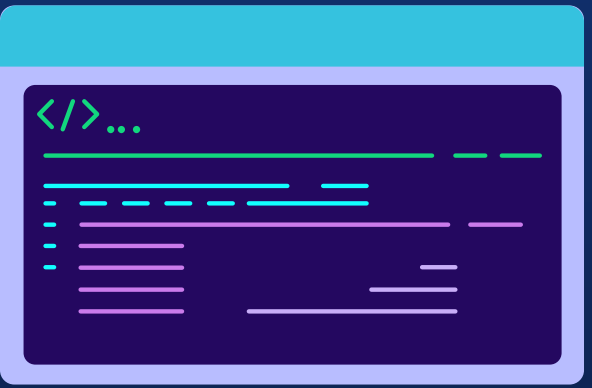
JWT



Utiliser les JWT avec NodeJS

JWT ?

JWT est un standard qui définit une façon de transmettre des infos de façon sécurisée entre plusieurs hôtes.

C’est un système utilisé pour l’authentification et l’autorisation en règle générale.

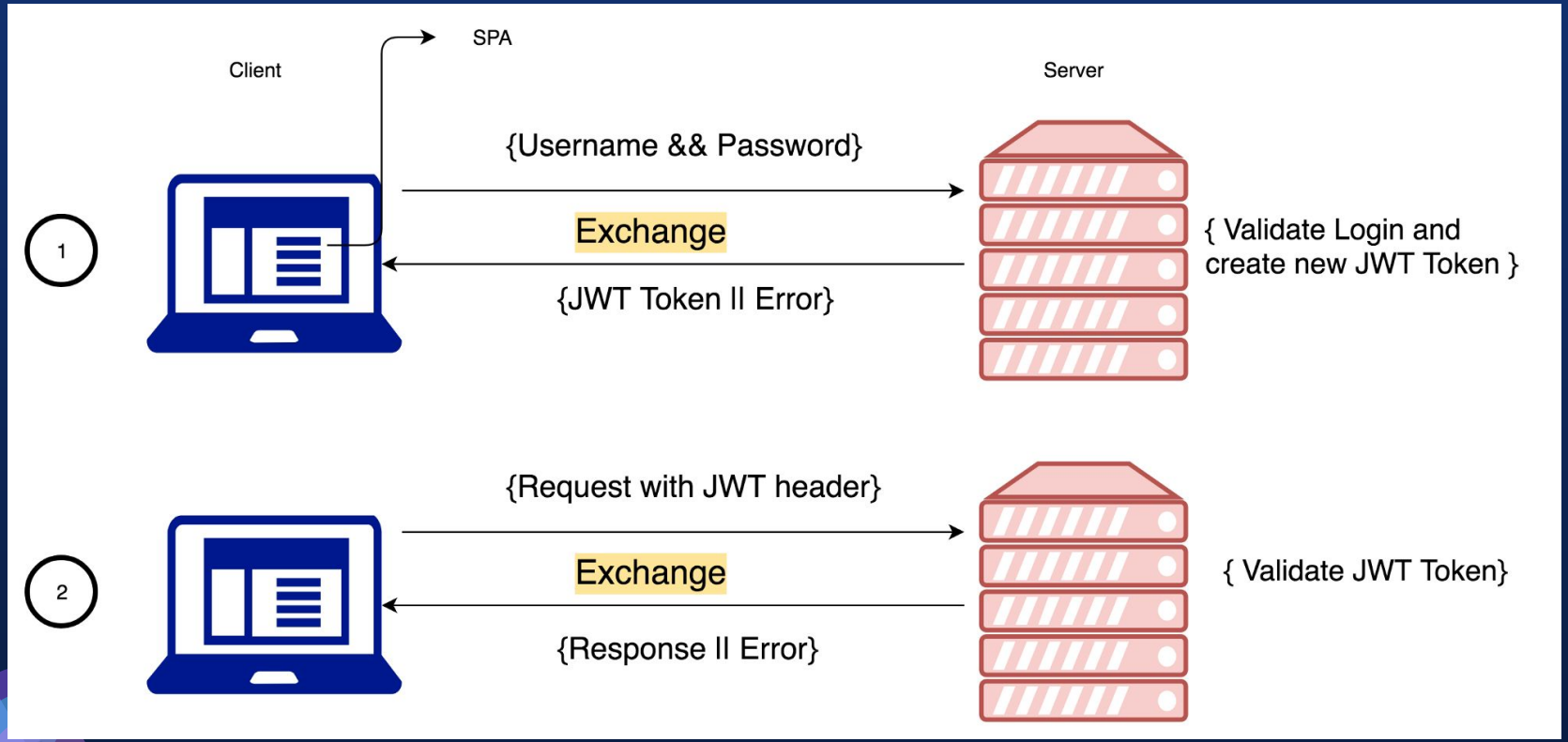
Un token JWT contient :

- Un header
- Un payload
- une signature

séparés par des points.

xxxxx.yyyyy.zzzzz

JWT ?


SPA
Client
Server
{Username && Password}
Exchange
{JWT Token II Error}
1
{ Validate Login and
create new JWT Token }
{Request with JWT header}
Exchange
{Response II Error}
2
{ Validate JWT Token}

Contenu

Le header contient le type du token et le nom de l'algorithme qui a permis de créer la signature.

Ces infos sont fournies sous forme de JSON transformée en base64(URL).

Le payload contient des informations sur l'utilisateur fournies par l'utilisateur.

Le standard impose des informations :

iss: l'emetteur de la requête

exp: le temps d'expiration

sub: le sujet

aud: le destinataire

Contenu

Le header contient le type du token et le nom de l'algorithme qui a permis de créer la signature.

Ces infos sont fournies sous forme de JSON transformée en base64(URL).

Le payload contient des informations sur l'utilisateur fournies par l'utilisateur.

Le standard impose des informations :

iss: l'emetteur de la requête

exp: le temps d'expiration

sub: le sujet

aud: le destinataire

Signature

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

La signature est un hash des données et permet de vérifier que celle-ci n’a pas été trafiquée.

Utilisation

le jwt est envoyé par l'utilisateur dans le header Authorization :

Authorization: Bearer <token>

Le client reçoit son JWT lors de son login.

Demo

Créer le token

```
export const generateToken = (user: User) => {

 // create a jwt token that is valid for 7 days
 const token = jwt.sign({ sub: user },
process.env.JWT_SECRET ?? "secret", {
   expiresIn: "7d",
 });

 return {
   ...user,
   token,
 };
}
```

Ici on génère le token et on met `user` dans son payload. Si le user a un rôle, alors on pourra consulter celui-ci via le web token.

Il faut installer express-jwt avec npm.

Renvoyer le token à l'utilisateur

```
router.post("/login", async (req, res) => {
  const { error } = loginSchema.validate(req.body);
  if (error) {
        return res.status(400).json({ error: error.message });
  }
  const loginDTO = req.body as LoginDTO;

  const user = await User.findOne({ email: loginDTO.email });
  if (!user || !comparePassword(loginDTO.password, user?.password)) {
        return res.status(400).json({ error: "Invalid credentials" });
  }

  const userWithToken = generateToken(user.toObject());
        res.status(200).json(removePassword(userWithToken))
});
```

L'utilisateur aura besoin de son token pour faire des requêtes.

Tester la validité du token

```
export const jwt = () => {
  const secret = process.env.JWT_SECRET;
  if (!secret) {
        throw new Error("JWT_SECRET is not defined");
  }
  return expressjwt({ secret, algorithms: ["HS256"] }).unless({
        path: [
        // public routes that don't require authentication
        "/auth/login",
        "/auth/register",
        ],
  })
}
```

expressjwt se charge de vérifier que le token n’est pas corrompu.

Accéder au payload du token

```
router.get("/", async ( req, res) => {
 const users = await User.find();
 console.log((req as any).auth)
 return res
        .status(200)
        .json(removePassword(users.map((user) => removePassword(user.toObject()))));
});
```

le payload du token est stocké dans req.auth