

Les API d'authentification (OAuth, ..)

Table des matières

I. API authentications et JWT	3
II. Exercice : Quiz	6
III. OAuth	7
IV. Exercice : Quiz	13
V. Essentiel	14
VI. Auto-évaluation	15
A. Exercice	15
B. Test	15
Solutions des exercices	16

I. API authentications et JWT

Durée : 1 h

Environnement de travail : un PC

Contexte

Le web couvre de nombreux domaines et les sites ou applications stockent de plus en plus de données personnelles nécessitant de pouvoir identifier clairement, facilement et de façon sécurisée qui est l'utilisateur et quelles sont ces données.

Dans ce cadre, des solutions d'authentification ont commencé à faire leur apparition pour permettre de « *standardiser* » les procédures de connexion des utilisateurs.

Certaines de ces solutions sont prévues pour être déployées uniquement sur les sites du développeur, quand d'autres permettent une interopérabilité avec d'autres sites ou services, en limitant (ou non) l'accès aux données des utilisateurs.

Il s'agit par exemple, des connexions via un bouton permettant de se créer un compte sur un site A, à partir de son compte Google ou Facebook.

Ces solutions sont particulièrement intéressantes car les utilisateurs se connectent sur le site du service et non de l'application.

Dans ce cours, nous verrons les principales méthodes d'authentification notamment grâce à JWT et à OAuth.

Nous verrons aussi la connexion OAuth via un cas concret avec GitHub, pour mieux comprendre le fonctionnement

Un élément important pour n'importe quel site dynamique est l'identification, elle permet à l'utilisateur de se connecter et de signaler son identité auprès du site.

Pour cela, la norme est aujourd'hui au classique formulaire courriel et mot de passe qui se trouve sur une grande majorité des sites, avec des dérivés faisant appel à un nom d'utilisateur (username) ou à un numéro de téléphone.

Ces données consistent alors à identifier l'utilisateur auprès du serveur et des services qui lui sont associés et à garder cette identification pendant toute la visite de l'utilisateur sur l'application (et dans certains cas, au-delà)

Nous pouvons donc remarquer qu'il existe plusieurs étapes propres à l'identification :

- L'envoi des données utilisateurs,
- La réception de ces données et l'identification de l'utilisateur,
- L'enregistrement de cette identification,
- L'utilisation du service via les identifiants cryptés.

Pour cela de nombreuses solutions et méthodes existent, la première et la plus ancienne est le stockage des données de connexion encrypté par la session du navigateur.

Cette solution était utilisée il y a un certain temps mais n'est plus tellement utilisée, principalement dû à un problème de sécurité.

Elle a été remplacée par une variante consistant à faire travailler un jeton d'accès (Access token) qui sera l'élément stocké par le client :

- L'envoi des données utilisateurs,
- La réception de ces données et l'identification de l'utilisateur,
- L'enregistrement de cette identification et la création d'un token,
- L'utilisation du service via un token.

Le principal avantage du token est qu'il est spécifique à :

- 1 utilisateur
- 1 session
- 1 durée

Ainsi si un token est acquis par des personnes mal intentionnées, l'impact sera beaucoup moins grave qu'une fuite des identifiants.

C'est notamment dû à la durée de vie du token, le rendant par conséquent automatiquement non valide à partir de son temps d'expiration.

De plus, la majorité des utilisateurs utilisant le même mot de passe à plusieurs endroits (pratique pourtant déconseillée), une fuite du token ne divulguerait pas ce dernier.

Le token est donc devenu une norme incontournable de l'authentification.

Cette dernière possède alors 2 grandes méthodes très utilisées pour l'authentification, le JWT ou Json Web Token et le OAuth pour Open Authorization. Ces 2 méthodes ont des approches différentes de l'authentification bien que basée pour les 2 sur l'utilisation de Token.

Les 2 solutions sont par ailleurs totalement Open Source et peuvent se retrouver sur des sites particulièrement connus.

Ces méthodes sont des protocoles intégrés au web et faisant partie des normes de sécurité. Elles possèdent chacune une API qui lui est propre.

Définition

Une Api ou Application Programming Interface est une couche applicative permettant de connecter 2 services ensemble. Dans la majorité des cas l'API ne tient pas compte d'un langage en particulier et fonctionne avec l'ensemble des projets. Ainsi, par exemple, une API peut relier un service Python avec une requête provenant de code JavaScript sans problème.

Le JWT

Le Json Web Token est un protocole normé et permettant d'encoder des informations afin de conserver un Token à information multiple en base de données.

Ces informations multiples sont fournies au format JSON et peuvent ainsi être conservées.

Le JWT est donc un objet JSON (ou JavaScript Object Notation) qui se décompose en 3 parties :

- L'en-tête
- Les données principales
- La signature

Le premier permettra de transmettre le type d'encryptage et le type d'objet (ici obligatoire jwt)

Le second transite toutes les informations principales de l'objet JWT, par exemple, les données que le serveur fournira comme un username, des valeurs de validité, des valeurs de droit d'accès et plus largement tout ce qui sera défini par le serveur.

La troisième et dernière partie concerne la signature, elle permettra de vérifier la conformité et la validité du token avec le serveur. Cette partie est dépendante de l'algorithme notifié dans la première partie.

Attention

Malgré l'encodage, il convient encore une fois de ne pas stocker le mot de passe dans le Token généré, afin de limiter les risques en cas de fuite de la méthodologie de décodage.

Remarque

Bien que possédant une liberté énorme sur le contenu, il est recommandé de limiter les données pour ne pas surcharger le token et/ou créer de faille, les valeurs contenues dans le JWT doivent être exploitées.

Méthode	Génération d'un token JWT
----------------	----------------------------------

Ainsi pour les informations suivantes :

- En-tête : Algorithme HS256 et type JWT,
- Information : moment de création dans iat au format timestamp et nom de l'utilisateur (ici David Smith),
- Information de signature : Générer automatiquement.

Le token sera le suivant :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1YW1lIjoieGF2aWQgU21pdGgiLCJpYXQiOiJlMTYyMzkwMjJ9.Bj-iz6E2X3SHbu5tuwJRfM0zdJugvITGQl0jiGXAcI0

Les différentes parties du token sont séparées par les points (par exemple : « eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 » définira l'en-tête).

Le token transmet les informations aux serveurs via une simple chaîne de caractère encryptée.

Cette opération peut être réalisée facilement sur le site officiel (jwt.io) et via leur outil comme la capture ci-dessous :

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiaWoiRGF2aWQgU21pdGgiLCJpYXQiOiE1MjYyMzkwMjJ9.Bj-iz62E3XSHbu5tuvwJRfM0zdJugvITGQ10jiGXAc10
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "name": "David Smith",  "iat": 1516239822}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret) ☐ secret base64 encoded
```

 Signature Verified

SHARE JWT

Capture d'écran site jwt.io, création d'un token depuis le site

La plupart des APIs, créé automatiquement l'en-tête et la signature, ainsi, seules les informations (le Payload) doivent être fournies.

Remarque

Comme spécifié dans l'en-tête, le protocole JWT supporte plusieurs algorithmes d'encryptage (ou de signature). Ces derniers dépendent principalement de l'API qui sera exploitée et de sa compatibilité avec chacun d'entre eux. La page de présentation des différentes API de JWT sur le site officiel jwt.io présente cette information de façon nette avec la compatibilité sur la partie de droite de chaque carte représentant une API, comme sur l'image ci-dessous :

Node.js	Node.js	Node.js
MINIMUM VERSION 4.2.2		
<div> <div>Sign</div> <div>Verify</div> <div>iss check</div> <div>sub check</div> <div>aud check</div> <div>exp check</div> <div>nbf check</div> <div>iat check</div> <div>jti check</div> <div>typ check</div> </div> <div> <div>HS256</div> <div>HS384</div> <div>HS512</div> <div>PS256</div> <div>PS384</div> <div>PS512</div> <div>RS256</div> <div>RS384</div> <div>RS512</div> <div>ES256</div> <div>ES256K</div> <div>ES384</div> <div>ES512</div> <div>EdDSA</div> </div>	<div> <div>Sign</div> <div>Verify</div> <div>iss check</div> <div>sub check</div> <div>aud check</div> <div>exp check</div> <div>nbf check</div> <div>iat check</div> <div>jti check</div> <div>typ check</div> </div> <div> <div>HS256</div> <div>HS384</div> <div>HS512</div> <div>PS256</div> <div>PS384</div> <div>PS512</div> <div>RS256</div> <div>RS384</div> <div>RS512</div> <div>ES256</div> <div>ES256K</div> <div>ES384</div> <div>ES512</div> <div>EdDSA</div> </div>	<div> <div>Sign</div> <div>Verify</div> <div>iss check</div> <div>sub check</div> <div>aud check</div> <div>exp check</div> <div>nbf check</div> <div>iat check</div> <div>jti check</div> <div>typ check</div> </div> <div> <div>HS256</div> <div>HS384</div> <div>HS512</div> <div>PS256</div> <div>PS384</div> <div>PS512</div> <div>RS256</div> <div>RS384</div> <div>RS512</div> <div>ES256</div> <div>ES256K</div> <div>ES384</div> <div>ES512</div> <div>EdDSA</div> </div>
<div>Auth0</div> <div>View Repo</div>	<div>Filip Skokan</div> <div>View Repo</div>	<div>AWS</div> <div>View Repo</div>
npm install jsonwebtoken	npm install jose	npm install aws-jwt-verify

Présentation des solutions JWT avec node.js depuis jwt.io

Exercice : Quiz

[solution n°1 p.17]

Question 1

L'authentification en cryptant les noms d'utilisateurs et mot de passe sur le navigateur du client est une bonne méthode.

- ☐ Faux
- ☐ Vrai

Question 2

Un token peut être similaire à un autre.

- ☐ Vrai
- ☐ Faux

Question 3

Le token, dans la majorité des cas est spécifique :

- ☐ À un utilisateur
- ☐ À une session
- ☐ À une validité
- ☐ À un service

Question 4

JWT signifie :

- ☐ JSON Web Token
- ☐ Just Wanna Token
- ☐ Justice Wild Tokenizer
- ☐ JSON Weird Token

Question 5

Un JWT est composé de :

- ☐ 3 parties
- ☐ 2 parties
- ☐ 1 partie
- ☐ 5 parties

III. OAuth

L'Open Authorization ou OAuth est une méthode d'authentification énormément utilisée dans des projets Open Source comme le service Cloud auto héberger NextCloud et OwnCloud et dans des services propriétaires comme GitHub.

Sa notoriété vient de plusieurs avantages pouvant lui être attribués comme la sécurité d'authentification qu'elle offre, la facilité de mise en place et d'utilisation ou même le fait que cette méthode soit Open Source.

L'OAuth existe actuellement sous 2 formes, sa version 1 appelée le plus souvent OAuth et sa version 2 nommée OAuth2 dans la majorité des cas. Cependant malgré leur nom en commun les 2 versions sont totalement différentes et ne partagent que leurs visions de l'authentification, une réécriture de zéro ayant été réalisée.

Historiquement, OAuth 1 est sorti en octobre 2007, puis a été adoptée par de nombreux acteurs du numérique, notamment Google, très tôt, en 2008, Twitter en 2010, Salesforce et plusieurs autres

En octobre 2012, sort OAuth 2, cependant ce dernier n'est pas une version majeure de sa version 1 mais une méthode totalement repensée. Ainsi, tout ce qui était bloquant avec la première version, façonnera les solutions de la seconde version

Remarque

OAuth 1.0 était déjà reconnu comme une référence en termes de sécurité se basant sur les signatures numériques, mais était difficile d'accès pour les développeurs, notamment dû à une nécessité d'interopérabilité au niveau cryptographique.

Ainsi OAuth 1.0 travaille à l'aide de cryptographie avec les signatures numériques qui atteste de l'authenticité des requêtes et n'est pas dépendant du Secure HTTP.

OAuth 1.0 signe alors chacune des demandes avec la signature numérique fournie et invalide la requête en cas de signature non correspondante.

OAuth 2.0, à contrario, est dépendant du Secure HTTP, pouvant fragiliser la sécurité en cas de défaut sur la sécurisation de la connexion.

Le fonctionnement se fait grâce à des token, pouvant être copiés ou volés rendant le fonctionnement simplifié, mais créant une autre faille possible de sécurité.

OAuth 2.0 prend aussi en charge d'autres clients que les clients en web (la, ou OAuth 1.0 ne prenait que les clients web)

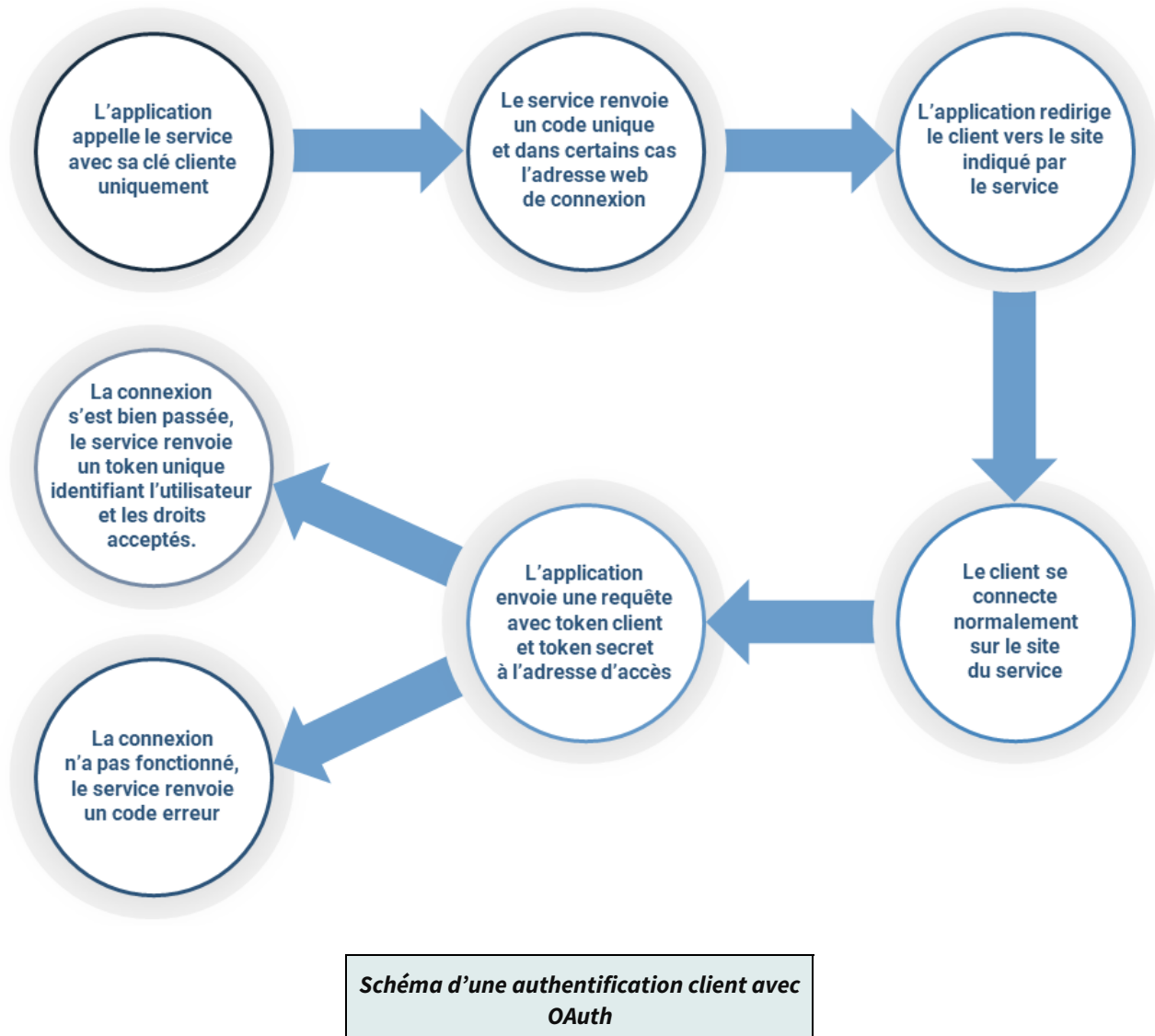
Malgré la réécriture de la méthodologie, le fonctionnement reste tout de même similaire.

Méthode

En méthode, OAuth 1 et 2 fonctionnent de la façon suivante :

1. Le demandeur (l'application) s'authentifie et s'enregistre auprès du service utilisant OAuth.
2. Le service fournit une clé secrète qui lui servira de signature, dans le cas de OAuth 1 et de token, dans le cas de OAuth2 (avec une clé client) pour l'intégralité de l'application.
3. L'application demandeur signe toute ces requêtes avec la clé fournie par le service.
4. Dans le cas, où la clé n'est pas valide, incomplète ou manquante, la requête entière est rejetée.

Pour le client, cela permet à l'application d'accéder à certaines informations présélectionnées sans pour autant devoir fournir son nom d'utilisateur et son mot de passe, l'utilisateur devant identifier par un Token unique comme sur le schéma ci-dessous montrant l'authentification OAuth 2.0.



Ainsi, comme visible ci-dessus, l'utilisateur ne fournira toujours que les données pour obtenir les droits d'accès et sera identifié par un token, donc la liaison avec ces données ne sera connue que par le service.

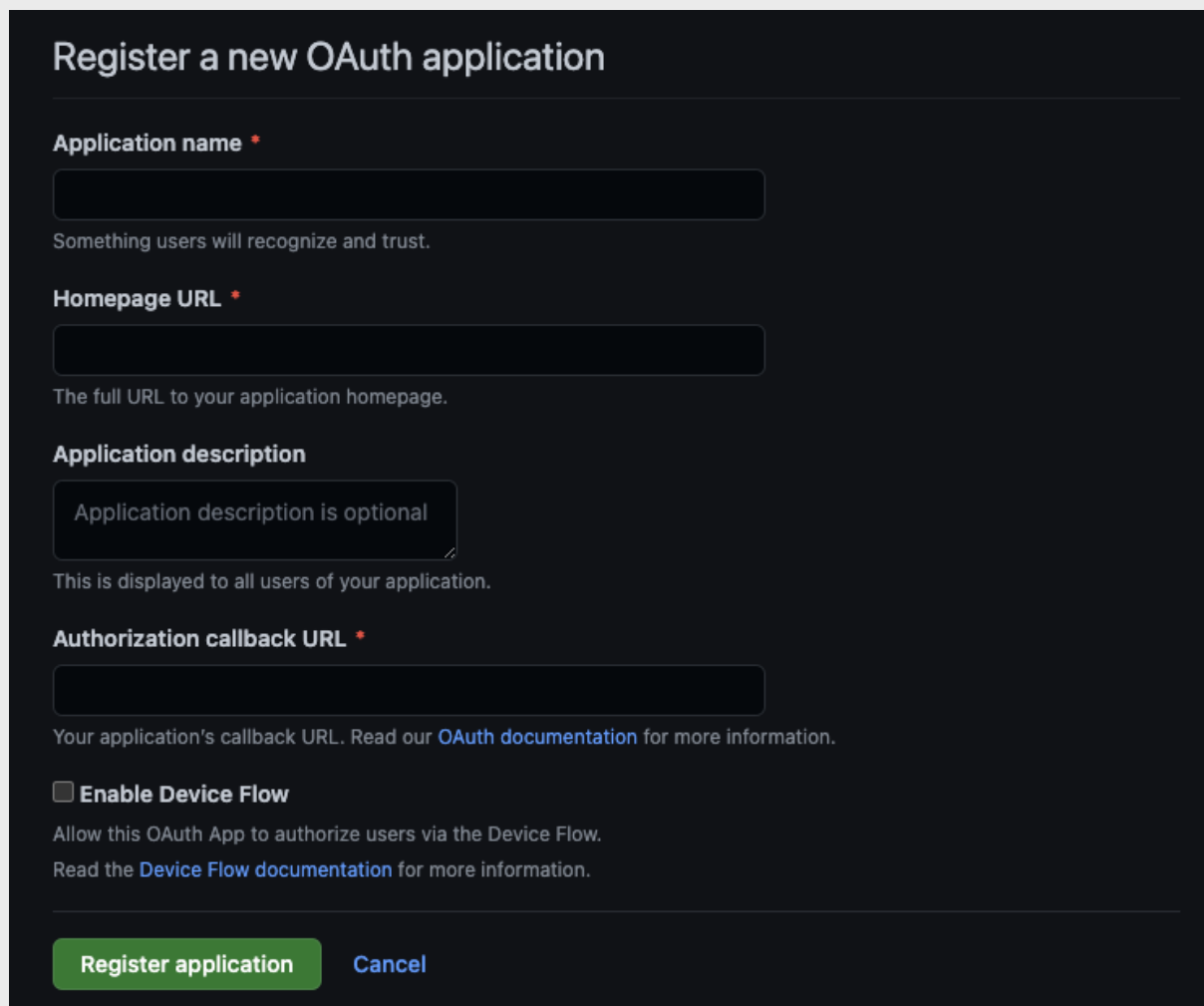
Exemple

Plus concrètement, en partant de la connexion via l'API GitHub qui exploite l'OAuth 2.0, la première étape consiste à se diriger vers la documentation d'authentification du service, pour GitHub, celle-ci se situe à cette adresse : GitHub Docs¹, il est ainsi possible de voir qu'il faut commencer par créer une clé OAuth propre à l'application, cette partie se fait via le lien : Sign in to GitHub², évidemment un compte développeur sur le site du service est nécessaire (dans le cas présent, tous les comptes utilisateur GitHub sont considérés comme développeur, dans certains cas, il s'agit de compte spécifique).

1 <https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps#web-application-flow>

2 <https://github.com/settings/applications/new>

La page suivante s'affiche alors :



The screenshot shows the 'Register a new OAuth application' page on GitHub. It features a dark-themed form with the following fields and options:

- Application name ***: A text input field with the placeholder text 'Something users will recognize and trust.'
- Homepage URL ***: A text input field with the placeholder text 'The full URL to your application homepage.'
- Application description**: A text area with the placeholder text 'Application description is optional' and a note 'This is displayed to all users of your application.'
- Authorization callback URL ***: A text input field with the placeholder text 'Your application's callback URL. Read our [OAuth documentation](#) for more information.'
- Enable Device Flow**: A checkbox with the label 'Enable Device Flow' and a note 'Allow this OAuth App to authorize users via the Device Flow. Read the [Device Flow documentation](#) for more information.'

At the bottom, there are two buttons: a green 'Register application' button and a blue 'Cancel' button.

***Capture d'écran de l'enregistrement d'application OAuth
GitHub***

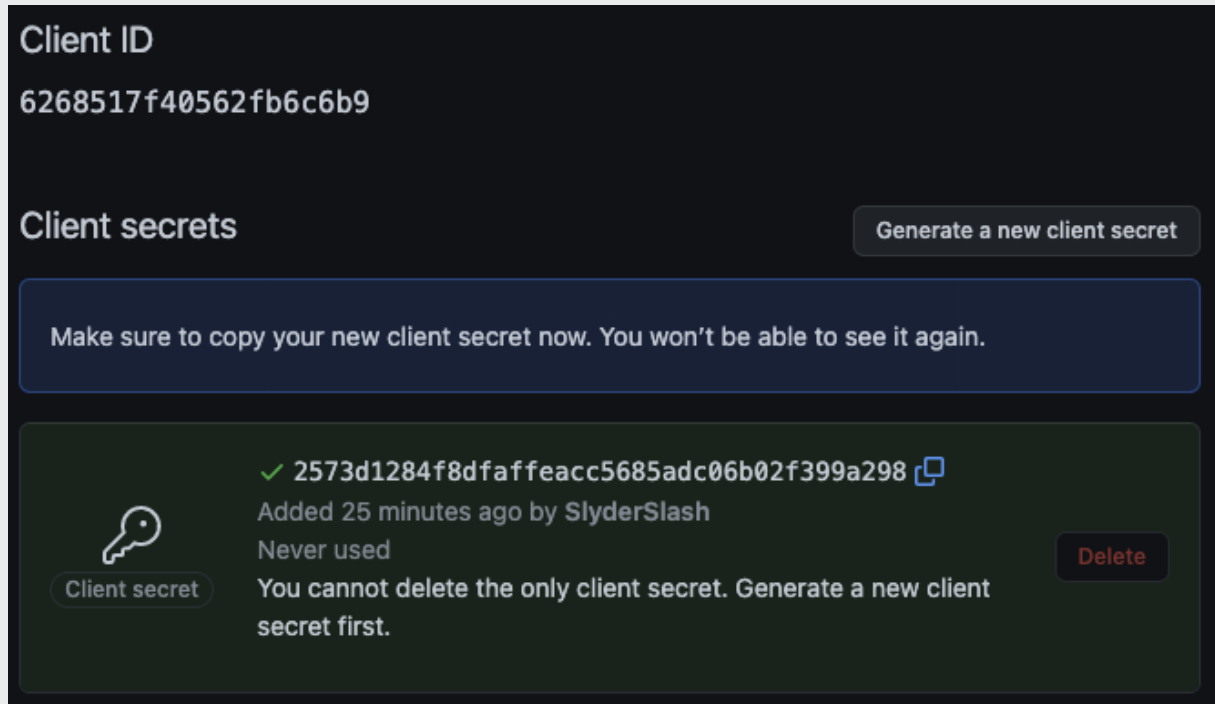
Ici, les champs ont tous une utilité :

- Application name : sera le nom identifiant votre application (celui-ci sera vue par les utilisateurs).
- Homepage URL : est la page d'accueil de l'application.
- Application description : est une description de l'application (dans le cas présent, ce champ n'est pas obligatoire, mais dans certains cas, il sert à faire valider une application par le service).
- Authorisation callback URL : sera l'URL appelé par le service après l'authentification de l'utilisateur.

Ainsi, les champs Application name et Homepage URL, renseignent principalement l'utilisateur sur l'application qui aura accès aux informations.

Le champ Authorisation callback URL, quant à lui, permet de rediriger l'utilisateur vers l'application après sa connexion.

Une fois, cela fait, GitHub fournit un token nommé Client ID qui sera le token client et fournit un token secret en cliquant sur « *Generate a new client secret* ».

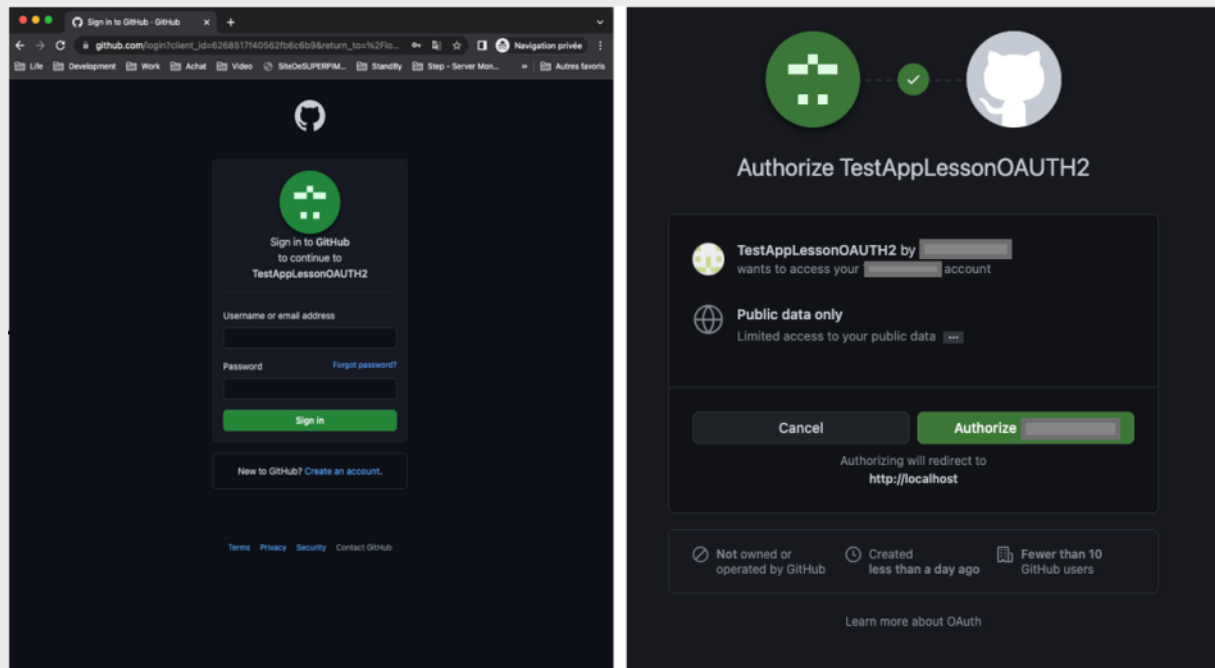


Capture d'écran des tokens OAuth GitHub

Ces tokens permettront d'identifier l'application.

Il est alors possible de faire une requête d'accès pour un utilisateur en appelant l'URL fourni par la documentation GitHub : Sign in to GitHub¹ et en fournissant l'identifiant client comme paramètre de méthode GET.

Ce qui, dans le cas actuel, donne : Sign in to GitHub to continue to TestAppLessonOAUTH2²³



1 <https://github.com/login/oauth/authorize>

2 https://github.com/login/oauth/authorize?client_id=6268517f40562fb6c6b9

3 https://github.com/login/oauth/authorize?client_id=6268517f40562fb6c6b9

Capture d'écran connexion du client via l'API OAuth GitHub

Le client est alors amené à se connecter, directement sur le site GitHub, le site lui fournit alors :

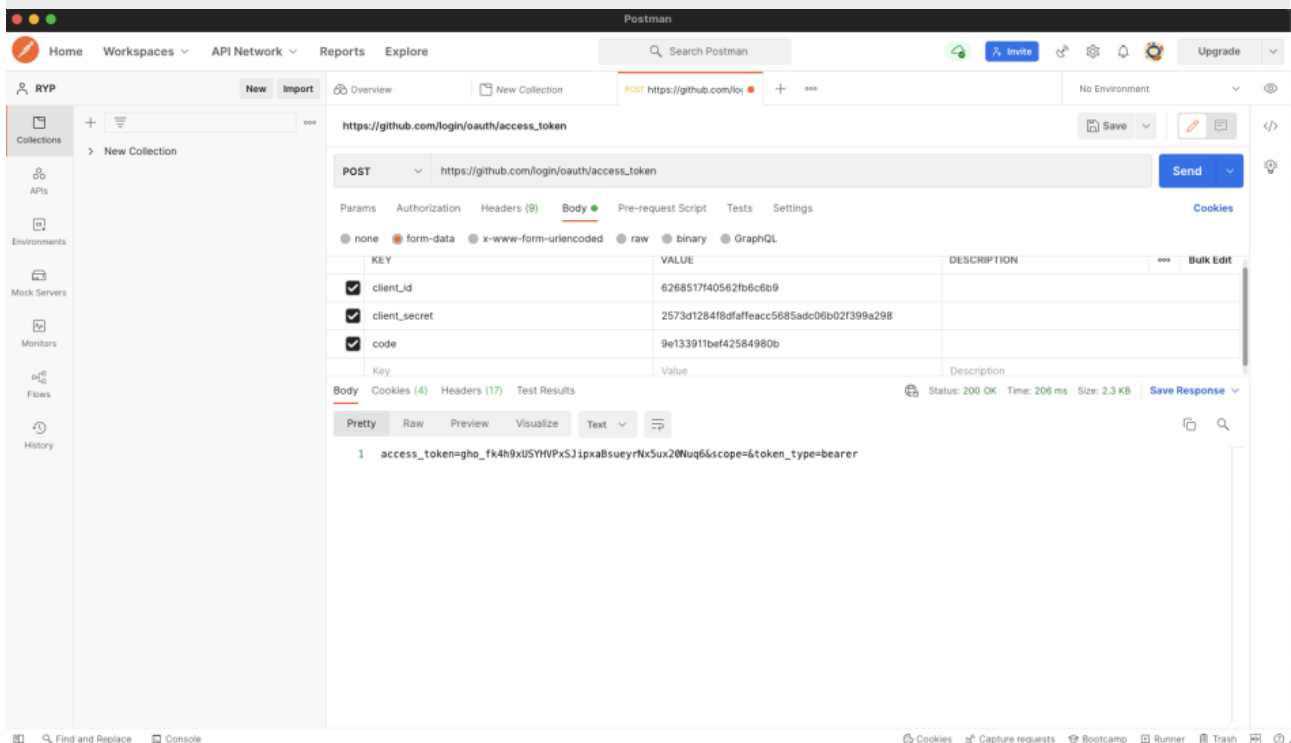
- Le nom de l'application (TestAppLessonOAUTH2 dans l'exemple ci-dessus)
- Le propriétaire de l'application (carrées gris)
- Le champ d'accès aux données (Public data only ici)
- L'adresse de redirection (localhost dans ce cas)

Une fois accepté, l'adresse de redirection est appelée via une méthode GET avec en paramètre nommé code, un code d'accès unique et temporaire.

Celui-ci n'est exploitable que par le propriétaire de l'application (car le token secret sera nécessaire) et permet uniquement d'accéder aux token de l'utilisateur, une fois toutes les données fournies.

L'application doit ensuite réaliser une requête POST avec : le token client, le token secret et le code obtenu.

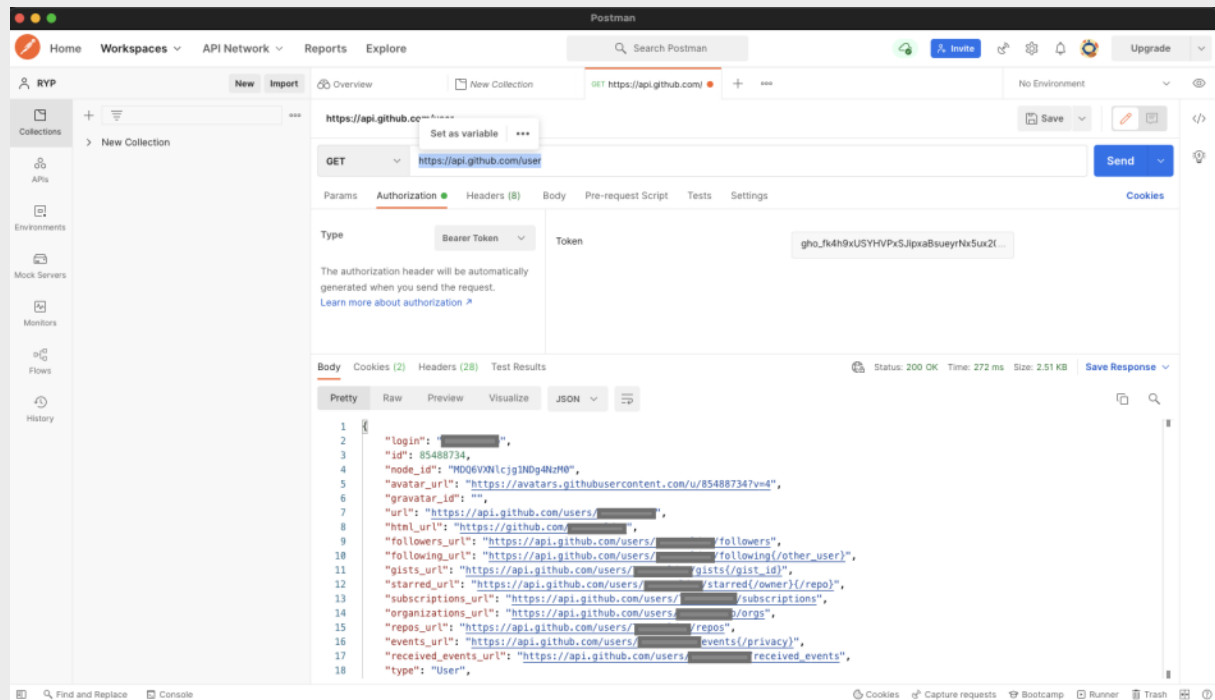
Par exemple, en effectuant la requête avec l'outil Postman :



Capture d'écran de Postman, récupération du token d'accès

Ici, le access_token fourni sera le token identifiant le client pour chacune de ses requêtes, le scope donne les accès spécifiques (ici aucun, uniquement les données publiques) et enfin le token_type permet de connaître la méthode d'identification.

Il est alors possible d'accéder aux données publiques de l'utilisateur via l'API de GitHub, par exemple, en récupérant les données utilisateurs via une requête à l'adresse : GitHub¹ et en passant le token utilisateur grâce à l'autorisation via un bearer token, comme ci-dessous sur Postman :



Capture d'écran de Postman, accès à l'API via l'access token

Ainsi, il est possible d'avoir accès à l'API GitHub, mais de nombreux services fonctionnent aujourd'hui avec ce procédé et de manière similaire, dans de nombreux domaines, par exemple :

- Les services de carte : HERE
- Les services de cloud : NextCloud
- La gestion de version : GitHub
- Les jeux vidéo : Blizzard

L'accès OAuth, peut aussi être ajouté à une API personnelle, ce qui lui permettra de se sécuriser via des méthodes standards.

Remarque

L'utilisation des requêtes API OAuth peut se faire via n'importe quel langage, il suffit uniquement de respecter la documentation du service et ainsi vérifier le type de requête (GET, POST, PUT), le schéma de connexion (bien suivre toute les étapes) et pouvoir récupérer les bonnes données en fonction des réponses.

Exercice : Quiz

[solution n°2 p.18]

Question 1

¹ <https://api.github.com/user>

OAuth signifie :

- ☐ Open Authorization
- ☐ Open Authentication
- ☐ Origin Authorization
- ☐ Origin Authentication

Question 2

OAuth est connu avec :

- ☐ 3 noms
- ☐ 1 nom
- ☐ 4 noms
- ☐ 2 noms

Question 3

Les différentes versions (1.0 et 2.0) de OAuth sont similaires et ne sont qu'une mise à jour majeure :

- ☐ Faux
- ☐ Vrai

Question 4

Une signature d'application est nécessaire avec OAuth 1.0 :

- ☐ Vrai
- ☐ Faux

Question 5

L'utilisateur final se connecte à partir de quel page pour accéder à un service utilisant OAuth

- ☐ Le site du service
- ☐ L'application

V. Essentiel

Les APIs et méthodes d'authentification sont primordiales car elles permettent de se connecter à des services externes mais aussi de protéger une API pour un projet. Dans certains cas, elles consistent uniquement en solution de gestion de Token avec une signature propre à l'application, dans d'autres cas il s'agit de méthodologie avancée permettant d'optimiser l'authentification, sur le plan sécuritaire mais aussi sur la facilité d'accès et de mise en place pour les développeurs.

Dans certains cas, la sécurité est priorisée et la facilité d'accès est plus complexe comme avec OAuth 1.0, et dans d'autres cas, la sécurité est moins importante pour donner accès à une meilleure facilité d'utilisation comme OAuth 2.0.

Il convient alors d'utiliser la méthodologie, la plus proche des besoins de l'application.

Malgré la multitude de solutions existantes, certaines ressortent de façon régulière, notamment le JWT ou JSON Web Token, possédant des méthodologies d'utilisation pour énormément de langage. L'OAuth, quant à lui, est basé sur une API REST et permet à travers de multiples échanges, d'authentifier un utilisateur, sans que l'application ne demande ses identifiants.

VI. Auto-évaluation

A. Exercice

Développeur dans une entreprise, votre responsable vous demande de tester la taille moyenne et le type de token que fournit JWT.

Question 1

[solution n°3 p.19]

Allez sur le site de JWT et effectuez un test avec la valeur de mot de passe (password) : « *breathing!89* » et avec un nom d'utilisateur (username) « *wind* », définissez ensuite les 3 parties du token fourni avec l'algorithme HS512

Toujours dans cette entreprise, votre employeur vous demande en combien d'étape *pour l'utilisateur final* se fera la connexion via la méthodologie OAuth et le site GitHub

Question 2

[solution n°4 p.20]

Trouvez le nombre d'étapes entre la demande de connexion par le client, jusqu'à la récupération de la fiche utilisateur sur GitHub via la méthode OAuth.

Combien d'étapes sont nécessaires ? Quelles sont-elles ? Quelle version de OAuth est utilisée ?

B. Test

Exercice 1 : Quiz

[solution n°5 p.20]

Question 1

Les API d'authentification sont toutes similaires.

- ☐ Faux
- ☐ Vrai

Question 2

JWT est une solution gratuite.

- ☐ Faux
- ☐ Vrai

Question 3

JWT supporte plusieurs algorithmes pour la signature.

- ☐ Vrai
- ☐ Faux

Question 4

OAuth 1.0 avait comme défaut(s) principalement :

- ☐ Sa difficulté d'accès pour les développeurs
- ☐ Sa limite de plate-forme couverte
- ☐ Sa sécurité faible
- ☐ Sa difficulté d'utilisation pour l'utilisateur final

Question 5


OAuth 2.0 a un fonctionnement avec des requêtes standards.

- ☐ Vrai
- ☐ Faux

Solutions des exercices


Exercice p. 6 Solution n°1**Question 1**

L'authentification en cryptant les noms d'utilisateurs et mot de passe sur le navigateur du client est une bonne méthode.

- ☒ Faux
- ☐ Vrai
-  Faux, en cas de fuite cela met en danger le mot de passe.


Question 2

Un token peut être similaire à un autre.

- ☐ Vrai
- ☒ Faux
-  Faux, un Token ou jeton d'accès est par définition unique.


Question 3

Le token, dans la majorité des cas est spécifique :

- ☒ À un utilisateur
- ☒ À une session
- ☒ À une validité
- ☐ À un service
-  Bien que pouvant être spécifique à un service, la plupart des token sont spécifiques à l'utilisateur, sa session et la validité du token.


Question 4

JWT signifie :

- ☒ JSON Web Token
- ☐ Just Wanna Token
- ☐ Justice Wild Tokenizer
- ☐ JSON Weird Token
-  JWT signifie JSON Web Token.


Question 5

Un JWT est composé de :

- ☒ 3 parties
 - ☐ 2 parties
 - ☐ 1 partie
 - ☐ 5 parties
-  Le JWT est composé d'un en-tête, d'un payload et d'une signature, soit 3 parties.


Exercice p. 13 Solution n°2**Question 1**

OAuth signifie :

- ☒ Open Authorization
 - ☐ Open Authentication
 - ☐ Origin Authorization
 - ☐ Origin Authentication
-  OAuth signifie Open Authorization.


Question 2

OAuth est connu avec :

- ☐ 3 noms
 - ☐ 1 nom
 - ☐ 4 noms
 - ☒ 2 noms
-  OAuth est connu à travers 2 noms principalement 1.0 et 2.0.


Question 3

Les différentes versions (1.0 et 2.0) de OAuth sont similaires et ne sont qu'une mise à jour majeure :

- ☒ Faux
 - ☐ Vrai
-  Faux, OAuth 2.0 est une réécriture totale de OAuth à partir du concept de OAuth 1.0.


Question 4

Une signature d'application est nécessaire avec OAuth 1.0 :

- ☒ Vrai
- ☐ Faux
-  Vrai, c'est la base de OAuth permettant son fonctionnement.

Question 5

L'utilisateur final se connecte à partir de quel page pour accéder à un service utilisant OAuth

- ☒ Le site du service
- ☐ L'application
-  L'utilisateur final se connecte sur le site du service, cela lui évite de partager ces identifiants avec l'application

p. 15 Solution n°3

Dans un premier temps, il est nécessaire d'aller sur le site jwt.io¹, une fois dessus, il est nécessaire de choisir le bon algorithme dans la liste déroulante et rentrer les informations au format JSON dans le PAYLOAD :

```
1 {  
2   "username" : "wind",  
3   "password": "breathing!89"  
4 }
```

Et de regarder le résultat obtenu dans la partie Encoded :

eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImJyZWZ0aGluZyE4OSJ9.MnKY5BG07H-eeQvyxc3qxWP68R9iO9eO6__DPFPa7x5SHbR-Ysgrs5Q

Celle-ci est découpé en 3 parties :

- L'en-tête : eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9
- Le Payload : eyJ1c2VybmFtZSI6ImJyZWZ0aGluZyE4OSJ9
- La signature : MnKY5BG07Hc1StQhcrMRnrGnoQz1pAJZ4rYx85zRu-eeQvyxc3qxWP68R9iO9eO6__DPFPa7x5SHbR-Ysgrs5Q

¹ <https://www.jwt.io/>

Algorithm HS512

Encoded
PASTE A TOKEN HERE

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IndpbnQpLCJwYXNkd29yZCI6ImJyZWFOaGluZyE4OSJ9.MnKY5BG07Hc1StQhcrMRnrGnoQz1pAJZ4rYx85zRu-eeQvyxc3qxWP68R9i09e06__DPFPa7x5SHbR-Ysgrs5Q
```

Decoded
EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "username": "wind",
  "password": "breathing!89"
}
```

VERIFY SIGNATURE

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-512-bit-secret
) ☐ secret base64 encoded
```

Capture d'écran

p. 15 Solution n°4

Pour connaître le nombre d'étapes pour la connexion de l'utilisateur final et quelle version de OAuth est nécessaire avec GitHub il est nécessaire de se rendre dans la documentation de ce dernier à l'adresse suivante :

GitHub Docs¹

La version utilisée est la version 2.0 comme notifié dans le premier paragraphe, il y'a au total 6 étapes :

- Envoi de l'utilisateur sur une page avec le client id,
- Récupération du code temporaire,
- Requête à GitHub avec le code temporaire, le client id et le secret,
- Récupération du token identifiant l'utilisateur,
- Appel à l'api utilisateur avec le token,
- Récupération et connexion de l'utilisateur.


Exercice p. 15 Solution n°5

Question 1

Les API d'authentification sont toutes similaires.

- ☒ Faux
- ☐ Vrai


¹ <https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps#web-application-flow>

-  Faux. Bien que principalement basée sur des systèmes de Token ou Signature, chaque méthodologie est unique.

Question 2

JWT est une solution gratuite.


- ☐ Faux
- ☒ Vrai

-  Vrai, il s'agit d'une méthode Open Source (attention néanmoins à respecter tous les termes de la licence).

Question 3

JWT supporte plusieurs algorithmes pour la signature.


- ☒ Vrai
- ☐ Faux

-  Vrai, JWT supporte, à l'heure actuelle, au total 14 algorithmes différents.

Question 4

OAuth 1.0 avait comme défaut(s) principalement :

- ☒ Sa difficulté d'accès pour les développeurs
- ☒ Sa limite de plate-forme couverte
- ☐ Sa sécurité faible
- ☐ Sa difficulté d'utilisation pour l'utilisateur final

-  OAuth 1.0 était principalement défaillant sur sa facilité d'accès pour les développeurs et sa limite, visant uniquement les plates-formes web. Sa sécurité était accrue.

Question 5

OAuth 2.0 a un fonctionnement avec des requêtes standards.

- ☒ Vrai
- ☐ Faux

-  Vrai, il s'agit d'appels GET et POST dans la majorité des cas.