ECE PARIS
ÉCOLE D'INGÉNIEURS

Json Web Tokens

2019/2020

APERCU

Dans ce TP vous serez amenés à mettre en place un pare-feu utilisant la technologie JWT (Json Web Tokens) pour sécuriser l'accès à l'API.

OBJECTIFS

1. Installer la librairie
2. Mettre en place un middleware de connexion
3. Créer un pare-feu

1. Installer la librairie

Pour ce TP, nous utiliserons la librairie **jsonwebtoken** qui permet de simplifier la création et la vérification des JWT ainsi que la librairie **fs** qui permet de lire des fichiers afin de charger la clé de chiffrement de l'application. Pour installer les librairies:

```
npm install jsonwebtoken fs
```

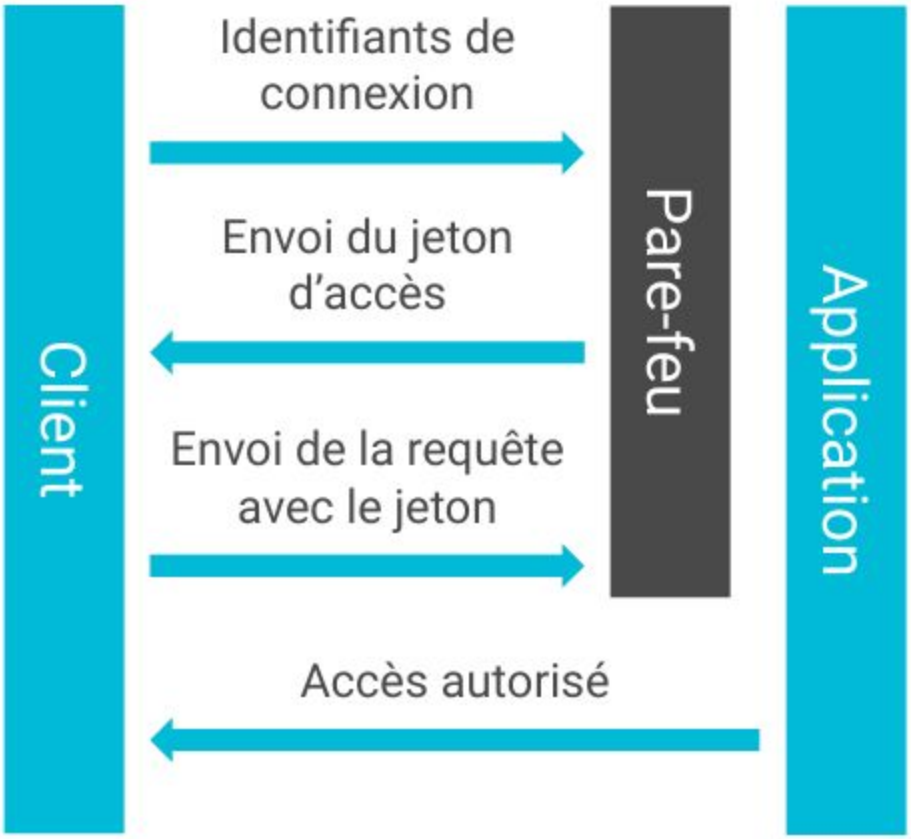
Ensuite pour importer les librairies:

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
```

2. Mise en place d'un middleware de connexion

Dans ce TP, nous allons mettre en place un pare-feu afin de sécuriser l'accès à l'API. Avant de procéder à la suite du TP, veuillez créez une table **users** dans laquelle seront les colonnes **username** avec le type **varchar** de 255 caractères et une colonne **password** avec le type **varchar** de 255 caractères.

Le procédé standard de création d'un JWT est le suivant:



Nous souhaitons permettre la connexion d'un utilisateur afin de générer un JWT (Json Web Token) qu'il pourra utiliser pour le reste de ses requêtes. Nous allons donc tout d'abord créer un middleware pour l'URL de connexion:

```
app.post('/login', function(req, res) {

});
```

Attention: Si l'on souhaite que ce middleware soit traité avant toute autre route, il faudra le placer avant les routes dans le code.

L'objectif est désormais de récupérer le nom d'utilisateur et le mot de passe afin de vérifier si l'utilisateur existe bien et que le mot de passe est correct:

```
let username = req.body.username;
let password = req.body.password;

let query = `SELECT * FROM users WHERE username='${username}' AND
password='${password}'`;
db.query(query, function(err, result, fields) {
    if (err) throw err;

    if (result.length > 0) {
        // Return JWT
    }
    else {
        res.send("Access denied");
    }
});
```

Maintenant il ne reste plus qu'à créer puis retourner le token JWT en fonction de la clé de chiffrement. Tout d'abord, créez un fichier texte **secret.key** qui contiendra votre clé secrète (exemple: "CeciEstMaCleSecrete101"). Avec la librairie **fs**, vous lirez ce fichier pour l'utiliser avec la librairie **jsonwebtoken**. Pour créer un jeton, rien de plus simple: la méthode **sign** permet de simplement le créer.

```
let secretKey = fs.readFileSync('secret.key');
let token = jwt.sign({ username: username}, secretKey);

res.send(token);
```

3. Création du pare-feu

Nous souhaitons donc vérifier pour chaque requête si une clef API est présente dans l'**en-tête** et si cette clef est correcte. Le pare-feu aura pour rôle de vérifier que l'option **X-Auth-Token** dans l'entête de la requête est bien présente et que le jeton est correct:

```
app.use(function(req, res, next) {
    if ("x-auth-token" in req.headers) {
        let token = req.headers["x-auth-token"];
        let secretKey = fs.readFileSync('secret.key');
        let decoded = jwt.verify(token, secretKey);
        let query = `SELECT * FROM users WHERE
username='${decoded.username}'`;

        db.query(query, function(err, result, fields) {
            if (err) throw err;

            if (result.length > 0) {
                next();
            }
            else {
                res.send("Access denied");
            }
        });
    } else {
        res.send("Access denied");
    }
});
```

Remarque: La fonction **next** permet de passer au prochain middleware, donc dans notre cas, cela permet d'appeler la route associée à la requête.