JSON WEB TOKEN (JWT)

FORMATEUR : MAHDI KELLOUCH

DÉFINITION

- JSON Web Token (JWT) est un standard ouvert défini dans la RFC 7519.
- Avec JWT, l'échange des informations se fait a travers des jetons (tokens).
- JWT définit un moyen de transmettre des informations - comme des éléments d'authentification et d'autorisation - entre deux parties : un client et un serveur.
- La communication avec JWT est sûre car chaque jeton émis est signé numériquement, de sorte que le serveur peut vérifier si le jeton est authentique ou a été falsifié.

LES UTILISATIONS DE JWT

Les JWT peuvent être utilisés de différentes manières :

- Authentification : Lorsqu'un utilisateur se connecte avec succès en utilisant ses informations d'identification, un jeton d'identification est renvoyé.
- Autorisation : Une fois qu'un utilisateur a réussi à se connecter, une application peut demander à accéder à des routes, des services ou des ressources (par exemple, des API) au nom de cet utilisateur. Pour ce faire, elle doit transmettre à chaque demande un jeton d'accès, qui peut se présenter sous la forme d'un JWT.
- Echange d'information : Les JWT sont un bon moyen de transmettre des informations en toute sécurité entre des parties, car ils peuvent être signés, ce qui signifie que vous pouvez être sûr que les expéditeurs sont bien ceux qu'ils prétendent être. En outre, la structure d'un JWT vous permet de vérifier que le contenu n'a pas été falsifié.

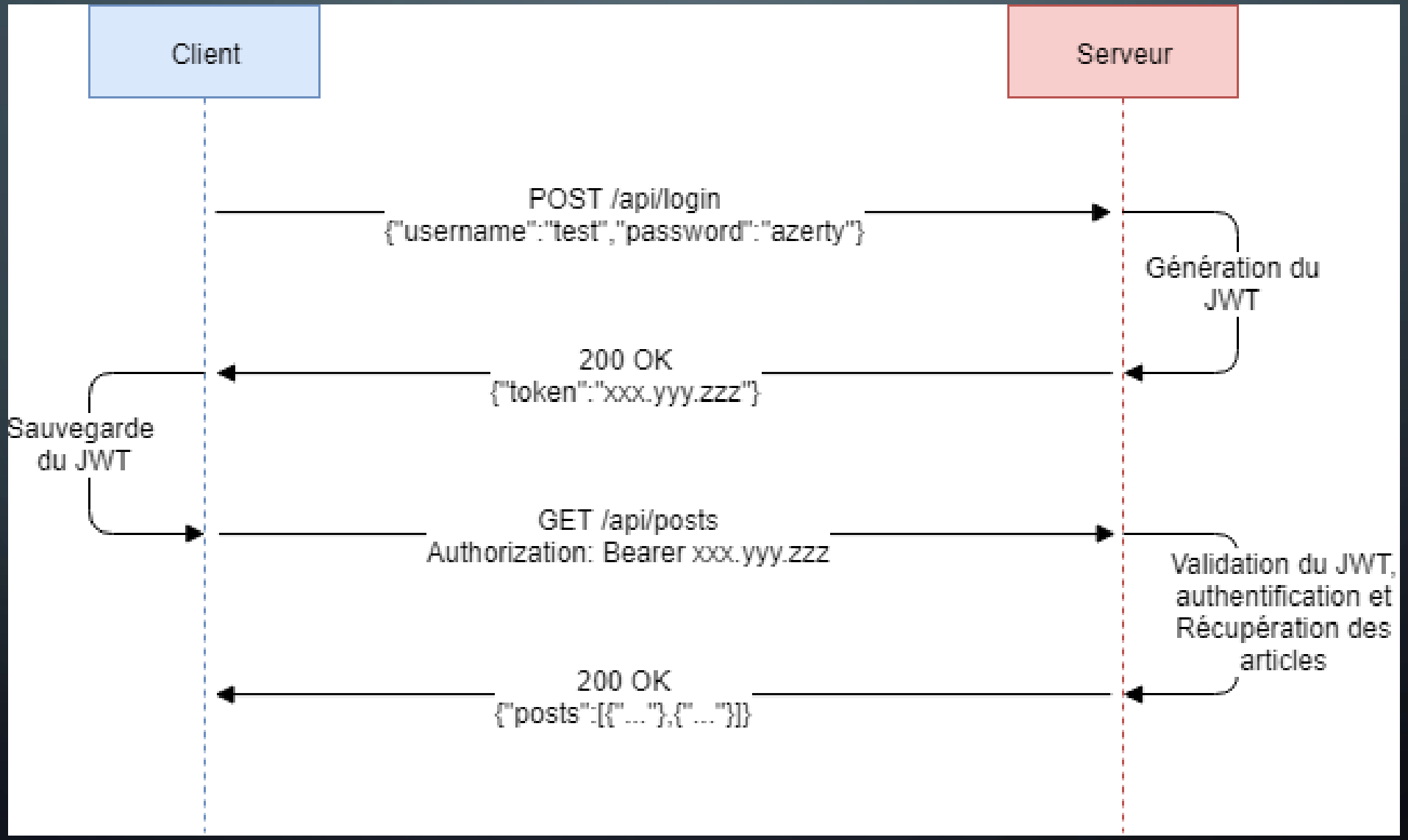
FONCTIONNEMENT DE JWT

JWT

FONCTIONNEMENT DE JWT

1. Les « JSON Web Token » ou JWT sont des jetons générés par un serveur lors de l'authentification d'un utilisateur sur une application Web, et qui sont ensuite transmis au client.
2. Ils seront renvoyés avec chaque requête HTTP au serveur, ce qui lui permettra d'identifier l'utilisateur.
   - Pour ce faire, les informations contenues dans le jeton sont signées à l'aide d'une clé privée détenue par le serveur. Quand il recevra à nouveau le jeton, le serveur n'aura qu'à comparer la signature envoyée par le client et celle qu'il aura générée avec sa propre clé privée et à comparer les résultats. Si les signatures sont identiques, le jeton est valide.

FONCTIONNEMENT DU JWT



STRUCTURE D'UN JETON

JWT

STRUCTURE D'UN JWT

- Un JWT signé se compose de trois parties codées en base64 et séparées par un point :

- **Exemple :**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

HEADER : ENTÊTE

- L'en-tête se compose généralement d'un objet JSON de deux membres :
  - **Typ :** le type du jeton, qui est donc JWT
  - **Alg :** l'algorithme de hachage utilisé, comme HMAC SHA256 ou RSA.
- Cette partie est encodé en base64

Par exemple :

```
{
      "typ" : "JWT",
      "alg" : "RS256"
}
```

PAYLOAD : CHARGE UTILE

- La charge utile contient les déclarations sous le format JSON.
- Les déclarations sont des informations sur une entité (généralement, l'utilisateur) et des métadonnées supplémentaires.
- Cette partie est encodé en base64

Par exemple :

```
{
   “iat“: 1480929282,
   “exp“: 1480932868,
   “name“:“Username“
}
```

SIGNATURE

- La signature est utilisé pour valider que le jeton est digne de confiance et n'a pas été modifie tout au long du chemin.
- La partie signature est créé par la prise en compte de l'en-tête codé, la charge utile codée, un secret, l'algorithme spécifié dans l'en-tête, et signer le tout.

```
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
Secret
)
```

INSTALLATION DE JWT

JWT

INSTALLATION

- Pour créer et vérifier les jetons JWT, nous allons utiliser la bibliothèque **jsonwebtoken**
- La commande pour installer le package jsonwebtoken :

npm install jsonwebtoken

- Pour utiliser cette bibliothèque dans nos fichier, il suffit d'ajouter l'importation suivante :

import jwt from '**jsonwebtoken**'

GÉNÉRATION DES JETONS

JWT

GÉNÉRATION DE JETON

- La génération d'un jeton se fait a l'aide de la fonction **jwt.sign**.
- Cette fonction accepte trois paramètres :
  - Le secret du jeton
  - L'élément de données à hacher dans le jeton
  - L'heure d'expiration du jeton (Optionnel)

GÉNÉRATION DE JETON – EXEMPLE 1

```
app.get("/api/login", (req, res) => {
  const user = {
    id: 1,
    username: "brad",
    email:'brad@gmail.com'
  };

  jwt.sign({ user: user }, "secret_key", (err, token) => {
    res.json({
      token,
    });
  });
});
```

GÉNÉRATION DE JETON - EXEMPLE 2

```
const JWT_SECRET = "MY_SECRET";
const vusername = "root";
const vpassword = "admin";

app.post("/api/login", (req, res) => {
  const { username, password } = req.body;

  if (username === vusername && password === vpassword) {

    jwt.sign({ user: "root" }, JWT_SECRET, (err, token) => {
      if (!err) res.json({ token });
      else res.status(401).json({ message: err });
    });

  } else {

    return res
      .status(401)
      .json({ message: "username et password non fournies" });
  }
});
```

GÉNÉRATION DE JETON - EXEMPLE 3

```
const JWT_SECRET = "MY_SECRET";
const vusername = "root";
const vpassword = "admin";

app.get("/api/login", (req, res) => {
  const { username, password } = req.body;

  if (username === vusername && password === vpassword) {
    jwt.sign({ user: "root" }, JWT_SECRET, {expiresIn: '30s'},
     (err, token) => {
      if (!err) res.json({ token });
      else res.status(401).json({ message: err });
    });
  } else {
    return res
      .status(401)
      .json({ message: "username et password non fournies" });
  }
});
```

PACKAGE CRYPTO

- Pour générer un secret non prévisible, on utilise la bibliothèque crypto comme le montre l'exemple suivant :

```
require('crypto').randomBytes(48).toString('hex')
```

- La valeur générée par cette ligne de code peut être stocker dans le fichier .env

AUTHENTIFICATION D'UN JETON

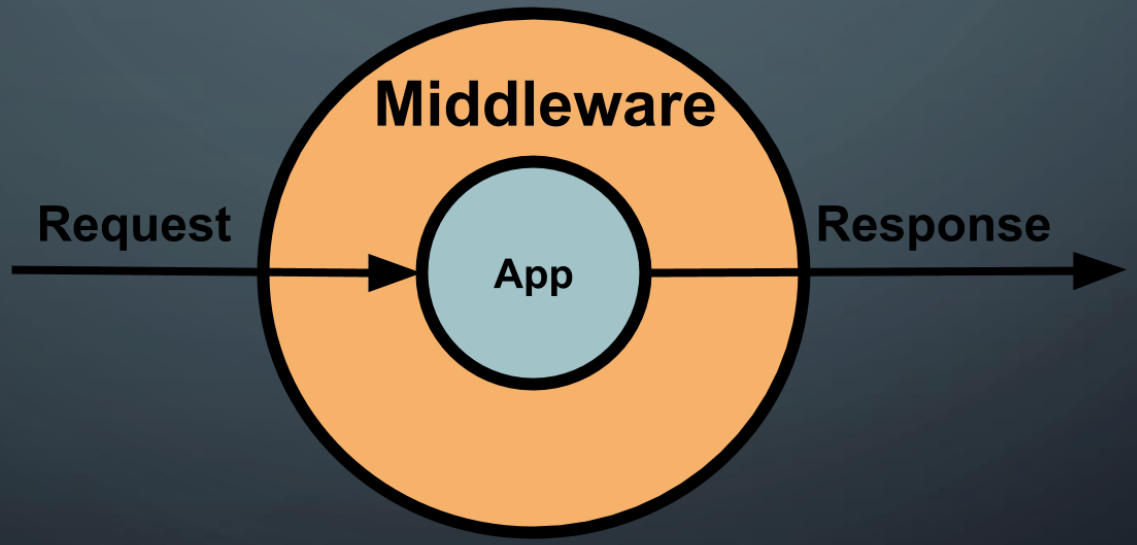
JWT

LA NOTION DE MIDDLEWARE

- Un middleware désigne littéralement tout ce que vous placez au milieu d'une couche du logiciel et d'une autre.
- Les middlewares Express sont des fonctions qui s'exécutent pendant le cycle de vie d'une demande adressée au serveur Express.
- Chaque middleware a accès à la demande et à la réponse HTTP pour chaque route (ou chemin) à laquelle il est rattaché.
- Les middelwares sont capables de modifier les demandes (l'objet req), les objets de réponse, et peuvent également mettre fin au cycle de réponse.

Pour simplifier, vous pouvez considérer les middlewares comme un groupe de fonctions qui s'exécutent dès qu'une demande est adressée au serveur.

LA NOTION DE MIDDLEWARE



AUTHENTIFICATION D'UN JETON

- Pour authentifier un jeton, nous allons utilise la notion de middleware dans Express.js.
- Le principe est le suivant : lorsqu'une demande est adressée à une route spécifique, les variables (req, res) peuvent être envoyées à une fonction intermédiaire avant celle spécifiée dans le route app.get((req, res) => {}).
- Le middleware est une fonction qui prend comme paramètres (req, res, next).
  - Le req est la requête envoyée (GET, POST, DELETE, PUT, etc.).
  - La res est la réponse qui peut être renvoyée à l'utilisateur d'une multitude de façons (res.sendStatus(200), res.json(), etc.).
  - Le next est une fonction qui peut être appelée pour déplacer l'exécution au-delà du middleware et dans la réponse réelle du serveur app.get.

EXEMPLE DE FONCTION D'AUTHENTIFICATION MIDDLEWARE

```
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1]

    if (token == null) return res.sendStatus(401);

    jwt.verify(token, JWT_SECRET, (err, userData) => {
        if(err)
            return res.sendStatus(403);
        else{
            req.user = userData;
            next();
        }
    })
}
```

UTILISATION DE LA FONCTION

- Et un exemple de requête qui utiliserait cet middleware ressemblerait à quelque chose comme ceci :

```
app.get('/api/data', authenticateToken, (req,res) => {
    res.json({data:'Mes donnees...'})
})
```