# SimpleIdServer OPENID & OAUTH2.0

## Paramètres de la requête « authorization »

### OAUTH2.0

|  |  |  |
| --- | --- | --- |
| Paramètre | Description | URL |
| response\_type | Indique les paramètres qui doivent être retournés par le endpoint d’authorization par exemple : code, token ou id\_token. | <https://tools.ietf.org/html/rfc6749> |
| client\_id | Identifiant du client | <https://tools.ietf.org/html/rfc6749> |
| redirect\_uri | URL de redirection enregistré dans le client. | <https://tools.ietf.org/html/rfc6749> |
| scope | Scope dont la requête d’autorisation souhaite accéder. | <https://tools.ietf.org/html/rfc6749> |
| state | Valeur opaque utilisée par le client afin d’avoir une consistance entre la requête et la réponse. Si elle est présente, la valeur est retournée par le serveur d’autorisation. | <https://tools.ietf.org/html/rfc6749> |
| response\_mode | Informer le serveur d’autorisation du mécanisme qui doit être utilisé pour retourner la réponse à une requête d’autorisation.  Exemple de valeurs : query, fragment. | <https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html> |
| ui\_locales | Liste des langages préférées. Ce paramètre est utilisé par l’interface graphique. | Initialement introduit par OPENID, ce paramètre est présent dans OAUTH2.0 |

### OPENID

|  |  |  |
| --- | --- | --- |
| Paramètres | Description | URL |
| max\_age | Temps maximum de l’authentification.  Temps écoulé > date d’authentification + *max\_age* (seconds) alors authentifier l’utilisateur  Lorsque le paramètre *max\_age* est présent alors inclure dans l’*id\_token* le claim *auth\_time*. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| id\_token\_hint | Correspond à « id\_token » qui a précédemment été généré par le serveur d’autorisation.  Le subject doit être le même que celui de l’id\_token\_hint  Le paramètre *id\_token\_hint* devrait être présent si *prompt=none* sinon une erreur est retournée  Le serveur d’autorisation doit être listé dans le claim *audience* de l’identity token. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| claims | Demander des claims spécifiques au serveur d’autorisation.  Lorsqu’un claim dit « essential » est reçu et qu’aucun consent n’a été donné alors l’utilisateur final est redirigé vers la page d’authentification  Lorsqu’un claim dit « essential » est reçu et que sa valeur n’existe pas alors une erreur est levée  Le claim auth\_time doit être présent dans l’id\_token lorsqu’il est spécifié dans le paramètre claims | <https://openid.net/specs/openid-connect-core-1_0.html#ClaimsParameter> |
| request | Passer les paramètres d’une requête d’autorisation sous la forme d’un token JWT | <https://openid.net/specs/openid-connect-core-1_0.html#RequestObject> |
| request\_uri | L’objet « request » est récupéré par l’URL. | <https://openid.net/specs/openid-connect-core-1_0.html#RequestUriParameter> |
| nonce | Clef de session inclus dans l’id\_token, éviter les attaques « replay »  Utilisé pour associer une session du client avec un identity token. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| display | Comment le serveur d’autorisation doit afficher les page web.  Les valeurs possibles sont : page, popup, touch ou wap. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| prompt | Indique au serveur d’autorisation si l’utilisateur doit être : authentifié à nouveau, afficher les « consents » etc …  Les valeurs possibles sont : login, none, consent ou select\_account. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| login\_hint | Ce paramètre peut être utilisé par le serveur d’autorisation pour remplir le formulaire d’authentification.  Le login\_hint peut correspondre à l’adresse email ou à l’identifiant de l’utilisateur. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |
| acr\_values | Liste de « Authentication Context Class Reference (ACR). ».  Les claims « acr » & « amr » doivent être présents dans l’identity token. | <https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest> |

#### Le paramètre request

Est une requête d’autorisation représentée sous la forme d’un jeton JWT.

Les paramètres obligatoires de la requête d’autorisation sont :

* Response\_type
* Client\_id
* Scope = openid

Ils doivent être présents dans la requête d’autorisation ainsi que dans le jeton JWT.

Lorsque le jeton est signé alors les claims « iss » et « aud » doivent être inclus. Iss doit être égale à l’identifiant du client.

Durant la vérification de la signature du jeton JWS, le paramètre « alg » doit être le même que la propriété « RequestObjectSigningAlg » présente chez le client.

#### Le paramètre request\_uri

Consiste à passer le paramètre « request » par référence, c’est-à-dire de récupérer sa valeur à partir d’une URL.

#### Le paramètre prompt = select\_account

Un utilisateur peut avoir plusieurs identités / sessions.

Chaque session doit être stockée dans un cookie séparée et la session courante doit être stockée dans un autre cookie.

Les sessions sont stockées en mémoire et répliquées dans le cluster SimpleIdServer.

#### sLe paramètre acr\_values

ACR VALUES est un paramètre optionnel utilisé par le client pour spécifier un ou plusieurs « Authentication Context Class » (ACR).

Un « Authentication Context Class » indique le niveau d’assurance (Level Of Assurance). Plus il est élevé plus on est certain de l’identité de l’utilisateur final.

**Exemples** :

|  |  |
| --- | --- |
| Acr\_value | Description |
| Mod-pr | Mitiger les attaques de type « phishing » des « credentials ».  L’utilisateur s’authentifie avec son appareil mobile en validant le message de confirmation qui s’affiche. Aucune autre information n’est nécessaire pour utiliser la clef secrète de l’appareil.  Moyen d’authentification : appareil mobile. |
| Mod-md | Mitiger les attaques de type « phishing » et prouve bien que l’utilisateur final est en possession de l’appareil car il doit être débloqué par un code PIN.  Moyens d’authentification : appareil mobile + secret (code PIN | empreinte etc …) |

<https://ldapwiki.com/wiki/Identity%20Assurance%20Level>

#### Le scope offline\_access

Le scope « *offline\_access* » est utilisé afin de récupérer un « *refresh token* ». Ce dernier est utilisé afin de récupérer un « *access token* » qui sera utilisé pour récupérer les informations de l’utilisateur même si ce dernier n’est pas connecté.

Lorsque le serveur d’autorisation reçoit le scope « *offline\_access* », les vérifications suivantes sont effectuées :

1. S’assurer que le paramètre « prompt » contient « consent ». Ignorer le scope « offline\_access ».
2. Doit avoir le consent de tous les clients d’utiliser un «*offline\_access* ».
3. Le scope doit être ignoré sauf si le response\_type = code.

## Paramètre d’un identity token

|  |  |  |
| --- | --- | --- |
| Paramètres | Règles | Supporté |
| iss | URL du fournisseur d’identité | OK |
| sub | Identifiant unique de l’utilisateur final | OK |
| aud | Contient le *client\_id* du relying party. Indique la liste des clients qui peuvent consommer l’id\_token | OK |
| exp | Date d’expiration. Nombre de secondes écoulées depuis le 1er janvier 1970 | OK |
| iat | Date de création. Nombre de secondes écoulées depuis le 1ier janvier 1970 | OK |
| auth\_time | Date d’authentification de l’utilisateur. Nombre de secondes écoulées depuis le 1ier janvier 1970 | OK |
| Lorsque le paramètre max\_age est renseigné alors auth\_time doit être renseigné | OK |
| nonce | Si la requête possède nonce, alors la valeur doit être incluse dans l’id\_token | OK |
| azp | Indique le client pouvant présenter l’*id\_token*. Utile dans le cas où un client transfère un token à un autre client | OK |
| at\_hash | HASH (access token) | OK |
| c\_hash | HASH (authorization code) | OK |
| acr | Indique le niveau d’authentification | OK |
| amr | Indique les moyens d’authentification qui ont été utilisés | OK |

### Subject identifier types

#### Public

Fournit le même « subject » pour tous les clients.

#### Pairewise

Un sujet différent est généré par client, afin d’éviter une corrélation involontaire, laisser la possibilité d’exécuter des actions sans avoir la permission.

Règles :

* Si la propriété « *sector\_identifier\_uri* » n’a pas été renseignée par le client alors il faut utiliser « *redirect\_uris* ». Si plus d’un hôte distinct est spécifié dans la propriété « *redirect\_uris* » alors le client doit impérativement enregistrer sa propriété « *sector\_identifier\_uri* ».
* Si « *sector\_identifier\_uri* » est renseigné alors son hôte est utilisé comme identifiant de secteur. Cette valeur est utilisée pour calculer l’identifiant « pairwise ».
* « *sector\_identifier\_uri* » doit être une URL « *https* » qui pointe sur un fichier JSON contenant une liste de « *redirect\_uri* ». Cette liste doit être incluse dans « *redirect\_uris* ».

La propriété « *sector\_identifier\_uri* » est utilisée afin qu’un regroupement de sites puisse avoir un « subject » consistant.

Fournit la possibilité de regrouper un ensemble de services. Permet à l’administrateur du service de pouvoir changer le nom de domaine sans avoir besoin de réenregistrer tous les utilisateurs.

## Service UserInfo

Par défaut le endpoint UserInfo retourne les « claims » dans un objet JSON sauf si la génération d’un token JWT a été demandée explicitement durant l’enregistrement du client.

Lorsqu’un claim n’est pas présent alors il doit être supprimé de la réponse, il ne peut pas être présent avec comme valeur NULL ou vide.

Les deux types de réponse sont : :application/json et application/jwt.

|  |  |
| --- | --- |
| Fonctionnalités | Supporté |
| Retourner un token JWS | OK |
| Retourner un token JWE | OK |
| Les claims sont retournés | OK |
| Le paramètre « claims » est supporté | OK |

### Claims agrégés et distribués

#### Claims agrégés

La réponse du endpoint UserInfo contient les claims externes sous un format JWT.

Les paramètres suivants ont été ajoutés dans la réponse :

* \_claim\_names : { « claim » :  «source » }
* \_claim\_sources : { « source » : { « JWT » : « valeur » } }

#### Claims distribués

La réponse du endpoint UserInfo contient une URL ainsi qu’un access token qui pourra être utilisé par le client afin de récupérer les claims.

Les paramètres suivants ont été ajoutés dans la réponse :

* « \_claim\_names » : { « claim » : « source » }
* « \_claim\_sources » :{ « source » : { « endpoint » : « url », « access\_token » : « token » } }

## Self-Issued OpenID provider

Le fournisseur d’identité est installé sur la machine de l’utilisateur final par exemple un téléphone portable. Cas d’utilisation : utilise l’empreinte digitale afin d’authentifier un utilisateur.

L’identifiant « iss » doit être <https://self-issued.me>.

Plus d’informations : <https://nat.sakimura.org/wp-content/uploads/2013/08/openid-connect-selfissued-1_0.html>

## Service d’enregistrement d’un client

Le RFC « <https://tools.ietf.org/html/rfc7591> » décrit le protocole pour enregistrer dynamiquement un client OAUTH2.0.

### Protocole

Le protocole est constitué de plusieurs étapes :

1. Le client récupère un « access token »
2. Ce dernier est utilisé pour intéragir avec le point d’entrée « client registration
3. Le serveur d’autorisation retourne : les méta données du client, un identifiant unique et un ensemble de « credentials ».

### Les méta données

|  |  |  |
| --- | --- | --- |
| Paramètre | Description | Règles |
| redirect\_uris | Liste d’URLS de redirection utilisée durant le grant type « implicite » ou « code d’authorisation » | Obligatoire si le grant\_type est égale à « code » ou « implicit ». |
| token\_endpoint\_auth\_method | Indique la méthode d’authentification utilisée pour accéder au point d’entrée « token ». | Peut être égale à :   * none : client public qui ne possède aucun secret. * client\_secret\_post * client\_secret\_basic   Si la valeur n’est pas renseignée alors la valeur par défaut est « client\_secret\_basic ». |
| grant\_types | Liste de grant types | Les valeurs possibles sont :   * authorization\_code * implicit * password * client\_credentials * urn :ietf :params :oauth :grant-type :jwt-bearer * url :ietf :params :oauth :grant-type :saml2-bearer   Si la valeur n’est pas renseignée alors la valeur par défaut est « authorization\_code » |
| response\_types | Liste de réponses que le client pourra utiliser sur le point d’entrée « authorization ». | Les valeurs possibles sont :   * code * token   Si la valeur n’est pas renseignée alors la valeur par défaut est « code » |
| client\_name | Nom du client qui sera affiché durant le processus d’autorisation | Si la valeur n’est pas renseignée alors la valeur par défaut est le « client\_id ».  Cette valeur peut être traduite. |
| client\_uri | URL du site web qui donne des informations à propos du client. | La valeur de ce champ doit pointer vers une page web.  Il doit être possible de traduire. |
| logo\_uri | Référence un logo du client | Doit pointer vers une image valide.  Il doit être possible d’internationaliser l’image. |
| scope | Les scopes que le client pourra utiliser. | Si la valeur n’est pas renseignée alors un ensemble de scopes par défaut pourra être renseigné par le serveur d’autorisation. |
| Contacts | Données qui pourront être utilisées pour contacter les personnes responsables du client. |  |
| tos\_uri | URL qui pointe vers une page « Terms Of Service ».  Ce document décrit la relation contractuelle qui existe entre entre le client et l’utilisateur final. | Doit pointer vers une page web valide.  Cette valeur peut être traduite. |
| policy\_uri | URL qui pointe vers une page « Policy document ».  Ce document décrit la façon dont l’organisation collecte, utilise et retient les informations personnelles de l’utilisateur final. | Doit pointer vers une page web valide.  Cette valeur peut être traduite. |
| jwks\_uri | URL qui référence un ensemble de documents JWK (possèdent les clefs publiques). | L’URL doit pointer vers une page web valide.  Les clefs JWK sont utilisées pour valider les signatures ou décrypter des requêtes.  Le paramètre « jwks\_uri » & « jwks » ne doivent pas être présents en même temps. |
| jwks | Object JSON qui contient une liste de clefs JWK.  Ce paramètre est généralement utilisé par une application native qui ne peut pas héberger une page web. |  |
| software\_id | Contrairement au « client\_id », cet identifiant unique généré par le développeur ne varie pas selon instances du programme. |  |
| software\_version | La version doit changer lors d’une mise à jour du programme. |  |
| software\_statement | Contient les méta données du client. Token signé JWT. |  |

Certains paramètres comme « client\_name », « tos\_uri » peuvent avoir différentes traductions. Durant le processus d’enregistrement, une traduction peut être passé par le client de cette façon :

* « nom paramètre »# « langue » par exemple : client\_name#en

Internationaliser les paramètres « client\_name », « tos\_uri »

#### Réponse

La réponse contient l’identifiant du client ainsi que le secret. Voici les paramètres retournés :

* client\_id : identifiant du client.
* *client\_secret* : secret du client
* *client\_id\_issued\_at*: date à laquelle l’identifiant du client a été créé.
* *client\_secret\_expires\_at : date à laquelle le secret expire.*
* Toutes les autres données doivent aussi être retournées.
* Si le paramètre « software statement » est inclus dans la requête alors il doit être présent dans la réponse.

Le code HTTP retourné est « 201 ».

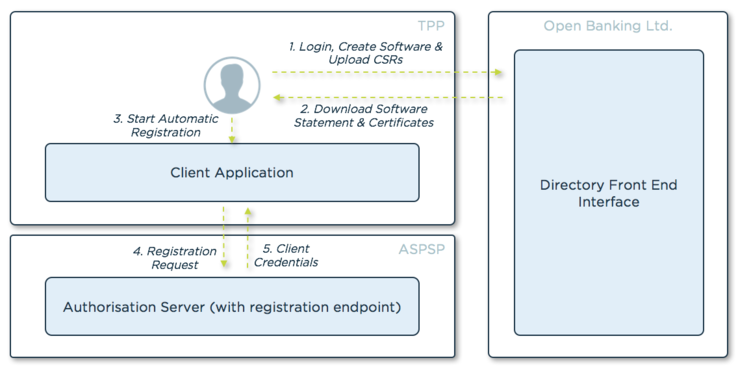
### Software statement

Le paramètre « software statement » est un token JWT. Lorsqu’il est passé en tant que paramètre à la requête d’enregistrement il doit:

* Signé ou MAC (JWS)
* Posséder le claim « iss »
* Il est recommandé de passer le claim  « software\_id » pour permettre une corrélation entre les différentes instances du programme.

Le paramètre « software statement » est typiquement distribué par toutes les instances de l’application. Une façon de récupérer ce paramètre est d’enregistrer l’application en appelant l’API software et récupérer un « software statement ».

#### CAS D’ UTILISATION



1. Un PTC (Primary Technical Contract) accède au site web de « OpenBanking », enregistre un nouveau logiciel, génère et télécharge les certificats.
2. Le PTC obtient un « Software Statement »
3. Le PTC déclenche l’enregistrement vers le ASPSP.
4. Le TPP créé une requête d’enregistrement et inclus le « software statement »
5. ASPSP vérifie la requête & créé le client OAUTH.

Afin de valider la signature JWT il faut :

1. Décoder la requête JWT, sans valider la signature.
2. Extraire le « software statement » du token JWT.
3. Valider le « software statement » avec le endpoint JWKS.

« OPEN BANKING DIRECTORY » peut être utilisé pour vérifier le client.

### Enregistrer un client openid

Le RFC se trouve : <https://openid.net/specs/openid-connect-registration-1_0.html>

Liste des paramètres différents par rapport à OAUTH2.0 :

|  |  |  |
| --- | --- | --- |
| Paramètre | Description | Règles |
| redirect\_uris |  | Un client web qui utilise le grant type implicite doit :   * Utiliser HTTPS * Le nom de l’hôte ne doit pas être localhost   Un client natif peut enregistrer une URL de redirection avec un schéma qui lui est propre ou utiliser http://localhost |
| grant\_types |  | Les valeurs possibles sont « authorization\_code », « implicit » ou « refresh\_token ». |
| application\_type | Type d’application | Optionnel  La valeur par défaut est web.  Peut prendre comme valeur « native » ou « web ». |
| sector\_identifier\_uri | URL utilisé afin de calculer le sujet « pairwise » | Optionnel  L’URL doit avoir un schéma « https » |
| subject\_type | Type de sujet | Optionnel  Les valeurs possibles sont : « pairwise » ou « public »  La valeur par défaut est configurable dans les options du serveur. |
| id\_token\_signed\_response\_alg | Algorithme alg JWS. | Obligatoire si le paramètre response\_types possède la valeur « id\_token ».  La valeur par défaut est RS256. |
| id\_token\_encrypted\_response\_alg | Algorithme alg JWE. | Par défaut aucun cryptage n’a lieux. |
| id\_token\_encrypted\_response\_enc | Algorithme enc JWE | Si le paramètre “id\_token\_encrypted\_response\_alg » est spécifié alors la valeur par défaut est « A128CBC-HS256 » |
| userinfo\_signed\_response\_alg | Algorithme alg JWS | Par défaut le résultat est retourné en JSON. |
| userinfo\_encrypted\_response\_alg | Algorithme alg JWE | Par défaut aucun cryptage n’a lieux. |
| userinfo\_encrypted\_response\_enc | Algorithme enc JWE | Si le paramètre “userinfo\_encrypted\_response\_alg” est spécifié alors la valeur par défaut est “A128CBC-HS256” |
| request\_object\_signing\_alg | Algorithme alg JWS utilisé par le client afin de signer l’objet requête. | Par défaut aucun algorithme n’est utilisé. |
| request\_object\_encryption\_alg | Algorithme alg JWE utilisé par le client pour crypter l’objet requête.  La clef symétrique utilisée est le « client\_secret » | Par défaut aucun cryptage n’a lieux. |
| request\_object\_encryption\_enc | Algorithme enc JWE | Si le paramètre « request\_object\_enryption\_alg » est spécifié alors la valeur par défaut est « A128CBC-HS256 ». |
| default\_max\_age | Temps maximum d’authentification. | La valeur par défaut provient de la configuration. |
| require\_auth\_time | Indique si le claim  « auth\_time » est requis. | La valeur par défaut est false. |
| default\_acr\_values | Liste des ACR par défaut. |  |
| initiate\_login\_uri | A FAIRE | A FAIRE |
| request\_uris | A FAIRE | A FAIRE |

#### Nouveau PARAMETRES

|  |  |  |
| --- | --- | --- |
| Paramètre | Description | Règles |
| token\_signed\_response\_alg |  |  |
| token\_encrypted\_response\_alg |  |  |
| token\_encrypted\_response\_enc |  |  |

# SCIM2.0

Le « media-type » d’une requête SCIM doit être égale à « *application/scim+json* ».

Une ressource SCIM est représentée sous la forme d’un object JSON. Dans chaque représentation d’une ressource il y a un attribut « schemas » indiquant les schémas SCIM, cet attribut est utilisé afin de déterminer la liste des attributs contenu dans la ressource.

La liste des schémas peut être récupérée via l’URL relative « /schemas ».

## Les attributs communs

Les ressources SCIM ont des attributs en commun

|  |  |
| --- | --- |
| Attribut | Description |
| id | Identifiant unique d’une ressource SCIM généré par le service.  Chaque représentation d’une ressource doit inclure un identifiant. |
| externalId | Identifiant de la ressource défini par le client. |
| meta | Contient les méta données d’une ressource :   * **resourceType** : Type de ressource * **created** : Date de création * **lastModified** : Dernière date de modification * **location** : URL de la ressource * **version** : Numéro de version de la ressource (repris dans l’ETAG). Si une représentation possède un numéro de version (ETAG) et que le processus de génération de la version ne respecte pas les caractéristiques de « ‘strong validator » alors l’ETAG doit être préfixé par « W/ » (case sensitive). |
| Schemas | Cet attribut est requis et contient la liste des schémas. |

## créer une ressources

<https://tools.ietf.org/html/rfc7644#section-3.3>

<https://tools.ietf.org/html/rfc7644#section-3.12>

Pour créer une ressource, une requête HTTP POST doit être exécutée sur l’un des points d’entrées « /Users » ou « /Groups ». Les attributs sont traités de la manière suivante :

* Les attributs dont la « mutability » est égale à « readOnly » doivent être ignorés.
* Les attributs dont la « mutability » est égale à « readWrite » et qui ne sont pas présents dans la requête peuvent avoir une valeur par défaut.
* Les clients qui veulent redéfinir la valeur par défaut peuvent spécifier « null » ou une liste vide « [] ».

Lorsque la ressource est créée :

* HTTP HEADER :
  + Status code = 201
  + Location = URL de la ressource.
  + ETAG : Concerne le numéro de version de la ressource.
* HTTP BODY :
  + L’identifiant de la ressource.
  + L’identifiant externe si besoin
  + Les méta données
    - resourceType : User
    - created : date de création
    - lastModified : dernière date de modification
    - location : URL de la ressource
    - version : Numéro de version.
  + Propriétés de la dernière représentation de la ressource.

Si un conflit est détecté durant la création de la ressource alors :

* HTTP HEADER
  + Status code = 409 (Conflict)
* HTTP BODY :
  + scimType = uniqueness

Si la requête ne respecte par le schéma alors renvoyer l’erreur :

* HTTP HEADER
  + Status code = 400
* HTTP BODY :
  + scimType = « invalidSyntax »
  + detail = « Request is unparsable, syntactically incorrect, or violates schema”

# TODOLIST

|  |  |
| --- | --- |
| Tâches |  |
| Ajouter un endpoint « OAUTH » pour pouvoir ajouter un client. | OK |
| Vérifier si c’est possible d’utiliser « response\_type »=  « token » avec l’API OAUTH2.0 | OK |
| Supporter le paramètre « software\_statement » | OK |
| Ajouter les règles de validation OPENID (enregistrement d’un client) | OK |
| Vérifier tous les algorithmes utilisés pour l’encryption et la signature d’un identity token | OK |
| Vérifier tous les algorithmes utilisés pour l’encryption et la signature du paramètre « request ». Vérifier si le endpoint JWKS expose que les clefs publiques. | OK |
| Supporter le paramètre « response\_types » | OK |
| Générer le token JWS. Le endpoint JWKS du serveur OPENID doit retourner la clef publique utilisée pour vérifier la signature | OK |
| Générer le token JWE. Le endpoint JWKS du client doit retourner la clef publique utilisée pour encrypter JWE. | OK |
| Le endpoint JWKS du serveur OPENID doit posséder les clefs publiques que le client puisse encrypter la requête. | OK |
| Rediriger l’utilisateur vers un AMR par défaut | OK |
| Gérer l’annulation d’un « consent » | OK |
| Supporter le paramètre « prompt ». Il y a une boucle infinie sur « Consents » lorsque « prompt » = « consent » | OK |
| Corriger le module « pwd » | OK |
| Supporter « prompt »= « select\_account » | OK |
| Supporter le paramètre display | OK |
| Supporter le paramètre prompt | OK |
| Supporter le paramètre login\_hint | OK |
| Supporter le paramètre acr\_values  Développer l’écran d’authentification par SMS.  Créer une classe de base pour aider à la création d’un nouveau moyen d’authentification. | OK |
| Nettoyer GIT :   1. Ajouter script « ps1 » pour construire la solution 2. Créer deux Nuget packages    1. OPENID    2. OAUTH 3. Refactorer le projet « MutiAccount » pour utiliser « Cookie ». Même pas nécessaire de faire le « MultiAccount », juste nécessaire d’utiliser cookie avec l’event. 4. Créer un Nuget package « Bootstrap4 » 5. Effacer tout l’historique et avoir une branche « release/1.0 ». | OK |
| Ajouter automatiquement les commentaires | OK |
| Ajouter de l’ASYNC dans tous les contrôleurs | OK |
| Ajouter des scopes par défaut dans OPENID | OK |
| Ajouter des JWKS par défaut dans OAUTH | OK |
| Modifier NUSPEC pour copier les fichiers | <https://stackoverflow.com/questions/40469058/nuget-content-files-in-net-core-solution-not-getting-copied-when-installing-thr> |
| Pouvoir créer une ressource SCIM2.0 | A FAIRE |
| Pouvoir récupérer une ressource SCIM2.0 | A FAIRE |
| Pouvoir supprimer une ressource SCIM2.0 | A FAIRE |
| Ajouter les événements | A FAIRE |
| Ajouter un benchmark concernant les performances | A FAIRE |

Trouver un moyen simple de transformer les fichiers « .pp » dans NUGET.

Un « id\_token » est encrypté et retourné de cette façon :

1. Le token est signé et produit un token JWS.
2. Le token JWS est ensuite encrypté et donne un token JWE.
3. Le Relying Party (RP) enregistre ses clefs publiques durant le processus d’enregistrement.
4. Le document JWKS doit contenir :
   1. Clef publique pour vérifier la signature
   2. Clef publique pour encrypter le contenu.

Un client qui doit vérifier un identity token doit récupérer la clef publique du document JWKS du serveur OPENID.

Un client qui doit décrypter un identity token doit utiliser sa clef privée.

Vérifier signature :

<https://connect2id.com/products/nimbus-jose-jwt/examples/jws-with-rsa-signature>

Crypter :

<https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-rsa-encryption>