

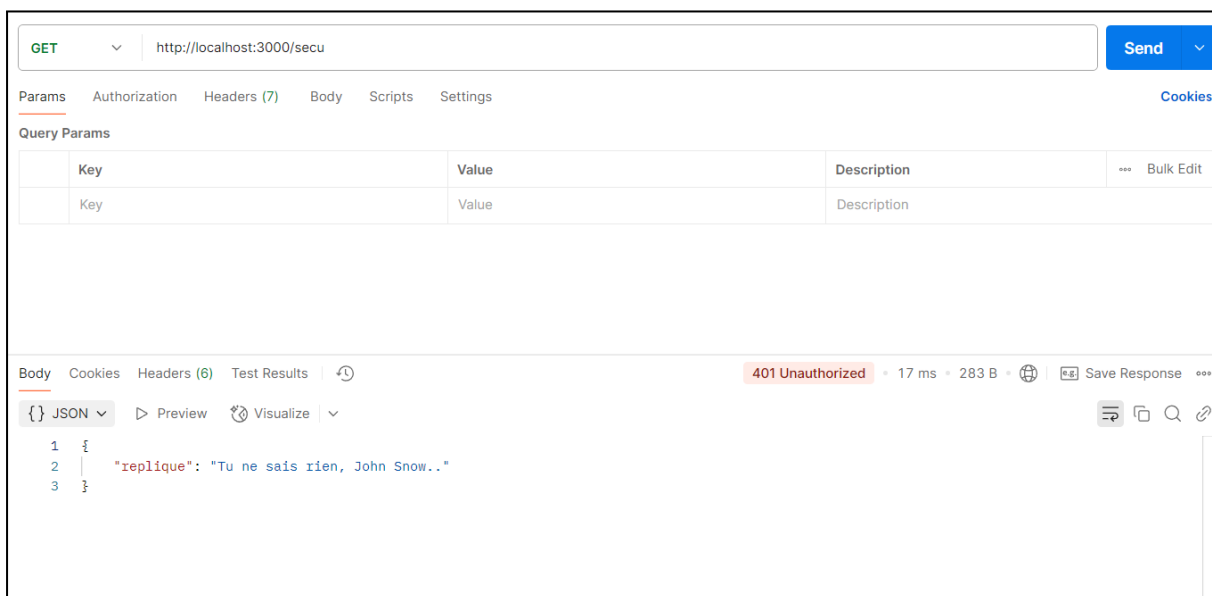
R6.A.05 Développement Avancé

TP4: Authentification et Autorisation – 11/02/2025

Étape 1 – « De base... »

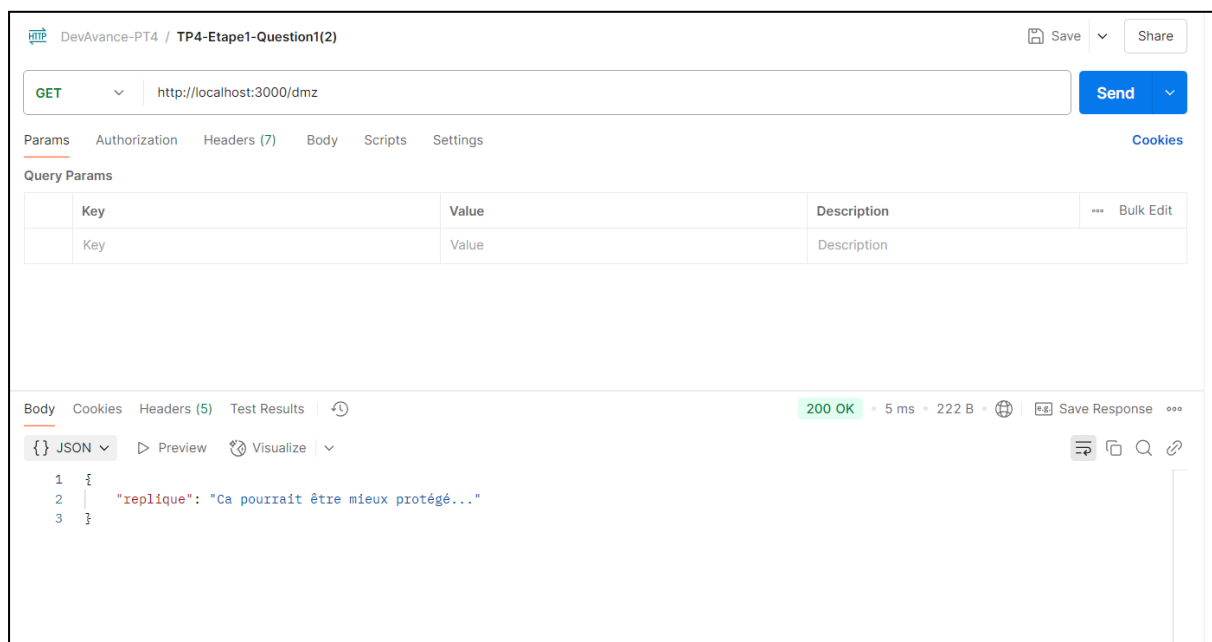
Etape 1 question 1:

Avec le logiciel Postman, sans préciser de paramètre supplémentaire, j'ai essayé d'accéder aux end-points `http://localhost:3000/secu` et <http://localhost:3000/dmz>, en créant une collection dédié. Voici les résultats obtenus:



Postman interface showing a GET request to `http://localhost:3000/secu`. The response is a 401 Unauthorized status with a response time of 17 ms and 283 B. The response body is a JSON object:

```
{  "replique": "Tu ne sais rien, John Snow.."}
```



Postman interface showing a GET request to `http://localhost:3000/dmz`. The response is a 200 OK status with a response time of 5 ms and 222 B. The response body is a JSON object:

```
{  "replique": "Ca pourrait être mieux protégé..."}
```

Etape 1 question 2

Nous avons encodé les identifiants de connexion de la façon suivante.

Encodage au format Base64

Il suffit de saisir vos données et d'appuyer sur le bouton d'encodage.

Tyrion.wine

1 Pour encoder des binaires (comme des images, des documents, etc.), utilisez le formulaire de téléchargement de fichiers un peu plus bas sur cette page.

UTF-8

Jeu de caractères de destination.

LF (Unix)

Séparateur de nouvelle ligne de destination.

☐ Encodez chaque ligne séparément (utile lorsque vous avez plusieurs entrées).

☐ Divisez les lignes en segments de 76 caractères (utile pour MIME).

☐ Effectuez un encodage sûr pour les URL (utilise le format Base64URL).

☒ Mode direct OFF

Encodage en temps réel alors que vous tapez ou collez (prenant en charge uniquement le jeu de caractères UTF-8).

> ENCODAGE <

Encodage de vos données dans la zone ci-dessous.

VHlyYW9uOndpbmU=

L'authentification est réussie. Voici le résultat obtenu sur postman :

DevAdvance-PT4 / TP4-Etape1-Question1

Save

Share

GET

http://localhost:3000/secu

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

<input checked="" type="checkbox"/> Host	<calculated when request is sent>	
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.43.0	
<input checked="" type="checkbox"/> Accept	/*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> Authorization	Basic VHlyYW9uOndpbmU=	
	Key	Value
		Description

Body

Cookies

Headers (5)

Test Results

200 OK • 6 ms • 225 B • Save Response

{}

JSON

Preview

Visualize

```
1 {
2   |   "replique": "Un Lannister paye toujours ses dettes !"
3   }
```

Etape 1 question 3

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/secu
- Auth Type:** Basic Auth
- Username:** Tyrion
- Password:** wine
- Status:** 200 OK
- Response Body (JSON):**

```
{  "replique": "Un Lannister paye toujours ses dettes !"}
```

Authorization est cochée automatiquement dans Headers à la suite de cela.

On reteste ensuite les end-points et on confirme bien qu'une erreur dans la saisie du username/password rejette l'authentification ("wineee" au lieu de "wine").

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/secu
- Auth Type:** Basic Auth
- Username:** Tyrion
- Password:** wineee
- Status:** 401 Unauthorized
- Response Body (JSON):**

```
{  "replique": "Tu ne sais rien, John Snow.."}"
```

Etape 1 question 4

La méthode `.after()` permet d'exécuter du code uniquement après que tous les plugins aient été enregistrés. En d'autres termes, elle garantit que les routes ou les fonctionnalités ajoutées dépendent bien des plugins chargés auparavant.

Etape 1 question 5

```
fastify.route({
  method: 'GET',
  url: '/autre',
  handler: async (req, reply) => {
    return {
      replique: 'Autre route !'
    };
  }
});
```

On remarque l'absence de `onRequest: fastify.basicAuth`, car on souhaite que la route soit accessible sans authentification

Étape 2 – Prouves qui tu es !

Etape 2 question 1 :

Créer une nouvelle clé RSA de 2048 bits appelé `server.key` :

```
openssl genrsa -out server.key 2048
```

Etape 2 question 2 :

Dans mon terminal cmd j'ai entré:

```
C:\Users\bsikr\BUT3\Dev_Avancee\DevAvance-TP4>openssl req -new -key server.key -out server.req
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:nadir.fr
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

ensuite j'ai renommé `server.req` en `server.csr`

J'ai signé avec la commande: `openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt`

Enfin j'ai testé le bon fonctionnement avec la commande suivante et Postman

openssl s_server -accept 4567 -cert server.crt -key server.key -www -state

```
PS C:\Users\bsikr\BUT3\Dev_Avancee\DevAvance-TP4> openssl s_server -accept 4567 -cert server.crt -key server.key -www -state
Using default temp DH parameters
ACCEPT
```

The screenshot shows the Postman interface for a GET request to `https://localhost:4567/`. The response status is **200 ok** with a response time of 20 ms and a size of 4.34 KB. The response body is displayed in HTML format, showing the output of the `s_server` command. The output includes the command itself, a message about TLS renegotiation, supported ciphers, and a list of supported TLS versions and cipher suites.

```
<HTML><BODY BGCOLOR="#ffffff">
<pre>
s_server -accept 4567 -cert server.crt -key server.key -www -state
This TLS version forbids renegotiation.
Ciphers supported in s_server binary
TLSv1.3      :TLS_AES_256_GCM_SHA384      TLSv1.3      :TLS_CHACHA20_POLY1305_SHA256
TLSv1.3      :TLS_AES_128_GCM_SHA256      TLSv1.2      :ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1.2      :ECDHE-RSA-AES256-GCM-SHA384  TLSv1.2      :DHE-RSA-AES256-GCM-SHA384
TLSv1.2      :ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2      :ECDHE-RSA-CHACHA20-POLY1305
TLSv1.2      :DHE-RSA-CHACHA20-POLY1305  TLSv1.2      :ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1.2      :ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2      :DHE-RSA-AES128-GCM-SHA256
TLSv1.2      :ECDHE-ECDSA-AES256-SHA384  TLSv1.2      :ECDHE-RSA-AES256-SHA384
TLSv1.2      :DHE-RSA-AES256-SHA256      TLSv1.2      :ECDHE-ECDSA-AES128-SHA256
TLSv1.2      :ECDHE-RSA-AES128-SHA256    TLSv1.2      :DHE-RSA-AES128-SHA256
TLSv1.0      :ECDHE-ECDSA-AES256-SHA      TLSv1.0      :ECDHE-RSA-AES256-SHA
TLSv1.0      :DHE-RSA-AES256-SHA         TLSv1.0      :ECDHE-ECDSA-AES128-SHA
TLSv1.0      :ECDHE-RSA-AES128-SHA       SSLv3       :DHE-RSA-AES128-SHA
TLSv1.2      :RSA-PSK-AES256-GCM-SHA384  TLSv1.2      :DHE-PSK-AES256-GCM-SHA384
TLSv1.2      :RSA-PSK-CHACHA20-POLY1305  TLSv1.2      :DHE-PSK-CHACHA20-POLY1305
```

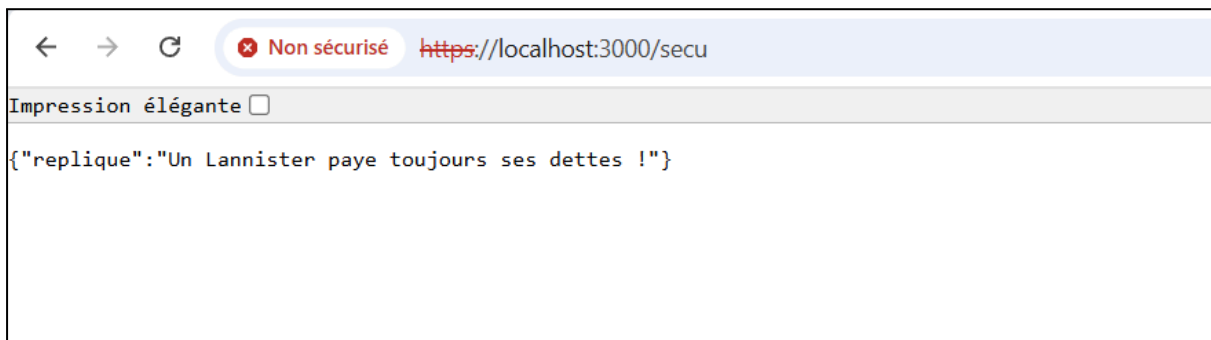
Etape 2 question 3 :

```
const options : {...} = {
  http2: true,
  https: {
    key: fs.readFileSync(path.join(separator: './server.key')),
    cert: fs.readFileSync(path.join(separator: './server.crt')),
    //allowHTTP1: true
  },
  logger: true
};
const fastify : FastifyInstance<...> & PromiseLike<...> = Fastify(options)
```

En ajoutant les options ci-dessus, nous avons fait évoluer le service web pour que l'accès s'effectue en https avec la clé privée et le certificat que nous avons généré.

allowHTTP1 a été ajouté car Postman ne semble pas supporter http2 convenablement. Toutefois, lorsqu'on teste depuis le navigateur, le allowHTTP1 n'est pas nécessaire.

Voici le résultat de la requête <https://localhost:3000/secu> :



Étape 3 – Un jeton dans la machine

Après avoir cloné le dépôt de l'enseignant

- Générer les clés de chiffrement avec openssl en ECDSA compatible avec la norme JWT. Vous les placerez dans le dossier .ssl du projet.

```
openssl ecparam -name prime256v1 -genkey -noout -out .ssl/ec_private.pem
openssl ec -in .ssl/ec_private.pem -pubout -out .ssl/ec_public.pem
```

- Compléter l'enregistrement du plugin fastifyJwt dans le fichier src/plugins/jwt.js.

```
export default fp( fn: async function (app, opts) : Promise<void> {

  const privateKey = fs.readFileSync('.ssl/ec_private.pem', 'utf8');
  const publicKey = fs.readFileSync('.ssl/ec_public.pem', 'utf8');

  app.register(fastifyJwt, {
    secret: {
      private: privateKey,
      public: publicKey
    },
    sign: {
      algorithm: 'ES256',
      issuer: 'info.iutparis.fr'
    },
    verify: {
      algorithms: ['ES256'] // Autorise uniquement ES256
    }
  })
})
```

- Compléter le handler addUser() dans le cas où il s'agit d'un nouveau compte utilisateur dans le fichier src/controllers/login.js :

```
else{  
  let newUser : {...} = {  
    email: email,  
    password: hashedPassword  
  }  
  users.push(newUser)  
  
  res.status(201).send({  
    user :newUser,  
    message: "utilisateur créé avec succès"  
  })  
}  
}
```

- Écrire le handler loginUser() dans le fichier src/controllers/login.js pour signer et renvoyer un jeton pour un utilisateur valide.

```
export const loginUser = async function (req, res) {  
  const { email, password } = req.body;  
  const hashedPassword =  
    createHash("sha256").update(password).digest("hex");  
  let user = users.find((u) => u.email === email && u.password  
    === hashedPassword);  
  
  if (!user) {  
    return res.status(401).send({  
      message: "Utilisateur non-identifié"  
    });  
  }  
  
  const token = await res.jwtSign({ email }, {expiresIn : "1h"});  
  return res.send({  
    message: "Connexion réussie",  
    token  
  });  
}
```

En testant sur Postman, on constate que l'ajout d'un user ainsi que son authentification fonctionnent correctement.

POST http://localhost:3000/signup Send

Params Authorization Headers (11) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "email": "user@example.com",
3   "password": "motdepasse"
4 }
```

Body Cookies Headers (5) Test Results 201 Created • 3 ms • 337 B Save Response

POST http://localhost:3000/signin Send

Params Authorization Headers (11) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "email": "user@example.com",
3   "password": "motdepasse"
4 }
```

Body Cookies Headers (5) Test Results 200 OK • 4 ms • 423 B Save Response

Body Cookies Headers (5) Test Results 200 OK • 4 ms • 423 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "message": "Connexion réussie",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnVzZXJyQGV4YW1wbGUuY29tIiwiaWF0IjoxNzU5MjYyNTgzLCJleHAiOjE3MzkyNzYxODN9.OHZXZofif7duIpvR6n6aj_5Dq_yE6h4-HtuauU6Ub91jVZ6sQcmpESw31nWUdPqwVJIKFFGvkCgFPGWEEC9VMQ"
4 }
```