



27/02/2025

Introduction et Maîtrise de JSON Web Token (JWT)

Sécurisation des échanges et
authentification des utilisateurs
avec JWT



MAHDI KELLOUCH
ISMO TETOUAN

Table des matières

Introduction.....	2
1. Définition et Utilisation de JWT	2
1.1 Qu'est-ce qu'un JWT ?.....	2
1.2 Utilisation courante des JWT.....	2
2. Fonctionnement du JWT	2
2.1 Génération et transmission d'un JWT.....	2
2.2 Vérification du JWT.....	2
3. Structure d'un JWT	2
4. Installation et Utilisation de JWT en Node.js.....	3
4.1 Installation de la bibliothèque jsonwebtoken	3
4.2 Génération d'un JWT en Node.js.....	3
4.3 Vérification d'un JWT	3
5. Utilisation des JWT avec Express.js.....	3
5.1 Création d'un middleware d'authentification.....	3
5.2 Utilisation du middleware dans Express.....	3
6. Exercices et Applications.....	3
Exercice 1 : Génération et vérification d'un JWT.....	3
Exercice 2 : Création d'un middleware Express	5
Exercice 3 : Expiration et renouvellement du JWT	7
7. Quiz.....	9
Conclusion	10

JSON Web Token (JWT)

Introduction

Le JSON Web Token (**JWT**) est un standard ouvert (*RFC 7519*) permettant l'**échange sécurisé** d'informations entre deux parties (**client** et **serveur**) sous forme de jetons. Ces jetons sont signés numériquement, garantissant ainsi leur authenticité et leur intégrité.

1. Définition et Utilisation de JWT

1.1 Qu'est-ce qu'un JWT ?

Un JSON Web Token (**JWT**) est une chaîne encodée en **base64** contenant des informations structurées sous format JSON. Il est utilisé principalement pour :

- **L'authentification** : identification d'un utilisateur après connexion.
- **L'autorisation** : contrôle d'accès aux ressources protégées.
- **L'échange d'informations** : transmission sécurisée de données signées.

1.2 Utilisation courante des JWT

- **Authentification** : Un **JWT** est généré et envoyé au client après une connexion réussie.
- **Autorisation** : Le client utilise ce **JWT** pour accéder aux ressources sécurisées.
- **Échange sécurisé** : Les informations contenues dans un **JWT** sont signées, garantissant leur intégrité.

2. Fonctionnement du JWT

2.1 Génération et transmission d'un JWT

1. L'utilisateur s'authentifie auprès du serveur.
2. Le serveur génère un JWT et l'envoie au client.
3. Le client stocke le JWT (souvent dans localStorage ou cookies).
4. À chaque requête, le client envoie le JWT dans l'en-tête HTTP.
5. Le serveur vérifie la signature et extrait les informations du JWT.

2.2 Vérification du JWT

1. Le serveur récupère le JWT envoyé par le client.
2. Il décode le JWT et vérifie la signature avec la clé secrète.
3. Si la signature est valide, les informations contenues dans le JWT sont considérées comme authentiques.

3. Structure d'un JWT

Un JWT est composé de trois parties séparées par des points (.) :

1. **Header (En-tête)** : Contient l'algorithme de signature et le type du token.
2. **Payload (Charge utile)** : Contient les informations sur l'utilisateur et les métadonnées.
3. **Signature** : Assure l'intégrité du token en utilisant la clé secrète.

Exemple de JWT encodé en base64 :

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJEwLCJyb2xlIjoiYWRTaW4iLCJpYXQiOiJlMmZlc3ODI0MDB9.xxxxxx
```

4. Installation et Utilisation de JWT en Node.js

4.1 Installation de la bibliothèque jsonwebtoken

```
npm install jsonwebtoken
```

4.2 Génération d'un JWT en Node.js

```
const jwt = require('jsonwebtoken');
const secretKey = 'secret';
const token = jwt.sign({ userId: 1, role: 'admin' }, secretKey, { expiresIn: '1h' });
console.log(token);
```

4.3 Vérification d'un JWT

```
const decoded = jwt.verify(token, secretKey);
console.log(decoded);
```

5. Utilisation des JWT avec Express.js

5.1 Création d'un middleware d'authentification

```
const authMiddleware = (req, res, next) => {
  const token = req.headers['authorization'];
  if (!token) return res.sendStatus(403);
  jwt.verify(token, secretKey, (err, decoded) => {
    if (err) return res.sendStatus(403);
    req.user = decoded;
    next();
  });
};
```

5.2 Utilisation du middleware dans Express

```
app.get('/protected', authMiddleware, (req, res) => {
  res.json({ message: 'Accès autorisé', user: req.user });
});
```

6. Exercices et Applications

Exercice 1 : Génération et vérification d'un JWT

- Écrire un script Node.js qui génère un JWT avec un rôle utilisateur.

- Vérifier ce JWT et afficher son contenu.

Solution attendue :

```
// jwt-exemple.js

const jwt = require('jsonwebtoken');

// Clé secrète pour signer le token
const SECRET_KEY = 'ma_cle_super_secrete_2024';

// Fonction pour générer un JWT
function generateToken(userData) {
  const payload = {
    userId: userData.id,
    role: userData.role,
    email: userData.email,
    iat: Math.floor(Date.now() / 1000),
    exp: Math.floor(Date.now() / 1000) + (60 * 60) // Expire dans 1 heure
  };

  return jwt.sign(payload, SECRET_KEY);
}

// Fonction pour vérifier et décoder un JWT
function verifyToken(token) {
  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    return {
      valid: true,
      data: decoded
    };
  } catch (error) {
    return {
      valid: false,
      error: error.message
    };
  }
}

// Données utilisateur de test
const user = {
  id: 123,
  role: 'admin',
  email: 'admin@example.com'
};

// Test de génération
console.log('1. Génération du token :');
const token = generateToken(user);
console.log(token);
console.log('\n');
```

```
// Test de vérification
console.log('2. Vérification et décodage du token :');
const verification = verifyToken(token);
console.log(JSON.stringify(verification, null, 2));
```

Exercice 2 : Création d'un middleware Express

- Implémenter un middleware Express.js qui protège une route spécifique en validant un JWT.
- Tester avec Postman ou PowerShell.

Solution attendue :

```
// auth-middleware.js
const express = require('express');
const jwt = require('jsonwebtoken');

const app = express();
app.use(express.json());

const SECRET_KEY = 'cle_secrete_2024';

// Middleware d'authentification
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Token manquant' });
  }

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) {
      return res.status(403).json({ message: 'Token invalide' });
    }
    req.user = user;
    next();
  });
};

// Route de login pour obtenir un token
app.post('/login', (req, res) => {
  const { username, role } = req.body;

  const user = { username, role };
  const token = jwt.sign(user, SECRET_KEY, { expiresIn: '1h' });

  res.json({ token });
});

// Route protégée
```

```

app.get('/protected', authenticateToken, (req, res) => {
  res.json({
    message: 'Accès autorisé',
    user: req.user
  });
});

// Route publique
app.get('/public', (req, res) => {
  res.json({ message: 'Route publique accessible à tous' });
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Serveur démarré sur le port ${PORT}`);
});

// test-requests.ps1

# 1. Obtenir un token (Login)
$loginBody = @{
  username = "john"
  role = "admin"
} | ConvertTo-Json

$response = Invoke-RestMethod -Uri "http://localhost:3000/login" -Method Post -Body
$loginBody -ContentType "application/json"
$token = $response.token

# 2. Accéder à la route protégée avec le token
$headers = @{
  Authorization = "Bearer $token"
}

$protectedResponse = Invoke-RestMethod -Uri "http://localhost:3000/protected" -Method Get
-Headers $headers
$protectedResponse

// Pour tester avec Postman :

```

1. Login (POST <http://localhost:3000/login>)

```

{
  "username": "john",
  "role": "admin"
}

```

2. Route protégée (GET <http://localhost:3000/protected>) Headers:

Authorization: Bearer <votre_token>

Exercice 3 : Expiration et renouvellement du JWT

- Implémenter un mécanisme permettant de renouveler un JWT expiré.

Solution attendue :

```
// token-refresh.js
const express = require('express');
const jwt = require('jsonwebtoken');

const app = express();
app.use(express.json());

const ACCESS_TOKEN_SECRET = 'access_secret_2024';
const REFRESH_TOKEN_SECRET = 'refresh_secret_2024';

// Stockage des refresh tokens (en pratique, utilisez une base de données)
let refreshTokens = [];

// Génération des tokens
const generateTokens = (user) => {
  const accessToken = jwt.sign(user, ACCESS_TOKEN_SECRET, { expiresIn: '15m' });
  const refreshToken = jwt.sign(user, REFRESH_TOKEN_SECRET, { expiresIn: '7d' });

  refreshTokens.push(refreshToken);

  return {
    accessToken,
    refreshToken
  };
};

// Login route
app.post('/login', (req, res) => {
  const { username } = req.body;
  const user = { username };

  const tokens = generateTokens(user);
  res.json(tokens);
});

// Refresh token route
app.post('/refresh-token', (req, res) => {
  const { refreshToken } = req.body;

  if (!refreshToken) {
    return res.status(401).json({ message: 'Refresh Token requis' });
  }

  if (!refreshTokens.includes(refreshToken)) {
    return res.status(403).json({ message: 'Refresh Token invalide' });
  }
}
```



```

    jwt.verify(refreshToken, REFRESH_TOKEN_SECRET, (err, user) => {
      if (err) {
        return res.status(403).json({ message: 'Refresh Token invalide' });
      }

      // Générer un nouveau access token
      const accessToken = jwt.sign({ username: user.username }, ACCESS_TOKEN_SECRET, {
expiresIn: '15m' });
      res.json({ accessToken });
    });
  });

// Logout route
app.post('/logout', (req, res) => {
  const { refreshToken } = req.body;
  refreshTokens = refreshTokens.filter(token => token !== refreshToken);
  res.json({ message: 'Logout réussi' });
});

// Middleware de vérification du token d'accès
const verifyToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Access Token requis' });
  }

  jwt.verify(token, ACCESS_TOKEN_SECRET, (err, user) => {
    if (err) {
      if (err.name === 'TokenExpiredError') {
        return res.status(401).json({ message: 'Token expiré, veuillez rafraîchir'
});
      }
      return res.status(403).json({ message: 'Token invalide' });
    }
    req.user = user;
    next();
  });
};

// Route protégée
app.get('/protected', verifyToken, (req, res) => {
  res.json({ message: 'Accès autorisé', user: req.user });
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Serveur démarré sur le port ${PORT}`);
});

```

// Pour tester le système de refresh token :

1. Login pour obtenir les tokens :

```
$loginBody = @{  
    username = "john"  
} | ConvertTo-Json  
  
$response = Invoke-RestMethod -Uri "http://localhost:3000/login" -Method Post -Body  
$loginBody -ContentType "application/json"  
$accessToken = $response.accessToken  
$refreshToken = $response.refreshToken
```

2. Accès à la route protégée :

```
$headers = @{  
    Authorization = "Bearer $accessToken"  
}  
  
$protectedResponse = Invoke-RestMethod -Uri "http://localhost:3000/protected" -Method Get  
-Headers $headers
```

3. Rafraîchir le token :

```
$refreshBody = @{  
    refreshToken = $refreshToken  
} | ConvertTo-Json  
  
$refreshResponse = Invoke-RestMethod -Uri "http://localhost:3000/refresh-token" -Method  
Post -Body $refreshBody -ContentType "application/json"  
$newAccessToken = $refreshResponse.accessToken
```

4. Déconnexion :

```
$logoutBody = @{  
    refreshToken = $refreshToken  
} | ConvertTo-Json  
  
$logoutResponse = Invoke-RestMethod -Uri "http://localhost:3000/logout" -Method Post -Body  
$logoutBody -ContentType "application/json"
```

7. Quiz

1. Quel est le rôle principal d'un JWT ?

a) Stocker des mots de passe

b) Transmettre des informations de manière sécurisée

- c) Générer des clés API
2. Quelle est la structure d'un JWT ?
- a) Un seul bloc de texte
 - b) Trois parties séparées par des virgules
 - c) Trois parties séparées par des points
3. Quelle commande permet d'installer jsonwebtoken en Node.js ?
- a) `npm install jwt`
 - b) `npm install jsonwebtoken`
 - c) `npm get jsonwebtoken`
4. Dans Express.js, où doit-on inclure le middleware d'authentification JWT ?
- a) Avant les routes sécurisées
 - b) Après la réponse du serveur
 - c) Dans le client

Conclusion

Le **JSON** Web Token est une méthode efficace et sécurisée pour gérer l'authentification et l'autorisation dans les applications modernes. Son adoption est large, notamment dans les architectures **microservices** et les applications Single Page Applications (**SPA**). En maîtrisant son fonctionnement, vous êtes en mesure d'améliorer la sécurité et l'efficacité de vos applications web.