

COMPTE RENDU TRAVAUX PRATIQUES – SUJET 4 Authentication et Autorisation

Étape 1 – Mise en place de l'authentification basique

Dans un premier temps, j'ai mis en place un système d'authentification basique (Basic Auth) afin de sécuriser certaines routes du serveur Fastify.

1) Installation et configuration du projet

J'ai commencé par récupérer et installer le projet via `npm install`, puis j'ai démarré le serveur avec `npm start`, ce qui m'a permis d'accéder aux routes en local (`http://localhost:3000`).

2) Ajout de l'authentification basique

L'objectif était de protéger une route spécifique par un identifiant et un mot de passe.

Grâce au plugin `@fastify/basic-auth`, j'ai mis en place une authentification restreignant l'accès à un utilisateur nommé *Tyrion* avec le mot de passe *wine*.

3) Création et test des routes

- `/dmz` : route accessible sans authentification.
- `/secu` : route sécurisée par Basic Auth.

Une tentative d'accès à `/secu` sans authentification retournait une erreur *401 Unauthorized*. En renseignant *Tyrion/wine* dans l'onglet *Authorization (Basic Auth)*, j'obtenais bien la réponse attendue.

4) Ajout d'une route publique /autre

J'ai ajouté une route supplémentaire `/autre`, accessible librement sans authentification.

La fonction `after()` s'assure que tous les plugins, y compris `@fastify/basic-auth`, sont enregistrés avant l'ajout des routes qui en dépendent, évitant ainsi d'éventuelles erreurs.

Étape 2 – Passage en HTTPS

Dans cette étape, l'objectif était de sécuriser les échanges entre le serveur et les clients en activant HTTPS.

1) Génération d'un certificat SSL auto-signé

J'ai généré un certificat SSL auto-signé avec OpenSSL, créant ainsi :

- Une clé privée (`server.key`).
- Un certificat SSL (`server.crt`).

```
openssl genpkey -algorithm RSA -out server.key
openssl req -new -key server.key -out server.csr
openssl x509 -req -days 365 -in server.csr -signkey server.key
```

2) Modification du serveur pour activer HTTPS

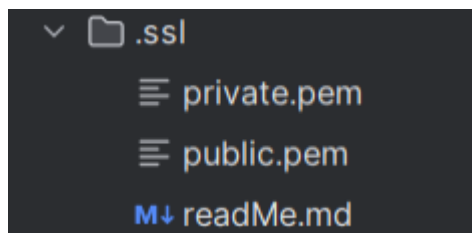
J'ai adapté le fichier server.js pour que Fastify utilise ces fichiers SSL, forçant ainsi l'utilisation de HTTPS.

3) Tests et validation

Après avoir redémarré le serveur, j'ai tenté d'accéder aux routes via `https://localhost:3000`.

Le certificat étant auto-signé, Postman affichait un avertissement SSL. En désactivant la vérification SSL dans les paramètres, tout fonctionnait normalement.

Étape 3 – Un jeton dans la machine 1) Génération des Clés ECDSA Pour assurer la sécurité des jetons JWT, nous avons utilisé l'algorithme ES256, qui nécessite une clé privée pour signer les jetons et une clé publique pour les vérifier. Ces clés ont été générées avec OpenSSL et stockées dans le dossier .ssl.

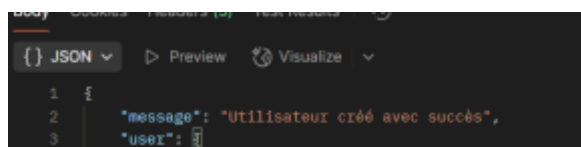


2) Configuration du Plugin JWT Afin de gérer les jetons JWT, nous avons enregistré un plugin Fastify qui permet :

- De signer les jetons lors de la connexion.
- De vérifier l'authenticité des jetons pour les routes protégées. Cette configuration permet d'assurer que seuls les utilisateurs ayant un jeton valide peuvent accéder aux ressources sécurisées.

3) Inscription des Utilisateurs (/signup) L'API permet aux utilisateurs de s'inscrire en fournissant une adresse email et un mot de passe. Lors de l'inscription :

- Le mot de passe est haché avant d'être stocké.
- Un rôle aléatoire est attribué (soit admin, soit utilisateur).



4) Connexion et Génération du Jeton JWT (/signin) Une fois inscrit, un utilisateur peut se connecter avec son email et son mot de passe. Si les informations sont correctes :

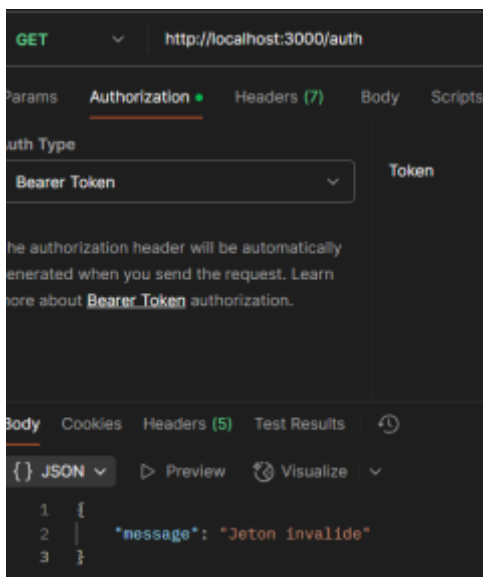
- Un jeton JWT est généré et envoyé en réponse.
- Ce jeton contient les informations de l'utilisateur (email et rôle).
- Il est utilisé pour accéder aux routes protégées. Si les informations sont incorrectes, l'API renvoie une erreur d'authentification.



```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFpbCI6Imxha3NhbkBnbWVpbC5jb291LlciLCJyb2x1IjoidXRpbG1zYXRleXIiLCJpYXQ1OiJlE3HzkyWzE2OT19NjB4LmF-QypH8k5TagdkVjGGZ_vh_dRA1V0swg258NtYnkQX5PPaUJXdeN03aUvM21rXvdA9uD10-cr0pLAdxQ"
3 }
```

5) Vérification du Jeton JWT (/auth) Une route protégée permet de tester l'accès aux utilisateurs authentifiés. L'utilisateur doit inclure son jeton JWT dans la requête.

- Si le jeton est valide, l'API retourne les informations de l'utilisateur et son rôle.
- Si le jeton est invalide ou absent, l'accès est refusé avec un code d'erreur 401



```
GET http://localhost:3000/auth

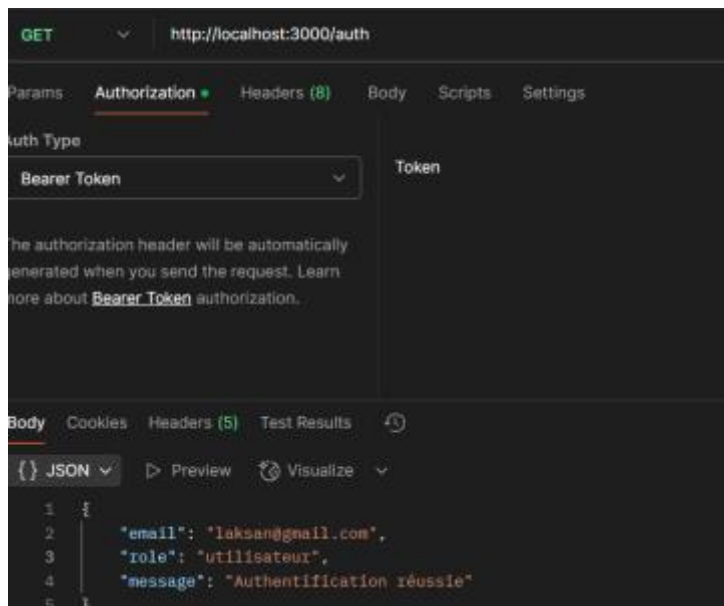
Params  Authorization  Headers (7)  Body  Scripts

Auth Type
Bearer Token  Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Body  Cookies  Headers (5)  Test Results

1 {
2   "message": "Jeton invalide"
3 }
```



6) Gestion des Rôles Une distinction a été mise en place entre les administrateurs et les utilisateurs classiques :

- Un administrateur a accès à certaines fonctionnalités spécifiques.
- Un utilisateur classique a des accès restreints.