Article
Dmitrii Kuznetsov · Oct 7, 2019 12m read

Open Exchange

OAuth Authorization and InterSystems IRIS: Taming Trust Protocols

How can you allow computers to trust one another in your absence while maintaining security and privacy?

“ A Dry Martini ”, he said. “ One. In a deep champagne goblet. ”
“ Oui, monsieur. ”
“ Just a moment. Three measures of Gordons, one of vodka, half a measure of Kina Lillet. Shake it very well until it ’ s ice-cold, then add a large thin slice of lemon peel. Got it? ”
"Certainly, monsieur." The barman seemed pleased with the idea.
Casino Royale, Ian Fleming, 1953

OAuth helps to separate services with user credentials from “ working ” databases, both physically and geographically. It thereby strengthens the protection of identification data and, if necessary, helps you comply with the requirements of countries' data protection laws.

With OAuth, you can provide the user with the ability to work safely from multiple devices at once, while "exposing" personal data to various services and applications as little as possible. You can also avoid taking on "excess" data about users of your services (i.e. you can process data in a depersonalized form).

If you use Intersystems IRIS, you get a complete set of ready-made tools for testing and deploying OAuth and OIDC services, both autonomously and in cooperation with third-party software products.

OAuth 2.0 and OpenID Connect

OAuth and OpenID Connect — known as OIDC or simply OpenID — serve as a universal combination of open protocols for delegating access and identification — and in the 21st century, it seems to be a favorite. No one has come up with a better option for large-scale use. It's especially popular with frontenders because it sits on top of HTTP(S) protocols and uses a JWT (JSON Web Token) container.

OpenID works using OAuth — it is, in fact, a wrapper for OAuth. Using OpenID as an open standard for the authentication and creation of digital identification systems is nothing new for developers. As of 2019, it is in its 14th year (and its third version). It is popular in web and mobile development and in enterprise systems.

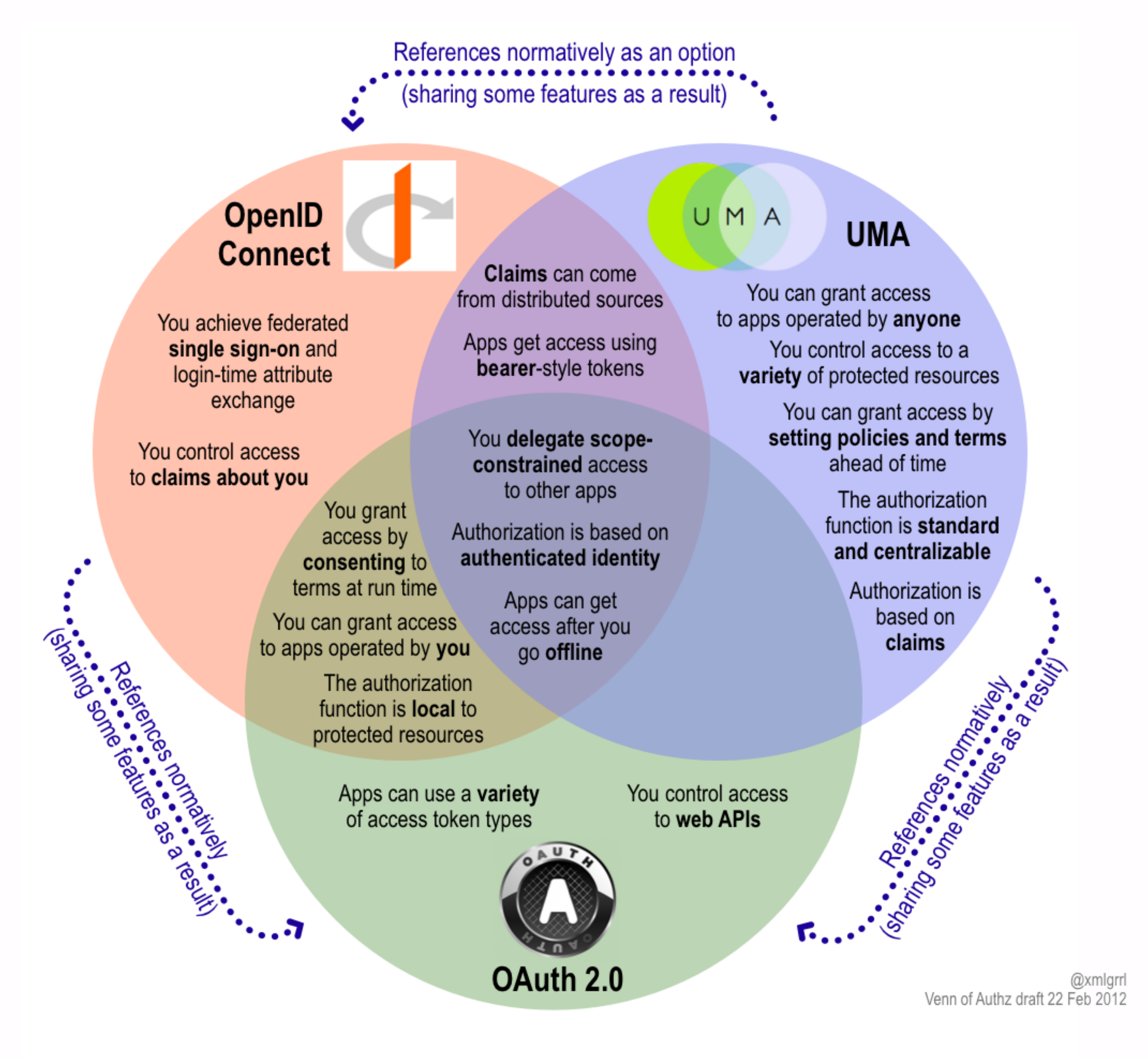
Its partner, the OAuth open standard for delegating access, is 12 years old, and it's been nine years since the relevant RFC 5849 standard appeared. For the purposes of this article, we will rely on the current version of the protocol, OAuth 2.0, and the current RFC 6749. (OAuth 2.0 is not compatible with its predecessor, OAuth 1.0.)

Strictly speaking, OAuth is not a protocol, but a set of rules (a scheme) for separating and transferring user identification operations to a separate trusted server when implementing an access-rights restriction architecture in software systems.

Be aware: OAuth can't say anything about a specific user! Who the user is, or where the user is, or even whether the user is currently at a computer or not. But OAuth makes it possible to interact with systems without user participation, using pre-issued access tokens. This is an important point (see "User Authentication with OAuth 2.0" on the OAuth site for more information).

The User-Managed Access (UMA) protocol is also based on OAuth. Using OAuth, OIDC and UMA together make it possible to implement a protected identity and access management (IdM, IAM) system in areas such as:

- Using a patient's HEART (Health Relationship Trust) personal data profile in medicine.
- Consumer Identity and Access Management (CIAM) platforms for manufacturing and trading companies.
- Personalizing digital certificates for smart devices in IoT (Internet of Things) systems using the OAuth 2.0 Internet of Things (IoT) Client Credentials Grant.


References normatively as an option
(sharing some features as a result)
OpenID Connect
UMA
You achieve federated **single sign-on** and login-time attribute exchange
You control access to **claims about you**
Claims can come from distributed sources
Apps get access using **bearer**-style tokens
You can grant access to apps operated by **anyone**
You control access to a **variety** of protected resources
You can grant access by **setting policies and terms** ahead of time
The authorization function is **standard and centralizable**
Authorization is based on **claims**
You **delegate scope-constrained** access to other apps
Authorization is based on **authenticated identity**
Apps can get access after you go **offline**
You grant access by **consenting** to terms at run time
You can grant access to apps operated by **you**
The authorization function is **local** to protected resources
Apps can use a **variety** of access token types
You control access to **web APIs**
OAuth 2.0
References normatively (sharing some features as a result)
References normatively (sharing some features as a result)
@xmlgrrl
Venn of Authz draft 22 Feb 2012

[A New Venn Of Access Control For The API Economy](https://community.intersystems.com)

Above all, do not store personal data in the same place as the rest of the system. Separate authentication and authorization physically. And ideally, give the identification and authentication to the individual person. Never store them yourself. Trust the owner's device.

Trust and Authentication

It is not a best practice to store users' personal data either in one ’ s own app or in a combined storage location along with a working database. In other words, we choose someone we trust to provide us with this service.

It is made up of the following parts:

- The user
- The client app
- The identification service
- The resource server

The action takes place in a web browser on the user's computer. The user has an account with the identification service. The client app has a signed contract with the identification service and reciprocal interfaces. The resource server trusts the identification service to issue access keys to anyone it can identify.

The user runs the client web app, requesting a resource. The client app must present a key to that resource to gain access.
If the user doesn ’ t have a key, then the client app connects with an identification service with which it has a contract for issuing keys to the resource server (passing the user on to the identification service).

The Identification Service asks what kind of keys are required.

The user provides a password to access the resource. At this point, the user has been authenticated and identification of the user has been confirmed, thus providing the key to the resource (passing the user back to the client app), and the resource is made available to the user.

Implementing an Authorization Service

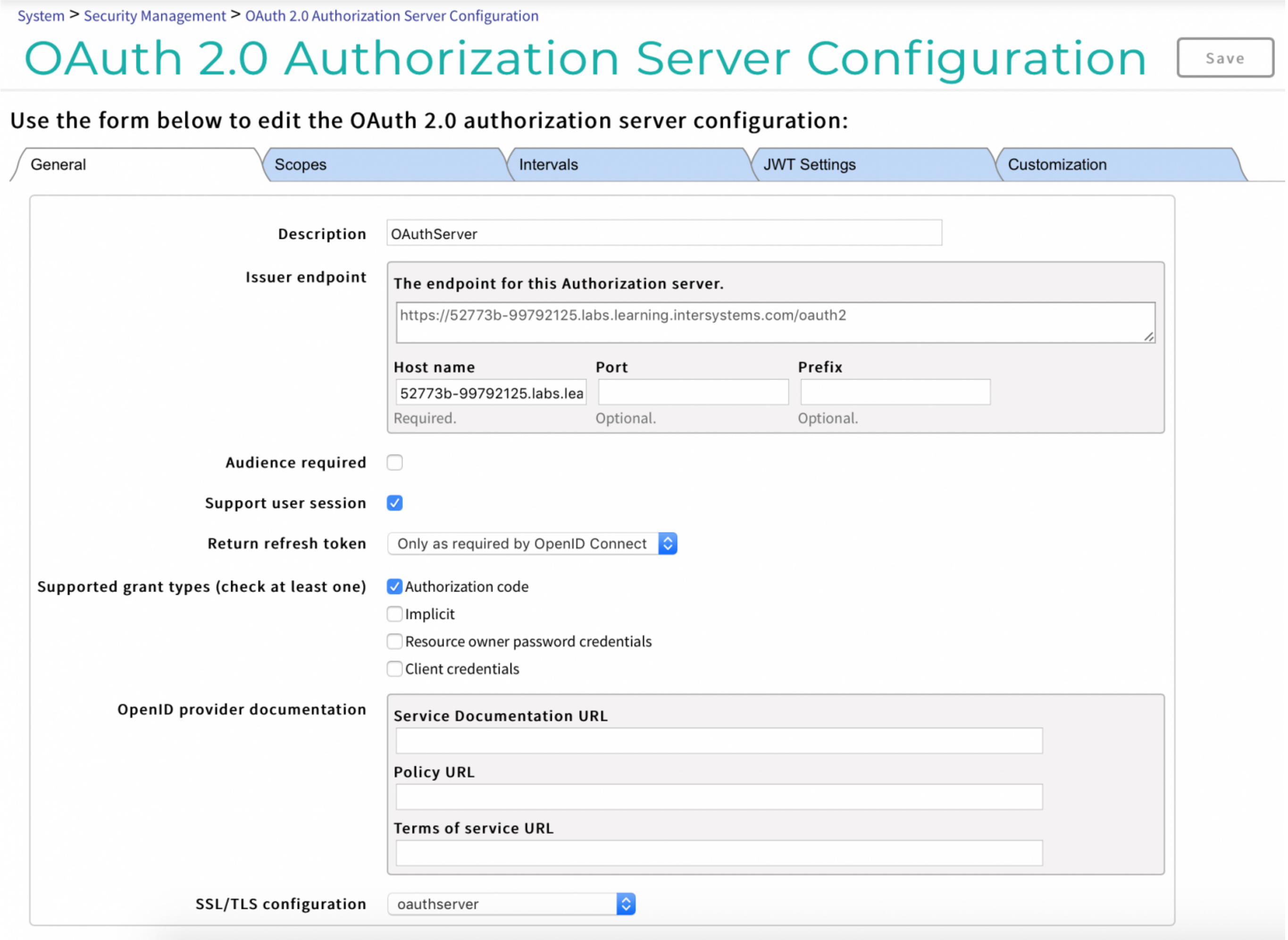
On the Intersystems IRIS platform, you can assemble a service from different platforms as needed. For example:

1. Configure and launch an OAuth server with the demo client registered on it.
2. Configure a demo OAuth client by associating it with an OAuth server and web resources.
3. Develop client apps that can use OAuth. You can use Java, Python, C#, or NodeJS. Below is an example of the application code in ObjectScript.

There are multiple settings in OAuth, so checklists can be helpful. Let's walk through an example. Go to the IRIS management portal and select the section System Administration > Security > OAuth 2.0 > Server.

Each item will then contain the name of a settings line and a colon, followed by an example or explanation, if necessary. As an alternative, you can use the screenshot hints in Daniel Kutac's three-part article, InterSystems IRIS Open Authorization Framework (OAuth 2.0) implementation - part 1, part 2, and part 3.

Note that all of the following screenshots are meant to serve as examples. You ’ ll likely need to choose different options when creating your own applications.


System > Security Management > OAuth 2.0 Authorization Server Configuration
Save
General
Scopes
Intervals
JWT Settings
Customization

OAuth 2.0 Authorization Server Configuration

Use the form below to edit the OAuth 2.0 authorization server configuration:

Description: OAuthServer

Issuer endpoint: The endpoint for this Authorization server.
https://52773b-99792125.labs.learning.intersystems.com/oauth2

Host name	Port	Prefix
52773b-99792125.labs.lea		
Required.	Optional.	Optional.

Audience required ☐

Support user session ☑

Return refresh token: Only as required by OpenID Connect

Supported grant types (check at least one)
☑ Authorization code
☐ Implicit
☐ Resource owner password credentials
☐ Client credentials

OpenID provider documentation
Service Documentation URL:
Policy URL:
Terms of service URL:

SSL/TLS configuration: oauthserver

On the General Settings tab, use these settings:

- Description: provide a description of the configuration, such as "Authorization server".
- The endpoint of the generator (hereinafter EPG) host name: DNS name of your server.
- Supported permission types (select at least one):
  - Authorization code
  - Implicit
  - Account details: Resource, Owner, Password
  - Client account details
- SSL/TLS configuration: oauthserver

On the Scopes tab:

- Add supported scopes: scope1 in our example

On the Intervals tab:

- Access Key Interval: 3600
- Authorization Code Interval: 60
- Update Key Interval: 86400
- Session Interruption Interval: 86400
- Validity period of the client key (client secret): 0

On the JWT Settings tab:

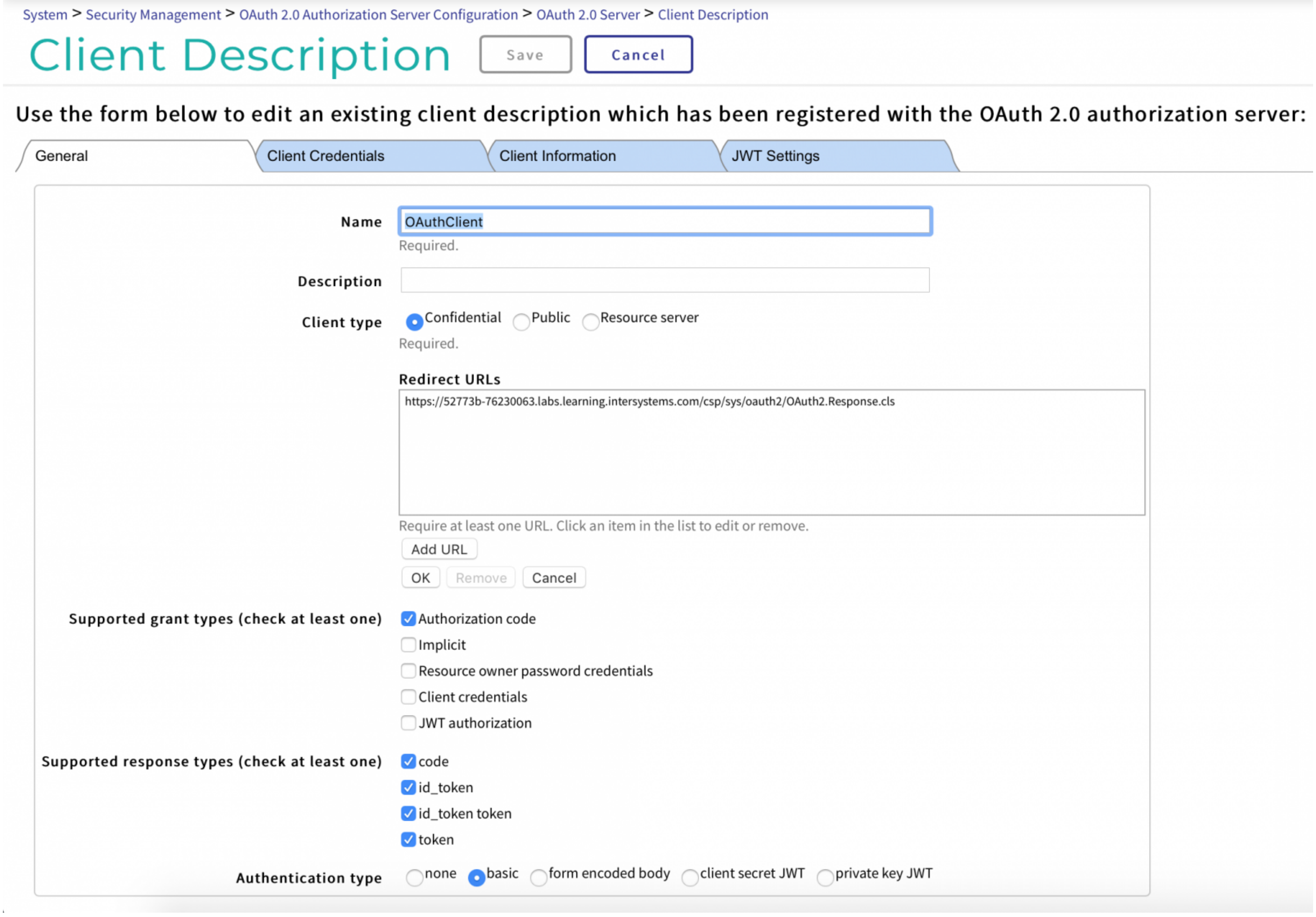
- Entry algorithm: RS512
- Key Management Algorithm: RSA-OAEP
- Content Encryption Algorithm: A256CBC-HS512

On the Customization tab:

- Identify Class: %OAuth2.Server.Authenticate
- Check User Class: %OAuth2.Server.Validate
- Session Service Class: OAuth2.Server.Session
- Generate Key Class: %OAuth2.Server.JWT
- Custom Namespace: %SYS
- Customization Roles (select at least one): %DBIRISSYS and %Manager

Now save the changes.

The next step is registering the client on the OAuth server. Click the Customer Description button, then click Create Customer Description.


System > Security Management > OAuth 2.0 Authorization Server Configuration > OAuth 2.0 Server > Client Description
Client Description
Save
Cancel
General
Client Credentials
Client Information
JWT Settings
Add URL
OK
Remove
Cancel

Use the form below to edit an existing client description which has been registered with the OAuth 2.0 authorization server:

Name: OAuthClient
Required.

Description:

Client type: ● Confidential ○ Public ○ Resource server
Required.

Redirect URLs
https://52773b-76230063.labs.learning.intersystems.com/csp/sys/oauth2/OAuth2.Response.cls
Require at least one URL. Click an item in the list to edit or remove.

Supported grant types (check at least one)
☑ Authorization code
□ Implicit
□ Resource owner password credentials
□ Client credentials
□ JWT authorization

Supported response types (check at least one)
☑ code
☑ id_token
☑ id_token token
☑ token

Authentication type: ○ none ● basic ○ form encoded body ○ client secret JWT ○ private key JWT

On the General Settings tab, enter the following information:

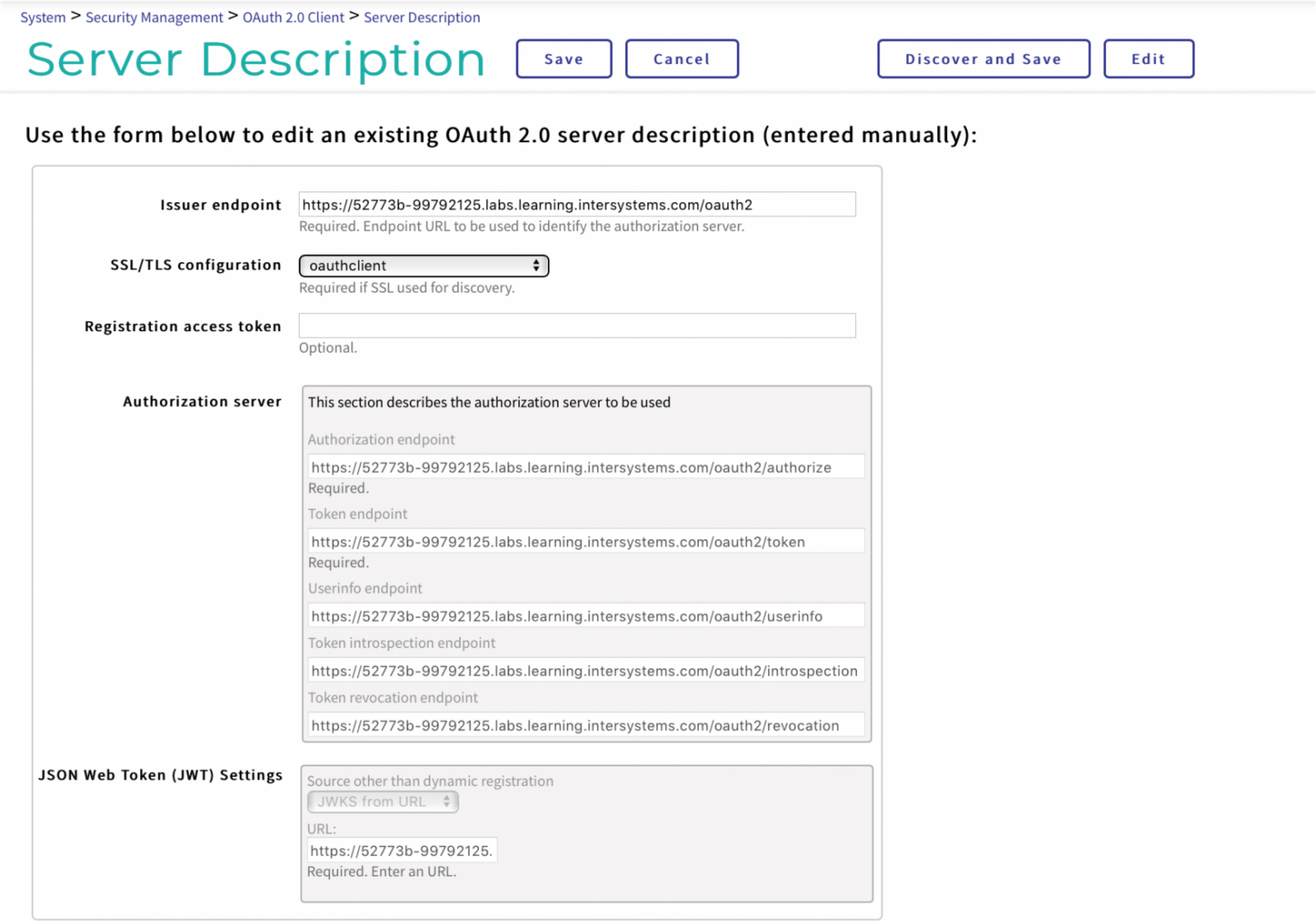
- Name: OAuthClient
- Description: provide a brief description
- Client Type: Confidential
- Redirect URLs: the address of the point to return to the app after identification from oauthclient.
- Supported grant types:

  - Authorization code: yes
  - Implicit
  - Account details: Resource, Owner, Password
  - Client account details
  - JWT authorization
- Supported response types: Select all of the following:
  - code
  - id_token
  - id_token key
  - token
- Authorization type: Simple

The Client Account Details tab should be auto-completed, but ensure the information here is correct for the client. On the Client Information tab:

- Authorization screen:
  - Client name
  - Logo URL
  - Client homepage URL
  - Policy URL
  - Terms of Service URL

Now configure the binding on the OAuth server client by going to System Administration > Security > OAuth 2.0 > Client.



System > Security Management > OAuth 2.0 Client > Server Description
Server Description
Save
Cancel
Discover and Save
Edit

Use the form below to edit an existing OAuth 2.0 server description (entered manually):

Issuer endpoint: https://52773b-99792125.labs.learning.intersystems.com/oauth2
Required. Endpoint URL to be used to identify the authorization server.

SSL/TLS configuration: oauthclient
Required if SSL used for discovery.

Registration access token:
Optional.

Authorization server
This section describes the authorization server to be used

Authorization endpoint
https://52773b-99792125.labs.learning.intersystems.com/oauth2/authorize
Required.

Token endpoint
https://52773b-99792125.labs.learning.intersystems.com/oauth2/token
Required.

Userinfo endpoint
https://52773b-99792125.labs.learning.intersystems.com/oauth2/userinfo

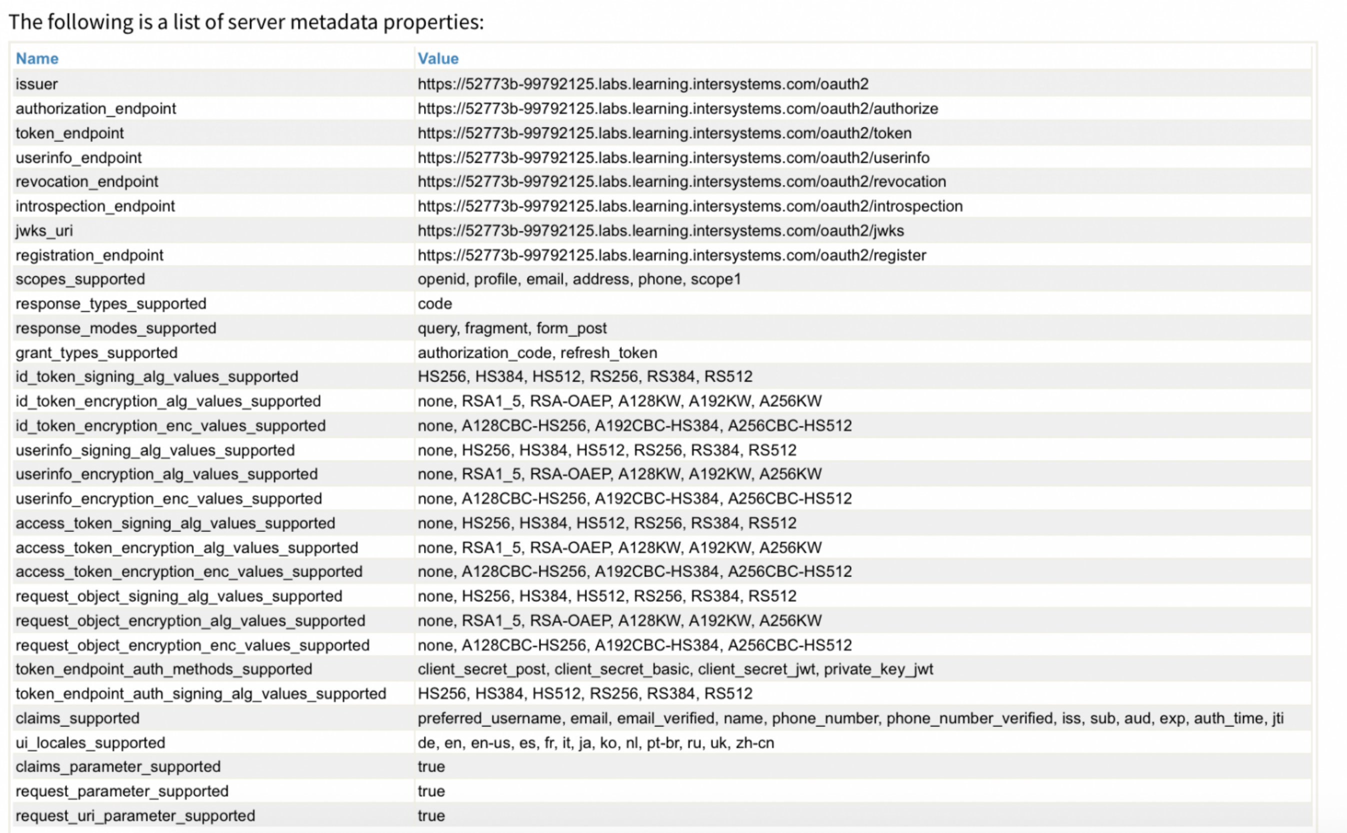
Token introspection endpoint
https://52773b-99792125.labs.learning.intersystems.com/oauth2/introspection

Token revocation endpoint
https://52773b-99792125.labs.learning.intersystems.com/oauth2/revocation

JSON Web Token (JWT) Settings
Source other than dynamic registration
JWKS from URL
URL:
https://52773b-99792125.
Required. Enter an URL.

Create a Server Description:

- The endpoint of the generator: taken from the general parameters of the server (see above).
- SSL/TLS configuration: choose from the preconfigured list.
- Authorization server:
  - Authorization endpoint: EPG + /authorize
  - Key endpoint: EPG + /token
  - User endpoint: EPG + /userinfo
  - Key self-test endpoint: EPG + /revocation
  - Key termination endpoint: EPG + /introspection
- JSON Web Token (JWT) settings:
  - Other source besides dynamic registration: choose JWKS from URL
  - URL: EPG + /jwks



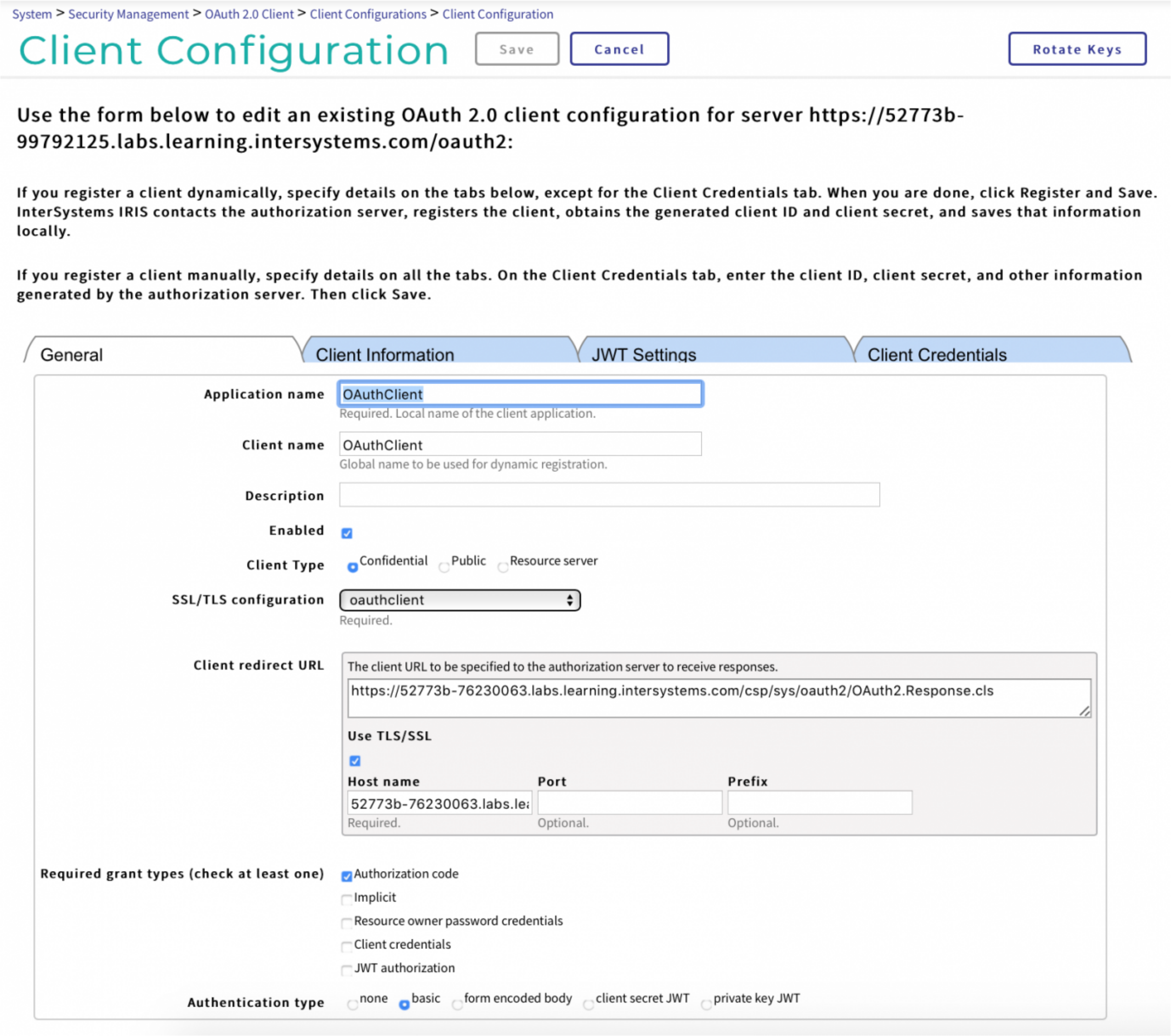
The following is a list of server metadata properties:

Name	Value
issuer	https://52773b-99792125.labs.learning.intersystems.com/oauth2
authorization_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/authorize
token_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/token
userinfo_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/userinfo
revocation_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/revocation
introspection_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/introspection
jwks_uri	https://52773b-99792125.labs.learning.intersystems.com/oauth2/jwks
registration_endpoint	https://52773b-99792125.labs.learning.intersystems.com/oauth2/register
scopes_supported	openid, profile, email, address, phone, scope1
response_types_supported	code
response_modes_supported	query, fragment, form_post
grant_types_supported	authorization_code, refresh_token
id_token_signing_alg_values_supported	HS256, HS384, HS512, RS256, RS384, RS512
id_token_encryption_alg_values_supported	none, RSA1_5, RSA-OAEP, A128KW, A192KW, A256KW
id_token_encryption_enc_values_supported	none, A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
userinfo_signing_alg_values_supported	none, HS256, HS384, HS512, RS256, RS384, RS512
userinfo_encryption_alg_values_supported	none, RSA1_5, RSA-OAEP, A128KW, A192KW, A256KW
userinfo_encryption_enc_values_supported	none, A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
access_token_signing_alg_values_supported	none, HS256, HS384, HS512, RS256, RS384, RS512
access_token_encryption_alg_values_supported	none, RSA1_5, RSA-OAEP, A128KW, A192KW, A256KW
access_token_encryption_enc_values_supported	none, A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
request_object_signing_alg_values_supported	none, HS256, HS384, HS512, RS256, RS384, RS512
request_object_encryption_alg_values_supported	none, RSA1_5, RSA-OAEP, A128KW, A192KW, A256KW
request_object_encryption_enc_values_supported	none, A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
token_endpoint_auth_methods_supported	client_secret_post, client_secret_basic, client_secret_jwt, private_key_jwt
token_endpoint_auth_signing_alg_values_supported	HS256, HS384, HS512, RS256, RS384, RS512
claims_supported	preferred_username, email, email_verified, name, phone_number, phone_number_verified, iss, sub, aud, exp, auth_time, jti
ui_locales_supported	de, en, en-us, es, fr, it, ja, ko, nl, pt-br, ru, uk, zh-cn
claims_parameter_supported	true
request_parameter_supported	true
request_uri_parameter_supported	true

From this list, for example, you can see (scopessupported and claimssupported) that the server can provide the OAuth-client with different information about the user. And it's worth noting that when implementing your application, you should ask the user what data they are ready to share. In the example below, we will only ask for permission for scope1.

Now save the configuration.

If there is an error indicating the SSL configuration, then go to Settings > System Administration > Security > SSL/TSL Configurations and remove the configuration.


System > Security Management > OAuth 2.0 Client > Client Configurations > Client Configuration
Save
Cancel
Rotate Keys
General
Client Information
JWT Settings
Client Credentials

Client Configuration

Use the form below to edit an existing OAuth 2.0 client configuration for server https://52773b-99792125.labs.learning.intersystems.com/oauth2:

If you register a client dynamically, specify details on the tabs below, except for the Client Credentials tab. When you are done, click Register and Save. InterSystems IRIS contacts the authorization server, registers the client, obtains the generated client ID and client secret, and saves that information locally.

If you register a client manually, specify details on all the tabs. On the Client Credentials tab, enter the client ID, client secret, and other information generated by the authorization server. Then click Save.

Application name: OAuthClient
Required. Local name of the client application.

Client name: OAuthClient
Global name to be used for dynamic registration.

Description:

Enabled: ☑

Client Type: ● Confidential ○ Public ○ Resource server

SSL/TLS configuration: oauthclient
Required.

Client redirect URL: The client URL to be specified to the authorization server to receive responses.
https://52773b-76230063.labs.learning.intersystems.com/csp/sys/oauth2/OAuth2.Response.cls

Use TLS/SSL ☑

Host name	Port	Prefix
52773b-76230063.labs.le		
Required.	Optional.	Optional.

Required grant types (check at least one):
☑ Authorization code
□ Implicit
□ Resource owner password credentials
□ Client credentials
□ JWT authorization

Authentication type: ○ none ● basic ○ form encoded body ○ client secret JWT ○ private key JWT

Now we're ready to set up an OAuth client:
System Administration > Security > OAuth 2.0 > Client > Client configurations > Create Client configurations
On the General tab, use these settings:

- Application Name: OAuthClient
- Client Name: OAuthClient
- Description: enter a description
- Enabled: Yes
- Client Type: Confidential
- SSL/TCL configuration: select oauthclient
- Client Redirect URL: the DNS name of your server
- Required Permission Types:
  - Authorization code: Yes
  - Implicit
  - Account details: Resource, Owner, Password
  - Client account details
  - JWT authorization
- Authorization type: Simple

On Client Information tab:

- Authorization screen:
    - Logo URL
    - Client homepage URL
    - Policy URL
    - Terms of Service URL
- Default volume: taken from those specified earlier on the server (for example, scope1)
- Contact email addresses: enter addresses, separated by commas
- Default max age (in seconds): maximum authentication age or omit this option

On the JWT Settings tab:

- JSON Web Token (JWT) settings
- Creating JWT settings from X509 account details
- IDToken Algorithms:
    - Signing: RS256
    - Encryption: A256CBC
    - Key: RSA-OAEP
- Userinfo Algorithms
- Access Token Algorithms
- Query Algorithms

On the Client Credentials tab:

- Client ID: as issued when the client registered on the server (see above).
- Client ID Issued: isn't filled in
- Client secret: as issued when the client registered on the server (see above).
- Client Secret Expiry Period: isn't filled in
- Client Registration URI: isn't filled in

Save the configuration.

Web app with OAuth authorization

OAuth relies on the fact that the communication channels between the interaction participants (server, clients, web application, user's browser, resource server) are somehow protected. Most often this role is played by protocols SSL/TLS. But OAuth will work and on unprotected channels. So, for example, server Keycloak, by default uses HTTP protocol and does without protection. It simplifies working out and debugging at working out. At real use of services, OAuth protection of channels should be included strictly obligatory is written down in the documentation Keycloak. Developers InterSystems IRIS adhere to a more strict approach for OAuth - use SSL/TSL is obligatory. The only simplification - you can use the self-signed certificates or take advantage of built-in IRIS service PKI (System administration >> Security >> Public key system).

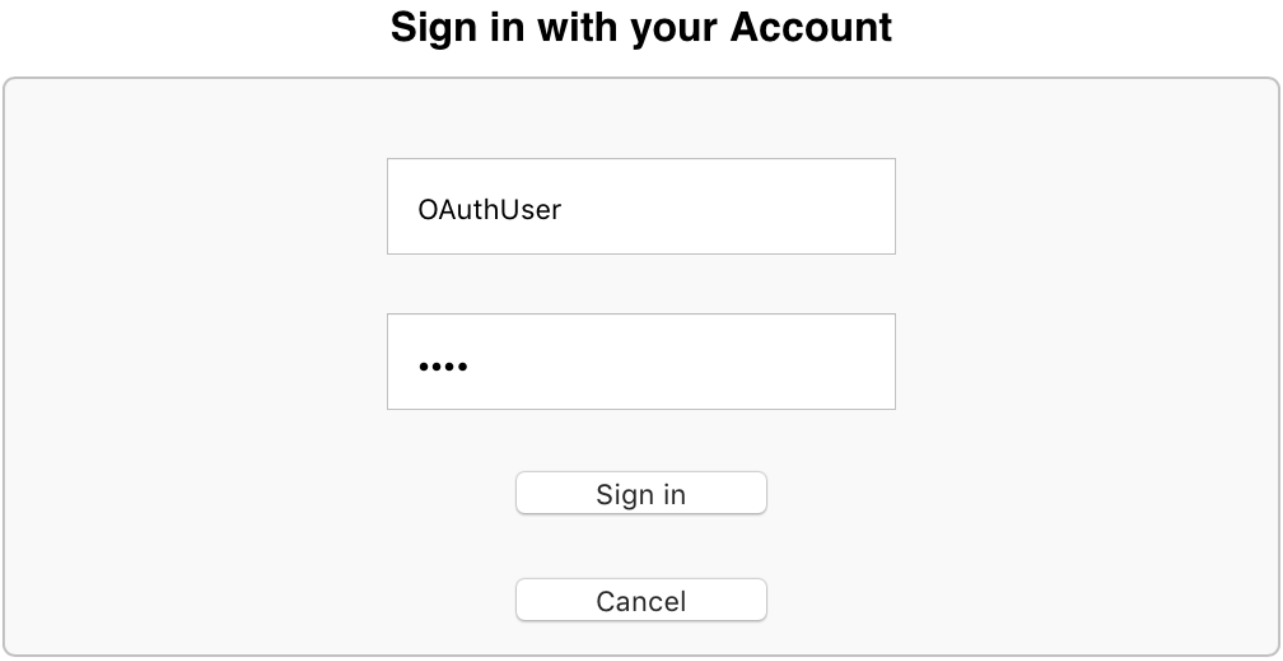
Verification of the user's authorization is made with the explicit indication of two parameters - the name of your application registered on the OAuth server, and in the OAuth client scope.

```
Parameter OAUTH2APPNAME = "OAuthClient";
set isAuthorized = ##class(%SYS.OAuth2.AccessToken).IsAuthorized(
..#OAUTH2APPNAME,
.sessionId,
"scope1",
.accessToken,
.idtoken,
.responseProperties,
.error)
```

In the lack of authorization, we prepare a link to the request for user identification and obtaining permission to work with our application. Here we need to specify not only the name of the application registered on the OAuth server and in the OAuth client and the requested volume (scope) but also the backlink to which point of the web application to return the user.

```
Parameter OAUTH2CLIENTREDIRECTURI = "https://52773b-76230063.labs.learning.intersyste
ms.com/oauthclient/"
set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
..#OAUTH2APPNAME,
"scope1",
..#OAUTH2CLIENTREDIRECTURI,
.properties,
.isAuthorized,
.sc)
```

We use IRIS and register users on the IRIS OAuth server. For example it is enough to set to the user only a name and the password.
At transfer of the user under the received reference, the server will carry out the procedure of identification of the user and inquiry at it of the permissions for operation by the account data in the web application, and also will keep the result in itself in global OAuth2.Server.Session in the field %SYS:



Sign in with your Account

OAuthUser

••••

3. Demonstrate the data of an authorized user. If the procedures are successful, we have, for example, an access token. Let's get it:

```
set valid = ##class(%SYS.OAuth2.Validation).ValidateJWT(
.#OAUTH2APPNAME,
accessToken,
"scope1",
.aud,
.JWTJsonObject,
.securityParameters,
.sc
)
```

The full working code of the OAuth example:

```
Class OAuthClient.REST Extends %CSP.REST
{
Parameter OAUTH2APPNAME = "OAuthClient";
Parameter OAUTH2CLIENTREDIRECTURI = "https://52773b-76230063.labs.learning.intersyste
ms.com/oauthclient/";
// to keep sessionId
Parameter UseSession As Integer = 1;
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
    <Routes>
     <Route Method="GET" Url = "/" Call = "Do" />
    </Routes>
}
ClassMethod Do() As %Status
{
    // Check for accessToken
    set isAuthorized = ##class(%SYS.OAuth2.AccessToken).IsAuthorized(
     ..#OAUTH2APPNAME,
     .sessionId,
     "scope1",
     .accessToken,
     .idtoken,
     .responseProperties,
     .error)
    // to show accessToken
    if isAuthorized {
     set valid = ##class(%SYS.OAuth2.Validation).ValidateJWT(
      ..#OAUTH2APPNAME,
      accessToken,
      "scope1",
      .aud,
      .JWTJsonObject,
      .securityParameters,
      .sc
     )
     &html< Hello!<br> >
      w "You access token = ", JWTJsonObject.%ToJSON()
     &html< </html> >
     quit $$$OK
    }
    // perform the process of user and client identification and get accessToken
    set url = ##class(%SYS.OAuth2.Authorization).GetAuthorizationCodeEndpoint(
     ..#OAUTH2APPNAME,
     "scope1",
     ..#OAUTH2CLIENTREDIRECTURI,
     .properties,
     .isAuthorized,
     .sc)
    if $$$ISERR(sc) {
      w "error handling here"
```

```
       quit $$$OK
    }
    // url magic correction: change slashes in the query parameter to its code
    set urlBase = $PIECE(url, "?")
    set urlQuery = $PIECE(url, "?", 2)
    set urlQuery = $REPLACE(urlQuery, "/", "%2F")
    set url = urlBase _ "?" _ urlQuery
 &html<
     <html>
      <h1>Authorization in IRIS via OAuth2</h1>
      <a href = "#(url)#">Authorization in <b>IRIS</b></a>
     </html>
 >
    quit $$$OK
}
}
```

You can also find a working copy of the code on the InterSystems GitHub repository: https://github.com/intersystems-community/iris-oauth-example.

If necessary, enable the advanced debug message mode on the OAuth server and OAuth client, which are written to the ISCLOG global in the %SYS area:

```
set ^%ISCLOG = 5
set ^%ISCLOG("Category", "OAuth2") = 5
set ^%ISCLOG("Category", "OAuth2Server") = 5
```

For more details, see the IRIS Using OAuth 2.0 and OpenID Connect documentation.

Conclusion

As you've seen, all OAuth features are easily accessible and completely ready to use. If necessary, you can replace the handler classes and user interfaces with your own. You can configure the OAuth server and the client settings from configuration files instead of using the management portal.

#JSON #OAuth2 #Security #Tutorial #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/oauth-authorization-and-intersystems-iris-taming-trust-protocols