

# Réalisation d'une API RESTful connectée à MongoDB Atlas

REALISE PAR

- René Lothaire BAZIE

SOUS LA SUPERVISION DE

M. PREIRA



Année universitaire : 2021/2022

CYCLE INGENIEUR DE CONCEPTION – NoSQL

# **PLAN**

## INTRODUCTION

- I.) Choix et description du jeu de données
- II.) Choix du framework
- III.) Description de la procédure déploiement de la base de données sur  
Mongo Atlas
- IV.) Liste des requêtes d'interrogation simples et analytiques proposées  
dans l'API
- V.) Présentation et explication des codes source écrits
- VI.) La documentation de l'API

## CONCLUSION

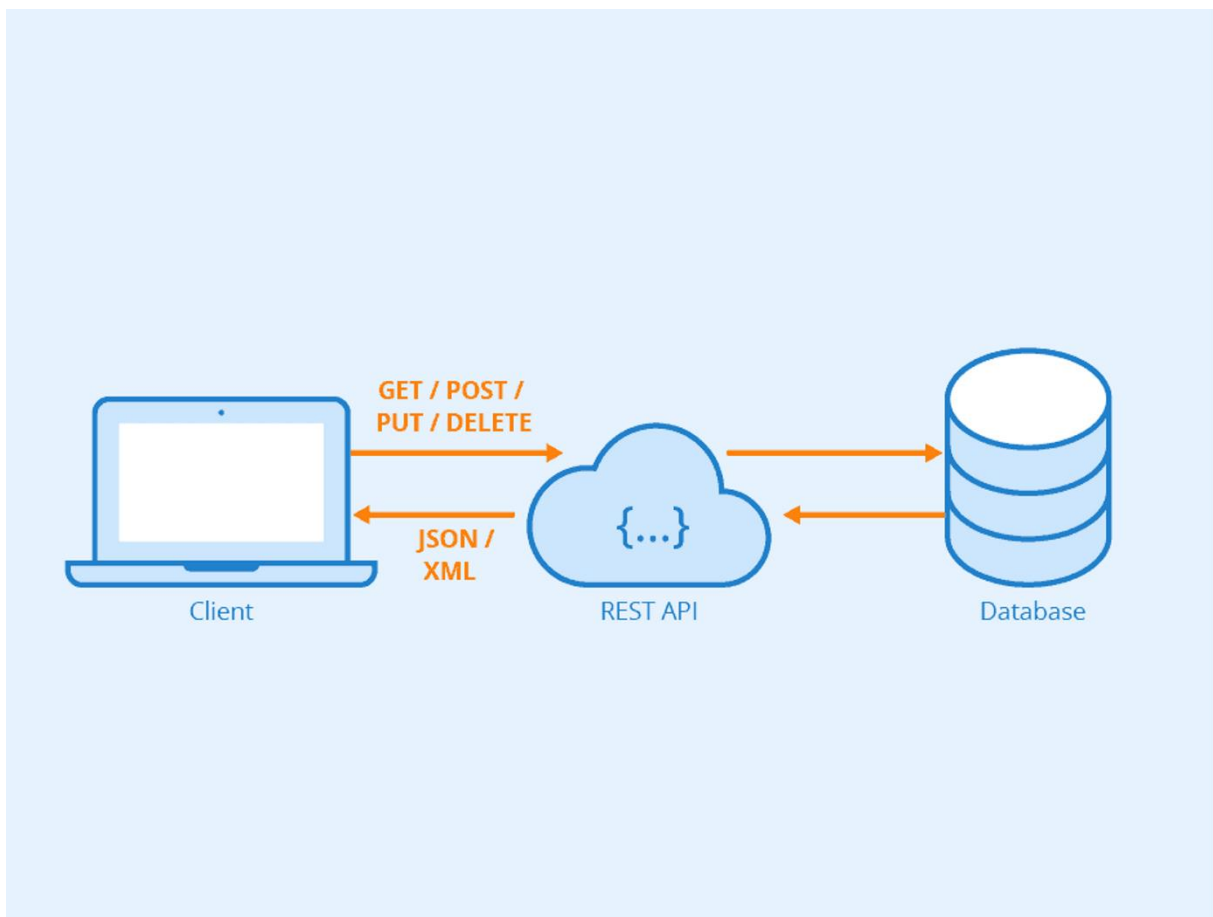
### **Table des matières**

INTRODUCTION.....	3
I.) Choix et description du jeu de données .....	4
II. Choix du framework .....	5
III.) Description de la procédure déploiement de la base de données sur Mongo Atlas .....	6
IV.) Liste des requêtes d'interrogation simples et analytiques proposées dans l'API .....	12
V.) Présentation et explication des codes source écrits .....	14
V.1.) La partie des « importations » .....	14
V.2.) La partie « création de l'objet » .....	15
V.3.) La partie « connexion à la base de données » .....	15
V.4.) La partie concernant « la méthode GET » .....	15
V.5.) La partie concernant la méthode POST .....	17
V.6.) La partie concernant la méthode DELETE .....	18
V.7.) La partie concernant la méthode PUT .....	20
V.8.) La partie « débogage et exécution de l'application » .....	21
VI.) La documentation de l'API .....	21
VI.1) Les applications à prendre en compte .....	22
VI.2) Pycharm .....	22
VI.3) POSTMAN .....	28
CONCLUSION .....	34

## INTRODUCTION

API est un acronyme pour “Application Programming Interface” ou Interface de programmation d’application en français. Il s’agit d’une interface permettant l’interaction entre différentes applications. Elle définit quels appels ou requêtes peuvent être réalisés et comment les réaliser : le format des données à utiliser, la structure de la réponse, les conventions à respecter etc.

Les APIs sont à la base de toutes les interactions entre différentes applications. De très nombreuses entreprises et organisations proposent aujourd’hui des APIs pour interagir avec leurs applications. Cela permet ainsi aux développeurs d’applications tierces de réaliser des opérations comme transmettre ou accéder à des données d’une application à une autre via cette API.



Une API REST permet de manipuler de la donnée via une interface applicative

## I.) Choix et description du jeu de données

### I.1.) Choix du jeu de données

Notre choix de jeu de données s'est porté sur le classement universitaire : `universities_ranking.json`.

En effet, Les données présentées sous les statistiques clés sont celles fournies par l'université elle-même dans sa soumission au classement mondial des universités du Times Higher Education. Il représente les données de l'année universitaire 2019/20 et peut varier des années suivantes ou antérieures.



### I.2.) Description du jeu de données

Notre jeu de données `universities_ranking.json` est composé de plusieurs documents dont :

- Le rang
- Nom/Titre
- Pays/Région
- Nombre d'étudiants ETP

Il s'agit du nombre d'étudiants équivalents temps plein à l'université.

- Nombre d'étudiants par personnel

Il s'agit du rapport entre le nombre d'étudiants équivalents temps plein et le nombre de membres du personnel académique ; ceux impliqués dans l'enseignement ou la recherche.

- % d'étudiants internationaux

Le pourcentage d'étudiants provenant de l'extérieur du pays de l'université.

- Ratio femmes/hommes : le ratio d'étudiants femmes/hommes à l'université.

## II. Choix du framework

Il existe plusieurs Frameworks basés sur de nombreux langages de programmation vers lesquels nous pouvons nous orienter. Il y a par exemple, Spring de Java, Express.js basé sur Node.js, Ruby on Rails de Ruby ou encore Flask de Python.

Dans cette chronique, nous allons utiliser Python avec comme Framework, Flask.

### *Pourquoi Flask ?*

- Flask est un cadre de travail (framework) Web pour Python. Ainsi, il fournit des fonctionnalités permettant de construire des applications Web, ce qui inclut la gestion des requêtes HTTP et des canevas de présentation.
- Les applications Flask sont construites à partir de canevas très simples et sont donc plus adaptées au prototypage d'APIs.
- Le Framework est flexible
- Son déploiement est rapide et la phase de création peut commencer quasi instantanément

### **III.) Description de la procédure déploiement de la base de données sur Mongo Atlas**

MongoDB Atlas est la solution cloud de Database as a Service (DBaaS). Atlas nous déploie un serveur MongoDB managé sur un cloud Amazon Web Services, Google Cloud Platform ou Microsoft Azure, dans la région de notre choix. Nous aurons le choix de la taille de notre cluster tout en ayant l'avantage d'avoir notre base de données managée par l'équipe d'ingénieurs de MongoDB.

MongoDB Atlas inclut toutes les fonctionnalités qui ont fait de MongoDB la meilleure base de données NoSQL. Il intègre des optimisations et les meilleures pratiques automatisées et perfectionnées acquises grâce à la gestion et l'optimisation de milliers de déploiements MongoDB.

Cela nous permet de garantir les performances, la disponibilité et la sécurité de nos déploiements MongoDB sur AWS. MongoDB Atlas nous permet de moins nous soucier de la gestion de notre base de données et de vous concentrer davantage sur la création de notre application.

Tels sont les bénéfices de MongoDB Atlas.

Le déploiement de la base de données sur Mongo Atlas s'effectue suivant plusieurs étapes qui seront énoncées dans les lignes suivantes :

## Etape 1 : Création d'un cluster partagé

[CLUSTERS](#) > [CREATE A SHARED CLUSTER](#)

### Create a Shared Cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).

**PREVIEW** Serverless


Dedicated


**FREE** Shared


## Etape 2 : Choix du fournisseur du cloud, de la région, du stockage et du nom du cluster

Cloud Provider & Region

AWS, Paris (eu-west-3) ▼










★ Recommended region ⓘ


🏷️ Paid tier region ⓘ


NORTH AMERICA

 N. Virginia (us-east-1) ★


 Oregon (us-west-2) ★


 Ohio (us-east-2) ★ 🏷️


 N. California (us-west-1) 🏷️


 Montreal (ca-central-1) 🏷️


EUROPE

 Frankfurt (eu-central-1) ★


 Stockholm (eu-north-1) ★

 Ireland (eu-west-1) ★


 Paris (eu-west-3) ★


 London (eu-west-2) ★ 🏷️


AUSTRALIA

 Sydney (ap-southeast-2) ★

ASIA

 Tokyo (ap-northeast-1)

 Singapore (ap-southeast-1) ★

 Hong Kong (ap-east-1) ★

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted ^

Additional Settings

MongoDB 5.0, No Backup ^

Cluster Name

Cluster0 ^

**FREE**

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

[Back](#)

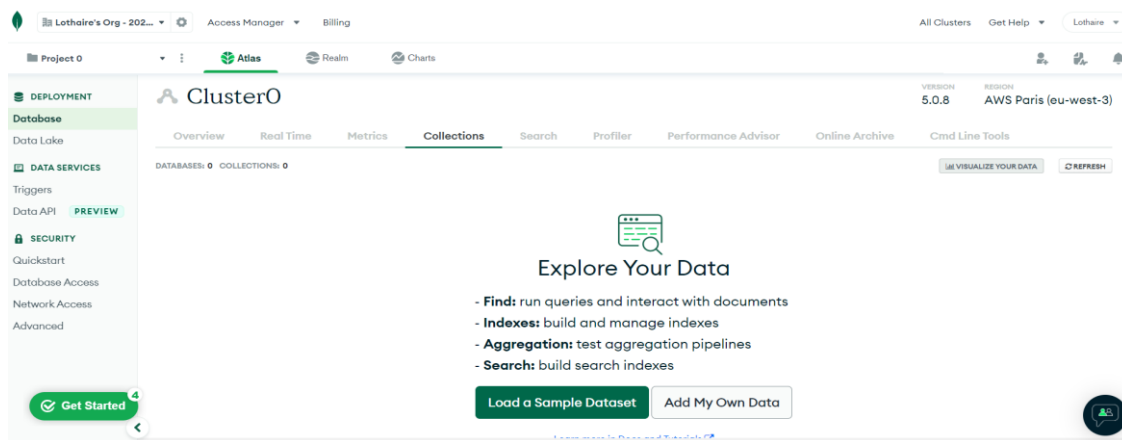
Create Cluster

CYCLE INGENIEUR DE CONCEPTION – NoSQL

7

Puis on clique sur « create cluster »

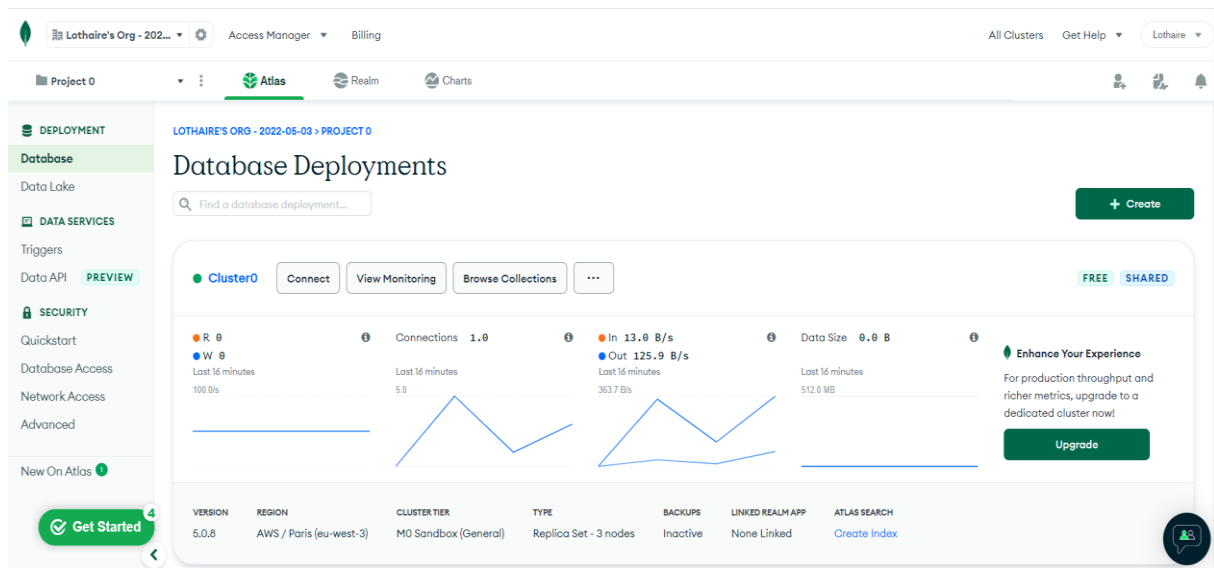
On se retrouvera avec l'image de la fenêtre suivante :



**Etape 3** : Création de la base de données ; elle se fait globalement en deux sous-étapes :

- L'insertion du nom pour notre database
- L'insertion du nom pour notre collection se trouvant dans la base de données





**Etape 4** : puis vient la partie « setup connection security »

Nous avons besoin de sécuriser notre cluster sur mongoDB Atlas avant de l'utiliser Cette procédure se fait en deux temps :

- L'ajout d'une adresse IP de connexion
- La création d'un utilisateur de base de données avec bien sûr un mot de passe

Ci-dessous, les captures d'écran des deux parties ;

#### 1 Add a connection IP address

**IP Address**

**Description (Optional)**

#### 2 Create a Database User

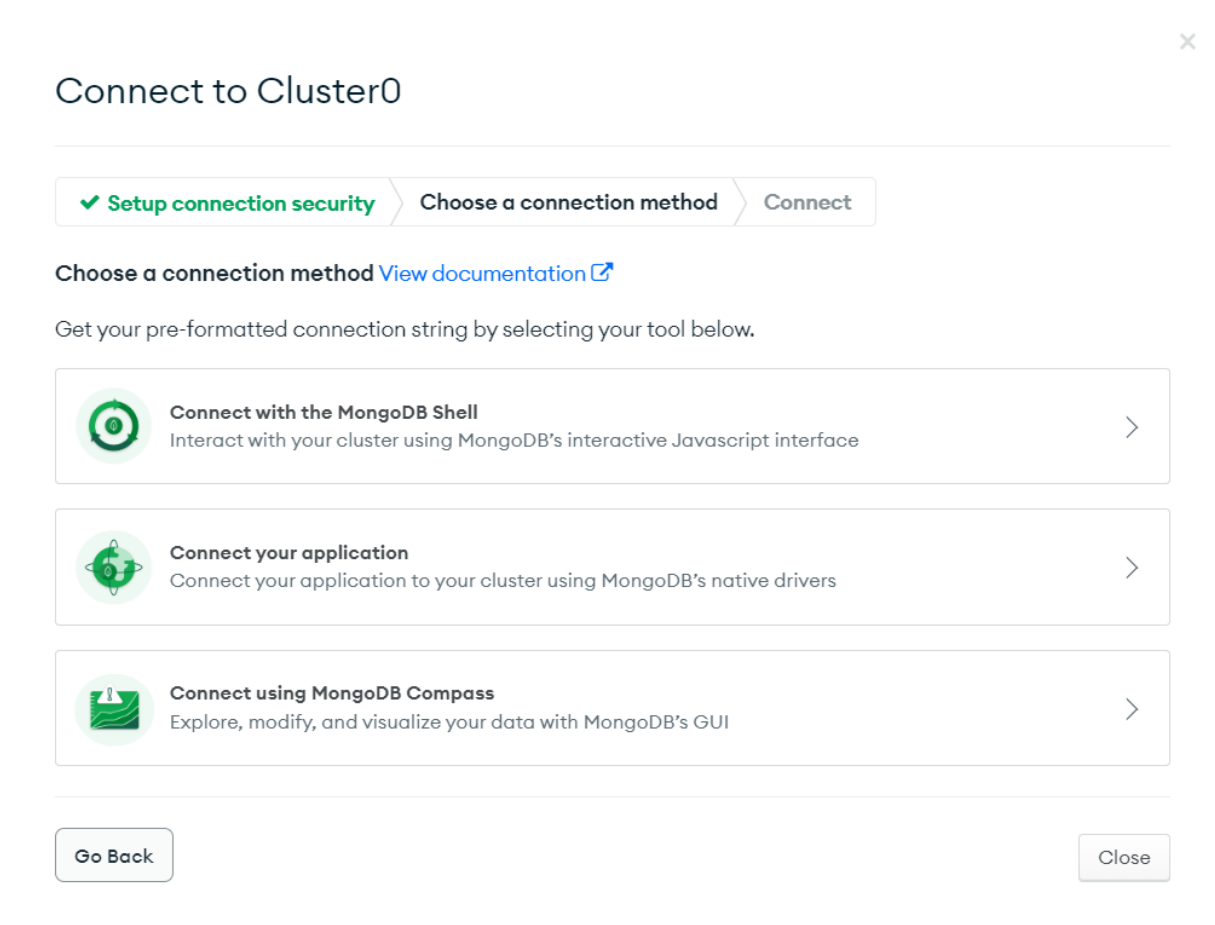
This first user will have **atlasAdmin** permissions for this project.

Keep your credentials handy, you'll need them for the next step.

**Username**

**Password**

## Etape 5 : Le choix de la méthode de connexion



## Etape 6 : La phase finale consistant en la connexion à notre cluster

Elle contient deux parties à savoir :

- La sélection du driver et de la version
- L'ajout de la chaine de connexion dans le code d'application

La capture d'écran dans la page suivante ;

## Connect to Cluster0

✓ Setup connection security

✓ Choose a connection method

Connect

### 1 Select your driver and version

DRIVER

Python

VERSION

3.6 or later

### 2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://Rene:<password>@cluster0.m0c58.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **Rene** user. Replace **myFirstDatabase** with the name of the database that connections will use by default. Ensure any option params are [URL encoded](#).

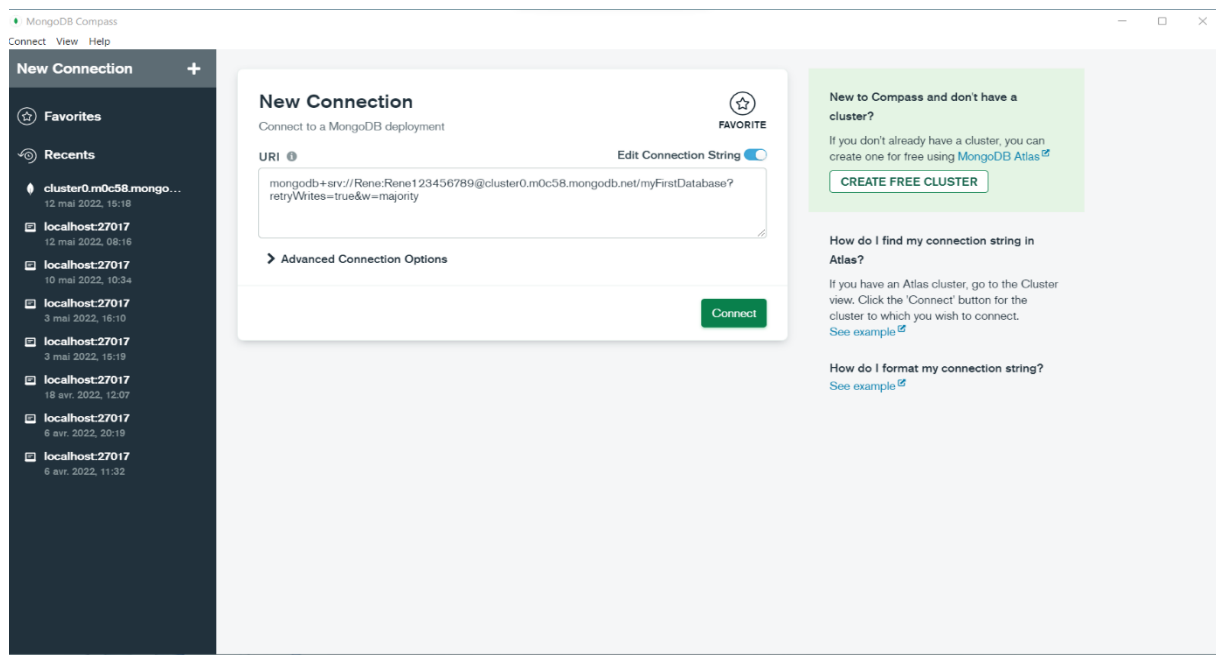
Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

Close

## Etape 7 : Connexion à mongoDB Compass

Nous avons la possibilité de connecter notre base de données à mongoDB Compass en entrant l'URI de notre cluster ;



## IV.) Liste des requêtes d'interrogation simples et analytiques proposées dans l'API

- La méthodologie la plus répandue de conception des API s'appelle REST.
- L'aspect le plus important de REST est qu'elle est basée sur quatre méthodes définies par le protocole HTTP : GET, POST, PUT et DELETE.
- Celles-ci correspondent aux quatre opérations standard effectuées sur une base de données : READ, CREATE, UPDATE et DELETE.

### Description des méthodes

Les méthodes de ressources correspondent aux différents types de traitements que nous pouvons effectuer lors d'une requête API REST. Il existe principalement 4 méthodes courantes, qui correspondent aux méthodes CRUD, à savoir :

POST : la méthode qui sert à publier, envoyer des informations au serveur ;

GET : celle pour récupérer des informations venant du serveur. Elle peut retourner plusieurs formats de réponses, comme nous l'avons souvent répété au cours de cette chronique ;

PUT : il s'agit de la méthode à utiliser lorsque l'on effectue une mise à jour auprès d'un serveur ;

DELETE : qui, comme son nom l'indique permet de supprimer une entrée sur un serveur.

Cependant, il existe d'autres méthodes que l'on peut utiliser lors d'un appel d'une API REST comme PATCH, HEAD (l'équivalent du "Test d'existence"), OPTIONS (l'équivalent de "Lister les commandes disponibles"), TRACE et CONNECT.

## **Liste des requêtes**

Nous vous proposons les requêtes http que nous avons créé et utilisé :

- <http://127.0.0.1:5000/>
- [http://127.0.0.1:5000/GET/number\\_usa\\_universities](http://127.0.0.1:5000/GET/number_usa_universities)
- [http://127.0.0.1:5000/GET/number\\_universities\\_students](http://127.0.0.1:5000/GET/number_universities_students)
- <http://127.0.0.1:5000/GET/universitiespercountries>
- [http://127.0.0.1:5000/GET/number\\_strange\\_students](http://127.0.0.1:5000/GET/number_strange_students)
- <http://127.0.0.1:5000/add>
- [http://127.0.0.1:5000/delete\\_universities](http://127.0.0.1:5000/delete_universities)
- [http://127.0.0.1:5000/update\\_universities](http://127.0.0.1:5000/update_universities)

## V.) Présentation et explication des codes source écrits

Dans cette partie, nous présenterons notre code source écrit en python avec bien sûr l'aide de l'outil Flask.

Nous subdivisons notre code source principalement en huit grandes parties :

- La partie des « importations »
- La partie « création de l'objet »
- La partie « connexion à la base de données »
- La partie « concernant la méthode GET »
- La partie « concernant la méthode POST »
- La partie « concernant la méthode DELETE »
- La partie « concernant la méthode PUT »
- La partie « débogage et exécution de l'application »

### V.1.) La partie des « importations »

```
1  from bson import json_util
2
3  from flask import Flask
4  from flask import request
5  from flask import jsonify
6
7  from pymongo import MongoClient
8
9  import json
10
```

Nous avons donc importé principalement 6 packages qui nous ont été d'ailleurs d'une importance capitale.

Bson → Binary json

Flask → Pour envoyer des requêtes HTTP à des fonctions Python

Request → Pour effectuer des requêtes HTTP

Jsonify → Pour retourner des données json

Pymongo → permet d'utiliser trois types d'objets via votre IDE python : les clients, les bases de données et les collections.

MongoClient → qui nous aide à nous connecter à notre cluster

## V.2.) La partie « création de l'objet »

```
19 |  
20 | app = Flask(__name__)  
21 |  
22 |
```

Il s'agit là de la créer l'objet application Flask. Cela est nécessaire pour pouvoir faire appel aux méthodes.

## V.3.) La partie « connexion à la base de données »

```
22 |  
23 | cluster = MongoClient("mongodb+srv://Rene:Rene123456789@cluster0.m0c58.mongodb.net/myFirstDatabase?retryWrites=true&w=majority")  
24 |  
25 |  
26 | db = cluster["universities_db"]  
27 | collection = db["universities_ranking"]  
28 |
```

Dans cette partie, nous avons d'abord fourni l'url de l'atlas MongoDB pour connecter python à MongoDB.

Sur la ligne 26 il s'agit du lien vers notre base de données universities\_db et sur la ligne 27 nous avons la connexion à notre collection sous format json.

## V.4.) La partie concernant « la méthode GET »

Pour cette méthode, nous avons créé 5 routes, chacune faisant référence à une fonction qui retourne des éléments spécifiques.

- 1<sup>re</sup> route

```
69 | @app.route('/', methods=['GET'])  
70 | def home():  
71 |     return '''<p><h2 align = "center" color = "red" > <font color = "blue">BIENVENU SUR LA PLATEFORME DEVELOPPE PAR </font> </p></h2>  
72 | <p><h3 align = "center">Dinin René Lothaire BAZIE</h3></p>  
73 | '''  
74 |  
75 |
```

Dans l'image ci-dessus, l'instruction indique à Flask que la fonction home correspond au chemin /. Elle retourne un texte avec la structuration suivante :



- 2<sup>ème</sup> route

```
@app.route('/GET/usa_universities', methods=['GET'])
def getUniversities():
    USA_universities = list(collection.find({"location": "United States"}))
    return "LES UNIVERSITES AMERICAINES : \n \n " + json.dumps(USA_universities, default=json_util.default)
```

En ce qui concerne cette route, l'instruction indique à Flask que la fonction getUniversities correspond au chemin /GET/usa\_universities.

Cette requête retourne toutes les universités américaines.

La fonction json.dumps() convertit un objet Python en une chaîne json. Il est similaire au dictionnaire en python.

- 3<sup>ème</sup> route

```
@app.route('/GET/number_universities_students', methods=['GET'])
def getUniversitiesNumberStudent():
    USA_universities_Number = collection.count_documents({"number students": {"$gte": "20,000"}})
    return "NOMBRE D'UNIVERSITES AVEC PLUS DE 20000 ETUDIANTS : \n \n " + \
        json.dumps(USA_universities_Number, default=json_util.default)
```

Dans cette partie, l'instruction indique à Flask que la fonction getUniversitiesNumberStudent correspond au chemin /GET/number\_universities\_students.

La fonction recherche donc les universités où le nombre d'étudiants est supérieur à 20000 et nous retourne le nombre.



- 4<sup>ème</sup> route

```
@app.route('/GET/number_strange_students', methods=['GET'])
def StrangeStudents():
    strange_students = collection.count_documents({"perc_intl_students":{"$gte":"75%"}})
    return "NOMBRE D'UNIVERSITES OU LES 3/4 DES ETUDIANTS SONT DES ETRANGERS: \n \n " \
        + json.dumps(strange_students, default=json_util.default)
```

Ici nous avons une route qui grâce une fonction nommée StrangeStudents(), compte le nombre d'universités où les  $\frac{3}{4}$  des étudiants sont des internationaux (étrangers).

- 5<sup>ème</sup> route

```
@app.route('/GET/universitiespercountries', methods=['GET'])
def universitiespercountries():
    u = list(collection.aggregate([
        {"$group":{"_id":"$location", "nombre": {"$sum": 1}}},
        {"$sort": {"nombre": -1}}
    ]))
    return "NOMBRE D'UNIVERSITES PAR PAYS ET PAR ODRE DECROISSANT : \n \n " \
        + json.dumps(u, default=json_util.default)
```

Ici la fonction universitiespercountries() correspond au chemin suivant : /GET/universitiespercountries.

Là il s'agit d'une agrégation ; notre fonction retourne le nombre d'universités par pays et par ordre décroissant.

## V.5.) La partie concernant la méthode POST

Cette méthode sert à publier, envoyer des informations à la base de données.

```
96     @app.route('/add', methods=['POST'])
97
98     def api_response():
99
100         if request.method == 'POST':
101             message = add_university()
102             return message
103
```

```
def add_university():
    new = json.loads(request.data)
    ranking = new.get("ranking")
    title = new.get("title")
    location = new.get("location")
    number_students = new.get("number_students")
    students_staff_ratio = new.get("students_staff_ratio")
    perc_intl_students = new.get("perc_intl_students")
    gender_ratio = new.get("gender_ratio")

    condition = collection.find_one({"ranking": ranking})

    if condition:
        return jsonify({"message": "This university is already mentioned"})
    else:
        collection.insert_one({"ranking": ranking, "title": title, "location": location, "number_students": number_students,
                                "students_staff_ratio": students_staff_ratio, "perc_intl_students": perc_intl_students, "gender_ratio": gender_ratio})
    return jsonify({'msg': "University successfully ADDED !"})
```

L'instruction indique à Flask que la fonction `add_university` correspond au chemin `/add`.

- La première fonction fait une vérification de la méthode grâce à la structure de contrôle « if ». Si la méthode correspondante est POST alors on effectue la deuxième fonction dont nous parlerons dans le second tiret.

- La 2<sup>ème</sup> fonction, elle, est celle qui représente véritablement l'utilisation de POST. Grâce à `json.loads()` qui prend un objet fichier et renvoie l'objet json, nous avons pu entrer les différents champs de notre jeu de données. Une condition a été introduite vérifiant si le champ « ranking » que nous avons entré existe déjà ou non dans le jeu de données : si oui, un message s'affichera indiquant que l'université existe déjà ; sinon le document créé sera inséré grâce bien sûr à la fonction `insert_one` avec un message indiquant que le document a bien et bel été ajoutée.

## V.6.) La partie concernant la méthode DELETE

La méthode DELETE comme son nom l'indique, permet supprimer une entrée.

```

129
130     @app.route('/delete_universities', methods=['DELETE'])
131
132     def api_delete():
133         if request.method == 'DELETE':
134             message = remove()
135             return message
136

```

```

def remove():

    new = json.loads(request.data)
    ranking = new.get("ranking")
    title = new.get("title")
    location = new.get("location")
    number_students = new.get("number_students")
    students_staff_ratio = new.get("students_staff_ratio")
    perc_intl_students = new.get("perc_intl_students")
    gender_ratio = new.get("gender_ratio")

    condition = collection.find_one({"ranking": ranking})

    if condition:
        collection.delete_one(
            {"ranking": ranking, "title": title, "location": location, "number_students": number_students,
             "students staff ratio": students_staff_ratio, "perc intl students": perc_intl_students,
             "gender ratio": gender_ratio})
        return jsonify({'msg': "university successfully REMOVED !"})
    else :
        return jsonify({"message": "this ranking doesn't exist in the database"})

```

L'instruction indique à Flask que la fonction remove correspond au chemin /delete\_universities.

Tout comme dans la méthode POST, il y'a deux fonctions :

- La 1<sup>re</sup> qui vérifie que la méthode correspondante est DELETE avant de passer à la seconde fonction.

- La 2<sup>ème</sup> fonction qui utilise également json.loads afin de pouvoir entrer les champs. Le changement se produit au niveau de la structure de contrôle.

Si la valeur du champ « ranking » du document entré existe, alors le document sera supprimé grâce à la méthode delete\_one avec le message suivant : « university successfully removed ».

Sinon, un autre message apparait indiquant que le « ranking » entré n'existe pas.

## V.7.) La partie concernant la méthode PUT

```
@app.route('/update_universities', methods=['PUT'])  
  
def api_update():  
    if request.method == 'PUT':  
        message = update()  
        return message
```

```
def update():  
  
    new = json.loads(request.data)  
    ranking = new.get("ranking")  
    title = new.get("title")  
    location = new.get("location")  
    number_students = new.get("number_students")  
    students_staff_ratio = new.get("students_staff_ratio")  
    perc_intl_students = new.get("perc_intl_students")  
    gender_ratio = new.get("gender_ratio")  
  
    condition = collection.find_one({"ranking": ranking})  
  
    if condition:  
        #return jsonify(collection)  
        collection.update_one(  
            {"ranking": ranking}, {"$set": {"title": title, "location": location, "number_students": number_students,  
                "students staff ratio": students_staff_ratio,  
                "perc intl students": perc_intl_students,  
                "gender ratio": gender_ratio}}, upsert=False)  
        return jsonify({'msg': "university successfully UPDATED !"})  
  
    else :  
        return jsonify({'msg': "University does not exist !!"})
```

L'instruction indique à Flask que la fonction update correspond au chemin /delete\_universities.

Il y'a aussi deux fonctions :

La 1<sup>re</sup> étant similaire aux autres fonctions dans les méthodes POST et DELETE

La 2<sup>ème</sup> qui utilise également json.loads afin de pouvoir entrer les champs.

Si le champ « ranking » du document entré existe, alors le document sera mis à jour grâce à la méthode update\_one avec le message suivant : « university successfully updated ».

Sinon, un autre message apparait indiquant que le document entré n'existe pas.

## V.8.) La partie « débogage et exécution de l'application »

```
235
236     app.config["DEBUG"] = True
237
238 ►     if __name__ == "__main__":
239         app.run()
240
```

A la ligne 236 nous avons le débogage, permettant d'avoir une description plus précise des erreurs et aux lignes 238 et 239, nous avons l'exécution de l'application.

## VI.) La documentation de l'API

Le but des API est d'être utilisée par de nombreux développeurs, souvent même extérieurs à notre projet ou entreprise.

Ainsi, une documentation accessible et facilement exploitable est une condition préalable au développement des API. Il est important d'avoir une documentation toujours à jour quand le code/les fonctionnalités de l'API évoluent.

- Sans documentation, même la meilleure API est inutilisable.
- Une documentation doit être associée à l'API, qui décrit les ressources ou fonctionnalités disponibles via l'API et fournit des exemples concrets d'URLs de requête et de code.
- Chaque paragraphe doit fournir un exemple de requête http.

### JUMP TO

#### Introduction

```
POST /add/euromed
DEL /delete_universities
PUT update_universities
GET /GET/universitiespercountr
GET /GET/number_universities_stu
  ents
GET /GET/usa_universities
GET http://127.0.0.1:5000/
GET /GET/number_strange_students
```

**DOCUMENTATION DE LA COLLECTION SUR POSTMAN**

## VI.1) Les applications à prendre en compte

La réalisation de cette API implique l'installation certaines applications à savoir :

- Pycharm ou Visual Studio (choix porté sur pycharm)
- Postman
- MongoDB compass (avec l'aide mongoDB Atlas)

## VI.2) Pycharm

Pycharm offre les qualités suivantes :

*-Assistance intelligente pour Python* : PyCharm fournit la saisie automatique de code intelligente, des inspections de code, la mise en évidence d'erreur ;

*-Capacités de développement à distance* : Exécutez, déboguez, testez et déployez des applications sur des hôtes distants ou des machines virtuelles ;

*-Outils intégrés pour développeurs* : Une large gamme d'outils prêts à l'emploi : un débogueur et un testeur intégrés ; un profileur Python ;

*-Développement multitechnologies* : Outre Python, PyCharm prend en charge les langages JavaScript, SQL, HTML/CSS, les langages de modèles, AngularJS, Node.js, et d'autres.

### a.) Référence du langage utilisé

Ce manuel de référence décrit la syntaxe et la "sémantique de base" du langage.

De ce fait, le langage utilisé est le langage *Python*. Les versions acceptées sont 3.6 et plus...

### b.) Installation des modules Python

Ce manuel de référence de la bibliothèque décrit la bibliothèque standard distribuée avec Python.

Il décrit également certains des composants facultatifs couramment inclus dans les distributions Python et aussi les packages nécessaires pour la réalisation de l'api.

PACKAGES	FONCTIONNALITES
Pip	Gestionnaire de paquets pour Python. C'est un outil qui permet d'installer et de gérer des bibliothèques et des dépendances qui ne sont pas distribuées dans le cadre de la bibliothèque standard
Venv	Outil standard pour créer des environnements virtuels et fait partie de Python depuis Python 3.3. À partir de Python 3.4, il s'installe par défaut dans tous les environnements virtuels créés.
Flask	Permet d'envoyer des requêtes HTTP à des fonctions Python
Request	Permet d'effectuer des requêtes HTTP
Jsonify	Permet de retourner des données sous formats JSON
Pymongo	Permet d'utiliser trois types d'objets via votre IDE python : les clients, les bases de données et les collections
Json	Langage léger d'échange de données textuelles
Dnspython	Un package très utile qui sert lors de la connexion à MongoDB
matplotlib	Permet de faire des courbes

### c.) Définition des Endpoints

Un Endpoint est ce qu'on appelle une extrémité d'un canal de communication. Autrement dit, lorsqu'une API interagit avec un autre système, les points de contact de cette communication sont considérés comme des Endpoints. Ainsi, pour les API, un Endpoint peut inclure une URL d'un serveur ou d'un service.

Il y a entre autres dans notre API :

- /
- /all\_universities
- /GET/usa\_universities
- /GET/number\_universities\_students
- /GET/universitiespercountries
- /GET/number\_strange\_students
- /add
- /delete\_universities
- /update\_universities

## d.) Opérations d'interrogation de l'API

### CREER UN OBJET

Pour créer un objet, on a besoin de l'instance suivante : Flask(\_\_name\_\_).

La variable \_\_name\_\_ est passée comme premier argument lors de la création d'une instance de l'objet Flask (une application Python Flask). Dans ce cas, \_\_name\_\_ représente le nom du package d'application et il est utilisé par Flask pour identifier des ressources.

### CONNECTER SA BASE DE DONNEES

La connexion à la base de données passe par :

- L'utilisation de l'outil de connexion à MongoDB, 'MongoClient' à qui on passera en paramètre l'URI de connexion à la base de données.
- Puis vient la phase de liaison à notre connexion de base de données
- Et enfin, la connexion à la collection de notre base de données.

### ❖ FORMAT DE CHAÎNE DE CONNEXION STANDARD

Cette section décrit le format standard de l'URI de connexion MongoDB utilisé pour se connecter à un serveur de base de données MongoDB. Le format est le même pour tous les pilotes MongoDB officiels.

Voici le schéma de connexion URI standard :

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]
```



❖ Les composants de cette chaîne sont :

mongodb://	Un préfixe requis pour identifier qu'il s'agit d'une chaîne au format de connexion standard.
Identifiant Mot de passe@	Optionnel. Si spécifié, le client tentera de se connecter à la base de données spécifique à l'aide de ces informations d'identification après s'être connecté à l'instance mongod.
Hôte1	Obligatoire. Il identifie une adresse de serveur à laquelle se connecter. Il identifie un nom d'hôte, une adresse IP ou un socket de domaine UNIX.
:port1	Optionnel. La valeur par défaut est : 27017 si elle n'est pas spécifiée.
hôteX	Optionnel. On peut spécifier autant d'hôtes que nécessaire. On spécifie plusieurs hôtes, par exemple, pour les connexions aux jeux de réplicas.
:portX	Optionnel. La valeur par défaut est : 27017 si elle n'est pas spécifiée.
/base de données	Optionnel. Le nom de la base de données à authentifier si la chaîne de connexion inclut des identifiants d'authentification sous la forme username : password@
?options	Options spécifiques à la connexion. Si la chaîne de connexion ne spécifie pas de base de données/, on doit spécifier une barre oblique (c'est-à-dire /) entre le dernier hôteN et le point d'interrogation qui commence la chaîne d'options.

### Les commandes utiles pour effectuer nos opérations

**@app.route()** : lie une URL à une fonction et prend en paramètre l'url et la méthode (get, post, put ou delete).

**Find / Find\_one** : permet de trouver un/des documents ou un/ des champs

**Insert\_one** : permet d'insérer un document dans une collection

**Delete\_one** : permet de supprimer un document

**Update\_one** : permet de mettre à jour les documents d'une collection

**count\_documents** : permet de compter les documents d'une collection

**\$gte** : qui est un opérateur de filtrage qui signifie supérieur ou égal

**aggregate** : qui agrège les données d'une collection. On distingue entre autres :

\$match (pour le filtrage), \$group (groupement des données), \$sort (pour le tri).

### CREER UNE METHODE GET

Avec la méthode GET, les données à envoyer au serveur sont écrites directement dans l'URL.

ACTION	Read
VERB http	GET
CHEMIN D'URL	<ol style="list-style-type: none"> <li>1. /</li> <li>2. /GET/usa_universities</li> <li>3. /GET/number_universities_students</li> <li>4. /GET/universitiespercountries</li> <li>5. /GET/number_strange_students</li> </ol>
DESCRIPTION	<ol style="list-style-type: none"> <li>1. Définit une URL unique pour lire La page « d'accueil » de l'API. Etant donné qu'il s'agit de la première page</li> <li>2. Définit une URL unique pour lire toutes les universités américaines</li> <li>3. Définit une URL unique pour lire le nombre d'universités où le nombre d'étudiants est supérieur à 20000.</li> <li>4. Définit une URL unique pour lire le nombre d'universités par pays et par ordre décroissant. Là, il s'agit d'une opération d'agrégation.</li> <li>5. Définit une URL unique pour lire le nombre d'universités où les <math>\frac{3}{4}</math> des étudiants sont des internationaux (étrangers).</li> </ol>

### CREER UNE METHODE POST

La méthode POST écrit les paramètres URL dans la requête HTTP pour le serveur.

ACTION	Create
VERB http	POST
CHEMIN D'URL	/add
DESCRIPTION	<p>Définit une URL unique pour créer une nouvelle université ; Si le rang donné (ranking) donné existe déjà alors l'action ne sera pas effectuée sur la base de données. Sinon, un message sera retourné indiquant que le document inscrit a bien et bel été incorporé dans la base de données.</p>

### CREER UNE METHODE DELETE

ACTION	Delete
VERB http	DELETE
CHEMIN D'URL	/delete_universities
DESCRIPTION	Définit une URL unique pour supprimer une université ; Si le rang donné (ranking) donné existe alors l'action sera effectuée sur la base de données sinon, un message sera retourné indiquant que le « ranking » n'existe pas dans la base de données.

### CREER UNE METHODE UPDATE

ACTION	Update
VERB http	PUT
CHEMIN D'URL	/update_universities
DESCRIPTION	Définit une URL unique pour mettre à jour une nouvelle université ; c'est-à-dire modifier certains champs d'un document

### DEBOGER UNE APPLICATION

La commande peut faire plus que simplement démarrer le serveur de développement. En activant le mode débogage, le serveur se rechargera automatiquement si le code change et affichera un débogueur interactif dans le navigateur si une erreur se produit lors d'une requête.

Ci-dessous, la commande pour l'activation du débogage :

```
app.config["DEBUG"] = True
```

Pour l'exécution de l'application, il faut dire qu'il existe deux possibilités mais nous n'en parlerons que d'une seule ; nous utilisons :

`app.run()`.

### Spécification des options du serveur

La méthode `app.run()` prend en charge plusieurs options, y compris toutes celles que vous pouvez fournir à la flask run commande, et quelques autres :

- Host → le nom d'hôte sur lequel écouter.
- Port → le port du serveur Web.
- Debug → si donné, activer ou désactiver le mode débogage.
- Load\_dotenv → charger les fichiers `.env` et `.flaskenv` les plus proches pour définir les variables d'environnement.

## VI.3) POSTMAN

Postman est un logiciel permettant de créer et de tester des requêtes HTTP. Il permet de les personnaliser dans les plus fins détails grâce à une interface ergonomique et intuitive. On peut choisir la méthode de la requête, entrer l'URL du serveur que vous voulez interroger, et rajouter tous les paramètres possibles pour une requête HTTP. Le logiciel tient un historique des différentes requêtes. Il est très utile pour tester une api. On l'utilise pour s'initier au protocole HTTP en créant différentes requêtes.

### ❖ Guide d'utilisation

Notre collection se nomme `MyCollection`.

Ci-dessous les images et les résultats des requêtes.

# GET

MyCollection / /GET/number\_usa\_universities Save ... Edit Comments

**GET** ▼ http://127.0.0.1:5000/GET/number\_usa\_universities Send ▼

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Body ▼ 200 OK 1207 ms 43.93 KB Save Response ▼

Pretty Raw Preview Visualize JSON ▼ Copy Search

```

1  {
2    "_id": {
3      "$oid": "627ef54f57532bea1dc126a2"
4    },
5    "ranking": 2,
6    "title": "Stanford University",
7    "location": "United States",
8    "number students": "16,223",
9    "students staff ratio": "7.4",
10   "perc intl students": "23%",
11   "gender ratio": "44 : 56"
12 },
13 {
14   "_id": {
15     "$oid": "627ef54f57532bea1dc126a3"
16   },
17   "ranking": 3,
18   "title": "Harvard University",
19   "location": "United States",
20   "number students": "21,261",
21   "students staff ratio": "9.3",
22   "perc intl students": "25%"
23 }

```

Cookies Capture requests Bootcamp Runner Trash

MyCollection / /GET/number\_universities\_students Save ... Edit Comments

**GET** ▼ http://127.0.0.1:5000/GET/number\_universities\_students Send ▼

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body ▼ 200 OK 115 ms 231 B Save Response ▼

Pretty Raw Preview Visualize HTML ▼ Copy Search

```

1  NOMBRE D'UNIVERSITES AVEC PLUS DE 20000 ETUDIANTS :
2
3  976

```

MyCollection / /GET/universitiespercountries

GET http://127.0.0.1:5000/GET/universitiespercountries Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Body 200 OK 10.22 s 3.28 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "_id": "United States",
4     "nombre": 181
5   },
6   {
7     "_id": "Japan",
8     "nombre": 116
9   },
10  {
11    "_id": "United Kingdom",
12    "nombre": 101
13  },
14  {
15    "_id": "China",
16    "nombre": 91
17  },
18  {
19    "_id": "India",
20    "nombre": 63
21  },
22  {
23    "_id": "Brazil"
```

Pretty Raw Preview Visualize JSON

```
22  {
23    "_id": "Brazil",
24    "nombre": 52
25  },
26  {
27    "_id": "Spain",
28    "nombre": 50
29  },
30  {
31    "_id": "Italy",
32    "nombre": 49
33  },
34  {
35    "_id": "Russian Federation",
36    "nombre": 48
37  },
38  {
39    "_id": "Germany",
40    "nombre": 48
41  },
42  {
43    "_id": "Iran",
44    "nombre": 47
```

MyCollection / /GET/number\_strange\_students

GET http://127.0.0.1:5000/GET/number\_strange\_students

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body 200 OK 624 ms 245 B Save Response

Pretty Raw Preview Visualize HTML

```

1 NOMBRE D'UNIVERSITES OU LES 3/4 DES ETUDIANTS SONT DES ETRANGERS:
2
3 112

```

## POST

MyCollection / /add/euromed

POST http://127.0.0.1:5000/add

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

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

```

1 {
2   "ranking": "1527",
3   "title": "ESMT",
4   "location": "Senegal",
5   "number_students": "1000",
6   "students_staff_ratio": "14.5",
7   "perc_intl_students": "49%",
8   "gender_ratio": "47 : 53"
9 }

```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 3.56 s Size: 211 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "msg": "university successfully added !"
3 }

```

## Conséquence sur la base de données Mongo :

Documents Aggregations Schema Explain Plan Indexes Validation

**FILTER** { field: 'value' } **OPTIONS** **FIND** **RESET** **↺** **⋮**

**ADD DATA** **VIEW** **≡** **{ }** **⌄**

Displaying documents 1521 - 1526 of 1526 **<** **>** **REFRESH**



universities\_db.universities\_ranking **1.5k** **1**  
DOCUMENTS INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

**FILTER** { field: 'value' } **OPTIONS** **FIND** **RESET** **↺** **⋮**

**ADD DATA** **VIEW** **≡** **{ }** **⌄**

Displaying documents 1 - 20 of 1527 **<** **>** **REFRESH**

**DELETE**

http://127.0.0.1:5000/delete\_universities **Save** **⋮**

**DELETE** **http://127.0.0.1:5000/delete\_universities** **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings **Cookies**

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

```
1 {
2   "ranking": "1527",
3   "title": "ESMT",
4   "location": "Senegal",
5   "number_students": "1000",
6   "students_staff_ratio": "14.5",
7   "perc_intl_students": "49%",
8   "gender_ratio": "47 : 53"
9 }
```

**Body** **Cookies** **Headers (5)** **Test Results** **Status: 200 OK** **Time: 154 ms** **Size: 213 B** **Save Response**

**Pretty** **Raw** **Preview** **Visualize** **JSON** **⋮**

```
1 {
2   "msg": "university successfully removed !"
3 }
```



# PUT

Overview MyCollection GET /all/universities POST /add/eu GET http://127.0.0.1:5000/ DEL /delete\_u PUT http://127.0.0.1:5000/update\_universities GET http://127.0.0.1:5000/ + ... No Environment

http://127.0.0.1:5000/update\_universities Save

PUT http://127.0.0.1:5000/update\_universities Sending...

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

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

```
1 {
2   "ranking":1,
3   "title":"University of Oxford",
4   "location":"United Kingdom",
5   "number_students":"25000 (updated)",
6   "students_staff_ratio":"11.1",
7   "perc_intl_students":"50% (updated)",
8   "gender_ratio":"45 : 55 (updated)"
9 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 437 ms Size: 213 B Save Response

## Conséquence sur la base de données :



Document avant la mise à jour

ADD DATA VIEW

```
_id: ObjectId('627ef54f57532bea1dc126a1')
ranking: 1
title: "University of Oxford"
location: "United Kingdom"
number students: "20,774"
students staff ratio: "11.1"
perc intl students: "41%"
gender ratio: "46 : 54"
```



Document après la mise à jour

ADD DATA VIEW

```
_id: ObjectId('627ef54f57532bea1dc126a1')
ranking: 1
title: "University of Oxford"
location: "United Kingdom"
number students: "25000 (updated)"
students staff ratio: "11.1"
perc intl students: "50% (updated)"
gender ratio: "45 : 55 (updated)"
```

## CONCLUSION

En somme, on crée une API quand :

- Nos données sont modifiées ou mises à jour fréquemment.
- Les utilisateurs n'ont besoin d'accéder qu'à une partie des données à la fois.
- Les utilisateurs ont à effectuer d'autres actions que simplement consulter
- Les données, par exemple contribuer, mettre à jour ou supprimer.
- Si on dispose de données qu'on souhaite partager avec le monde entier, créer une API est une façon de le faire.



# FIN