Scénario 0 - Identification et authorisation

Le **backend**

Il est fort probable qu'à l'heure actuelle, vous soyez raisonnablement à l'aise avec le concept général de connection sur une application Web, mobile ou autre : vous soumettez un formulaire avec un identifiant et un mot de passe, et si les informations fournies correspondent, l'application vous donne accès à ses services.

Vous savez aujourd'hui qu'une application Web au niveau du **frontend** peut envoyer plusieurs requêtes suivants la connection de l'utilisateur, mais on ne demande pas le mot de passe à chaque fois. Comment l'API est en mesure de s'assurer que nous avons bel et bien l'autorisation de faire ces requêtes ?

Nous abordons donc les concepts importants dans la gestion des accès : l'identification (*authentication* en anglais) et l'autorisation (*authorization* en anglais).

Identification
: Processus requérant le pairage d'un identifiant (courriel, *username*, numéro de téléphone) à son mot de passe associé. Le succès de cette étape d'identification permet d'augmenter les autorisations de l'utilisateur.

Autorisation
: Processus qui s'assure qu'un utilisateur identifié ou non accèdent uniquement aux informations qu'il a droit. Dans le contexte où l'utilisateur est identifié, le système d'autorisation utilisera ce qu'on appel un jeton (*token* en anglais).

Jeton
: Composé d'une valeur unique généré aléatoirement, associé à une date d'expiration et un utilisateur, celui-ci doit être envoyé par le client au serveur pour chaque requête qu'elle va envoyé dans l'en-tête `Authorization`.

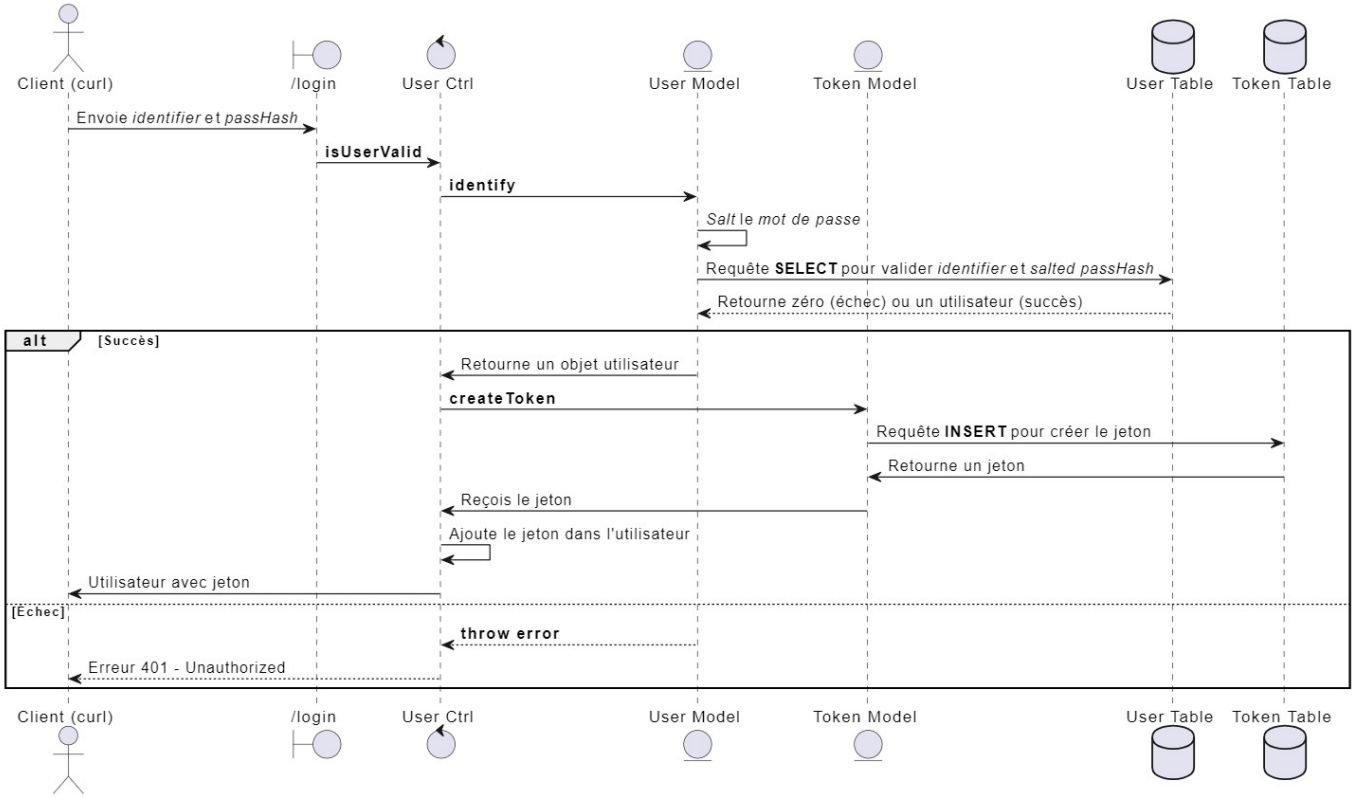
Maintenant, il est temps d'expliquer clairement les différents processus que nous allons avoir besoins de suivre afin de rendre notre API à la fois fonctionnelle mais le plus sécuritaire possible.

> NOTE - Il est important de comprendre que l'aspect sécurité est un domaine à part entière et appliquer toutes les bonnes pratiques dans le contextuel actuel prendrais probablement plusieurs semaines. Nous allons donc viser un *good enough*.

Processus de connection (*login*) vue du *backend*

Pré-requis : notre utilisateur est déjà dans la base de données.

Objectif : établir une connection avec notre utilisateur et recevoir notre jeton.

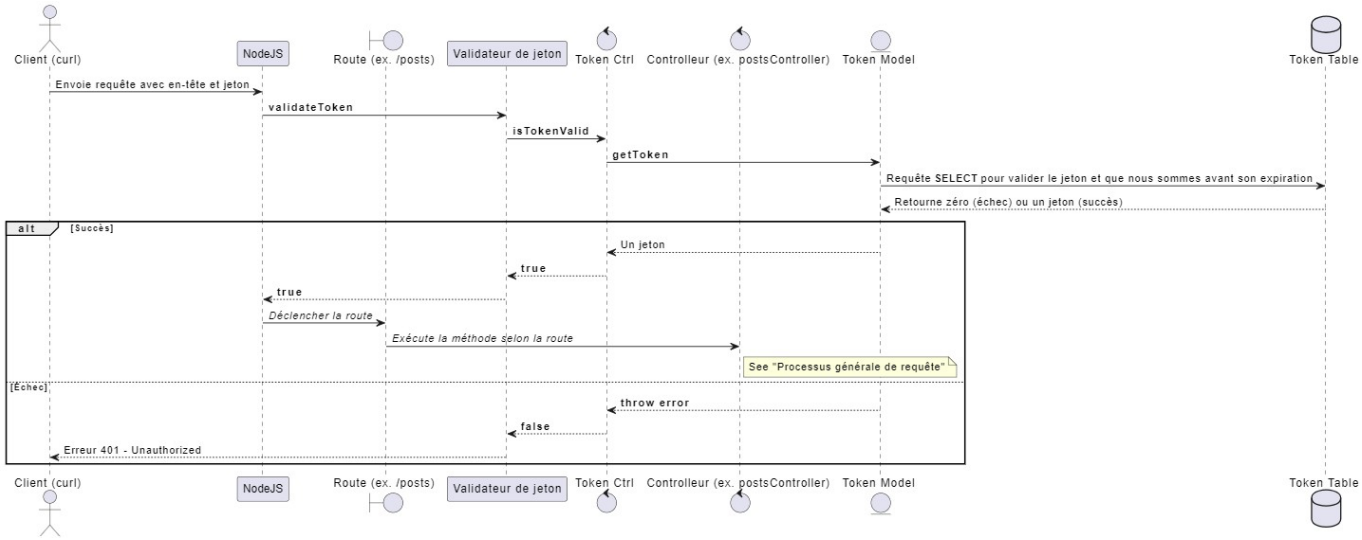


À cet étape, notre utilisateur à reçu les informations pertinentes de sont utilisateurs avec un jeton valide, ou un erreur HTTP 401 *Unauthorized*.

Processus d'autorisation d'une requête

Pré-requis : notre utilisateur est identifié.

Objectif : effectuer une requête quelconque (ex.: `/posts`).



À cet étape, notre utilisateur à reçu les informations pertinentes qu'il a demandé, ou un erreur HTTP 401 *Unauthorized* dans le cas où le jeton était pas le bon ou qu'il était expiré.

Processus de déconnexion

Le processus de déconnexion est très similaire à `Processus d'autorisation d'une requête` : il sera nécessaire de fournir le bon jeton et dans le délais prescrit. Toutefois, en cas de succès, on ne souhaite pas

supprimer le jeton de la table : on ne veut pas que la valeur unique du jeton soit réutilisée dans le future, même si les chances sont fortement improbable. Dans une tel situation, nous allons simplement changer la date d'expiration pour être à l'heure de déconnection.

> Pouvez-vous expliquez les raisons qu'on ne veut pas utiliser deux fois la même valeur unique du jeton ?

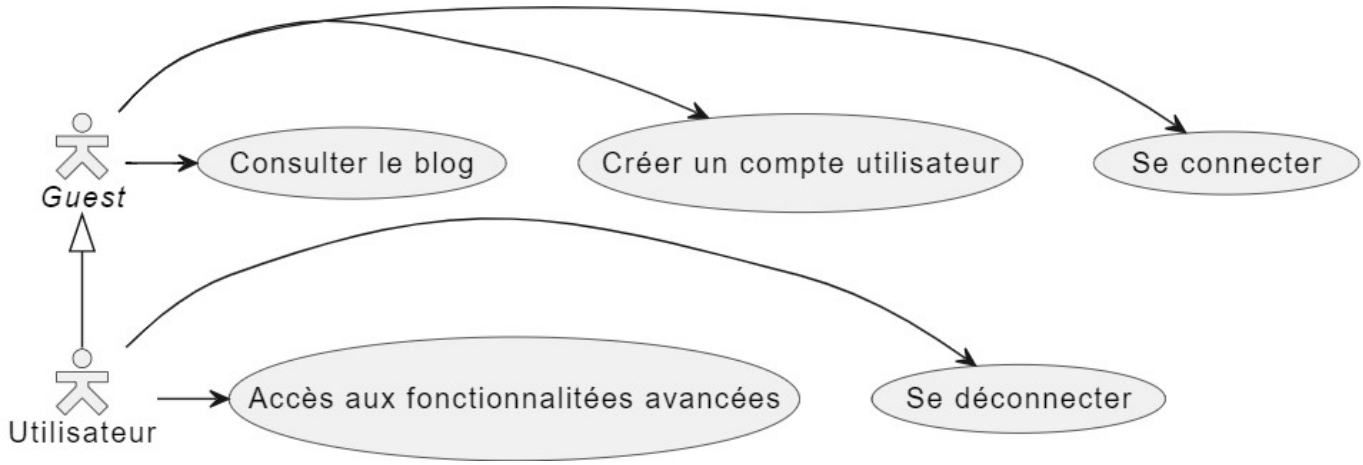
Comment gérer l'utilisateur non identifiée ?

Notre situation actuel (un blogue) est un exemple d'application qui permet, généralement, que des utilisateurs non identifiés s'y connectent pour consulter les différents articles disponibles. En effet, l'utilisateur n'aura que très peu de liberté (ajouter un commentaire, *like*, etc.), mais comment gérer l'accès aux articles ? C'est la même question pour accéder à la page de connexion ou la page de création de compte.

Il y aura donc un ensemble de route permise pour ce type d'utilisateurs, et il sera nécessaire de s'assurer que *Validateur de jeton* **ne devra pas** exécuter son processus de validation de jeton dans ces cas-là.

Schéma des cas d'utilisation

Pour bien visualiser les différentes fonctionnalités en liens avec l'identification et l'autorisation des utilisateurs, voici un schéma UML de cas d'utilisation qui permet de les visualiser rapidement.



L'accès aux fonctionnalitées avancées pourraient être représenter dans son propre schéma de cas d'utilisation, toutefois il sort du **scope** du présent travail. Remarquez que le liens entre les deux types d'utilisateurs est une flèche pleine signifiant que l'utilisateur à la même droit qu'un **guest** (dans notre cas, pas tout à fait, mais c'est pas grave).

> Il est facile dans notre cas de comprendre que nos utilisateurs se distingues par la présence ou non d'un jeton dans l'en-tête. Mais comment devrions-nous nous y prendre si nous avions des utilisateurs différents, comme des **writer**, des **reviewer** et des **admin** ?