Gabriel Dos Santos

TP API

API TP

QUESTIONS CREATION D'UNE API

Comment fonctionne l’architecture Node.js ?

Node.js permet d’exécuter du javascript côté serveur ce qui donne la possibilité par exemple de créer des API. Node à la particularité de pouvoir réalisé des évènements de façon asynchrone, c’est à dire quelle ne bloque pas les opérations en premier plan mais elle les rétrograde plutôt en arrière-plan afin de permettre la simultanéité des tâches. Node.js est aussi très apprécié pour ses nombreux packages disponibles et sa gestion efficiente des requêtes HTTP a souligné qu’il est notamment possible de créer avec, des serveurs.

Qu’est-ce qu’une API et comment elle fonctionne ?

Un API est tout simplement un outil qui permet de reprendre des services déjà existant afin d'éviter au programmeur de recoder un outil similaire (avec UBER on utilise google maps pour éviter de coder ce même service). Cela permet pour le programmeur de gagner du temps et pour l'entreprise qui met à disposition l'API de gagner de l'argent.

Une interface de programmation d'application a donc pour but de faire dialoguer plusieurs logiciels.

Une API permet à des programmes de communiquer entre eux via des appels d’API. Ces appels sont émis par un client API et reçu par un point de terminaison d’API. Pour permettre leur communication l’API doit utiliser un protocole spécifique, très souvent le HTPP. Les requêtes sont envoyées dans les différents points de terminaison puis renvoyé au client API. Pour permettre l’échange de ces données, l’API les formate sous un format JSON.

Qu’est-ce qu’un Middleware et lesquels vais-je utiliser :

Un middleware est tout simplement une fonction qui se place avant la redirection vers une route. C’est une sorte de sécurité que les requêtes HTTP doivent passer pour accéder à la route. On a par exemple des middlewares qui vont vérifier les données entrantes, qui vont authentifier l’utilisateur...

Middleware de validation de données entrantes :

Ce Middleware va me permettre de vérifier à l’aide d’express-validator les données entrées dans le body et de rediriger une erreur si celle-ci ne respectent pas les règles énoncées. Si elles correspondent elles donneront accès à la route spécifiée.

Middleware admin :

Comme son nom l’indique ce middleware va vérifier si l’utilisateur à les droit nécessaire, en l’occurrence ici être prof, pour accéder à l’intégralité des fonctions disponibles.

A voir Middleware qui authentifie la personne :

Ce middleware va permettre de donner l'accès à certaines données pour les personnes identifier avec un Token stockant la clé api présente dans le fichier .env

Quelles sont les mesures de sécurité mis en place, comment se protéger contre les attaques de type "Injection SQL" et pourquoi ?

D'un point de vue sécurité il est primordiale de placer les données sensibles dans un **fichier .env,** cela est plus qu'important car elle permet de garder ces données en dehors du codes sources. On peut y retrouver par exemple les données permettant l'accès à la database ou encore la clé API.

Le fait qu'on mette en place un **middleware de validation de données entrantes** est un bon point contre l'injection SQL car il permet de restreindre par des règles le contenu des différents champs saisis. De même pour le **middleware isAdmin** qui donne accès à certaines données sensibles uniquement aux individus autorisés (admin). On peut aussi citer le **middleware d'authentification** qui va permettre de donner l'accès à certaines données pour les personnes identifier avec un token précis.

Enfin on peut mentionner l'implémentation **des procédures stockées** permettant d'éviter la réalisation de requête autre que celle appelé et donc d'éviter un libre accès aux bases de données.

Il est important de se prémunir de cela car les bases de données stockent des données confidentielles, il faut donc impérativement les protégées.

DOCUMENTATION

1. Initialisation d’un nouveau projet Node.js et installation des dépendances

Npm init a pour but d’initialiser un nouveau projet Node.js et créé avec un fichier package.json. Ce fichier va stocker les métadonnées du projet (nom, version, auteur...), et ses dépendances.

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (tp_api)
version: (1.0.0)
description:
entry point: (server.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\Code FICHIER\IPSSI COURS\Api\TP_API\package.json:

{
  "name": "tp_api",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes) yes
```

D

Dotenv

Npm i dotenv est un module permettant d’importer des variables d’environnement à partir d’un fichier .env qu’on a au préalablement créé au sein de notre projet. Il y a plusieurs intéret à utiliser ce module. Il apporte dans un premier temps de la praticité mais aussi de la sécurité. En effet, il stock de manière sécurisée les données sensibles tel que les clé API par exemple.

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i dotenv

added 1 package, and audited 2 packages in 1s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Express

Npm i express permet d'installer le framework Espress. Il est très souvent utilisé car il fournit permet de gérer efficacement les requête HTTP, les route ou encore les vues

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i express

added 64 packages, and audited 66 packages in 3s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API>
```

MariaDB

Npm i mariadb nous permettra d'exploiter nos bases de données dans notre projet

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i mariadb

added 6 packages, and audited 72 packages in 2s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Bcrypt

Npm i bcrypt nous permet de mettre en place le hachage de nos mots de passe. Il nous permettra de définir la complexité de notre hachage mais aussi de comparé un mot de passe haché avec un mot lisible et ainsi vérifier s'ils correspondent.

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i bcrypt

added 60 packages, and audited 132 packages in 5s

16 packages are looking for funding
  run `npm fund` for details
```

Jsonwebtoken

Npm i jsonwebtoken va nous permettre de créer et vérifier des jsonwebtoke. Ces tokens sont générés lorsqu'un utilisateur se log et ils sont ensuite utilisés pour authentifier de manière sécurisé un utilisateur. Un token peut contenir des informations importantes sur l'utilisateur tel que son rôle ou ses différentes autorisations.

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i jsonwebtoken

added 13 packages, and audited 145 packages in 2s

16 packages are looking for funding
  run `npm fund` for details
```

Nodemon

Npm i nodemon permet un gain de praticité non négligeable car il permettra de redémarrer automatiquement notre projet afin d'appliquer nos changements. Cela nous évite de perdre du temps à lancer constamment manuellement notre projet.

```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i nodemon

added 25 packages, and audited 170 packages in 3s

19 packages are looking for funding
  run `npm fund` for details
```

Express-validator

Npm i express-validator nous permettra de pour attribuer des certaines conditions à nos valeurs entrantes

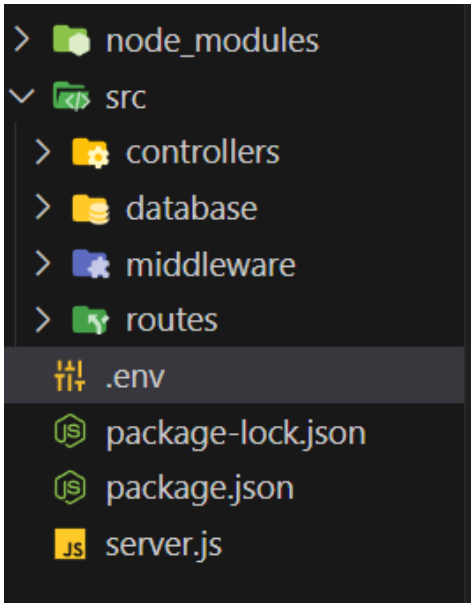
```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API> npm i express-validator

added 3 packages, and audited 174 packages in 4s

19 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Body-parser

Et enfin on finit par installer Body-Parser qui nous accorde le fait d'analyser le corps des requête http entrante. Lors de requête HTPP du style POST, PUT on envoie des données à travers le corps de la requête et body-parser va se charger de rendre ces données exploitables.

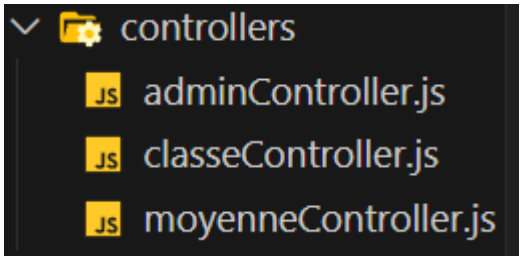
```
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API\src> npm i body-parser

up to date, audited 170 packages in 596ms

19 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Code FICHIER\IPSSI COURS\Api\TP_API\src>
```

2. Structure de mon projet



- **Dossier Database** : contiendra les fichiers de connexion à la base de données.
- **Dossier Routes** : contiendra la liste des différentes routes de notre API, on créera un fichier par entité de route (par exemple un fichier uniquement pour les utilisateurs, un autre pour les produits, etc.).
- **Dossier Controllers :** Contiendra les fonctions utilisées par nos routes !
- **Fichier Serveur.js :** Appellera nos routes, lancera le serveur
- **Dossier 'src' :** On mettra tous les fichiers dedans
- **Dossier middleware :** on y stockera notre middleware

3. Dossier controllers

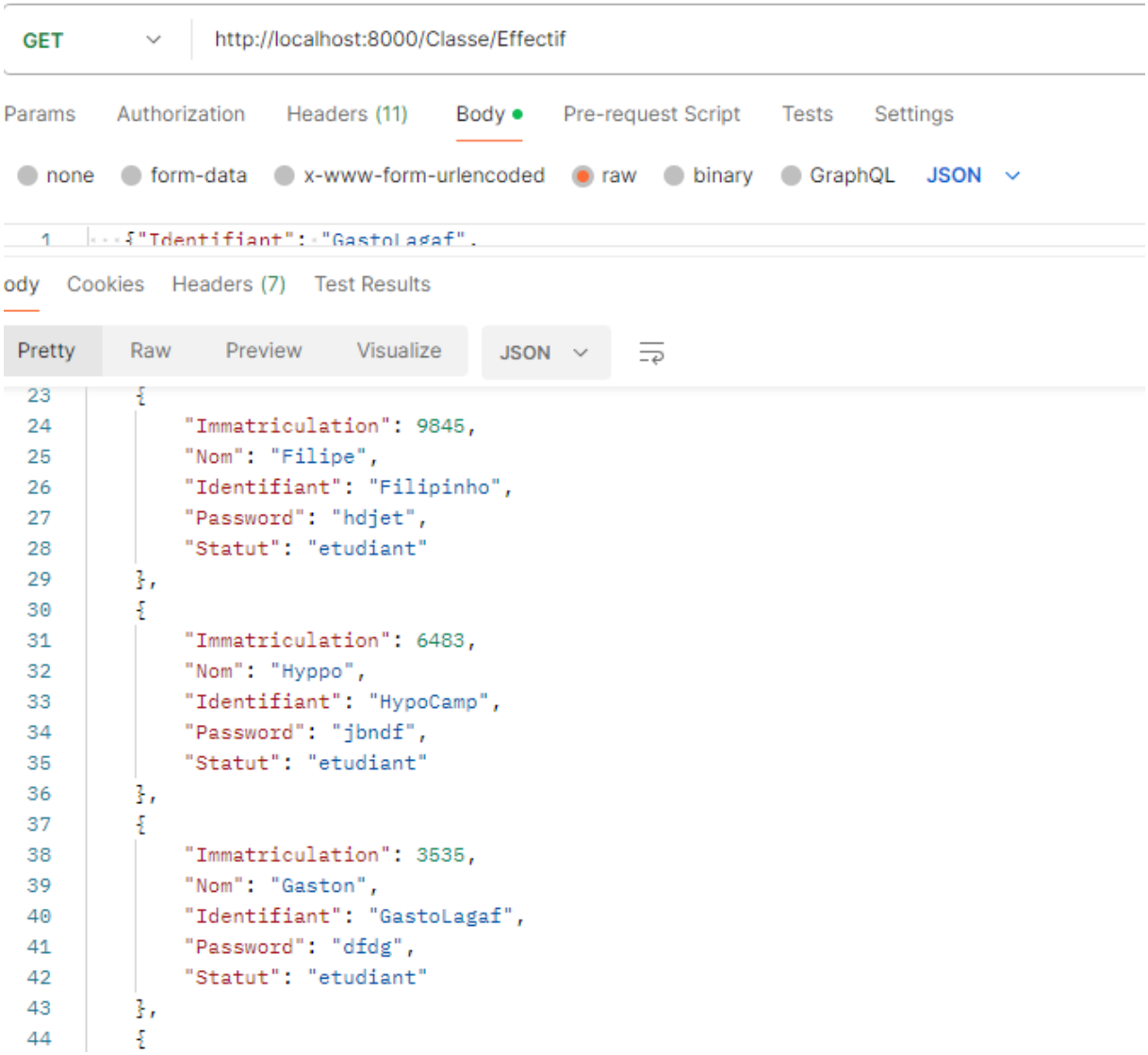


Le fichier **adminController** va nous permettre deux choses. Premièrement de vérifier le Token et dans un second temps d'extraire l'Identifiant de l'individu attribué au Token. On exportera cette fonction pour l'utiliser dans le **middlewareAdmin.**

```
const jwt = require('jsonwebtoken')


exports.getIdentifiantFromToken = (token) => {

    // on vérifie que le token
    try {
        const decoded =jwt.verify(token,process.env.API_KEY);
        // on décode le token pour avoir l'email
        return decoded.Identifiant;

    }catch(error) {
        return null;
    }

};
```

Dans le fichier **classeController** on retrouvera plusieurs fonctions. La première c'est getAllClasse qui permettra tout simplement d'afficher toutes les personnes qui composent la classe. Pour l'afficher on appellera notamment la procédure stockée allClasseProce().

A la suite de la documentation on mettra en place un accès davantage sécurisé pour cette fonction.



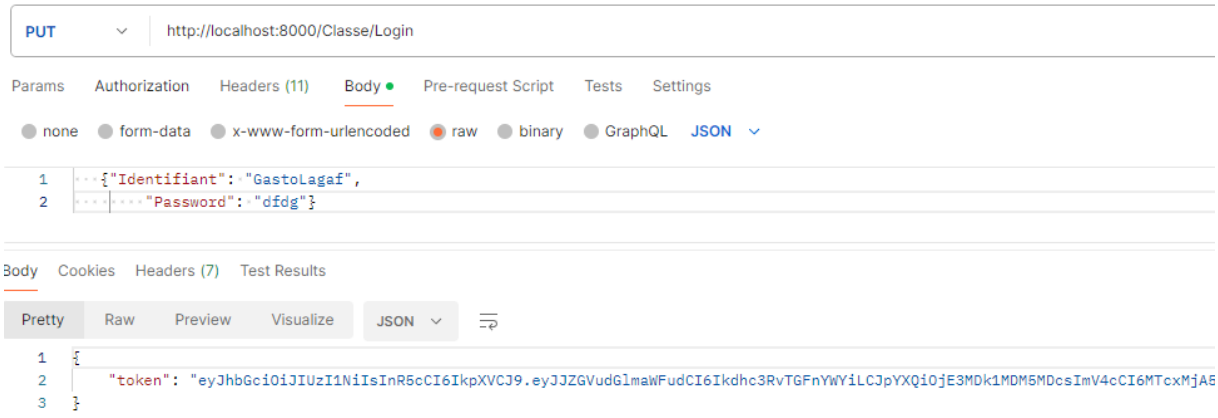
La dernière fonction et pas des moindres c'est putlogin() qui permet de log un individu. Cette fonction a plusieurs subtilités elle permet :

De vérifier si la personne existe ou non dans la bdd à l'aide de son identifiant

De log une personne qui ne sait jamais log auparavant (on arrive à le déterminer car une personne qui s'est déjà log possède dans la bdd un mp hashé)

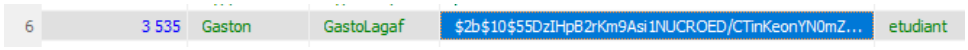
De log une personne qui s'est déjà log (possède un mp hashé dans la bdd)

Dans l'exemple je me log avec Gaston qui n'est pas encore connecté, ne pas oublier d'utiliser la méthode PUT sinon cela ne marchera pas :



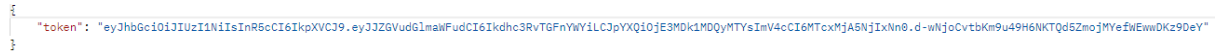
On a bien un Token qui se génère car son mp concorde et la console nous indique bien que la personne ne s'est jamais connectée auparavant et donc qu'il s'agit de sa première connexion :

On peut remarquer que son mp dans la bdd a bien été haché notamment grâce à l'opération PUT:

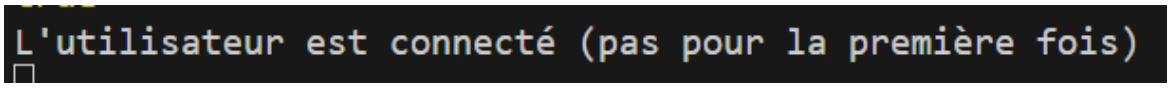


Si on souhaite se reconnecter avec le même utilisateur :

Un nouveau Token est généré :



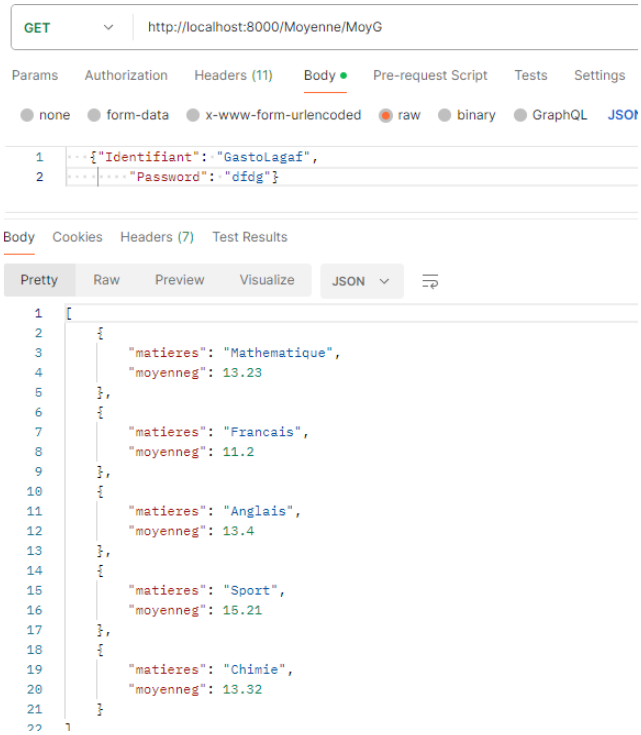
La console indique bien que l'utilisateur s'est connecté mais que ce n'est pas la première fois :



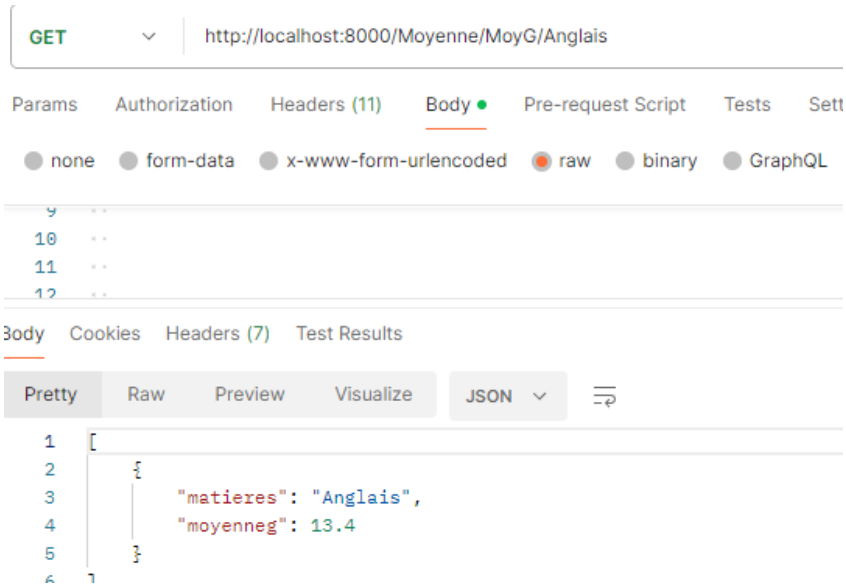
Si on se connecte avec un mauvais mot de passe :

Le dernier fichier controllers est le “moyenneController”. Il regroupe toutes les opérations CRUD. On a deux fonctions get, une qui permet d’afficher la moyenne générale de toutes les matières et une autre qui affiche selon l’URL tapé la moyenne générale d’une matière précise **(route dynamique)**.

Pour toutes les matières :

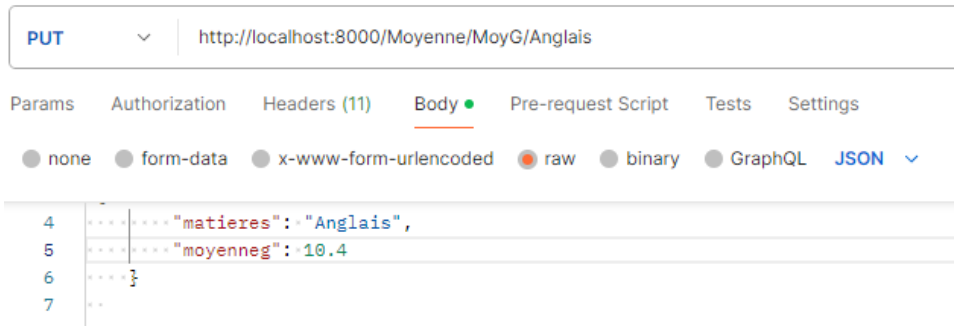


Pour une matière précise :

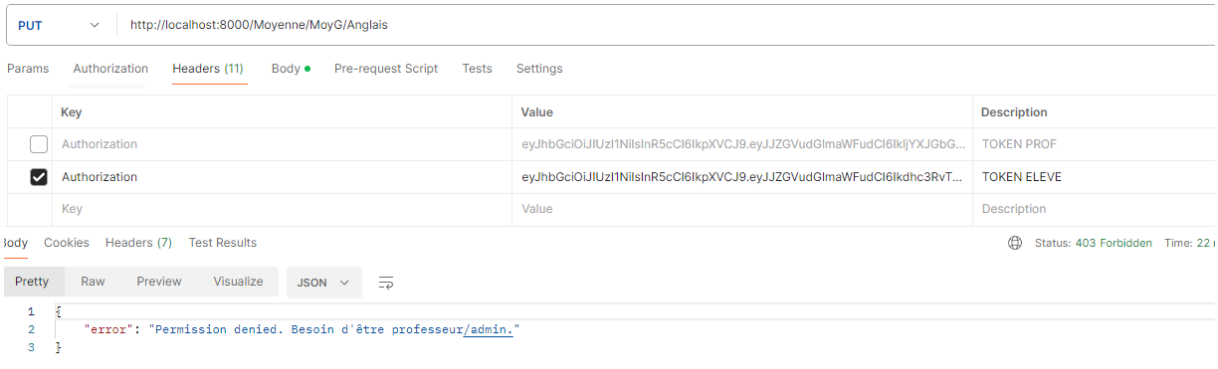


Ensuite on a une opération putMoyG qui permet à la personne qui est Admin (ici c'est le prof) de pouvoir mettre à jour les valeurs d'une matière, il peut aussi ajouter et supprimer une matière avec respectivement les fonctions postMatiere et deleteMatiere. Le fait qu'uniquement l'Admin puisse accéder à ces fonctionnalités, est dû au middlewareAdmin que j'ai créé et intégré à la route put, post et delete de mon fichier moyenneRoute. Ceci a pour but de renforcer la sécurité de mon API:

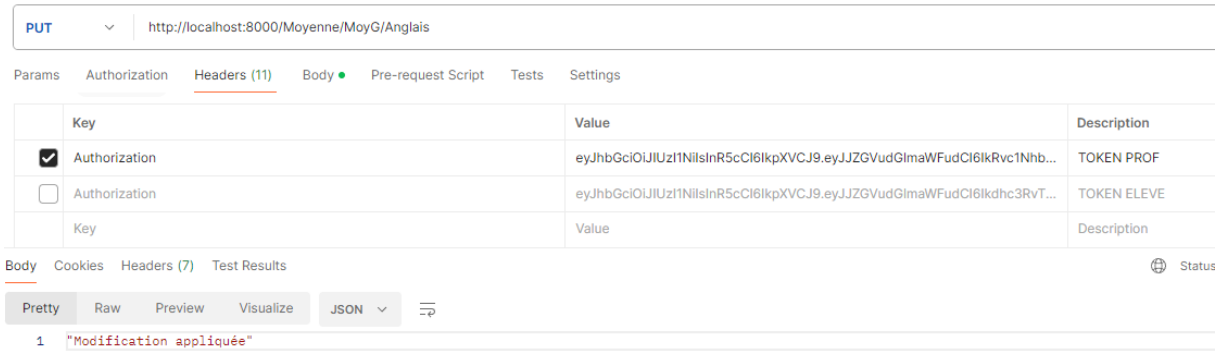
On essaye de modifier la moyenne générale en anglais en utilisant comme autorisation un Token d'élève.



On remarque qu'avec un Token élève on reçoit le message suivant :



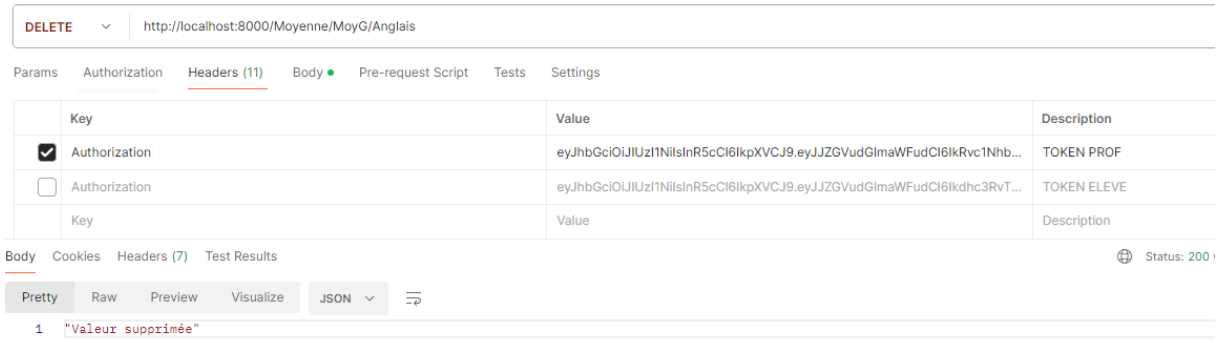
Alors qu'avec un Token de professeur l'opération se fait correctement :



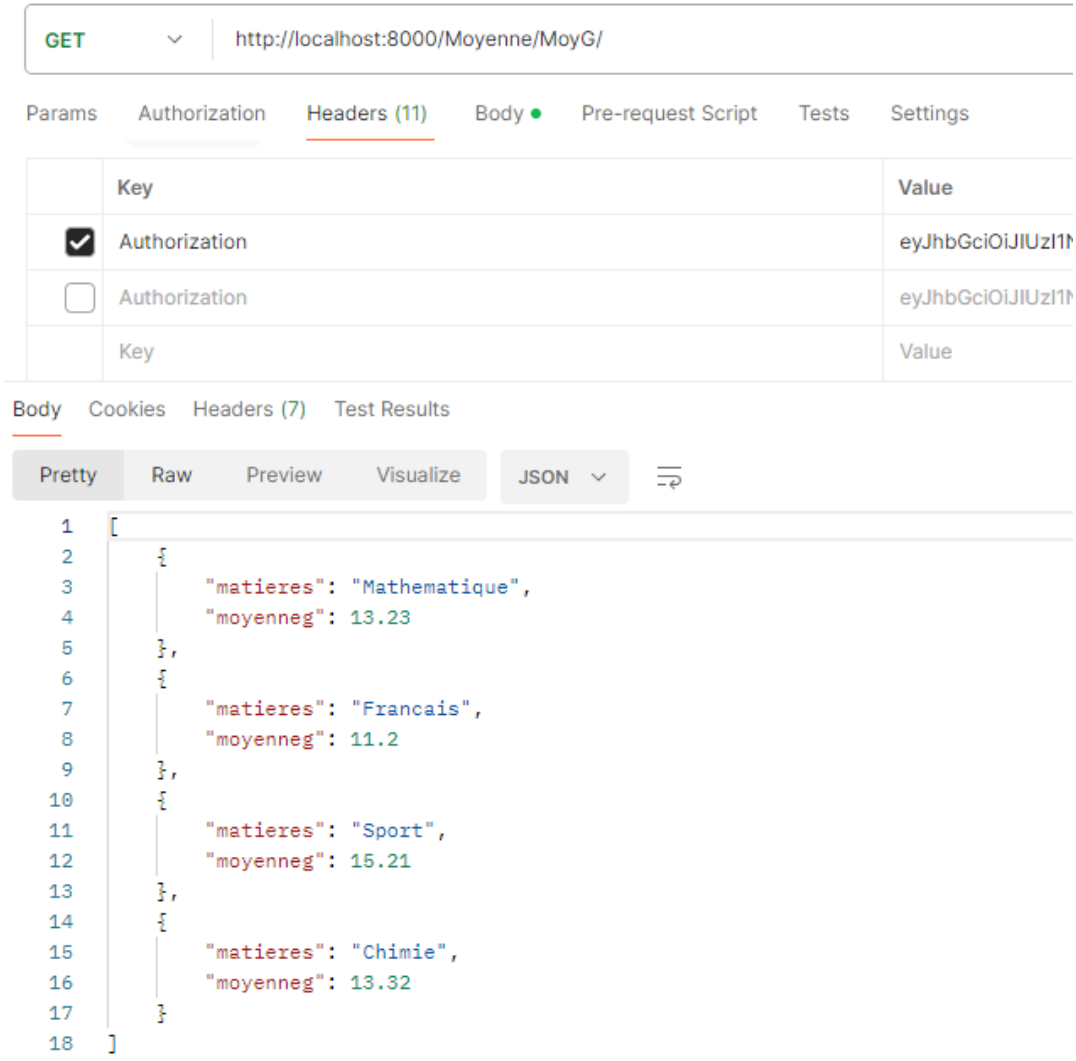
On peut constater que la moyenne d'anglais a bien été modifier.

Le principe est exactement le même avec POST et DELETE.

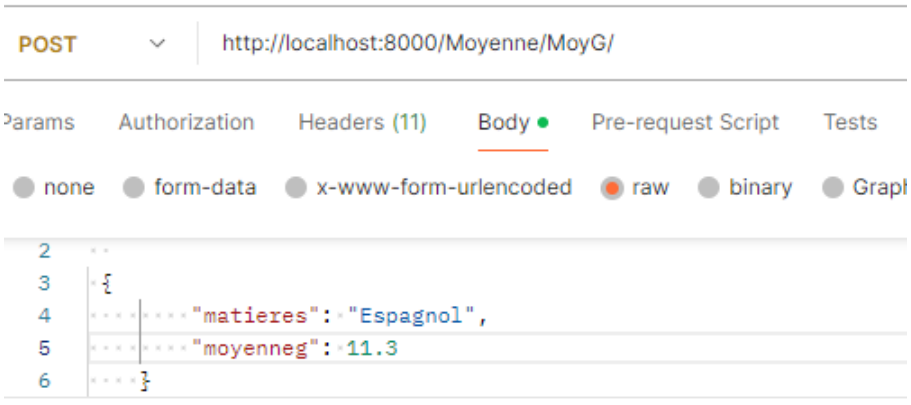
Pour DELETE, on décide de supprimer la valeur anglais :

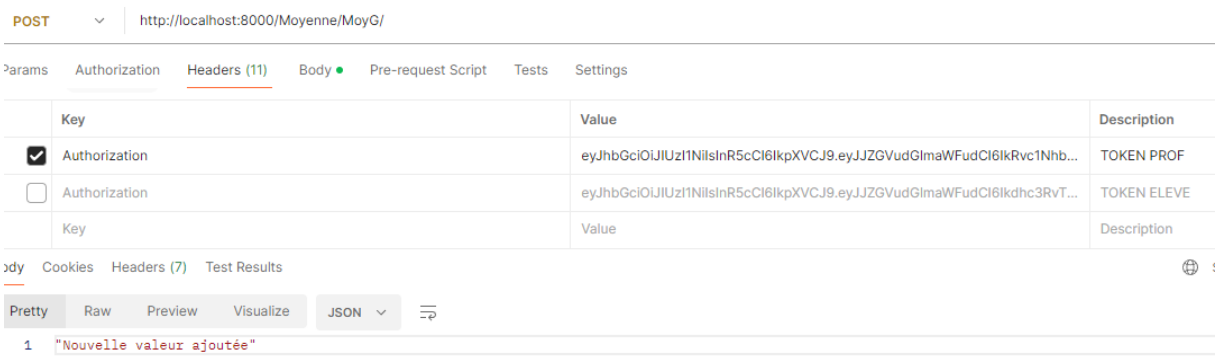


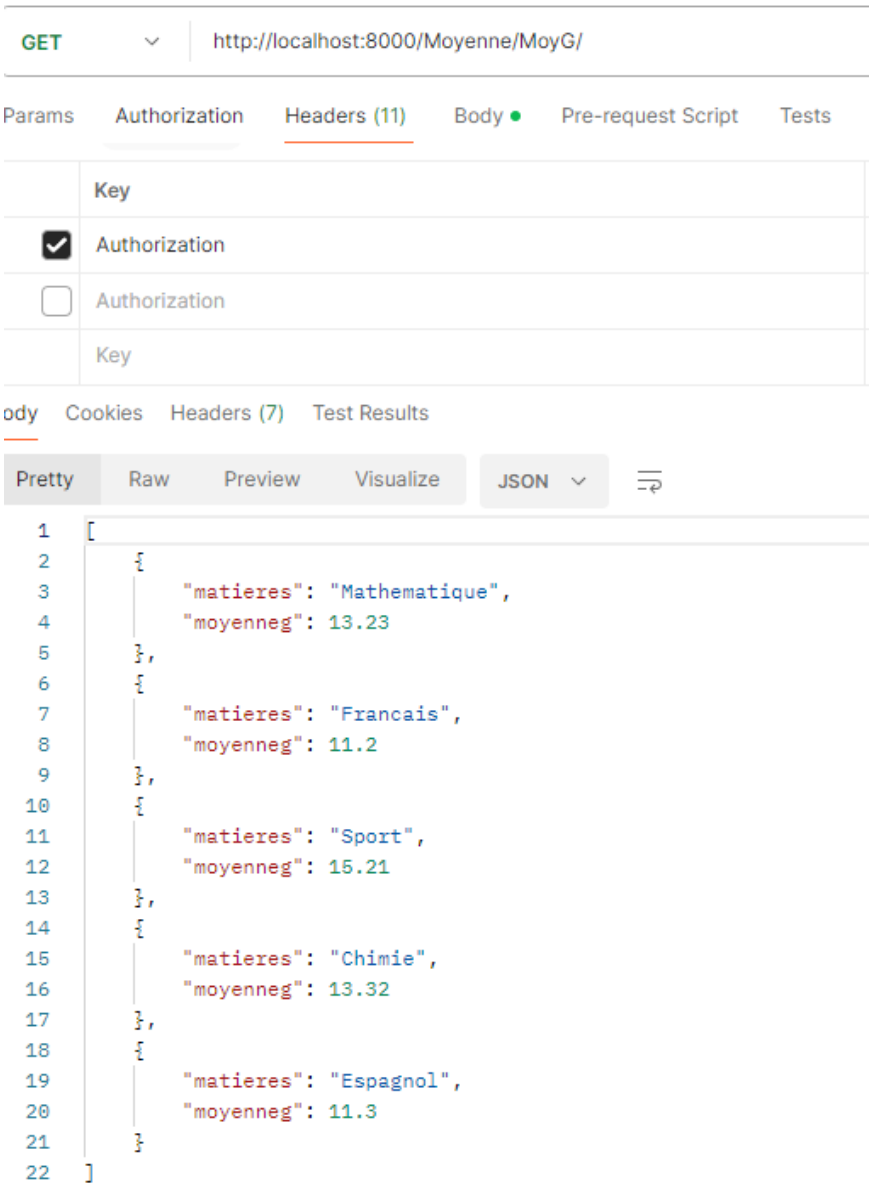
On constate que l'anglais a bien été supprimé :



Pour POST, on décide d'ajouter la valeur Espagnol :







La valeur “Espagnol” et sa “moyenneg” ont bien été ajouté.

4. Dossier databases

On retrouve dans mes fichiers de procédure stockées mes différentes requête SQL utilisées dans mes Controller. Dans le fichier classeProcedure on retrouve toutes les requêtes en relation avec la table classe et dans le fichier moyenneProcedure toutes les requêtes en relation avec la table moyenneg. Dans ce dossier databases on peut bien évidement retrouver notre fichier database qui permet de nous connecter à notre base de données (A SAVOIR QUE LES VARIABLES DE CONNEXION SONT EN CLAIR CAR JE NE PEUX PAS FAIRE AUTREMENT DU A UN BUG DE MON VS, J'AI TOUT DE MÊME RAJOUTE LES VARIABLES ENVIRONNEMENT AU FICHIER .ENV) et enfin notre base de données sous format sql pour que vous puissiez l'exploiter pour la correction.

5. Dossier middleware et routes

On a trois middlewares :

- MiddlewareAdmin stock la fonction isAdmin qui permet de récupérer le token ainsi que l'identifiant présent dedans avec les paramètres de requête ou les en-tête de requête et ensuite de le valider. Puis il utilise l'identifiant récupérer dans le token pour vérifier via une requête sql si l'identifiant est associé à un statut de prof/admin.
- MiddlewareAuth authentifie un individu via un token devant posséder la clé API.
- MiddlewareValeurEntrante permettra de vérifier si lors de la saisie du login et du mp les règles instaurées sont bien respectées.

On va introduire ensemble un middlewareAuth pour voir les étapes de sa mise en place. On va l'ajouter à l'ensemble des routes présentes dans le fichier moyenneRoute. Si un individu veut accéder à ces routes il devra obligatoirement détenir un token contenant la clé API présente dans notre.env.

Maintenant seul les élèves et le prof peuvent accéder à l'ensemble des matières (car ils disposent d'un token avec la bonne clé api) et uniquement le prof peut modifier, ajouter et supprimer une matière (car il dispose d'un token avec la bonne clé api et un identifiant associé au statut de prof).

Pour plus de sécurité on va aussi rajouter un middlewareAdmin à la route GET présente dans le fichier classeRoute. Cela va permettre uniquement au professeur d'accéder aux informations confidentielles de l'ensemble de la classe (identifiant, immatriculation, password...).

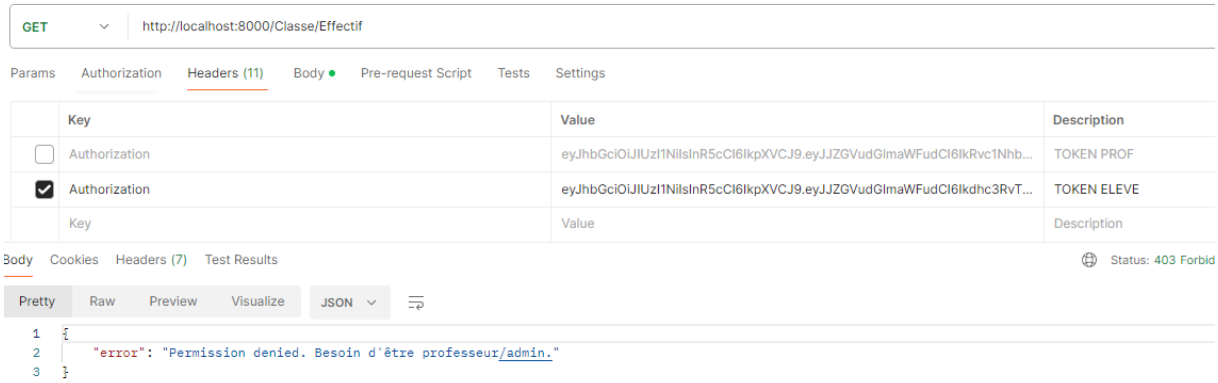
Fichier moyenneRoute

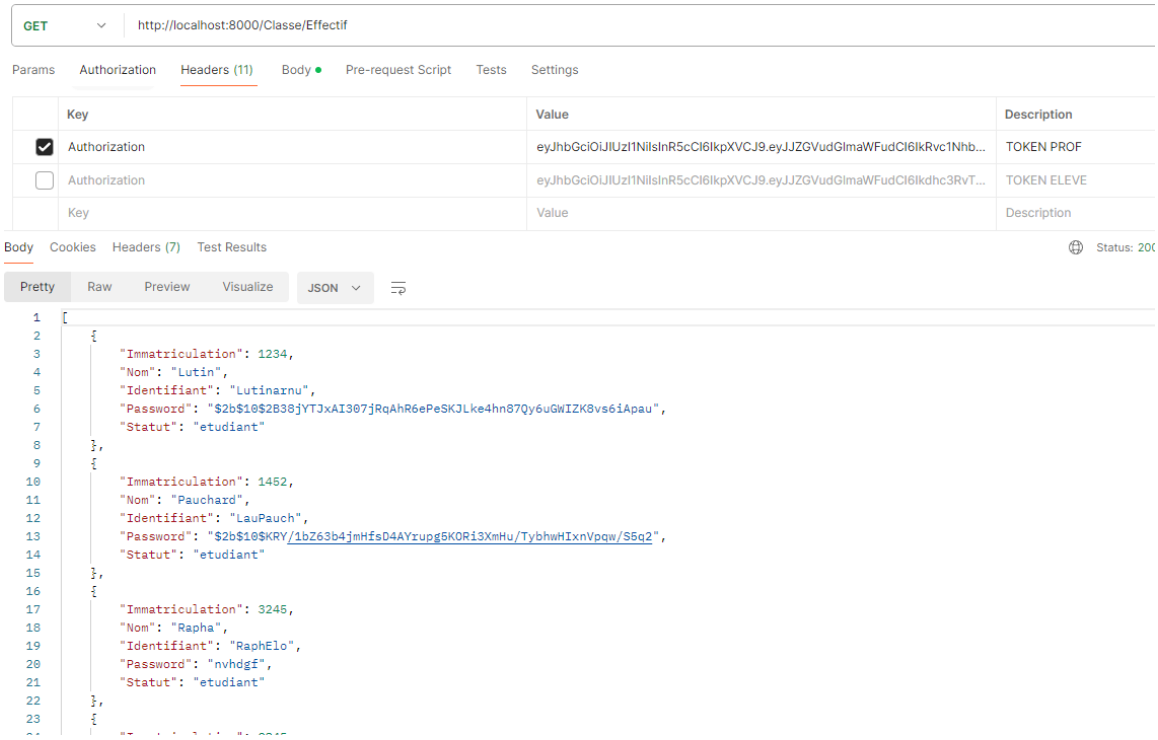
```
router.get("/MoyG",middlewareAuth.authentificator, moyenneController.getMoyG);

//On route vers le moyenneController et plus précisement la fonction
getMoyGBySubject lorsqu'on tappe /Moyenne/MoyG/"Nom_matière" et on effectue une
oppération get avant de router il faut passer par les différents middleware
router.get("/MoyG/:matieres",middlewareAuth.authentificator, moyenneController.
getMoyGBySubject);

//On route vers le moyenneController et plus précisement la fonction putMoyG
lorsqu'on tappe /Moyenne/MoyG/"Nom_matière" et on effectue une oppération put
//avant de router il faut passer par les différents middleware
router.put("/MoyG/:matieres",middlewareAuth.authentificator, middlewareAdmin.
isAdmin, moyenneController.putMoyG);

//On route vers le moyenneController et plus précisement la fonction
deleteMatiere lorsqu'on tappe /Moyenne/MoyG/"Nom_matière" et on effectue une
oppération delete avant de router il faut passer par les différents middleware
router.delete("/MoyG/:matieres",middlewareAuth.authentificator, middlewareAdmin.
isAdmin, moyenneController.deleteMatiere);

//On route vers le moyenneController et plus précisement la fonction
PostMatiere lorsqu'on tappe /Moyenne/MoyG/ et on effectue une oppération post
avant de router il faut passer par les différents middleware
router.post("/MoyG/",middlewareAuth.authentificator, middlewareAdmin.isAdmin,
moyenneController.postMatiere);
```

Fichier classeroute

```
const express = require("express"); // On va chercher le module express
const router = express.Router(); // On import le module router pour permettre
de router vers les controller
const classeController = require('../controllers/classeController');
//On importe le fichier classecontroller
const middlewareAdmin = require("../middleware/middlewareAdmin")

const { ValeurEntrante, regleLogin } = require('../middleware/
middlewareValeurEntrante');


//On route vers le classecontroller et plus précisement la fonction
getAllClasse. La présence du middlewareAdmin prouve qu'uniquement l'admin/prof
peut accéder à cette route
router.get('/Effectif',middlewareAdmin.isAdmin, classeController.getAllclasse);

//On route vers le classecontroller et plus précisement la fonction putlogin.
La présence des fonctions du middlewareValeurEntrante prouve que certaines
règles doivent être respecté lors de la saisie de l'identifiant et du mp
router.put('/Login', regleLogin(), ValeurEntrante, classeController.putlogin);

module.exports = router;
```

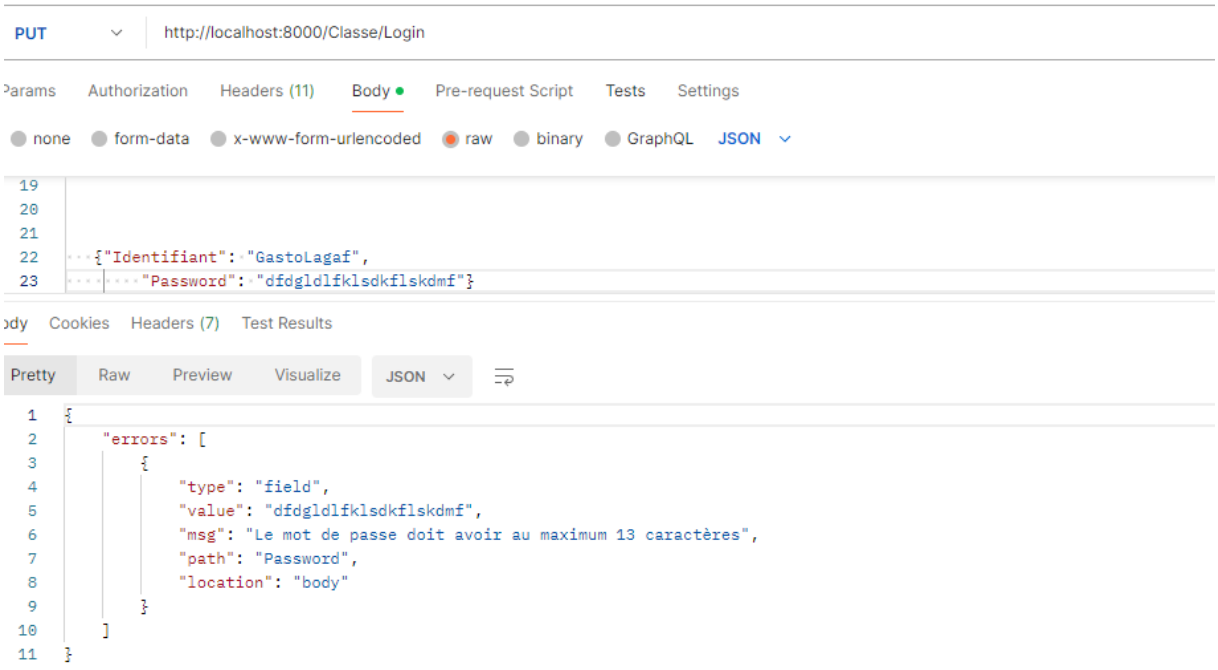
On constate que les élèves ne peuvent plus accéder à l'effectif de la classe contrairement au professeur :





On peut aussi tester si notre middlewareValeurEntrante fonctionne correctement :

Il marche correctement une erreur indiquant que le mp doit avoir au maximum 13 carractères a été renvoyé.



6. Fichier server.js, package.json et .env

Dans le fichier server.js on retrouve les différents modules permettant le traitement de données comme expresse.json ou encore bodyParser. On a aussi la méthode use qui permet de gérer les requêtes en destination d'un chemin d'URL. Et enfin on met sous écoute le serveur avec la méthode listen.

D'un point de vue sécurité il est primordiale de placer les données sensibles dans un **fichier .env,** cela est plus qu'important car elle permet de garder ces données en dehors du codes sources. On peut y retrouver par exemple les données permettant l'accès à la database ou encore la clé API

Et pour finir dans le fichier package.json on peut retrouver les liens vers les procédures stockées mais aussi nos différentes dépendances installées .