Sommaire

[Terminologies 4](#_Toc460858193)

[Benchmark 5](#_Toc460858194)

[Architecture 6](#_Toc460858195)

[Use cases of the website 8](#_Toc460858196)

[Protect resources 9](#_Toc460858197)

[Action : Move resources to sub directory 11](#_Toc460858198)

[Action : Remove one or more resources 11](#_Toc460858199)

[Action : Rename a resource 11](#_Toc460858200)

[Action : Move resources 11](#_Toc460858201)

[Action : Duplicate one or more resources 11](#_Toc460858202)

[Action : How to access to a resource 12](#_Toc460858203)

[Action : Create a file 12](#_Toc460858204)

[Action : Create a folder 12](#_Toc460858205)

[Action : Upload one or more resources 12](#_Toc460858206)

[Action : Refresh current working directory 12](#_Toc460858207)

[Action : Display keyboard shortcuts 13](#_Toc460858208)

[Action : Change visualization 13](#_Toc460858209)

[Action : Search resources in the current working directory 13](#_Toc460858210)

[Action : Navigate to the previous / next folder 13](#_Toc460858211)

[Action : Add / Edit authorization rules 13](#_Toc460858212)

[Action : Select all resources 16](#_Toc460858213)

[Edit UMA & OPENID assets 17](#_Toc460858214)

[Action : Add a client 18](#_Toc460858215)

[Action : Delete one or more clients 18](#_Toc460858216)

[Action : Edit a client 18](#_Toc460858217)

[Action : Delete one or more scopes 20](#_Toc460858218)

[Action : Display scope information 20](#_Toc460858219)

[Action : Delete one or more resource owners 20](#_Toc460858220)

[Action : Edit a resource owner 20](#_Toc460858221)

[Action : Delete one or more authorization policies 22](#_Toc460858222)

[Action : Display authorization policy 22](#_Toc460858223)

[Action : Delete one or more UMA resources 22](#_Toc460858224)

[Action : Display UMA resource 22](#_Toc460858225)

[Identity providers 24](#_Toc460858226)

[Action : Display providers 24](#_Toc460858227)

[Action : Enable / disable a provider 25](#_Toc460858228)

[Action : Add a provider 25](#_Toc460858229)

[Action : Edit provider 26](#_Toc460858230)

[Parameters 27](#_Toc460858231)

[Use cases of the Visual Studio extension 28](#_Toc460858232)

[Configurer les paramètres 28](#_Toc460858233)

[Générer une ou plusieurs ressources 28](#_Toc460858234)

[Rafraîchir la liste des ressources 29](#_Toc460858235)

[Protéger une ou plusieurs ressources 29](#_Toc460858236)

[Générer un proxy de sécurité 30](#_Toc460858237)

[Action : Rechercher une ou plusieurs ressources 30](#_Toc460858238)

[Action : Ajouter le proxy de sécurité 31](#_Toc460858239)

[Scénarios 32](#_Toc460858240)

[Premier scénario : Application lourde souhaite accéder à une opération protégée 32](#_Toc460858241)

[Identifier et catégoriser les entités 33](#_Toc460858242)

[Ajouter un client 34](#_Toc460858243)

[Ajouter une ressource 37](#_Toc460858244)

[Ajouter la politique d’autorisation 37](#_Toc460858245)

[Ajouter le rôle marketing au resource owner 38](#_Toc460858246)

[Développer 38](#_Toc460858247)

[Second scénario : une API souhaite accéder à une ressource protégée 41](#_Toc460858248)

[Identifier et catégoriser les entités 42](#_Toc460858249)

[Ajouter un client 42](#_Toc460858250)

[Ajouter une ressource 42](#_Toc460858251)

[Ajouter la politique d’autorisation 43](#_Toc460858252)

[Développer 43](#_Toc460858253)

[Troisième scénario : Limiter l’accès à certaines fonctionnalités du site 45](#_Toc460858254)

[Identifier et catégoriser les entités 46](#_Toc460858255)

[Ajouter un client 46](#_Toc460858256)

[Ajouter une ressource 47](#_Toc460858257)

[Ajouter la politique d’autorisation 47](#_Toc460858258)

[Ajouter le rôle « administrator » au resource owner 48](#_Toc460858259)

[Développer 48](#_Toc460858260)

[Installation 50](#_Toc460858261)

[Etapes obligatoires 50](#_Toc460858262)

[Installer GIT 50](#_Toc460858263)

[Installer le certificat 50](#_Toc460858264)

[Installation manuelle 51](#_Toc460858265)

[Installer avec docker 51](#_Toc460858266)

[Prérequis 51](#_Toc460858267)

[Lancer le déploiement 52](#_Toc460858268)

[Documentation technique 53](#_Toc460858269)

[Monitoring 53](#_Toc460858270)

[Nuget packages 53](#_Toc460858271)

[SimpleIdentityServer.Proxy 53](#_Toc460858272)

[SimpleIdentityServer.UmaIntrospection.Authentication 54](#_Toc460858273)

[SimpleIdentityServer.Uma.Authorization 54](#_Toc460858274)

[Load balancing 55](#_Toc460858275)

[Reste à faire 56](#_Toc460858276)

# Terminologies

|  |  |
| --- | --- |
| Words | Definitions |
| Resource | It can be anything that needs to be protected for example : a picture of a user or an API operation. |
| Folder | Resource which can contains one or more resources. |
| File | Unique resource. |
| Authorization policy | Can contains one or more security rules and is assigned to one or several resources. It is used by the UMA server to determine if an incoming request can execute the requesting operations (read, write or delete) on a protected resource. |
| Security rule | Belongs to an authorization policy. |
| Resource owner | An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user. |
| Scope | List of resources which can be accessed by a client. |
| Client | An application making protected resource requests on behalf of the resource owner and with its authorization. The term “client” does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices) |
| UMA server | Server which is conformed to the OPENID-RFC [[1]](#footnote-1) |
| OPENID server | Server which is conformed to the UMA-RFC [[2]](#footnote-2) |
| Requesting Party Token (RPT) | An UMA access token associated with a set of authorization data, used by the client to gain access to protected resources at the UMA server. |

# Benchmark

The following table lists the differences between our product and others: Lokit, Identity Server, Gluu server and AUTH0. It has been made in “29-08-2016”, if you noticed some differences don’t hesitate to contact-us by email.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Lokit** | **Identity Server** | **Gluu server** | **AUTH0** |
| Authors | Habart Thierry | Brock Allen & Dominick Baier | Gluu | Auth0 |
| Start date | October 2015 | January 2014 | March 2014 | November 2012 |
| Workflow Oauth2.0 | | | | |
| Client credentials | OK | OK | OK | OK |
| Password | OK | OK | OK | OK |
| Refresh token | OK | OK | OK | OK |
| Workflow OpenId | | | | |
| Implicit | OK | OK | OK | OK |
| Hybrid | OK | OK | OK | NOK |
| Other OPENID features | | | | |
| Register a client ([RFC](https://openid.net/specs/openid-connect-registration-1_0.html)) | OK | NOK | OK | NOK |
| Sign token ([JWS](https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41)) | OK | OK | OK | OK |
| Encrypt token ([JWE](https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-40)) | OK | NOK | NOK | NOK |
| Invalidate session | OK | OK | OK | NOK |
| Client authentication methods ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#ClientAuthentication)) | | | | |
| client\_secret\_basic | OK | OK | OK | OK |
| client\_secret\_post | OK | OK | OK | OK |
| client\_secret\_jwt | OK | NOK | NOK | NOK |
| private\_key\_jwt | OK | NOK | NOK | NOK |
| none | OK | NOK | NOK | NOK |
| Response modes | | | | |
| Query | OK | OK | OK | OK |
| Fragment | OK | OK | OK | OK |
| Form\_post | OK | OK | OK | OK |
| Other parameters | | | | |
| claims ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#ClaimsParameter)) | OK | OK | OK | NOK |
| request ([RFC](http://openid.net/specs/openid-connect-core-1_0.html#RequestObject)) | OK | OK | OK | NOK |
| Quality | | | | |
| Code coverage | 84% | Unknown | Unknown | Unknown |
| Nombre UTs | 633 | Unknown | Unknown | Unknown |
| UMA ([RFC](http://openid.net/specs/openid-heart-uma-2015-12-09.html)) | | | |  |
| UMA supported | OK | NOK | OK | NOK |
| UI | | | | |
| UI exists | OK | OK | OK | OK |
| CRUD opened assets | OK | OK | OK | OK |
| CRUD uma assets | OK | NOK | OK | NOK |
| Resource organized by urls | OK | NOK | NOK | NOK |
| Enable or disable external identity providers | OK | NOK | OK | OK |
| Deployments | | | | |
| Deployment methods | Docker or manually | Manually | Manually | Manually or hosted on the cloud |
| Others | | | | |
| Number of OPENID certifications | 5 | 4 | 5 | 2 |
| Preferred languages | C# | C# | Java | No preference |
| Tools or methods used to easily interact with APIs | Visual Studio extensions and nugget packages | Nuget packages | Unknown | Unknown |

# Architecture

The schema below shows the interactions between components.

Configuration API

Manager API

Uma API

Openid API

WebSite API

WebSite

UMA DB

OpenId DB

Configuration DB

Visual Studio extension

WebSite API DB

On a total of 11 components, there are 5 APIs, one visual studio extension, one website and four databases :

* Manager API : used by the clients to execute CRUD operations on OPENID assets for examples : client or resource owners.
* OpenId et UMA API : They are conformed to the RFCs OPENID and UMA.
* Configuration API : used by the clients to manage the OpenId API configuration for example : enable or disable external identity providers.
* WebSite API : an abstract layer which assigns Uris to resources.
* WebSite : it is used by an administrator to manage resource access.
* Visual studio extension : used by a .NET developer to easily interact with the different components in code.

The website is used by the administrator to manage the UMA and OPENID assets. Whereas the visual studio extension will be used by developers to access / protect resources such as API operations.

Use cases of both roles have been identified and they are described in the two next chapters.

# Use cases of the website

## Protect resources

<<include>>

<<include>>

<<include>>

<<include>>

<<include>>

Create a folder

Create a file

Move resources

Create a resource

Duplicate one or more resources

How to access to a resource

Rename a resource

<<include>>

Remove one or more resources

Move resources to sub directory

Administrator

Upload one or more resources

Display keyboard shortcuts

Search resources in the current working directory

Change visualization

Refresh current working directory

Administrator

Navigate to the previous / next folder

Add / Edit authorization rules

Select all resources

### Action: Move resources to sub directory

Select one or more resources and click on the item “” displayed in the context menu. Selected resources will be moved to a new sub directory.

### Action: Remove one or more resources

Select some resources and execute one of the following actions :

* Keyboard Shortcut : DEL
* Action *Delete* in the contextual menu 

### Action: Rename a resource

A resource can be renamed by executing one of the following actions:

* Keyboard Shortcut : F2
* Action *Rename* in in the contextual menu 

### Action: Move resources

Select several resources and drag & drop them into an existing folder. A special icon will be displayed in the bottom-right left corner.



### Action: Duplicate one or more resources

Select a resource and duplicate it by executing one of the actions :

* Display the context menu and click on the actions *copy* & *paste* or *duplicate*.



* Use the shortcuts “CTRL+C & CTRL+V“.

### Action: How to access to a resource

Select a resource and display its contextual menu. When the option “*how to access?*” is selected then a new window is displayed in front of the file explorer.



### Action: Create a file

Display the contextual menu of your current working directory and click on the option *new file* .

### Action: Create a folder

Do the same than before but instead of clicking on *new file* select the action *new folder* .

### Action: Upload one or more resources

Resources can be uploaded via different ways:

* Open a windows explorer and select several files, drag and drop them to the page “Protect your resources”.
* Click on the option upload files displayed in the contextual menu



### Action: Refresh current working directory

Either press F5 or click on the action *reload* in the contextual menu 

### Action: Display keyboard shortcuts

List of shortcuts can be displayed by pressing F1.



### Action: Change visualization

You can choose between vertical or horizontal resources visualization, to do that click on the icon  displayed in the menu bar.

### Action  Search resources in the current working directory

Accessible by pressing the keyboard shortcut CTRL+F or by typing your text in the search bar positioned at the top right-hand corner of the window.



### Action: Navigate to the previous / next folder

The navigation history is stored into your browser. Navigating to the previous / next folder can be done by executing one of the following actions :

* Use the buttons  displayed in the search bar
* Keyboard shortcut *Back* or *CTRL+←* : navigate to the previous folder
* Keyboard shortcut *CTRL+→* : navigate to the next folder.

### Action: Add / Edit authorization rules

If a resource is selected then it’s possible to add or edit its authorization rules. Click on the icon  or press the keyboard shortcut *CTRL+P*.

Rules and authorization policy were mentioned several times but they have never been clearly explained properly.

An authorization policy must contains at least one rule otherwise an error is displayed when attempting to create an empty one. They are used by the UMA server during the authorization process. Indeed when a client wants to access to a protected resource, he asked to the UMA resource an RPT token. The server internally decides to grant client access to a protected resource by executing the policy. The workflow is described in UMA website [[3]](#footnote-3).

Rules are exclusive and if at least one of them is satisfied then authorization is granted by the policy. Here the authorization policy formula is : . A rule is satisfied if the received claims values and / or clients are correct.

Policy structure overview :

Policy

In order to help you identifying policy and their rules, you can read the scenario below.

Problematic : Only the end-user thabart of the application Sample Client is allowed to view his bank account N°12345.

Solution : In the first place the identities should be identified and classified by their nature. There are four important information in the scenario, they are underlined in the text. Thanks to the decision table it was possible to classified them.

|  |  |
| --- | --- |
| Question | Nature |
| What is the identity of your resource to protect ? | Resource identifier |
| What is the identity of your client ? | Client Identifier |
| Which actions do-you want to perform on the resource ? | Permissions |
| Do-you want to restrict access to one or more users ? If yes, which information can be used to identify them ? For example : role or geographical position ? | Claims |
| Can-you identify the nature of your resource ? | Nature |

Classified information :

* Resource identifier : N°12345
* Client Id : Sample Client
* Permissions : view / read
* Claims : thabart
* Nature : BankAccount

When you have finished with the resource classification, you can go to the second step.

A resource needs to be created, first add a “BankAccount” folder into the “resources” directory, it should match with the nature of a resource.

Navigate to the new directory and create a file named “N°12345”, same value as the resource identifier.



Finally open the authorization policy editor and add a new rule. Below *allowed clients* select the authorized clients, in our case the client is “Sample Client”.

Under the sub-title “allowed claims” select “sub” et fill-in the field with “thabart”, then click on “Add” to add the claim into the list.

Persist your policy by clicking on *add rule* and on *save*.



A red square is displayed around a resource with an authorization policy applied on it.



### Action : Select all resources

To select all resources press *CTRL+A*.

You probably noticed that the actions available in the assets directory have not been described, it will be the object of the next chapter.

## Edit UMA & OPENID assets

<<include>>

<<include>>

Delete clients

Add a client

Administrator

Edit a client

Display scope information

Delete one or more scopes

Delete one or more resource owners

Edit resource owner

Delete or more authorization policies

Display authorization policy

Delete one or more UMA resources

Display UMA resource

### Action : Add a client

Open the folder *assets >> openid >> clients* and display its contextual menu.

Click on *add client * and fill-in the callback urls separated by a comma.

*Warning : urls should begin with https [[4]](#footnote-4)*

### Action : Delete one or more clients

Select several clients, display the contextual menu and click on *remove client*

.

### Action : Edit a client

Display the client information by clicking on *client information*  option in the contextual menu.



The client identity, secret and its callback urls are displayed on the new window.

Click on the link *Edit* to edit them.



The parameters are explained in the OPENID RFC [[5]](#footnote-5).

In most of the cases, advanced parameters don’t need to be updated except if you want to change the workflows. Here some uncommon scenarios in which advanced settings need to be updated:

* The client have a JWKS url, it is used by the OPENID server to decrypt and / or check the signature of the *request [[6]](#footnote-6)* parameter.
* Update the client authentication method.
* Change information displayed in the consent view.

If you want more information about the other advanced settings, you can contact-us or read the OPENID documentation[[7]](#footnote-7).

### Action : Delete one or more scopes

Open the directory *assets>>openid>>scopes*, select several scopes, display the context menu and click on remove scope .

### Action : Display scope information

Select one scope, open its contextual menu and click on *scope information* . The following information are displayed :

* Nature : Is-it an openid scope?
* Visibility : Is-it visible in the consent screen ?
* Contract : Is-is returned by the openid configuration endpoint */.well-known/openid-configuration*.

### Action : Delete one or more resource owners

Select some resource owners, display the context menu and click on remove resource owner .

### Action : Edit a resource owner

Select a resource owner, display its context menu and click on *resource owner info* button .

In the new window displayed in front, roles can be assigned to the resource owner. To do that, a local account needs to be created otherwise the message “not a local account” is displayed.

Why is-it useful to add roles ? Sometime it’s difficult to make an authorization policy based on the claims returned by external identity providers. If you want to limit the access of a resource only to certain roles but the claims are coming from facebook, the solution cannot be implemented simply because roles are not returned. But thanks to the website, claims returned by facebook can be enriched with roles. If we replace facebook by ADFS, additional roles don’t need to be specified because they are already returned.

Enrichment workflow :

Facebook

Hotmail

OPENID server

Client

Claims

Enriched claims

As we mentioned earlier a resource owner account can have two states :

External

Internal

Create a local account

Confirm his local account

Authenticate with external identity provider

When a resource owner is authenticated against one of the external identity providers : « Twitter », « Hotmail » or « GitHub » then an external account is automatically created.



To switch to a local account, click on the link *create a local account*:



### Action: Delete one or more authorization policies

Open the folder *assets>>uma>>authorization policies*, select several authorization policies, display the context menu and click on *remove authorization policy* .

### Action: Display authorization policy

Select an authorization policy, display its contextual menu and click on *authorization policy* . List of resources impacted by the policy and number of rules are displayed in the window.



### Action: Delete one or more UMA resources

Open the folder *assets>>uma>>resources*, select one or more resources, display the contextual menu and click on *remove resource* .

### Action: Display UMA resource

Display the contextual menu and click on resource info . Scopes and resource are displayed in the window :



## Identity providers

In the OpenId server, the end user can choose between one of the identity provider to authenticate himself: Hotmail, ADFS, GitHub or the Belgium id card.

When the claims are returned, they are used by the authorization policies to grant a request access to a protected resource.

If some claims are missing in the result, it’s always possible to enrich them with roles (refer to the chapter “edit a resource owner”).

List of actions available in the “connections” screen:

Enable / disable a provider

Edit a provider

Add a provider

Display providers

The following providers are configured by default: Microsoft, Linkedin, Google, GitHub, Facebook and the Belgium id card.

### Action: Display providers

Click on the connections tab to display the providers.



### Action  Enable / disable a provider

Providers can be enabled / disabled from the authentication page. To do that click on the button “On / Off” next to each provider.

By default an authentication page looks like:



New authentication page after the Facebook provider has been disabled:



Providers are managed by the administrator without worrying about restarting application.

### Action: Add a provider

Click on the *new identity provider* button to add a new one. When the window is displayed, try to fill-in all the fields. Use the decision table below to find all the values. When you have finished with it, click on the button *create* to persist the new provider.

|  |  |
| --- | --- |
| Fields | Description |
| Name | Name displayed is the authentication page |
| Callback path | Technical data used by the OPENID server. The value should start with « /signin-\* » otherwise there is no button displayed |
| Choose an identity provider type | Choose between OPENID, OAUTH2.0 or WS-Federation |

If there is no corresponding identity provider type, you can submit a new “enhancement” ticket to our website. The feature will be developed and deployed to your environment via Docker or XCOPY (refer to the Installation chapter)

If you want to use the resource owners which are stored in your database. We can help you writing some scripts to migrate from the old schema to the new one or create a new OPENID identity provider which takes your database as source.

### Action: Edit provider

Click on the provider title to edit it. You will be redirected to a new window where all the parameters are displayed. You will probably notice some differences between types, they are listed in the table:

|  |  |  |
| --- | --- | --- |
| Type | Parameters | Options |
| OAUTH2.0 | Callback path | TokenEndpoint |
| Namespace | AuthorizationEndpoint |
| Class | Scope |
| Code | ClientId |
| UserInformationEndpoint |
| ClientSecret |
| OPENID | Callback path | ClientId |
| Scope |
| ClientSecret |
| WellKnownConfigurationEndPoint |
| WSFEDERATION | Callback path | Realm |
| Namespace | IdEndpoint |
| Class |
| Code |

All parameters except namespace, code and class can be deduced with the configuration of your external identity providers for examples: the client identifier or secret. Those specific parameters are used by the OPENID server to parse xml or json and returns OPENID claims.

Those specific settings are not present in OPENID for a very simple reason, because a parser is not needed when OPENID is correctly implemented. The complete list of applications compliant with the standard is available online [[8]](#footnote-8) .

The parameter “code” presents in OAUTH2.0 and WS-Federation should contains a class with a method that respects a certain signature:

*public List<Claim> Process(JObject jObj) : OAUTH2.0*

*public List<Claim> Process(XmlNode node) : WS-Federation*

When the code has been written, the fields namespace and class name can be filled-in with their corresponding values. If we have the code below then the namespace is “Parser” and class name is “Example” :

*namespace Parser*

*{*

*public class Example*

*{*

*public List<Claim> Process(XmlNode node) { }*

*}*

*}*

## Parameters

Parameters are accessible via the “settings” tab. The token / authorization code expiration times can be updated.



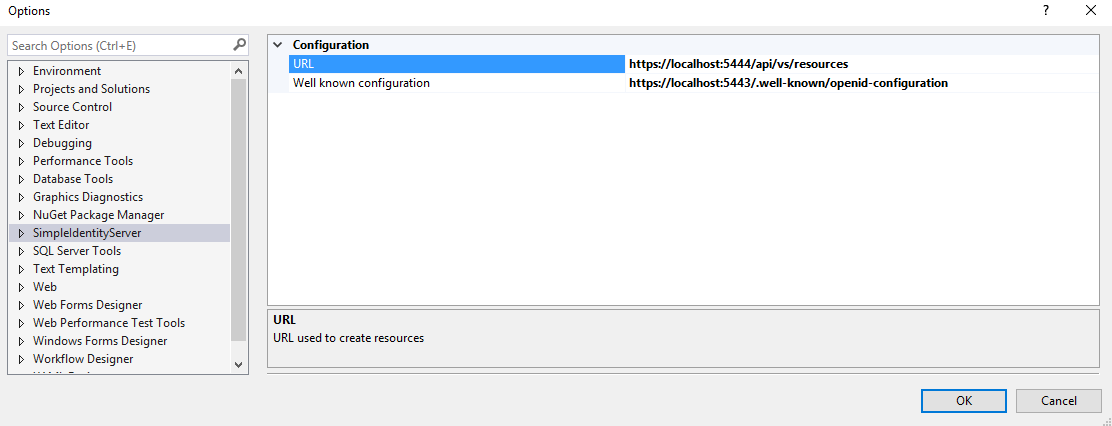
# Use cases of the Visual Studio extension

The extension is working only with Visual Studio version 2015. The previous versions will be supported in future release.

## Configure parameters

The configuration window is accessible via “Tools >> Options >> SimpleIdentityServer”. There are two URLs, the first one is used by the “generate resource” feature to generate a resource for each selected actions.

The other url is mostly used to retrieve an access token via client credentials, it is passed in the Authorization header to access to protected operations.



## Generate one or more resources

Choose any API project for which you want to protect its operations and display its contextual menu, then click on the option *generate resource* 

Don’t worry if it takes some times to load the operations. If after a while they are not displayed then close the panel window, select the project and reopen it again.

Here some features available in the window:

Refresh the resources

Protect one or more resources

Developer

### Refresh the resources

To refresh the resources, just click on the button *refresh*. It will take some times before they are displayed.



### Protéger une ou plusieurs ressources

Sélectionnez une ou plusieurs ressources que vous souhaitez protéger et indiquez le numéro de version de votre API. Ici, La notion de version est très importante car si des changements cassants ont été apportés à l’API, il suffit d’incrémenter le numéro et déployer la nouvelle version sans toucher à l’ancienne. Cliquez ensuite du « Protect » pour protéger les ressources sélectionnées.

L’opération va effectuer plusieurs tâches :

* Créer une ressource en suivant une convention de nommage :: « *Apis\<nom de l’assembly>\<numéro de version>\<nom du contrôleur>\<action> »*
* Ajouter les Nuget packages « SimpleIdentityServer.UmaIntrospection.Authentication » et « SimpleIdentityServer.Uma.Authorization »
* Ajouter un fichier de test « Startup\_Sample.cs » que vous pourrez réutiliser afin de protéger vos opérations.



Les détails techniques sont expliqués dans le chapitre « Documentation technique ».

## Générer un proxy de sécurité

Cette fonctionnalité est utilisée par le client afin de récupérer le RPT token. Ce dernier est passé en paramètre des requêtes qui souhaitent accéder à une ressource protégée. Pour afficher la fenêtre sélectionnez le projet du client et cliquez sur l’option « generate security proxy » dans le menu contextuel.

Rechercher une ou plusieurs ressources

Ajouter le proxy de sécurité

Développeur

### Action : Rechercher une ou plusieurs ressources

Tapez le nom de la ressource dans la bar de recherche et cliquez sur « Search ».



### Action : Ajouter le proxy de sécurité

Sélectionnez une ressource puis cliquez sur « Add ». Les tâches suivantes seront exécutées :

* Installer le Nuget package : « SimpleIdentityServer.Proxy »
* Ajouter deux fichiers au projet :
  + SecurityProxy\_\*.cs : possède une méthode statique qui retourne le RPT token.
  + AuthProvider.cs : possède une méthode statique qui retourne l’identity token.

# Scénarios

Cette partie décrit les problématiques de sécurité les plus fréquentes que l’on retrouve en entreprise et explique comment les résoudre.

Le code source se trouve ici : *https://github.com/thabart/SimpleIdentityServer.Samples.git*

## Premier scénario : Application lourde souhaite accéder à une opération protégée

**Contexte** : une entreprise d’e-commerce a développé en interne une application lourde. Elle est utilisée par son équipe de Marketing afin de récupérer des informations sur ses clients les plus fidèles. L’application a été développée en WPF et interroge un service web afin de récupérer la liste des clients. Seule cette application et les utilisateurs qui appartiennent au groupe «marketing » sont autorisés à récupérer la liste.

**Problème** : Comment l’application WPF peut accéder à l’opération protégée ?

**Solution** :

Le «workflow » que l’on souhaite implémenter est décrit dans le graphe ci-dessous. Il se compose de trois grandes étapes :

* *Identity token* : Récupérer un identity token avec le grant-type implicit [[9]](#footnote-9). Le token est ensuite retourné au client en tant que paramètre du callback.
* *RPT token* : Le client (application WPF) récupère un RPT Token [[10]](#footnote-10) pour cela il s’authentifie auprès du serveur UMA avec ses identifiants et passe l’identity token du resource owner. Ces informations sont requises par la politique d’autorisation, pour prendre la décision d’accorder ou non l’autorisation au client d’accéder à la ressource protégée. Pour plus d’informations concernant la politique d’autorisation référez à la description du cas d’utilisation : « Ajouter / Editer les règles d’autorisation ».
* *Vérification du RPT token*: Le RPT token est passé dans le header « Authorization » de la requête exécutée pour récupérer la liste des clients. Lorsqu’elle est reçue par l’API, le token est récupéré et vérifié en utilisant le point d’introspection [[11]](#footnote-11) offert par le serveur UMA.

Application WPF

Service API

Serveur OPENID

Récupérer Identity Token

Identity token

Serveur UMA

Authentifier utilisateur

Récupérer RPT token

RPT token

Récupérer clients

Nous vous épargnons les détails d’implémentation afin de faciliter la compréhension. Le workflow présenté est normalement plus complexe et possède beaucoup plus d’étapes intermédiaires.

Avant d’implémenter la solution, nous allons procéder par étapes de la même façon que le cas d’utilisation expliqué plus haut « ajouter / éditer les règles d’autorisation ».

Identifier et catégoriser les entités

Ajouter un client

Ajouter une ressource

Développer

Ajouter politique d’autorisation

Ajouter le rôle marketing au resource owner

### Identifier et catégoriser les entités

Dans un premier lieu il est nécessaire d’identifier les entités du contexte et de les catégoriser. Pour ce fait vous pouvez utiliser le tableau suivant.

|  |  |
| --- | --- |
| Questions | Type |
| Quelle est l’application consommateur du service que vous souhaitez protéger ? | Client |
| Quelle est l’opération du service que vous souhaitez protéger ? Identifiez le nom du service, sa version, l’entité business (client, produit) et l’opération. | Ressource : concaténation du nom du service, sa version, l’entité business ainsi que l’opération |
| Quelles sont les applications autorisées à accéder à la ressource ? | Liste des clients autorisés |
| Quelles sont les informations des resource owners acceptées pour accéder à la ressource ? | Liste des claims |

Voici ce que l’on obtient après analyse :

* Client : application WPF
* Ressource : ClientApi / v1 / Clients / Get
* Liste des clients autorisés : application WPF
* Liste des claims : rôle marketing.

Maintenant que les entités ont été identifiés vous pouvez utiliser le manager de ressources pour les créer.

### Ajouter un client

Ajoutez un nouveau client et éditez ses propriétés (lisez la description des cas d’utilisation « ajouter un client » & « éditer un client »). Dans la nouvelle fenêtre mettez à jour les propriétés avec les informations que vous avez réussi à extraire. Certains paramètres comme le nom & l’url de redirection sont assez facile à trouver ce qui n’est pas le cas des autres.

Essayez d’abord de deviner les grant\_types dont a besoin le client afin d’interagir avec les différents composants, les autres valeurs pourront ensuite être déduites.

Le workflow présenté ci-dessous requière une seule interaction avec le serveur OpenId. Elle débouche sur l’obtention d’un identity token et d’un access token valide pour les scopes « uma\_protection » et « uma\_authorization » (cet access token sera réutilisé plus tard lors de l’échange du RPT token et cette règle est toujours vraie).

Application WPF

Service API

Serveur OPENID

Récupérer Identity Token

Identity token & access token

Serveur UMA

Authentifier utilisateur

Récupérer RPT token

RPT token

Récupérer clients

Selon la façon dont est implémentée votre application WPF, le grant-type sera différent. Voici les relations entre le type d’implémentation et le grant type.

|  |  |  |  |
| --- | --- | --- | --- |
| Implémentation | Grant-Type | Avantages | Désavantages |
| Navigateur embarqué dans votre application | implicit | L’authentification est délégué au serveur OPENID | Pas le contrôle sur look & feel de la page d’authentification |
| Créer un formulaire | client\_credentials | Contrôle sur le look & feel | Le code de votre application doit être obfusqué pour ne pas exposer le ClientId et Clientsecret.  Cette approche n’est pas conseillé lorsqu’il n’y a pas de relation de confiance entre le client et le serveur. |

Lorsque les grant types ont été identifiés alors les autres paramètres peuvent être déduits. Pour ce fait aidez vous des tableaux ci-dessous :

|  |  |
| --- | --- |
| Grant type | Response types |
| Authorization code | Authorization code |
| Implicit flow | Token |
| Authorization code |
| Identity token |
| Client credentials |  |
| Password |  |
| Refresh token |  |

Relations entre les grant types et response types

|  |  |
| --- | --- |
| Type token | Scopes |
| Récupérer un RPT token | uma\_authorization |
| uma\_protection |
| Récupérer un identity token | Openid |
| Profile |
| Role |
| *La liste complète des scopes se trouve dans la documentation [[12]](#footnote-12)* |

Relations entre le type de token et les scopes

Lorsque vous avez tous les paramètres, la page d’édition peut être complétée de cette façon :

* Callback uris : <https://client.com>
* Grant types : implicit
* Response types : token, id\_token
* Scopes : openid, profile, role, uma\_authorization, uma\_protection



### Ajouter une ressource

La ressource peut être ajoutée de deux façons : avec l’interface graphique (voir les cas d’utilisation : « créer un nouveau répertoire » et « créer un nouveau fichier » ) ou l’extension Visual Studio (voir le cas d’utilisation : « Générer une ou plusieurs ressources »). Dans tous les cas la convention de nommage doit être respectée. Si la ressource est l’opération d’une API alors nous suggérons de respecter cette règle :

*Apis\<nom de votre application>\<numéro de version>\<nom de l’entité métier ou contrôleur>\<nom de l’opération>.*

Dans notre exemple nous obtenons « *Apis\ClientApi\v1\ClientsController\Get* ».

Si vous utilisez l’extension Visual Studio vous ne devez pas vous soucier de cette convention car elle est respectée.

Si la nature de la ressource est autre chose qu’une opération alors essayez de trouver une logique dans votre organisation. Par exemple si la ressource est l’image de profil d’un utilisateur, vous pouvez avoir quelque chose comme : « images\profile\user.png ».

Nous insistons sur l’importance de bien organiser vos ressources, si dès le départ vous partez sur une mauvaise organisation et que plus tard des corrections sont apportées. L’impact sera d’autant plus grand qu’il faudra mettre à jour les URLs dans toutes les applications et redéployer une nouvelle version.

### Ajouter la politique d’autorisation

Maintenant que le client et la ressource ont été créés, ajoutez la politique d’autorisation à la nouvelle ressource. Si vous n’y arrivez pas aidez-vous alors de la méthodologie qui est expliquée dans le chapitre « Action : Ajouter / Editer les règles d’autorisation ».

Voici le résultat attendu :



### Ajouter le rôle marketing au resource owner

Le rôle « marketing » doit être ajouté au resource owner autrement la politique d’autorisation ne passera jamais. Editez les propriétés du resource owner de votre choix et ajoutez lui le rôle « marketing ».



### Développer

Lorsque les entités ont été créées les modifications peuvent être apportées sur l’API et l’application WPF.

#### Modifier l’API

Pour rappel il existe deux types d’autorisation (pour plus de détails référez vous au chapitre « Documentation technique »)  :

* Convention: l’url de la ressource doit correspondre exactement à la structure du projet.
* Individuelle : limiter l’accès aux ressources en indiquant leurs URLs et leurs scopes.

L’implémentation technique marche seulement pour les projets ASP.NET CORE. Si vous travaillez sur une version antérieure, on peut vous aider à écrire votre implémentation. Les nuget packages « *SimpleIdentityServer.UmaIntrospection.Authentication* » et « *SimpleIdentityServer.Uma.Authorization* »  doivent être installées sur votre projet API.

L’autorisation par convention est activée en ajoutant le code suivant dans la procédure « ConfigureServices » de votre classe Startup.

// Authorization policy

services.AddAuthorization(options =>

{

// Add conventional uma authorization

options.AddPolicy("uma", policy =>

{

// policy.Requirements.Add(new ConventionalUmaAuthorizationRequirementTst(null));

policy.AddConventionalUma();

// options.AddPolicy("resourceSet", policy => policy.AddResourceUma("<url>", "<read>","<update>"));

});

});

L’opération « ClientController/Get » doit ensuite être décorée par l’attribut : 

#### Modifier l’application WPF

Avant de commencer n’oubliez pas d’installer le nuget package « SimpleIdentityServer.Proxy ».

Votre application WPF doit posséder une fenêtre d’authentification. Elle peut afficher la fenêtre d’autorisation du serveur OPENID dans un navigateur embarqué ou afficher son propre formulaire. Quelque soit la méthodologie choisie, il est important de stocker l’identity token qui a été récupéré.

Ensuite utilisez le code suivant pour récupérer le RPT token. N’oubliez pas bien évidemment de mettre à jour les propriétés.

public static async Task<string> GetRptToken(

string idToken,

string umaProtectionToken,

string umaAuthorizationToken)

{

var factory = new SecurityProxyFactory();

var proxy = factory.GetProxy(new SecurityOptions

{

ClientId = Constants.ClientInfo.ClientId,

ClientSecret = Constants.ClientInfo.ClientSecret,

UmaConfigurationUrl = "https://localhost:5445/.well-known/uma-configuration",

OpenidConfigurationUrl = "https://localhost:5443/.well-known/openid-configuration",

RootManageApiUrl = "https://localhost:5444/api"

});

try

{

var result = await proxy.GetRpt("resources/Apis/ClientApi/v1/ClientsController/Get", idToken, umaProtectionToken, umaAuthorizationToken, new List<string>

{

"execute"

});

return result;

}

catch (Exception ex)

{

return null;

}

}

Appelez la méthode GetRptToken en n’oubliant pas de passer en paramètre les tokens (identity & access) puis passez le résultat dans le header de votre requête.

var request = new HttpRequestMessage

{

Method = HttpMethod.Get,

RequestUri = new Uri("http://localhost:5100/api/clients")

};

request.Headers.Add("Authorization", $"Bearer {rptToken}");

Si vous voulez exécuter l’exemple complet ouvrez la solution « Scenario1\WpfClient » du repository « SimpleIdentityServer.Samples » et mettez à jour les propriétés dans les fichiers « UI\WpfClient\SecurityProxyClientApi » & « UI\WpfClient\Constants » & « Api\ClientApi\Startup ».

La page d’authentification est d’abord affichée. Dès lors que l’utilisateur avec le rôle « marketing » est authentifié alors les clients provenant de l’API sont listés.



Liste des clients :



## Second scénario : une API souhaite accéder à une ressource protégée

**Contexte** : Une équipe développe une nouvelle API qui récupère les données clientèles pour les enrichir avec leurs données marketing qui ont été récupérées depuis la création de l’entreprise comme par exemples : l’historique de navigation ou le nombre de plaintes.

**Problème** : Comment l’API peut-elle accéder à la liste des clients ?

**Solution** :

Le workflow ressemble beaucoup à celui du premier scénario. A la différence que cette fois-ci il n’est pas nécessaire de passer un identity-token au serveur UMA car il ne l’utilisera pas dans sa politique d’autorisation.

Récupérer RPT token

API

Service API

Serveur OPENID

Serveur UMA

Récupérer clients

Le processus d’implémentation est similaire à celui du premier scénario. Toutefois la tâche où la notion de resource owner intervient a été supprimée.

Identifier et catégoriser les entités

Ajouter un client

Ajouter une ressource

Développer

Ajouter politique d’autorisation

### Identifier et catégoriser les entités

Voici les entités que l’obtient après avoir utilisé le tableau de décision :

* Client : API
* Ressource : ClientApi / v1 / Clients / Get
* Liste des clients autorisés : API
* Liste des claims : aucun

### Ajouter un client

La plus part des paramètres peuvent facilement être déduits y-compris le grant type. Quel grant-type choisirez vous sachant qu’aucune interaction du resource owner n’est requise et qu’il y a une relation de confiance entre les deux APIs.

La bonne réponse est « client\_credentials », en effet vous n’encourez aucun risque de sécurité en le choisissant car le code est hébergé sur un serveur et devrait être inaccessible aux utilisateurs finaux.

Voici les paramètres que vous devez obtenir :

* Grant-types : client credentials
* Scopes : uma\_authorization et uma\_protection



### Ajouter une ressource

La ressource a déjà été créée dans le premier scénario.

### Ajouter la politique d’autorisation

Editez la politique d’autorisation de la ressource « resources\Apis\ClientApi\v1\ClientsController\Get » et ajoutez une nouvelle règle. Essayez de deviner les paramètres en vous appuyant sur la description du cas d’utilisation « Ajouter / Editer les règles d’autorisation ».

* Allowed clients : Marketing API
* Permissions : execute



### Développer

#### Modifier votre client API

Le Nuget package « SimpleIdentityServer.Proxy » doit être installé. Récupérez d’abord l’access token valide pour les scopes « uma\_protection » et « uma\_authorization » en utilisant le grant type « client credentials ».

private async Task<GrantedToken> GetAccessToken()

{

return await \_identityServerClientFactory.CreateTokenClient()

.UseClientSecretBasicAuth(\_clientId, \_clientSecret)

.UseClientCredentials("uma\_authorization", "uma\_protection")

.ResolveAsync("https://localhost:5443/.well-known/openid-configuration");

}

Passez ensuite le résultat en paramètre à la fonction ci-dessous afin de récupérer le RPT token. Ce dernier pourra ensuite être utilisé utilisé afin de récupérer la liste des clients.

public static async Task<string> GetRptToken(

string umaProtectionToken,

string umaAuthorizationToken)

{

var factory = new SecurityProxyFactory();

var proxy = factory.GetProxy(new SecurityOptions

{

UmaConfigurationUrl = "https://localhost:5445/.well-known/uma-configuration",

OpenidConfigurationUrl = "https://localhost:5443/.well-known/openid-configuration",

RootManageApiUrl = "https://localhost:5444/api"

});

try

{

var result = await proxy.GetRpt("resources/Apis/ClientApi/v1/ClientsController/Get", umaProtectionToken, umaAuthorizationToken, new List<string>

{

"execute"

});

return result;

}

catch (Exception ex)

{

return null;

}

}

Si vous voulez exécuter l’exemple complet ouvrez la solution « Scenario2\MarketingClient» du repository « SimpleIdentityServer.Samples » et mettez à jour les propriétés.

Lancez les deux projets « ClientApi » & « MarketingClient » puis naviguez vers l’url <http://localhost:5103/api/ratings>.



## Troisième scénario : Limiter l’accès à certaines fonctionnalités du site

**Contexte** : Les utilisateurs avec le rôle « administrator » peuvent voir le bouton « administrate » sur le site internet tandis que les autres ne le peuvent pas.

**Problème** : Comment limiter l’accès à certaines fonctionnalités selon le rôle des utilisateurs ?

**Solution** :

Comparé aux autres scénarios, le workflow présenté ci-dessous est assez simple :

* L’utilisateur s’authentifie auprès du serveur OPENID. Une fois authentifié alors les identity & access tokens sont retournés au site internet.
* Les permissions sont ensuite récupérées du serveur UMA et stockées dans un cookie. Les tokens qui ont été récupérés sont passés en paramètre de la requête.

Récupérer les permissions

Identity & access tokens

S’authentifie

Utilisateur

Serveur UMA

Site internet

Serveur OPENID

Comme les scénarios précédents nous allons suivre une méthodologie afin d’implémenter la solution.

Identifier et catégoriser les entités

Ajouter un client

Ajouter une ressource

Développer

Ajouter politique d’autorisation

Ajouter le rôle « administrator » au resource owner

### Identifier et catégoriser les entités

Voici les entités que l’obtient après avoir utilisé le tableau de décision :

* Client : WebSite
* Ressources : WebSite \ Scenario3 \ Home \ Admin
* Liste des clients autorisés : WebSite
* Liste des claims : rôle « administrator »

### Ajouter un client

Une fois de plus les paramètres sont assez faciles à déduire excepté le grant type. Le choix dépend encore une fois de la façon dont vous souhaitez authentifier les utilisateurs finaux. Si votre choix se porte sur le développement d’un formulaire « login » & « mot de passe » alors le grant type choisi sera « password ». Faîtes bien attention si vous choisissez ce grant-type car il est nécessaire d’avoir une relation de confiance avec le fournisseur d’identité et les identifiants du client « Id » & « secret » doivent être cachés du public.

Si au contraire vous souhaitez déléguer cette problématique d’authentification au serveur OPENID, choisissez alors le grant-type « implicit ». Lors de l’authentification l’utilisateur sera redirigé vers le serveur OPENID qui retournera ensuite les tokens. Notre préférence est toujours de choisir le grant-type le plus fiable et facile à implémenter, c’est pour cela que ce dernier a été sélectionné.

Faîtes bien attention de ne pas oublier les scopes obligatoires « uma\_authorization » & « uma\_protection » car il y a une interaction avec le serveur UMA.

Paramètres du client :

* Callback url : <https://localhost:5105/Authenticate/Callback>
* Grant types : implicit
* Response types : token & id\_token
* Scopes : uma\_protection, uma\_authorization, role, openid, profile



### Ajouter une ressource

Ajouter la ressource « WebSite \ Scenario3 \ Home \ Admin ».



### Ajouter la politique d’autorisation

Ajoutez une politique d’autorisation à la ressource. Essayez encore une fois de deviner les paramètres en vous appuyant sur le contexte ainsi que sur la description du cas d’utilisation : « Ajouter / Editer les règles d’autorisation ».

* Allowed clients : WebApplicationScenario3
* Permissions : read
* Allowed claims : rôle « administrator »



### Ajouter le rôle « administrator » au resource owner

. Editez les propriétés du resource owner de votre choix et ajoutez lui le rôle « administrator ».



### Développer

Je vous invite à jeter un œil au projet «Scenario3 \ WebApplication » du repository « SimpleIdentityServer.Samples ». Mettez à jour les paramètres du fichier « Constants.cs » et lancez l’application.

Si vous-vous connectez avec un utilisateur qui possède le rôle « administrator » alors la permission « Admin » est affichée, autrement il n’y a que trois permissions.



Permissions d’un utilisateur « administrator »



Permissions d’un utilisateur non « administrator »

# Installation

Le guide part du principe que l’architecture va être installée sur la même machine. Gardez à l’esprit qu’il est tout à fait possible de déployer séparément les différents composants. Cela peut être utile si souhaitez mettre en place le load balancing, pour plus de précisions référez vous au chapitre « Documentation technique \ Load balancing ».

## Etapes obligatoires

Que vous choisissez d’installer le produit manuellement ou par Docker, il y a quelques prérequis à installer sur votre machine.

### Installer GIT

GIT doit être installé afin de récupérer les sources du projet, suivez le guide du site officiel : ([link](https://git-scm.com/downloads))



### Installer le certificat

Le certificat « LokitCA.cer » joint au produit doit être installé sur votre machine dans « Certificate Store \ Local User \ Trusted CA ». Sans le certificat le site web ne pourra pas interagir avec les APIs.



## Installation manuelle

TODO

## Installer avec docker

### Prérequis

Le produit est déployé sur plusieurs conteneurs Docker. La procédure est assez simple et se résume en quelques étapes. Mais avant de commencer vous devez vous assurer que les prérequis suivants ont bien été installés et configurés sur votre machine.

#### Installer Docker

Docker doit être installé, vous pouvez suivre le guide sur le site officiel ([lien](https://docs.docker.com/engine/installation/)) qui est très bien expliqué :



#### Configurer VirtualBox

Par défaut Docker utilise VirtualBox pour lancer sa machine virtuelle. Des règles de redirection de ports doivent être ajoutées sur la machine « default », sans quoi vous ne pourrez pas naviguer sur le site :

* Ouvrez VirtualBox et sélectionnez la machine « default »
* Cliquez sur « Configuration » puis sélectionnez l’onglet « network »
* Cliquez sur le bouton « port redirection » et assurez-vous de bien avoir les mêmes règles :



Note : Une règle de redirection manque dans la capture d’écran. Ajoutez la sinon les logs ne pourront pas être affichés sur le site :

Nom : Kibana

Protocol : TCP

IP hôte : 127.0.0.1

Port hôte : 5601

Port invité : 5601

### Lancer le déploiement

Toujours dans le même invité de commande, exécutez l’instruction « docker-compose up » et attendez que le produit se lance.

Une fois l’application installée et déployée, vous pouvez vous connecter en tant qu’administrateur au site web <http://localhost:4200>.

# Documentation technique

## Monitoring

Pour observer les événements naviguez sur l’onglet « Logs » du site. Si c’est la première que vous vous y rendez alors configurez un index de recherche avec cette valeur « *simpleidserver-\** » puis importez le fichier « Kibana-Exports\export.xml ». Le dashboard sera alors configuré et visible à partir du site.



La fenêtre est composée de 4 widgets :

* Affiche tous les événements qui ont lieux dans le serveur OPENID
* Suivre l’évolution du nombre de « tokens » qui ont été accordés au fil du temps.
* Voir le nombre d’erreur d’autorisations ainsi qu’un autre widget qui affiche les erreurs d’autorisation.

## Nuget packages

### SimpleIdentityServer.Proxy

Installez le sur votre client afin de pouvoir facilement récupérer un RPT token et l’utiliser pour accéder à une ressource protégée.

### SimpleIdentityServer.UmaIntrospection.Authentication

Installez le sur un projet ASP.NET CORE qui contient des ressources sensibles. Il authentifie les requêtes qui ont un RPT token dans l’Authorization header.

### SimpleIdentityServer.Uma.Authorization

Installez le sur un projet ASP.NET CORE afin de mettre en place la politique d’autorisation.

## Load balancing

Pour mettre en place le load balancing il est nécessaire de déployer les APIs sur différentes machines. Ces derniers peuvent tourner sur n’importe quel OS (Linux, Windows ou MAC) grâce au Framework DOTNET CORE [[13]](#footnote-13). Choisissez parmi l’une des solutions (ou autre) de load balancing  (HalProxy etc …) et implémentez cette architecture :

Load balancer

OpenId #2

OpenId #1

Load balancer

Configuration #1

Configuration #2

Load balancer

WebSiteAPI #2

WebSiteAPI #1

Load balancer

UMA #1

UMA #2

Load balancer

Manager #1

Manager #2

# Reste à faire

|  |  |  |
| --- | --- | --- |
| Tâche | Priorité | Avancement |
| Ajouter un nouveau type de fournisseur d’identité SAML | 5 | DONE |
| Ajouter le code pour les fournisseurs d’identité OPENID | 5 | DONE |
| Afficher le bouton pour créer un fournisseur d’identité externe | 5 | DONE |
| Ajouter un fournisseur d’identité externe | 5 | DONE |
| Afficher le nom de l’utilisateur & son image & un bouton de déconnection | 5 | DONE |
| Vérifier le rôle de l’utilisateur | 5 | DONE |
| Vérifier pourquoi tous les fournisseurs d’identité ne sont pas affichés dans l’application WPF | 5 | DONE |
| Lorsque le client ne peut pas être authentifié alors retourner un message d’erreur lisible | 5 | DONE |
| Modifier le titre « GenerateResourceCommand » | 5 | DONE |
| Décrire le premier scénario | 5 | DONE |
| Afficher les clients | 5 | DONE |
| Utiliser un seul token pour plusieurs accès | 5 | DONE |
| Décrire le second scénario | 5 | DONE |
| Décrire le troisième scénario | 5 | DONE |
| Supprimer la propriété « include sub » de UmaIntrospection | 5 | DONE |
| Utiliser WS-Federation AUTH0 | 5 | DONE |
| Ajouter une partie dans le document sur Kibana | 5 | DONE |
| Ajouter la partie EASY DEPLOYMENT | 5 | DONE |
| Héberger le site internet | 5 | TODO |
| Déployer la solution sur azure (avec docker) | 5 | TODO |
| Sécuriser l’API | 5 | TODO |

Afin de créer une machine virtuelle sur Microsoft Azure avec docker d’installé dessus alors lisez la documentation : <https://azure.microsoft.com/fr-fr/documentation/articles/virtual-machines-linux-dockerextension/>

1. RFC OPENID : http://openid.net/specs/openid-connect-core-1\_0.html [↑](#footnote-ref-1)
2. RFC UMA : https://docs.kantarainitiative.org/uma/rec-uma-core.html [↑](#footnote-ref-2)
3. RPT token : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.5.1 [↑](#footnote-ref-3)
4. Redirect\_uri validation rules : <https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata> [↑](#footnote-ref-4)
5. Client parameters : <https://openid.net/specs/openid-connect-registration-1_0.html#ClientMetadata> [↑](#footnote-ref-5)
6. Request parameter : <http://openid.net/specs/openid-connect-core-1_0.html#JWTRequests> [↑](#footnote-ref-6)
7. OPENID documentation : <http://openid.net/specs/openid-connect-core-1_0.html> [↑](#footnote-ref-7)
8. Applications compliants with OPENID : http://openid.net/certification/ [↑](#footnote-ref-8)
9. Implicit grant type : http://openid.net/specs/openid-connect-implicit-1\_0.html [↑](#footnote-ref-9)
10. Récupérer RPT token : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.5.1 [↑](#footnote-ref-10)
11. Introspection endpoint : https://docs.kantarainitiative.org/uma/rec-uma-core.html#rfc.section.3.4.1 [↑](#footnote-ref-11)
12. Liste des scopes : http://openid.net/specs/openid-connect-core-1\_0.html#ScopeClaims [↑](#footnote-ref-12)
13. DOTNET CORE : https://www.microsoft.com/net/core#windows [↑](#footnote-ref-13)