# Notes sur le module : *Sécurité Applicative.*

Armel Pitelet

27 février 2022

## Table des matières



## 1 Généralités

Note :
Il est préférable de ne jamais faire de la sécurité à la main ! Authentification et authorisation (surtout plublic) sont des exemple de choses à ne pas faire à la main.

Remarque : *Sur la prmeière sécurité en Web.*
Elle est dans le s du https. Cette norme demande un certificat ssl.

Remarque : *Sur les certificats de sécurité.*
Il en existe deux grandes familles : les simples (my-app.ipme.com) ou les wildcard (*.ipme.com). Certaine entreprises sont authorisées à émettre des certificats validés. Ces denrier sont reconnue par la plupart des navigateurs. Les certificats sont tous issus de certificats racines (en nombre limité), la confiance vient de la. Un certificat est composé de deux partie, une clef publique et une clef privée. C'est la clef privée qui permet la génération de sous certificat. L'intéret du certificat est de chiffré une page. Il possède une date de validité. En générale il vaut mieux ne pas laisser trainer les clefs privées

Note :
Pour de l'authentification il ne faut pas utiliser de Get (https). Il vaut mieux passer par un post car le body de celui ci est sécurisé, ce qui n'est pas le cas du header du Get.

Note :
Les certificats Self-sign sont des certificats qui ne dérive pas d'une racine connue et vérifié.
Note :
Pour générer des certificats : **openssl**.

Remarque : *Sur un système d'authorisation dans l'api.*
Cela est nécessaire car en générale on fait de l'API Stateless. Pour faire de l'authorisation on peut passer par de la Session ou du Token.

Remarque : *Sur les Session.*
c'est un dictionnaire avec identifiant. C'est le serveur qui maintien la session via un cookie. Attention car ce n'est pas évident à mettre en place avec une API STateless.

Remarque : *Sur l'authentification Toekn Based.*
Voir **OAuth 2.0**. On a en fait trois composant distincts CLient, Api, Idp (identity Provider). C'est l'Idp qui valide une entité et retourne un Token JWT (JSON Web Token). Le token fait office d'identité. Contrairement au Session, ici c'est le client qui maintien son token, pas le serveur.
Note :
Sur le **jwt**. Le token est encrypté et ce site permet de le lire. Il ne faut pas mettre d'id ou de chose qui change dans un token car ces derniers ont une date validé. La donné en base en rapport avec l'utilisateur peut changer .

Note :
Pour faire de l'identification en dotnet, on peut passer par **identity server** ou bien **OpenIddict** (plus custom mais moins facile à prendre en main). On peut également voir **Auth0** qui en plus gèr ele multi tenant (une adresse peut se connecter à plusieurs comptes différents).

Note :
en plus de **OAuth 2.0** il existe aussi l apossibilité de rajouter **OpenId** par dessus. L'interet est de pouvoir utiliser un système de scope qui permet de donner des droit spécialiser (comme lire un article spécifique...).

Remarque : *Sur les **CORS**.*
Une autre couche de sécurité qui permet de gérer les droits au niveau de l'API. Il est également possible d'utiliser sa pour masquer certaine infos dans les headers.

Note :
Le sports http et https sont des port réservé en web !
Note :
Sue le couple login/mdp, il ne faut pas renvoyer d'info précise sur l'existence de l'email ou du mdp.

Note :
Un autre système d'authentification : **keycloak**.

# 2 Sur Auth0

Remarque : *Sur la notion de tenant.*
En gros c'est un tenant pour une api.

Remarque : *Sur l'onglet application.*
Ici une application est un composant qui va utiliser Auth0. Par exemple l'api et/ou les fronts. Le *Domain* est le nom du tenant associée, le Client Id est une clef publique, la clef privée sert lors des discussions de machine à machine.

Remarque : *Sur les Single Page Application.*
Angular en est une. Il s'agit d'une application qui génère un fichier. ID token expiration donne

la durée de vie d'un token d'authentification.

Remarque : *Pour pouvoir utiliser Auth0 dans angular.*
Il faut passer par une librairie dédiée via un npm install. Il faut créer un nouveau module : app-authentification.modules.ts et importer ici le module (Authmodule.forRoot(domain : MonDomaine, clientId : monIdDeClient)). Il faut ensuite importere ce module dans le module principale. Une fois ces méchanismes mis en place il suffit d'utiliser un service le AuthService. On peut l'utiliser ensuite pour faire authService.LoginWithRedirect() qui demande une nouvelle page. On peut faire aussi via un popup au lieu d'une nouvelle fenêtre. Pour povoir utiliser correctement cette fonction il faut du coté de Auth0 configuré l'url de callback. Il faut également configurer le moyen d'authentification (password par exemple). Il faut également donner l'url de l'application en WebOrigin

Remarque : *L'annotation authorize.*
Au niveau d'un controller permet de demander l'authentification pour avoir accès à un endpoint donné. Pour faire cela il faut installer micorsoft.AspNetCore.Authentification.JwtBearer. Puis builder.services.AddAuthentification(JwtBearerDefault.AuthenticationScheme).AddJwtBearer (options => options.Authority = "domainDeAuth0" ; options.Audience ="adressDeLAPI" ;
options.TokenValidationParamaters = new ToeknValidationParameters { NameClaimType = ClaimTypes.NameIdentifier}). On peut utiliser une annotation [AllowAnonymous] pour bypasser l'authorisation. Si ce système est activé il faut spécifier à angular d'utiliser un httpInterceptor : {allowedList : {uri ="apiUrl*", tokenOptions {audience : "adressAPI", scope : "read :current_user" }}}. L'étoile demande une injection de token via header et le scope ?=. Attention ces tokesn options ne semblent pas fonctionner correctement.

Note :
l'audience dans un token est la liste des appel possibles
Finalement pour l'API puisse discuter avec un systême elle a aussi besoin d'être passé sous forme d'app dans Auth0. Ici c'est uniquement pour autoriser la validation des tokens par l'api.

# 3 Sur la gestion des Roles

Remarque : *Pour bien gérer les roles.*
Il faut passer par une matrice de Role qui lie un role à différente permission sur différente fonctions. Cela force tt de même à gérer les rôles un peu partout dans l'application.

Remarque : *sur la gestion des dépendances dans angular.*
Il est possible de faire un npx npkill (package à installer) pour nettoyer tt les choses installer par un npm install. Cela permet de ne pas laisser node prendre trop de place lorsque l'on utlise pas les projets angular.

Remarque : *Sur les permissions.*
Pour gérer les droit d'accès à l'api on passe par les permissions dans les settings de l'API Auth0.

Remarque : *Pour faire de l'authentification basée sur les Roles.*
Dans les settings il faut activer les RBAC dans les Settings de l'API sur Auth0. Puis pour passer automatiquement les permissions lors de la connection (que l'on définit du coté de l'appli) on doit cocher la case du dessous (Add Permissions in the Access Token). Cela va modifier le scope dans le token d'identification et ajouter un champ permissions qui va contenir les permissions (au lieu

d'être dans le scope).

Remarque : *Sur la gestion des permissions au niveau de l'API.*

Il faut dans le builder.Service ajouter un AddAuthorization(o => o.AddPolicy("NomDePolice", policyBuilder => policyBuilder.RequiredAssertion(c => c.User.Identity...))) ou c est le context d'authorisation de la requête. Ici on peut faire un User.HasClaim("permissions", "NomDePermission"). Le context contient égalemenr un champs Ressource qui contien la requête http et donc le path. Si on est restfull on peut extraire le nom de la ressource pour s'en servir lors du check du nom de permission. Il est possible au niveau du builder de donner un AddRequirements pour ajouter en cascade des classes de Requirements spécifique. Attention contrairement à RequiredAssertion on ne peut pas faire de choses génériques avec sa.

Remarque : *Sur les claim.*

Lors de la config de l'autorisation, on peur récupérer les claims qui contiennent en fait le contenue du token d'authentification

Remarque : *Sur l'authentification et l'autorisation.*

Dans l'API l'authentification permet de gérer quel sont les tokens nécessaire pour l'identification. L'authorisation permet de définir des polices (que l'on peut ensuite appliquer aux controllers) qui vont vérifier certaine condition dans les tokens

Remarque : *Sur l'attribut Authorize.*

Si on décore un endpoint avec on force l'authentification. Les options que l'on peut donner à authorize permettent de demander des polices d'authorisation spécifiques pour accéder au endpoint (via Authorize(Policy = "MaPoliceAMoi")). Attention tt sa se joue dans le program.cs. Il faut déclarer les UseAuthtentication UseAuthorization dans le bonne ordre (d'abord autentification puis authorisation).

Remarque : *Sur le type* ***params****.*

Pour définir un argument de fonction. Cela permet de passer à la suite de params une Array d'un type donnée sans avoir à la passer comme une vrai liste. On peut passer les composants de la liste les uns après les autres dans l'appel de la fonction au lieu d'avoir à construire la liste avant de la passer.

Remarque : *Sur le AddRequirements.*

Il faut lui passer des objets qui implémentent IAuthorizationRequirement. Pour que cette classe fonctionne il faut qu'elle soit accompagné d'une AutorizationHandler<T (T est la classe de Requirement) qui doit implémenter un handle Requirement. Le handler va gérer l alogique de validation. La classe de requirement est là pour recevoir des paramètres au besoin. Il faut ensuite passer le Handler au service via un AddSingleton ou AddScoped. Attention par défaut si l'on n emet pas de Context.Succeed(requirement) dans le handler le comportement par défaut et d'utiliser un Context.Fail() avant le return Task.Completed. C'est lourd comme méthode mais en définissant un constructeur on peut aussi récupérer via DI la même chose que dans les controlleurs et donc faire de la validation dessus. ATtention à ne pas faire du singleton dans ce cas là.

Remarque : *Sur l'id et l'acces token.*

Access Token : Token qui n'est la que pour gérer le dialog avec une api (avec audience). C'est celui qui est envoyé pour communiquer avec une api. id_token est à destination des applis. C'est là que l'on a les informations sur les User par exemple (attention ce n'est pas sa qui correspond au raw json que l'on peut voir sur Auth0). Il existe également le refresh toekn qui permet de rerécupérer un access token sur le point d'expirer.