

# Chapitre 5

# Middlewares

# Express-Validator

# Introduction à Express Validator

- Bibliothèque de validation pour Express.js.
- Utilisée pour valider les entrées des requêtes.
- /validator/productValidator.js
- /validator/validate.js
- Importation : const { body, param } = require('express-validator');
- Exemple de Validation de Paramètre d'URL

```
● ● ●          Uploaded using RayThis Extension  
  
const validateIdParm = [  
    params('id').notEmpty().isNumeric()  
];
```

# Utilisation dans une route express

```
● ● ● Uploaded using RayThis Extension

const { validateIdParam, validateBodyParam } = require("../validator/productValidator")
const validate = require("../validator/validate") // Middleware de validation

router.post("", validateBodyParam, validate, store)

// dans productValidator.js
const validateBodyParam = [
    param('id').not().isString().notEmpty().isInt({min:0}),
    param('name').not().isString().notEmpty(),
    param('price').not().isString().notEmpty().isFloat({min:0})
];
// fichier validate
function validate(req, res, next) {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(422).json({ errors: errors.array() });
    }
    next();
}
```

# **Exercices**

# Chapitre 5

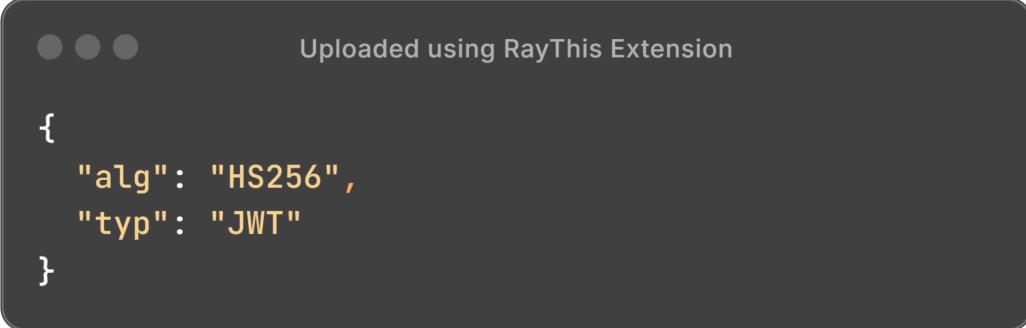
# Authentification JWT

# Qu'est ce que JWT ?

- JSON Web Token est un standard pour l'échange sécurisé d'informations
- Structure :
  - Header
  - Payload (données)
  - Signature

# Header

- Contient le type de token et l'algorithme de signature utilisé



Uploaded using RayThis Extension

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

# Payload

- Contient différentes informations possibles : sur l'utilisateur, permissions, émetteur du token, id, date d'émission, date d'expiration ...

...  
Uploaded using RayThis Extension

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022,  
  "exp": 1516242622  
}
```

# Signature

- Garantit l'intégrité du token via un chiffrage et une clé secrète
- La clé doit être confidentielle, et seulement côté serveur
- Elle permet de vérifier que le token vient bien du serveur

...  
Uploaded using RayThis Extension

```
// Exemple de code pour signer un JWT (JSON Web Token) avec HMAC SHA-256
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

# Avantages du JWT

- Stateless : pas de session serveur
- Rapide et sécurisé (via la signature)
- Utilisé sur web et mobile



# Flux d'authentification avec JWT

1. L'utilisateur s'authentifie avec ses identifiants
2. Le serveur vérifie puis génère un JWT
3. Le serveur renvoie le JWT au client
4. Le client inclut le JWT dans les headers lors de ses appels (Authorization Bearer)
5. Le serveur vérifie le token pour chaque requête protégée

# Mise en place avec Express.js

- Installation des packages (jsonwebtoken et bcryptjs pour le hash de password)
- Import des modules (require...)
- Création d'une route de la route login avec génération du token

```
● ● ●          Uploaded using RayThis Extension
// Exemple de "base de données" simulée
const users = [{ id: 1, username: 'testuser', password: bcrypt.hashSync('password123', 10) }];

// Route de connexion
app.post('/login', (req, res) => {
    const { username, password } = req.body;

    // Vérification si l'utilisateur existe
    const user = users.find(u => u.username === username);
    if (!user) return res.status(404).send('Utilisateur non trouvé');

    // Vérification du mot de passe
    const validPassword = bcrypt.compareSync(password, user.password);
    if (!validPassword) return res.status(401).send('Mot de passe incorrect');

    // Génération du JWT
    const token = jwt.sign({ id: user.id, username: user.username }, 'secret_key', {
        expiresIn: '1h'
    });

    // Envoi du token
    res.json({ token });
});
```

# Mise en place de la route register

```
● ● ●          Uploaded using RayThis Extension

const users = []; // Exemple de base simulée

// Route d'inscription
app.post('/register', (req, res) => {
    const { username, password } = req.body;

    // Vérifier si l'utilisateur existe déjà
    const existingUser = users.find(u => u.username === username);
    if (existingUser) return res.status(400).send('Utilisateur déjà existant');

    // Hacher le mot de passe
    const hashedPassword = bcrypt.hashSync(password, 10);
    // Créer un nouvel utilisateur
    const newUser = {
        id: users.length + 1, // auto-incrémenté
        username,
        password: hashedPassword
    };
    // Ajouter l'utilisateur dans la "base"
    users.push(newUser);

    res.status(201).send('Utilisateur créé avec succès');
});
```

# Mise en place d'un middleware de vérification du token

```
● ● ● Uploaded using RayThis Extension

function verifyToken(req, res, next) {
    // Récupérer le token à partir de l'en-tête
    const token = req.headers['authorization']?.split(' ')[1];
    // Vérifier que le token existe et est valide
    if (!token) return res.status(403).send('Token required');
    // Utiliser jwt pour vérifier le token
    // secret_key est la clé secrète utilisée pour signer le token
    jwt.verify(token, 'secret_key', (err, decoded) => {
        if (err) return res.status(401).send('Invalid token');
        // Si le token est valide, ajouter l'ID de l'utilisateur à la requête
        // decoded.id est l'ID de l'utilisateur contenu dans le token
        req.userId = decoded.id;
        next();
    });
}
```

# Protection d'une route

- Mise en place d'un middleware de vérification du token
- Si le client veut accéder à /protected il doit envoyer le token dans le header

```
...  
Uploaded using RayThis Extension  
  
// Route protégée accessible uniquement avec un JWT valide  
app.get('/protected', verifyToken, (req, res) => {  
    res.send(`Bienvenue, utilisateur ID: ${req.user.id}, username: ${req.user.username}`);  
});
```

# Exemple typique d'arborescence projet

```
mon-projet/
├── index.js          # Point d'entrée de l'application Express
├── package.json
└── config/
    └── db.js           # Fichier de configuration de la base de données (ex: MongoDB)
├── controllers/
    ├── authController.js # Contrôleur pour l'inscription, la connexion, etc.
    └── userController.js # Contrôleur pour les opérations sur les utilisateurs
├── middlewares/
    ├── authMiddleware.js # Middleware de vérification JWT
    └── validator.js      # Validateurs pour les requêtes (ex: express-validator)
├── routes/
    ├── authRoutes.js    # Routes liées à l'authentification
    └── userRoutes.js    # Routes utilisateur protégées
└── utils/
    └── generateToken.js # Fonction de génération de token JWT
```

# **Exercices**



# TP 3 : Validation des acquis

# **Chapitre 5**

## **JWT et Cookies**

### **PARTIE BONUS**

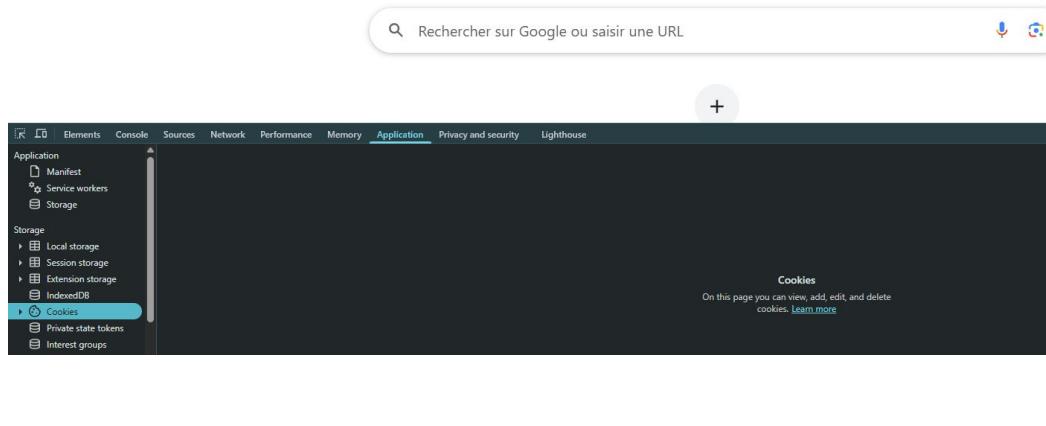
# C'est quoi un cookie ?

- Petit fichier enregistré sur le navigateur du client par le serveur ou le site web.
- Il permet de conserver des informations d'une page à l'autre
- Permet notamment :
  - Se souvenir d'un utilisateur
  - Enregistrer des préférences
  - Garder un panier d'achat
  - Stocker un JWT



# Où sont stockés les cookies ?

- Dans le navigateur, accessibles via l'onglet Applications > Cookies
- Envoyés automatiquement dans chaque requête HTTP
- Exemple d'en tête HTTP :
  - Cookie : token=xxxxx.yyyy.zzzzz



# JWT dans les cookies

- Pourquoi ne pas garder le JWT dans le localstorage ?
  - En cas d'attaque XSS (injection javascript)
  - Le token peut être volé
- On préfère un cookie, **non accessible en JavaScript** (cookie httpOnly)

# Nouveau flux d'authentification

1. Client envoie identifiants
  2. Serveur vérifie puis génère JWT
  3. Serveur stocke le JWT dans cookie httpOnly
  4. Le navigateur renvoie automatiquement le cookie aux appels suivants
  5. Serveur vérifie JWT dans le cookie puis accès ressource protégée
- 
- Le token n'apparaît plus dans les headers et **le client n'a rien à gérer**

# Mise en place dans Express : Route login

```
Uploaded using RayThis Extension

const jwt = require("jsonwebtoken");

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

  const token = jwt.sign({ username }, "secret_key", {
    expiresIn: "1d"
  });

  res.cookie("token", token, {
    httpOnly: true,
    secure: true,
    sameSite: "strict"
  });

  res.send("Authentifié avec JWT dans un cookie sécurisé");
});
```

# Middleware de vérification du JWT dans le cookie

```
function verifyJwtCookie(req, res, next) {
    const token = req.cookies?.token;
    if (!token) return res.status(403).send("Token required");

    jwt.verify(token, "secret_key", (err, decoded) => {
        if (err) return res.status(401).send("Invalid token");
        req.user = decoded;
        next();
    });
}
```

# Comparatif

Méthode	Avant	Après (cookies)
Transport JWT	Header (authorization bearer)	Cookie (httpOnly)
Visibilité JS	Oui (risque XSS)	Non
Adapté	API REST	Web
Niveau de sécurité	++	+++++

# **Exercices**