Une image contenant texte, Police, Graphique, calligraphie

Description générée automatiquement

**BUT 3 INFORMATIQUE**

**Parcours A FI**

**Réalisations d'applications**

**RAPPORT TP4**

**Authentification et Autorisation**

**Prénom et Nom de l’étudiant :** Akash Selvaratnam

**Groupe :** 303

**Promotion :** 2023-2024

# Sommaire

Table des matières

[Sommaire 2](#_Toc159169707)

[Étape 1 – « De base… » 3](#_Toc159169708)

[Étape 2 – Prouves qui tu es ! 6](#_Toc159169709)

[Étape 3 – Un jeton dans la machine 7](#_Toc159169710)

# Étape 1 – « De base… »

Au départ, j’ai pu prendre connaissance du code présent avec le server.js qui peut être lancé sur le port 3000 en localhost avec Fastify qui propose deux différentes requêtes GET <http://localhost:3000/dmz> et <http://localhost:3000/secu> et qui valide seulement les personnes ayant pour nom d’utilisateur Tyrion et mot de passe wine.

Pour l’étape 1, j’ai pu tester via Postman deux différentes requêtes Get <http://localhost:3000/dmz> et <http://localhost:3000/secu> et observez le résultat retourné.

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

Figure 1 : Requête <http://localhost:3000/dmz>

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Figure 2 : Requête <http://localhost:3000/secu>

Ensuite, j’ai pu réaliser une authentification pour la requête http://localhost:3000/secu sur Postman en indiquant le nom ainsi que le mot de passe que nous avons indiqué dans la fonction validate du fichier server.js afin que le retour « Un Lannister paye toujours ses dettes ! » puisse être envoyé, car il est envoyé seulement s'il respecte les authentifications.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Figure 3 : Configuration de l'Authorization

Le rôle de la fonction after() est d’être exécuté lorqu’e l’ensemble des plugins ont terminé de charger, il est toujours exécuté avant la fonction ready().

Dans la fonction after(), j’ai ajouté une nouvelle route /autre du type GET, mais à la différence de la route /secu cela doit être accessible sans les authentifications donc sans le onRequest.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Figure 4 : Fonction after()

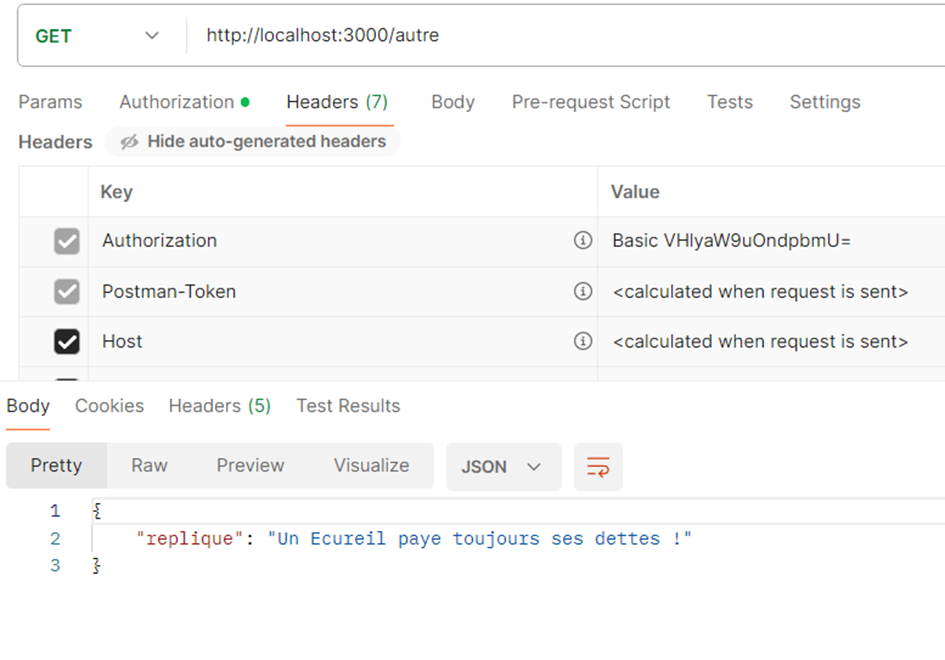


Figure 5 : Requête <http://localhost:3000/autre>

# Étape 2 – Prouves qui tu es !

Pour l’étape 2, j’ai créé une nouvelle clé RSA de 2048 bits nommé server.key avec la commande suivante openssl genrsa -out server.key 2048 .

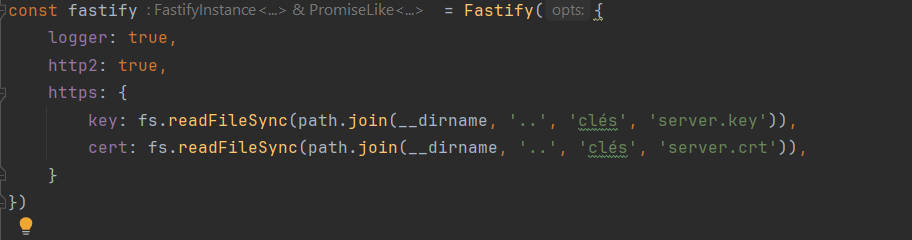
J’ai ensuite créé un nouveau Certificate Signing Request en signant mon certificat avec la clé privé créé précédemment à l’aide des commandes proposé sur les slides 35 et 36 et ensuite, j’ai pu tester le certificat généré via la commande proposé dans le cours permettant de vérifier le certificat sur le port 4567 avec l’application Postman.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Figure 6 : Test du Certificat sur Postman

J’ai ensuite configuré Fastify en HTTPS en indiquant le chemin vers ma clé privée ainsi que vers ma certification, j’ai du également importer des modules telles fs pour pouvoir lire les fichiers et le module path.



# 

# Étape 3 – Un jeton dans la machine

Pour l’étape 3, j’ai créé une clé privée ainsi qu’une clé publique avec openssl en ECDSA qui est compatible avec la norme JWT. Pour Savoir, si ma clé est compatible, j’ai consulté la documentation (lien : RFC 7519 - JSON Web Token (JWT) (ietf.org) qui m’a indiqué que P-256 était compatible avec l’algorithme ECDSA. J’ai ensuite lancé la commande suivante : openssl ecparam -list\_curves pour voir la clé correspondant à P-256 qui est prime256v1. J’ai pu ensuite créer ma clé privée et ma clé publique.

Pour pouvoir configuré mon fastifyJwt, j’ai tout simplement indiqué le chemin vers clé publique et le chemin vers ma clé privé.

Une image contenant texte, capture d’écran, Logiciel multimédia

Description générée automatiquement

Figure 7 : Configuration fastifyJwt

J’ai ensuite complété la fonction addUser() en créant un objet contenant le mail de l’utilisateur, son mot de passe haché ainsi que son rôle via la fonction random de la librairie Math. J’ai ajouté cet objet dans mon tableau d’utilisateur.

Une image contenant texte, capture d’écran, Logiciel multimédia, logiciel

Description générée automatiquement

Figure 8 : Fonction addUser()

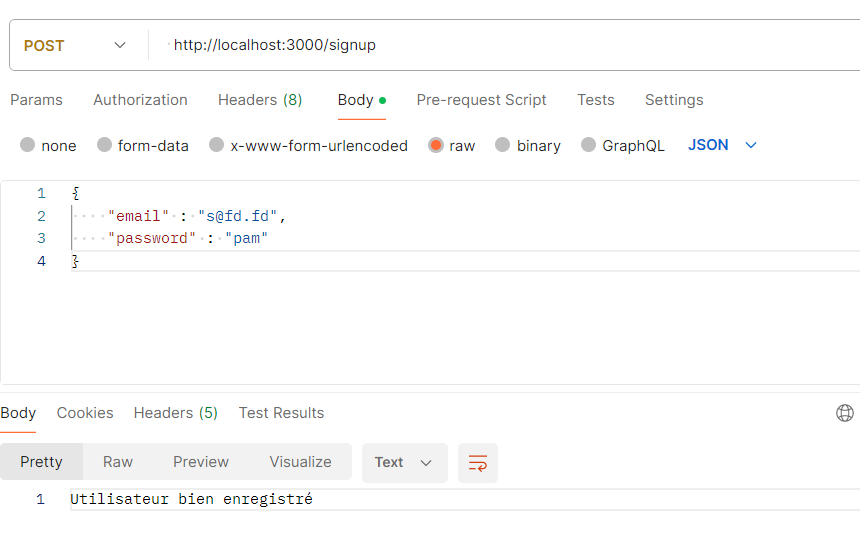


Figure 9 : Résultat Ajout d'un utilisateur

J’ai créé la fonction loginUser() en récupérant le mail ainsi que le mot de passe, puis en hachant le mot de passe et en vérifiant que l’utilisateur existe, si l’utilisateur existe alors je crée un jeton avec la fonction jwtSign proposé par Fastify qui contient le mail ainsi que le rôle de l’utilisateur en tant que payload et je renvoie cette information avec un statu 200 sinon je renvoie le message « utilisateur non-identifié » avec le code statu 401.

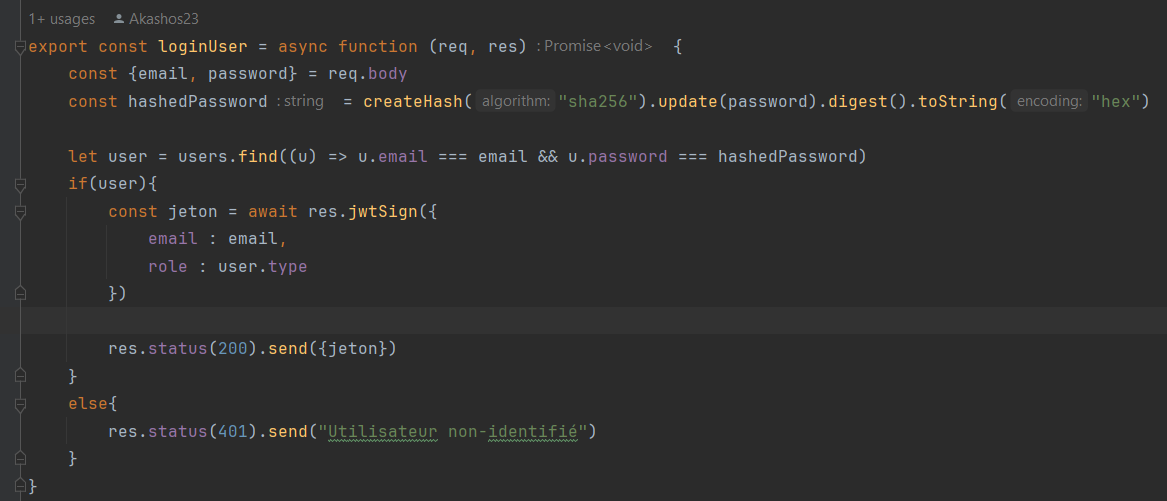


Figure 10 : Fonction loginUser()

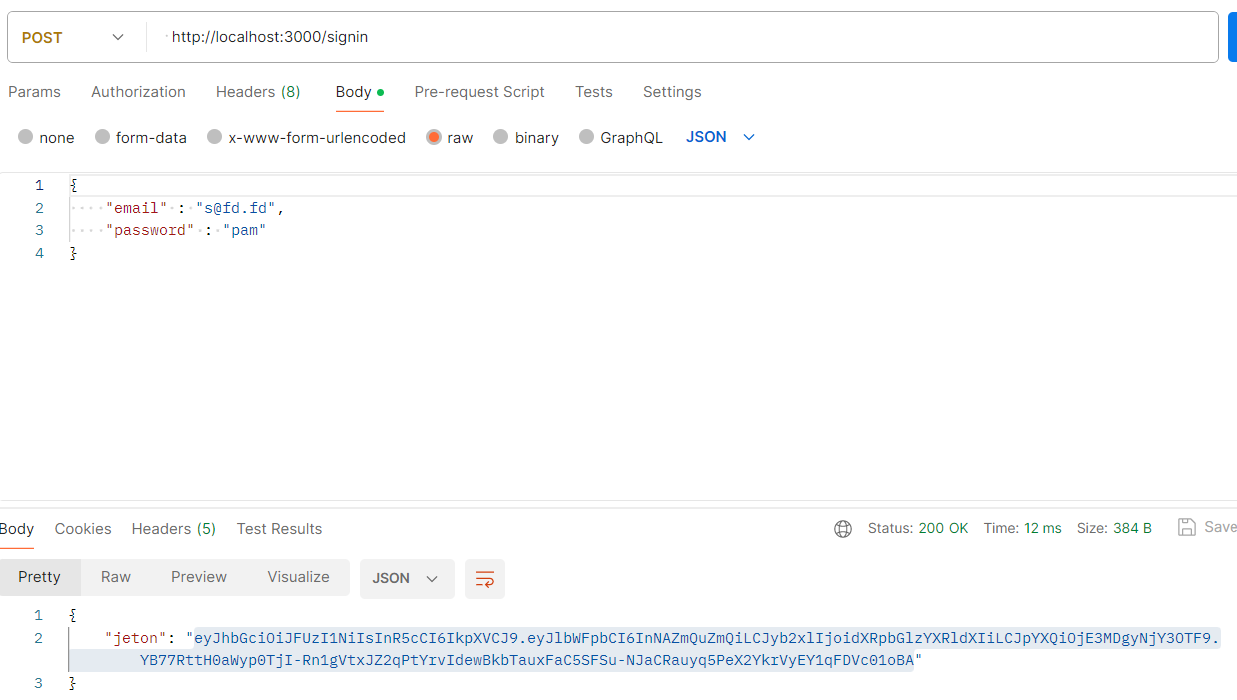


Figure 11 : Résultat connexion d'un utilisateur

J’ai ensuite complété le fastifyJwt dans le deuxième service de la même façon que le premier service en créant un répertoire .ssl qui contient mes clé privé et publique.

J’ai complété la fonction getAuthentificate() en indiquant sur Postman, dans authorization, OAuth 2.0 et en complétant le champ token avec le résultat que j’ai obtenue lors de la connexion.

Dans ma fonction getAuthenticate(), j’ai récupéré mon token puis j’exécute la fonction jwtVerify de la librairie Fastify qui prend en argument le token et qui permet de vérifier que le token est bien valide et existant.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Figure 12 : Fonction getAuthenticate

Une image contenant texte, logiciel, Police, nombre

Description générée automatiquement

Figure 13 : Résultat d'un token non valide

Pour finir, j’ai complété la fonction getAuthHandler() en récupérant le token puis en utilisant la fonction jwtVerify qui permet également de récupérer le contenu (mail, rôle) du token, je vérifie le rôle de l’utilisateur, si l’utilisateur est un admin alors j’envoie le message suivant « Full Access » sinon « Accès limité ».



Figure 14 : Fonction getAuthHandler

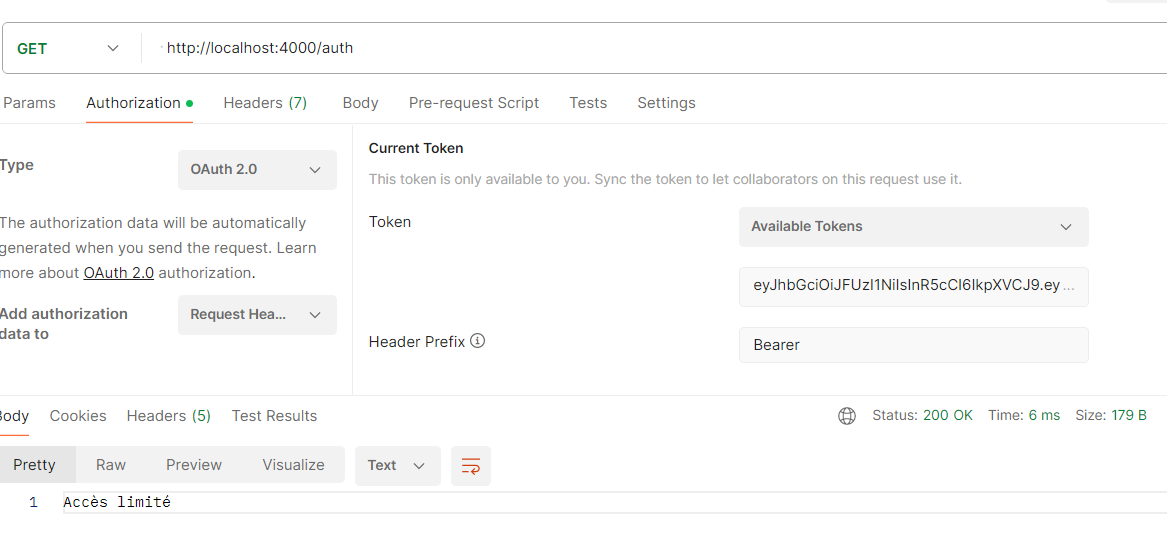


Figure 15 : Résultat d'une personne ayant le rôle d'utilisateur