta GlobalSummit2016Demo. Asegúrese de que tenemos habilitada la API del Drive API.



Ahora se definen las credenciales:



Y tenga en cuenta lo siguiente.

Authorized JavaScript – solamente permitimos a los scripts que se originan de forma local, y que tengan relación con la llamada a la página

Authorized redirect URIs – en teoría podemos redirigir nuestra aplicación del cliente hacia cualquier sitio, pero cuando utilizamos la implementación de InterSystems IRIS, OAUTH, debemos redirigirla hacia https://localhost/csp/sys/oauth2/OAuth2.Response.cls. Puede definir varias URI de redireccionamiento autorizadas, como se muestra en la captura de pantalla, aunque para esta demostración únicamente necesitamos la segunda entrada de las dos que hay disponibles

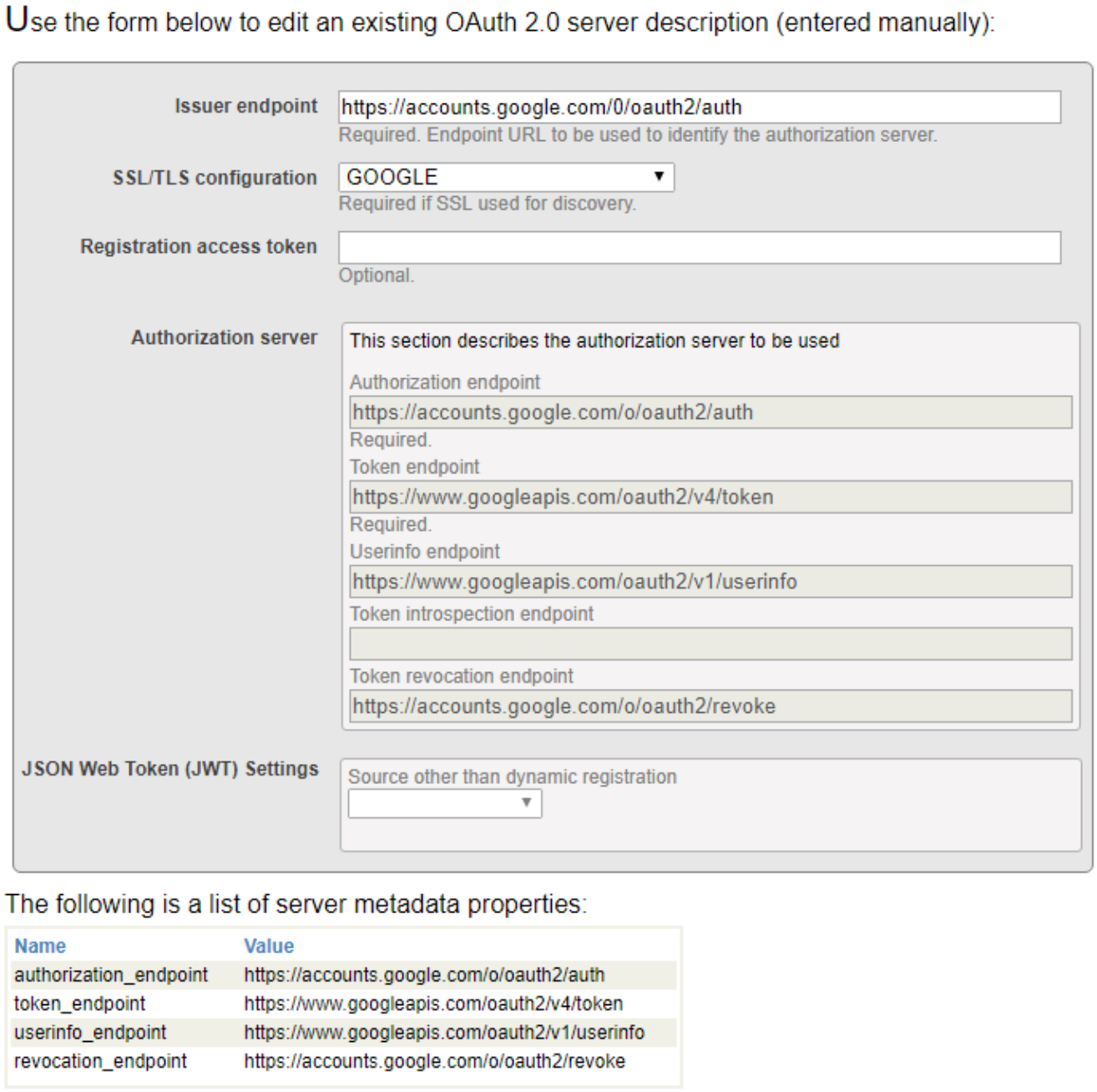
Por último, es necesario que se configure InterSystems IRIS como cliente del Servidor de Autorización de Google.

Configuración de Caché

La configuración como cliente de InterSystems IRIS OAUTH2 es un proceso en dos pasos. Primero, necesitamos crear una Configuración para el Servidor.

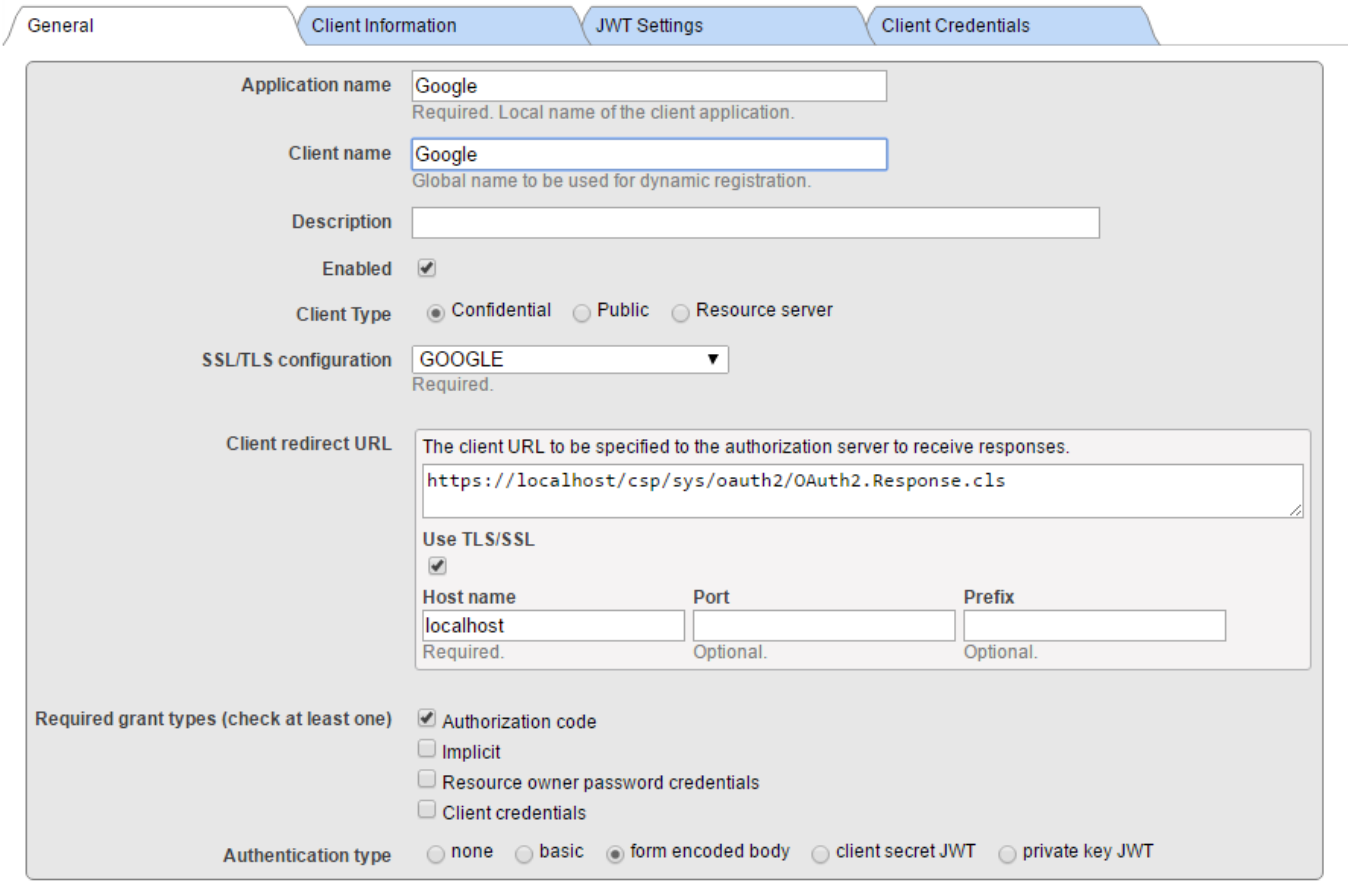
En SMP, vaya a **System Administration > Security > OAuth 2.0 > Client Configurations**.

Haga click en el botón **Create Server Configuration**, complete el formulario y guárdelo.

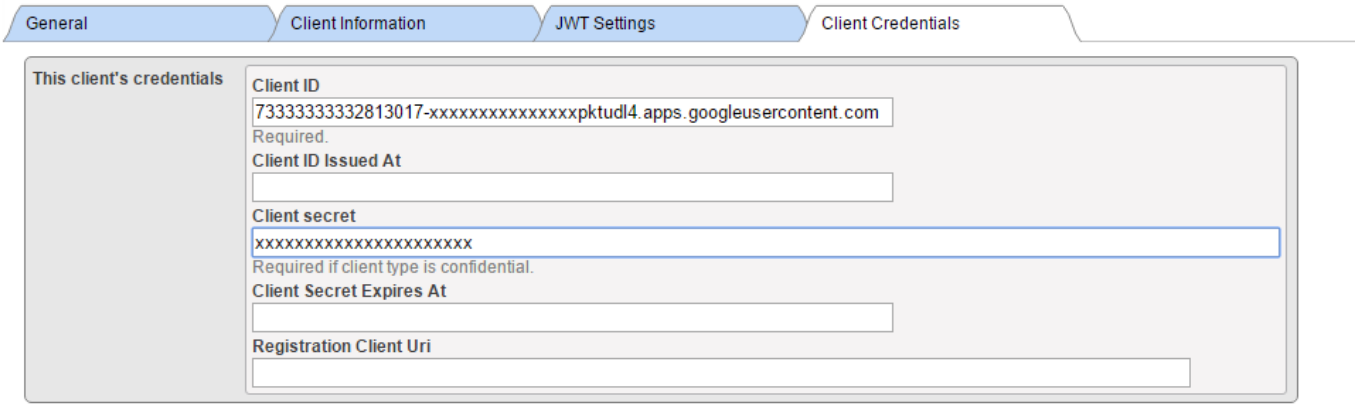


Toda la información que introdujo en el formulario está disponible en el sitio web para desarrolladores de Google ("Google Developers"). Tenga en cuenta que InterSystems IRIS es compatible con el reconocimiento automático de Open ID. Sin embargo, como no utilizamos ese protocolo, introdujimos toda la información de forma manual.

Ahora, haga clic en el enlace Client Configurations que se encuentra junto al Issuer Endpoint recién creado y haga clic en el botón **Create Client Configuration**.



Deje vacías las pestañas **Client Information** y **JWT Settings** (con los valores predeterminados) y complete los campos de la pestaña **Client credentials**.



Tenga en cuenta que creamos un Confidential Client (este es más seguro que el público), lo que significa que la información confidencial del cliente nunca sale de la aplicación cliente-servidor (nunca se transmite al navegador).

También, debe asegurarse de que la opción **Use SSL/TLS** está seleccionada, proporcione el nombre del servidor (debe ser localhost, ya que estamos redirigiendo la aplicación del cliente de manera local) y finalmente el puerto y el prefijo (esto es útil cuando existen varias instancias de InterSystems IRIS en el mismo equipo). De acuerdo a la información que se introdujo, la URL de redireccionamiento del cliente será registrada y se mostrará en la línea de arriba.

En la captura de pantalla anterior, proporcionamos una configuración SSL llamada GOOGLE. El nombre en sí solamente sirve para ayudarle a determinar cuál de las muchas posibles configuraciones de la SSL se utilizará para este canal de comunicación en particular. Caché utiliza las configuraciones SSL/TLS para almacenar toda la información necesaria para establecer un tráfico seguro con el servidor (en este caso, los URI de Google OAuth

2.0).

Consulte la documentación para obtener más información.

Proporcione los valores de Client ID y los de Client Secret que se obtuvieron del formulario para definir Google Credentials (si utiliza la configuración manual).

Ahora ya hemos terminado todos los pasos de la configuración y podemos avanzar hacia la programación de una aplicación CSP.

La aplicación del cliente

La aplicación del cliente es una sencilla aplicación web que se basa en CSP. Como tal, consiste en el código fuente del lado del servidor, que se define y ejecuta mediante el servidor web y una interfaz de usuario, y se muestra a los usuarios mediante un navegador web.

El servidor del cliente

El servidor del cliente es una aplicación sencilla de 2 páginas. Dentro de la aplicación, podemos:

· Juntar el redireccionamiento de la URL con el Servidor de Autorización de Google

· Realizar solicitudes a las APIs de Google Drive y de Google Calendar y mostrar el resultado

Página 1

Esta es una página de la aplicación, donde decidimos llamar a Google mediante sus recursos.

A continuación, se muestra un código minimalista, pero totalmente funcional, para representar la 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
  }
}
```

Explicamos brevemente el código:

1. Método OnPreHTTP - primero, debemos verificar si, por casualidad, ya obtuvimos un token de acceso válido como resultado de la autorización de Google (esto puede suceder, por ejemplo, cuando simplemente actualizamos la página). Si no es así, necesitaremos la autorización. Si ya tenemos un token, simplemente redirigiremos la página hacia la página que muestra los resultados

2. Método OnPage - solo lo utilizamos si no contamos con un token de acceso válido disponible. Para iniciar la comunicación, tenemos que realizar nosotros mismos la autenticación y la autorización con Google para que nos conceda el token de acceso

3. Definimos el objetivo de una cadena y las propiedades de la matriz que modifican el comportamiento del cuadro de diálogo para la autenticación de Google (necesitamos autenticarnos frente a Google antes de que éste nos otorgue su autorización según nuestra identidad)

4. Por último, recibimos la URL para una página de inicio de sesión en Google y se la presentamos al usuario seguida de la página de consentimiento

Una última cosa a tener en cuenta:

Especificamos la verdadera página de redireccionamiento en https://www.localhost/csp/google/Web.OAUTH2.Google2N.cls con el parámetro OAUTH2CLIENTREDIRECTURI. Sin embargo, ¡hemos usado la página del sistema de InterSystems IRIS con la estructura OAUTH para definir las credenciales de Google! El redireccionamiento se gestiona internamente mediante la clase de nuestro controlador OAUTH.

Página 2

En esta página se muestran los resultados de la autorización de Google y, si salió bien, recurriremos a las llamadas de la API de Google para recuperar los datos. Una vez más, es un código minimalista, pero totalmente funcional. Dejamos a la imaginación del lector una forma más estructurada para mostrar los datos que obtienen.

```
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>"
  }
}
}
```

Echemos un vistazo rápido al código:

1. En primer lugar, debemos comprobar si contamos con un token de acceso válido (que estamos autorizados)

2. Si es así, podemos hacer solicitudes a las APIs que ofrece Google, y están cubiertas por el token de acceso que se emitió

3. Para ello, utilizamos la clase estándar %Net.HttpRequest, pero añadimos el token de acceso a los métodos GET o POST, de acuerdo con la especificación de la API que se llamó

4. Como puede ver, la estructura OAUTH tiene implementado el método GetUserInfo() para su comodidad, pero puede recuperar directamente la información del usuario utilizando la especificación de la API de Google, de igual forma que lo hicimos con el método de ayuda RetrieveAPIInfo()

5. OAUTH intercambia los datos en formato JSON, simplemente leemos los datos que obtenemos y los volcamos en el navegador. Le corresponde al desarrollador de la aplicación analizar y dar formato a los datos recibidos, para que el usuario pueda verlos de una manera que sea útil para él. Pero esto va más allá del objetivo de esta demostración. (Aunque existen algunos comentarios en el código que muestran cómo realizar estos análisis)

Aquí se puede ver una captura de pantalla con los resultados, donde se muestran datos sin procesar en JSON.

Data from GetUserInfo API

{"id":"109305875729928072914","email":"isc.gs.2016@gmail.com","verified_email":true,"name":"GS ISC2016","given_name":"GS","family_name":"ISC2016","picture":"https://lh3.googleusercontent.com/-XdUIqdMkCWA/AAAAAAAAAAI/AAAAAAAAAAA/4252rscbv5M/photo.jpg","locale":"en"}

Data from /drive/v3/files

{ "kind": "drive#fileList", "files": [{ "kind": "drive#file", "id": "0B48SPOlG70UFYTZGTElmQklUMlk", "name": "Hi_there.txt", "mimeType": "text/plain" }, { "kind": "drive#file", "id": "0B48SPOlG70UFc3RhcnRlcl9maWxl", "name": "Getting started", "mimeType": "application/pdf" }] }

Data from /calendar/v3/users/me/calendarList

{ "kind": "calendar#calendarList", "etag": "\"1463437202338000\"", "nextSyncToken": "CNDh4rTQ38wCEhVpc2MuZ3MuMjAxNkBnbWFpbC5jb20=", "items": [{ "kind": "calendar#calendarListEntry", "etag": "\"1458124936568000\"", "id": "isc.gs.2016@gmail.com", "summary": "isc.gs.2016@gmail.com", "timeZone": "Europe/Prague", "colorId": "14", "backgroundColor": "#9fe1e7", "foregroundColor": "#000000", "selected": true, "accessRole": "owner", "defaultReminders": [{ "method": "popup", "minutes": 30 }], "notificationSettings": { "notifications": [{ "type": "eventCreation", "method": "email" }, { "type": "eventChange", "method": "email" }, { "type": "eventCancellation", "method": "email" }, { "type": "eventResponse", "method": "email" }] }, "primary": true }, { "kind": "calendar#calendarListEntry", "etag": "\"1458124893504000\"", "id": "#contacts@group.v.calendar.google.com", "summary": "Birthdays", "description": "Displays birthdays of people in Google Contacts and optionally 'Your Circles' from Google+. Also displays anniversary and other event dates from Google Contacts, if applicable.", "timeZone": "Europe/Prague", "colorId": "13", "backgroundColor": "#92e1c0", "foregroundColor": "#000000", "accessRole": "reader", "defaultReminders": [] }, { "kind": "calendar#calendarListEntry", "etag": "\"1458124776219000\"", "id": "en.czech#holiday@group.v.calendar.google.com", "summary": "Holidays in Czech Republic", "description": "Holidays and Observances in Czech Republic", "timeZone": "Europe/Prague", "colorId": "8", "backgroundColor": "#16a765", "foregroundColor": "#000000", "selected": true, "accessRole": "reader", "defaultReminders": [] }] }

Continúe con la parte 2, en la que se describe cómo actúa InterSystems IRIS como un servidor de autorización y como un proveedor de OpenID Connect.

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

#Autenticación #Control de acceso #OAuth2 #Seguridad #Caché #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/implementaci%C3%B3n-de-la-estructura-open-authorization-oauth-20-en-intersystems-iris-parte-1