Projet NoSQL

29 Mai 2022

Etudiant : *Moussa Steve. B Sanogo.*

Enseignant : *Jean-Marie Preira.*

INGC2, année Scolaire 2021/2022

Plan

Table des matières

Présentation du Projet

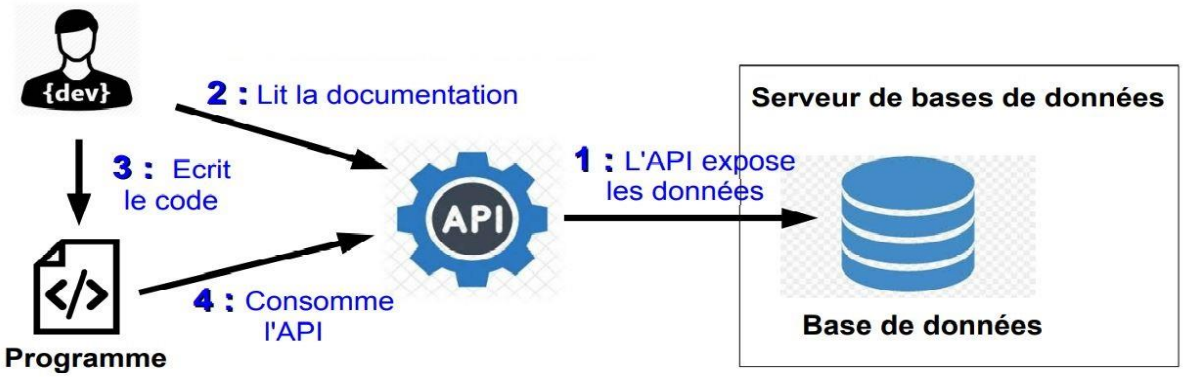
Á la fin du module de base de données NoSQL, il nous a été soumis deux projets au choix : le premier portant sur la réalisation d'une API RESTful connectée à MongoDB Atlas, le second avait pour objectif la réalisation d'un tableau de bord à l'aide MongoDB Charts. Nous avons décidé de réaliser le premier, car il nous semblait intéressant, mais aussi parce le concept d'API étant nouveau pour nous, il nous semblait abstrait.
Dans la suite de ce rapport il sera question d'apporter dans un premier temps des définitions sur des concepts de base, ensuite nous présenterons pas à pas les étapes de création de notre Cluster sur MongoDB Atlas puis de chargement du jeu de données. Après quoi nous présenterons l'environnement de travail et les différentes bibliothèques utilisées pour la réalisation de l'API. Enfin, nous présenterons la liste des requêtes d'interrogation simples et analytiques qui seront proposées dans l'API.

Présentation de Flask

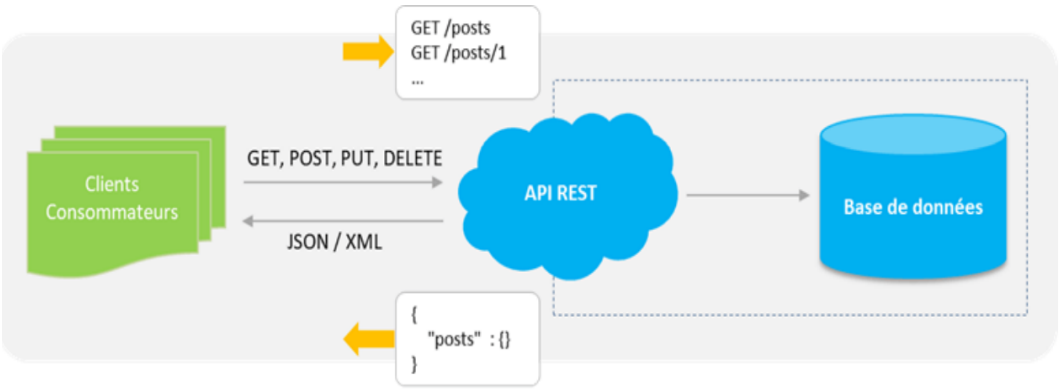
Flask est un micro Framework Web ***Python*** qui facilite la création d'une **application Web** complète. Un micro Framework d'un ensemble de modules qui permettent de développer plus rapidement en fournissant des fonctionnalités courantes. Lorsque vous concevez une application web, vous avez toujours besoin de gérer les **requêtes HTTP** d'un serveur web, d'afficher des pages web dynamiques, de gérer les cookies. Un Framework web vous fournit ces fonctionnalités pour commencer un projet sur des bases solides.
Flask est populaire car il est à la fois très puissant et très léger. Par exemple, il permet de créer une application web minimale en *7* lignes.

Qu'est-ce qu'une API RESTful

Une API, ou ***Application Programming Interface***, se définit comme un **ensemble de fonctions informatiques** par lesquelles **deux logiciels** vont **interagir** sans **intermédiation humaine**. Une API peut être publique ou privée, et permet d'exposé un service, elle rend disponible des fonctionnalités ou des données. En parallèle, les développeurs écrivent des programmes qui consomment ces APIs. Pour cela les développeurs s'appuient sur la documentation des APIs



Une ***API REST*** (***Representational State Transfer Application Program Interface***) est un style architectural qui permet aux logiciels de communiquer entre eux sur un réseau ou sur un même appareil, elles sont plus pour créer des ***services web***. Souvent appelés services web **RESTful**, REST utilise des méthodes **HTTP** pour **récupérer et publier** des données entre un périphérique **client** et un **serveur** même s'ils utilisent des systèmes d'exploitation et des architectures différents.
Le client peut demander des ressources avec un langage que le serveur comprend, et le serveur renvoie la ressource avec un langage que le client accepte. Le serveur renvoie la ressource au format ***JSON, XML texte***.
REST se base sur les URI (Uniform Resource Identifier) afin d'identifier une ressource. Une ressource est une information : une image, un document, une personne etc.

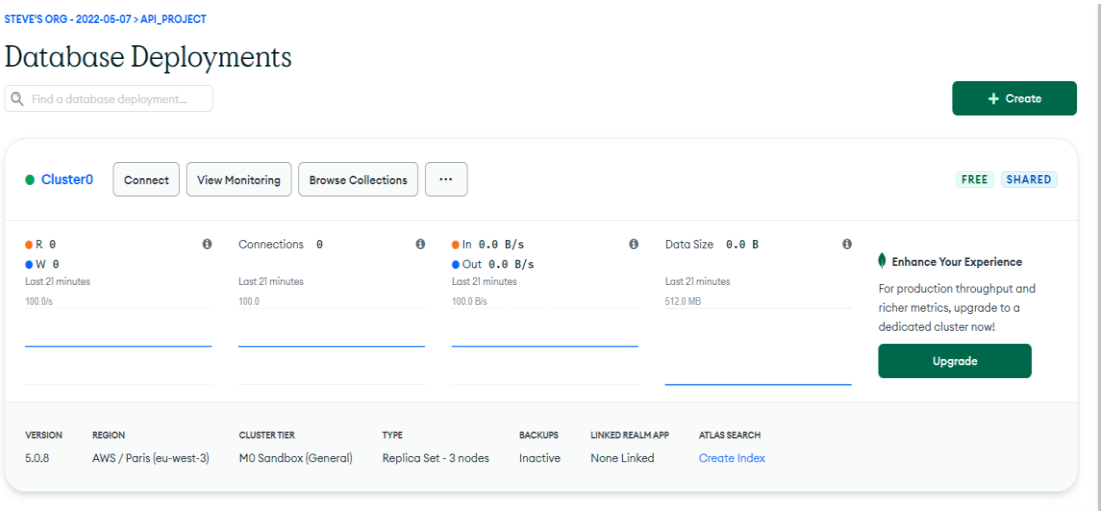


L'architecture RES

Création d'un cluster sur MongoDB Atlas

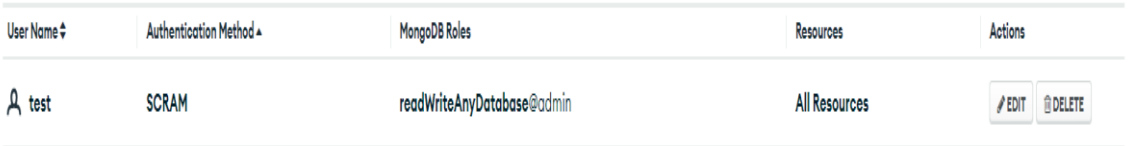
Création du Compte utilisateur

Avant toute chose il faut créer un compte utilisateur sur **MongoDB Atlas**, pour se faire cliquez ici. Après création de votre compte vous pouvez créer un projet, le nôtre a pour nom « ***API_Projet*** », puis associer à ce projet un cluster.
Le choix d'un cluster **partagé et gratuit** est plus adéquat pour l'apprentissage ou pour développer de petite application, choisissez par la suite le fournisseur cloud (**aws**, **Google Cloud** ou **Azure**) et la région.

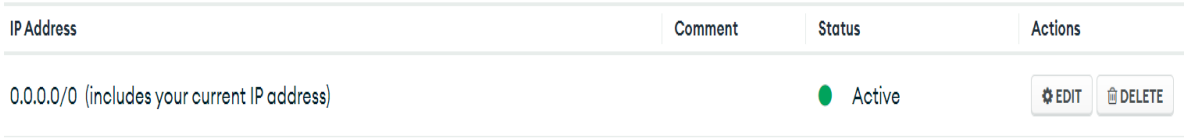


Création des utilisateurs du Cluster

Après cela il faut créer des utilisateurs avec leur différents droits d'accès, ces utilisateurs représentent les applications qui peuvent avoir accès au Cluster.
Dans notre cas nous avons créé un utilisateur ***« test »*** qui a le droit de ***lecture*** et d'***écriture*** sur toutes les base de données du Cluster, après s'être authentifié par son mot de passe.



Enfin il faut définir dans la section ***Network Access*** les adresses IP qui pourront avoir accès au cluster, afin de permettre l'accès à toute adresse IP, nous l'avons configuré tel que montrer dans la figure ci-dessus.



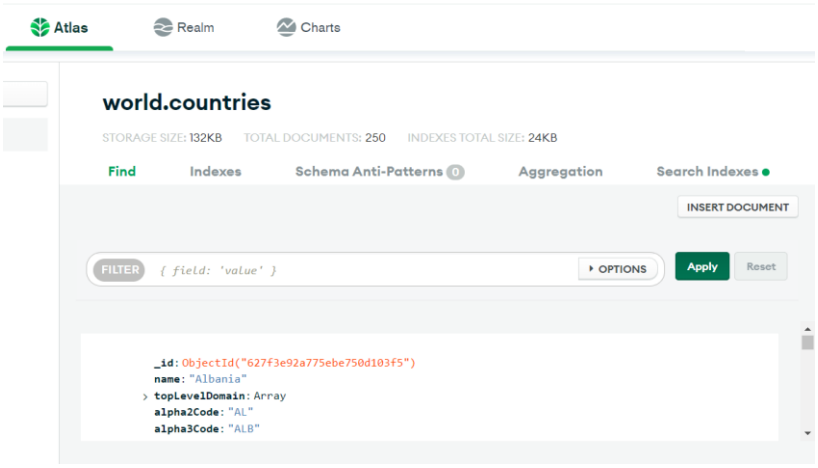
Choix et importation du jeu de données

Le jeu de donnée choisit a pour nom : « ***world »***, il contient un ensemble de document sur des pays du monde et comporte des champs comme le nom du pays « ***name*** », le nom de sa capitale « ***capital*** », le nombre de ses habitants « ***population*** », sa superficie « ***area*** », et l'indice de répartition des revenus de sa population « ***gini*** ». Pour le télécharger cliquer ici.

Pour l'importation du jeu de donnée rien de plus simple, il faut au préalable repérer le **nom d'host et le numéro de port** de **l'instance primaire** Mongo du cluster, puis grâce à l'outil ***mongoimport*** on se connecte à distance à ***Mongo Atlas*** sur notre base dans le cloud tout en précisant le nom de l'utilisateur et le mot de passe. Le nom d'utilisateur utilisé est celui créer précédemment ***« test ».***

```
C:\Users\HP>mongoimport --host cluster0-shard-00-02.kafpl.mongodb.net:27017 --db world --jsonArray --type json --collection countrie
--authenticationDatabase admin --ssl --username test --password AoBcoIIfoQtAuxjH < countries.json
2022-05-14T05:30:57.940+0000    connected to: mongodb://cluster0-shard-00-02.kafpl.mongodb.net:27017/
2022-05-14T05:30:58.634+0000    250 document(s) imported successfully. 0 document(s) failed to import.
```

Importation du jeu de données



Le jeu de données sur MongoDB Atlas.

Nous allons aussi créer une autre collection du nom de « ***update*** » qui contiendra les mises à jour apportées par un utilisateur sur des pays de notre collection « **countries** ». Pour un pays dont on modifiera une information, on ajoutera à une copie du document correspondant les champs « ***modificationDate*** », « ***modificationHour*** » et ***« processed »*** avant de le stocker dans ***« update »,*** ces champs permettront à l'administrateur de trier l'ensemble des documents à modifier par date décroissante par exemple, puis de marquer à chaque document traiter la valeur **True** au niveau du champ ***« processed »***.

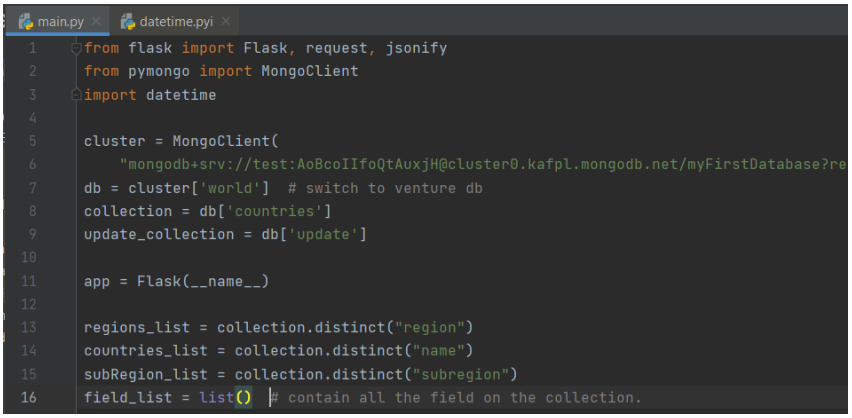
Ainsi donc lorsqu'un utilisateur modifiera des informations relatives à un pays, notre API la sauvegardera dans la collection **« update »** puis l'administrateur de la DB pourra juger de la **crédibilité** des mises à jour apportées avant leur intégration dans la collection principale.
Aussi la mise à jour d'un document ne concernera que quelques champs à savoir : ***« name », « capital », « population », « area »*** et ***« gini »***.

```
C:\Users\HP>mongosh "mongodb+srv://cluster0.kafpl.mongodb.net/myFirstDatabase" --apiVersion 1 --username test
Enter password: ***************
Atlas atlas-r5rgud-shard-0 [primary] myFirstDatabase> use world
switched to db world
Atlas atlas-r5rgud-shard-0 [primary] world> db.createCollection("update");
{ ok: 1 }
```

Connexion à la data base « world » et création de la collection « update »

Présentation de l'environnement de travail

PyCharm est un environnement de développement intégré (IDE) **Python** dédié, fournissant une large gamme d'outils essentiels pour les développeurs, étroitement intégrés pour créer un environnement pratique pour le développement productif de Python, du Web et de la science des données.
PyCharm est disponible en trois éditions : la version ***Community***, ***Professional***, et ***Edu.*** Les versions ***Community*** et ***Edu*** sont gratuites et open source, quant à la version Professional elle est payante. Pour notre projet nous avons utilisé la version Community, télécharger ***PyCharm*** ici.



Présentation de l'environnement de travail PyCharm

Bibliothèques utilisées

Installation de Flask

Après avoir créé notre projet « ***API_RESTful*** » sur **PyCharm**, nous avons téléchargé le package **flask** grâce à la commande « ***pip install flask*** » au niveau de l'invite de commande de notre projet. L'installation de flask comme nous l'avons mentionné plus haut nous permettra d'implémenter un serveur Web avec les différentes méthodes **HTTP** (GET, POST, UPDATE etc.) permettant au serveur de répondre aux différentes requêtes des utilisateurs. Après l'installation de ce package nous importerons les classe **Flask**, et utiliserons l'objet global **request**, et la fonction **jsonify.**

La classe **Flask** nous permet :

- ✓ De créer une instance, le premier argument du constructeur est le nom du module ou du package de l'application(__name__). Ceci est nécessaire pour que Flask sache où rechercher des ressources telles que des modèles et des fichiers statiques.

- ✓ D'utiliser la méthode ***route()*** qui permet **de lier** une **URI** à une **fonction** que nous définirons, en effet cette méthode indiquera à Flask quelle URI doit déclencher notre fonction. En dehors de l'URI que la méthode ***route()*** prend en paramètre, elle peut aussi recevoir en argument la méthode http utilisée : **POST**, **GET, PUT etc…**
  Lorsque le serveur de développement de Flask fonctionne avec les paramètres par défaut, il est accessible à l'adresse locale http://127.0.0.1:5000/.

- ✓ D'utiliser l'objet global **request** afin de pouvoir récupérer les différentes données envoyées par l'utilisateur vers notre API, ces données seront par la suite envoyées vers notre Cluster sur MongoDB Atlas. Ces données peuvent servir soit à **ajouter** soit à **modifier** un document dans notre collection « ***countries*** »

- ✓ D'utiliser la fonction **jsonify** du module Flask pour sérialiser les données au format **JSON** (JavaScript Object Notation), puis les encapsuler dans un objet de réponse.

Installation de Pymongo et importation de la classe MongoClient

La classe Python ***MongoClient*** permet aux développeurs d'établir des connexions à MongoDB en développement à l'aide d'instances clientes, elle facilite le codage et la connexion à MongoDB.
MongoClient fait partie de la bibliothèque ***Pymongo*** et peut être importé dans le code Python en utilisant le code « ***from pymongo import MongoClient*** »
Pour l'installation de ***Pymongo*** il faut exécuter la commande « ***python -m pip install pymongo*** » à partir de l'outil de commande présent dans ***PyCharm***.

Aussi il faut installer la bibliothèque ***dnspython*** pour l'URI **mongodb srv** à l'aide de la commande : « ***python -m pip install dnspython*** »

Après l'installation de ***PyMongo*** et l'importation de la classe ***MongoClient***, on crée une instance de cette classe avec un constructeur prenant en argument **nom d'host de l'instance primaire MongoDB** sur notre Cluster sur Atlas, le nom d'utilisateur et le mot de passe. Par la suite on peut se positionner sur la base de données « **world** » et la collection « **countries** ». Enfin plusieurs méthodes s'offrent à nous pour l'interrogation de la collection, pour plus de détail consulté ce lien.

```
cluster = MongoClient(

"mongodb+srv://test:AoBcoIIfoQtAuxjH@cluster0.kafpl.mongodb.net/myFirstDatabase?retryWrites=true&w=majority")
```

Connection au Cluster sur Atlas grâce à de MongoClient

Fonctionnalités de l'API

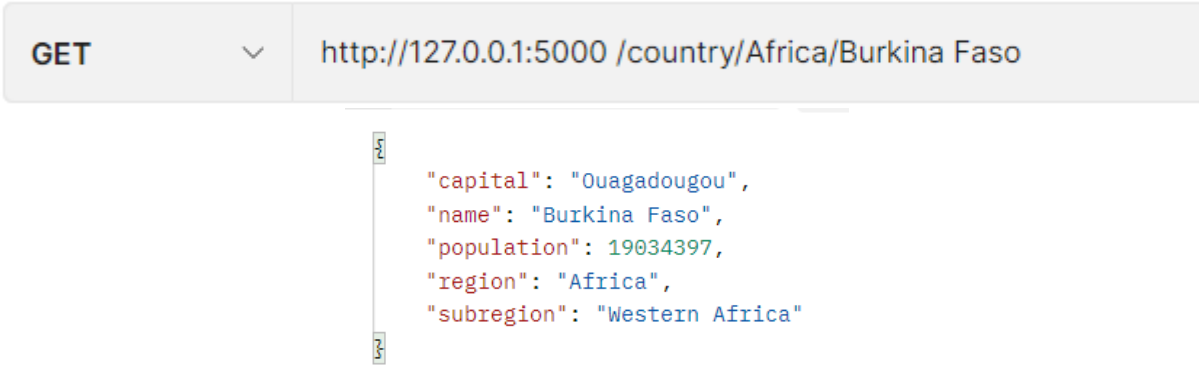
De façon général notre API, permet de fournir un ensemble d'informations utilise par rapport à un pays du monde : sa capitale, sa population, les langues parlées, la monnaie utilisée, sa superficie etc… Elle permet aussi d'avoir un ensemble de données en rapport avec une région d'un continent (ex East Africa, Western Africa etc…), d'un continent, ou plus globalement d'obtenir des chiffres par rapport au monde entier. Les informations obtenues par rapport à un continent, une région d'un continent ou au monde ne sont rien d'autre que le résultat d'opération de somme ou de moyenne effectuées sur certains champs de la collection selon des critères bien précis. Elles peuvent aussi consister à retourner un champ précis pour l'ensemble des pays appartenant à un continent, ou une région d'un continent ou plus globalement pour l'ensemble des pays de la collection.

Les requêtes d'interrogations

@app.route('/country/<region>/<name>', methods=['GET'])

Cette URI permet de retourner un document contenant quelques informations relatives à un pays, elle prend en paramètre le nom du contient ***<region>*** dans lequel se trouve le pays et son nom ***<name>*.**

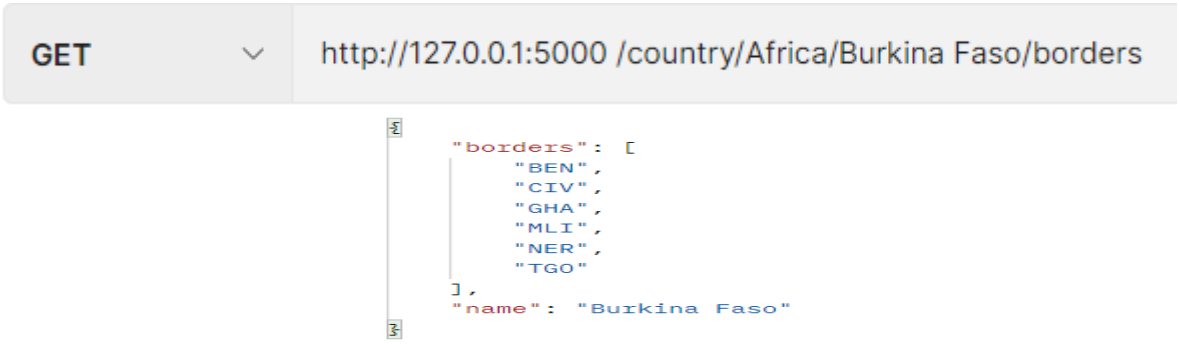
Exemple : Retournons des informations concernant le Burkina Faso.



@app.route('/country/<region>/<name>/<field>',methods=['GET'])

Cette URI permet de retourner une information précise ***<field>*** en rapport à un pays. Il faut bien sûr que le champ recherché soit contenu dans notre collection sinon nous indiquerons que nous ne sommes pas en mesure de retourner l'information demandé. Il faut également préciser lors de son usage le continent auquel appartient le pays ***<region>*.**

Exemple : Retournons les pays limitrophes du Burkina Faso.



@app.route('/countries/<region>/<field>', methods=['GET'])

Celle-ci permet de retourner tous les pays se trouvant dans un continent donnée **<region>** avec une information particulière ***<field>***.

Exemple : Retournons la population des différents pays d'Europe.

@app.route('/countries/<region>/<subregion>/<field>', methods=['GET'])

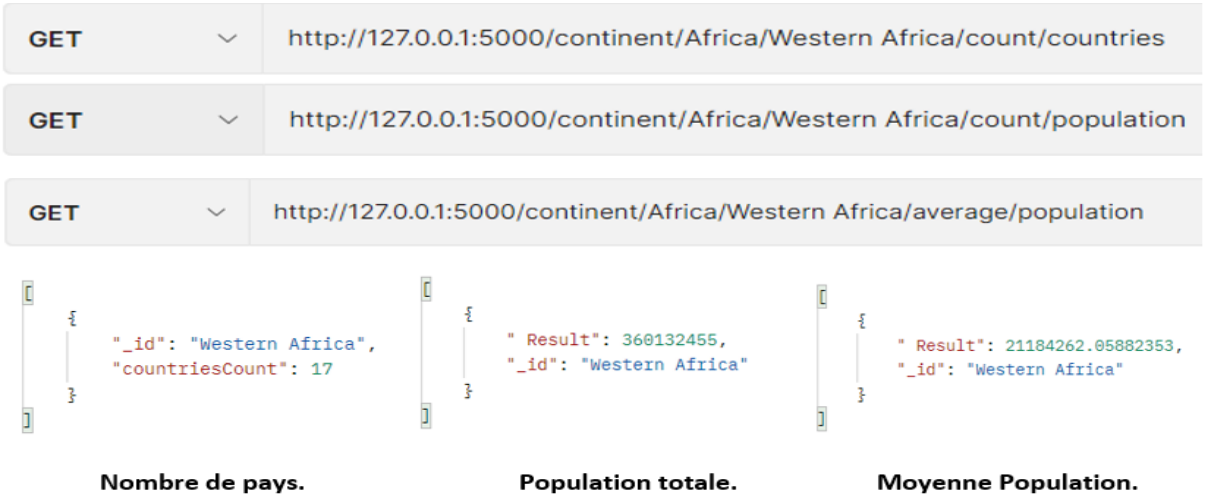
Cette URI est semblable à la précédente, cependant elle permet de retourner une information relative à tous les pays appartenant à une région (Est, Ouest, Sud) d'un continent donné, elle ne retourne donc pas tous les pays du continent concerné.

Exemple : Retrouvons tous les pays de l'Afrique de l'ouest.

@app.route('/continent/<region>/<subregion>/<operation>/<field>',methods=['GET'])

Cette URI permet de retourner un ensemble d'informations relatives aux pays appartenant à une sous-région ***<subregion>*** d'un continent donné ***<region>***, ces informations sont le résultat d'opération **<operation> :** de somme ou de moyenne sur un champ de recherche ***<field>*.** Cette URI peut donc gérer deux requêtes selon l'opération : ***count ou average***.

Exemple : Retournons le nombre de pays et la population totale de l'Afrique de l'ouest, puis retournons la moyenne de la population Ouest Africaine.

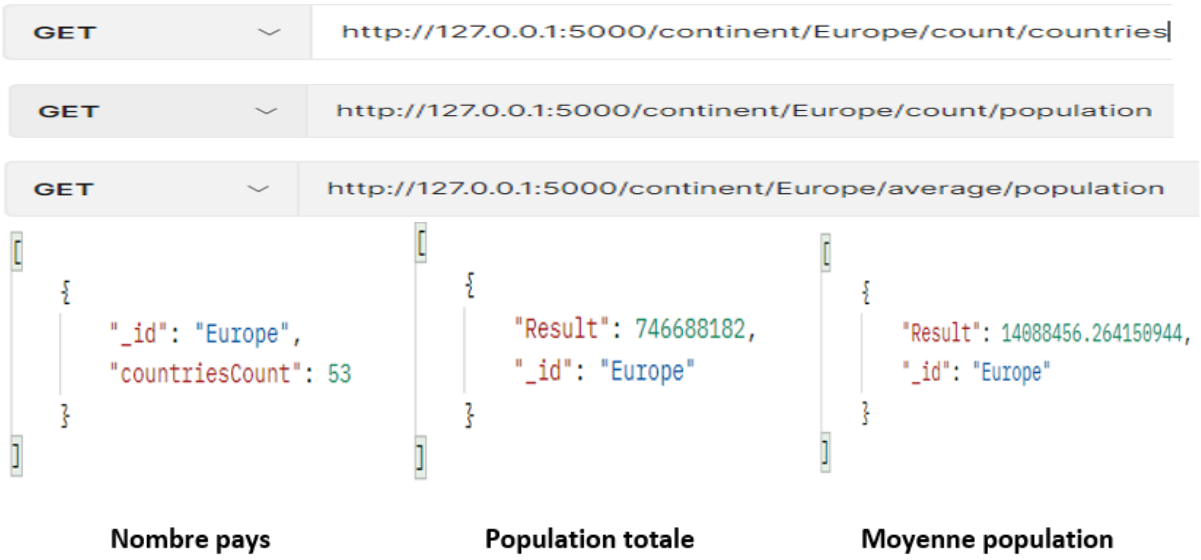


Nombre de pays. **Population totale.** **Moyenne Population.**

@app.route('/continent/<region>/<operation>/<field>', methods=['GET'])

Celle-ci permet d'obtenir les mêmes résultats que la précédente sauf que le niveau d'agrégation est étendu à un continent.

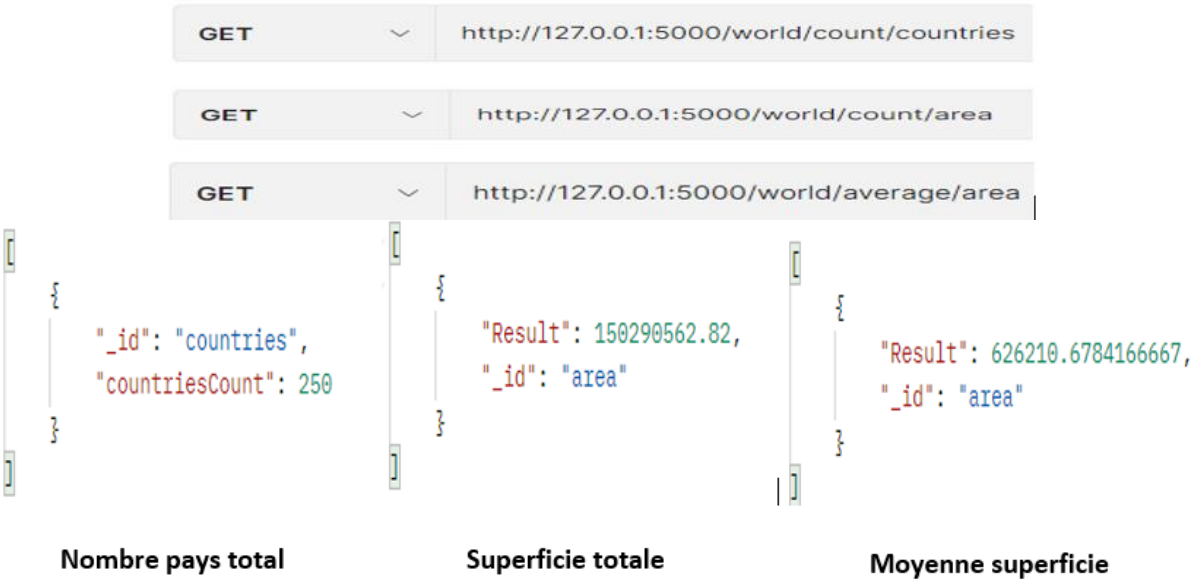
Exemple : Retournons le nombre de pays et la population totale de l'Europe. Puis retournons la moyenne de la population.



@app.route('/world/<operation>/<field>', methods=['GET'])

Cette URI est semblable aux deux (02) précédentes cependant le domaine d'agréation est maintenant étendu à tous les pays de la collection sans exception. Il faut également signaler que ces trois dernières URI permettant de retourner des résultats basés sur des agrégations ne concerne pas tous les champs de la collection, elles concernent naturellement que les champs quantitatifs parmi lesquels nous avons : ***<population>, <area>, <gini>, <countries>.***

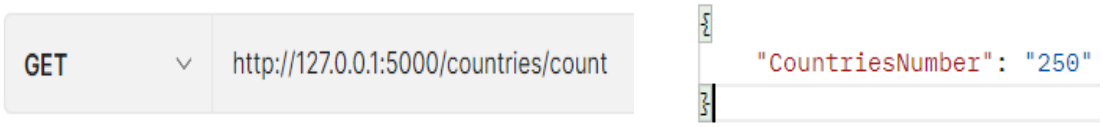
Exemple : Retournons le nombre de pays et la superficie totale de l'ensemble des pays contenu dans la collection. Puis retournons la moyenne de la superficie.



@app.route('/countries/count', **methods**=['GET'])

Cette URI permet de retourner le nombre de pays total répertorié dans notre collection.

Exemple : Déterminons le nombre total de pays dans notre collection.



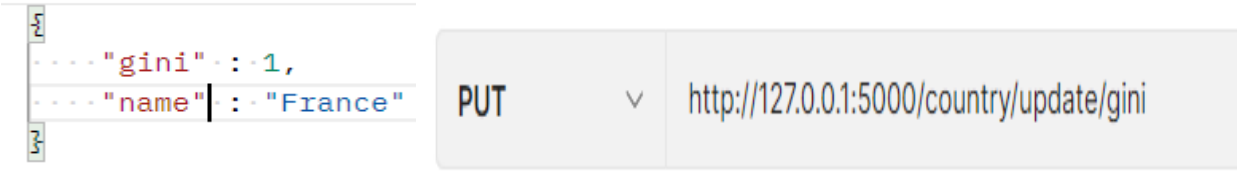
Nous avons donc au total **11** requêtes d'interrogation de notre base de données ***world***.

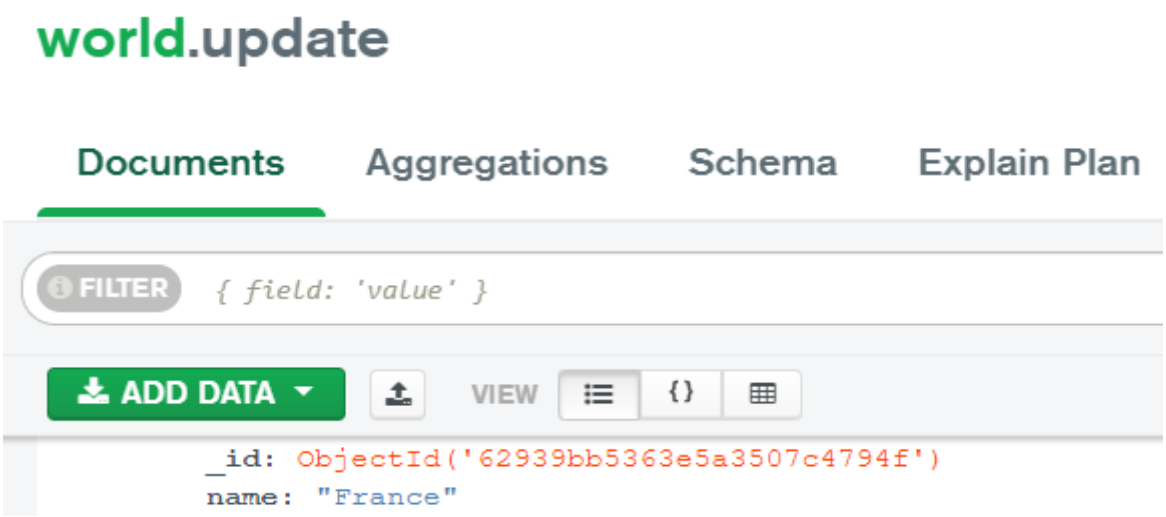
Les requêtes de mise à jour

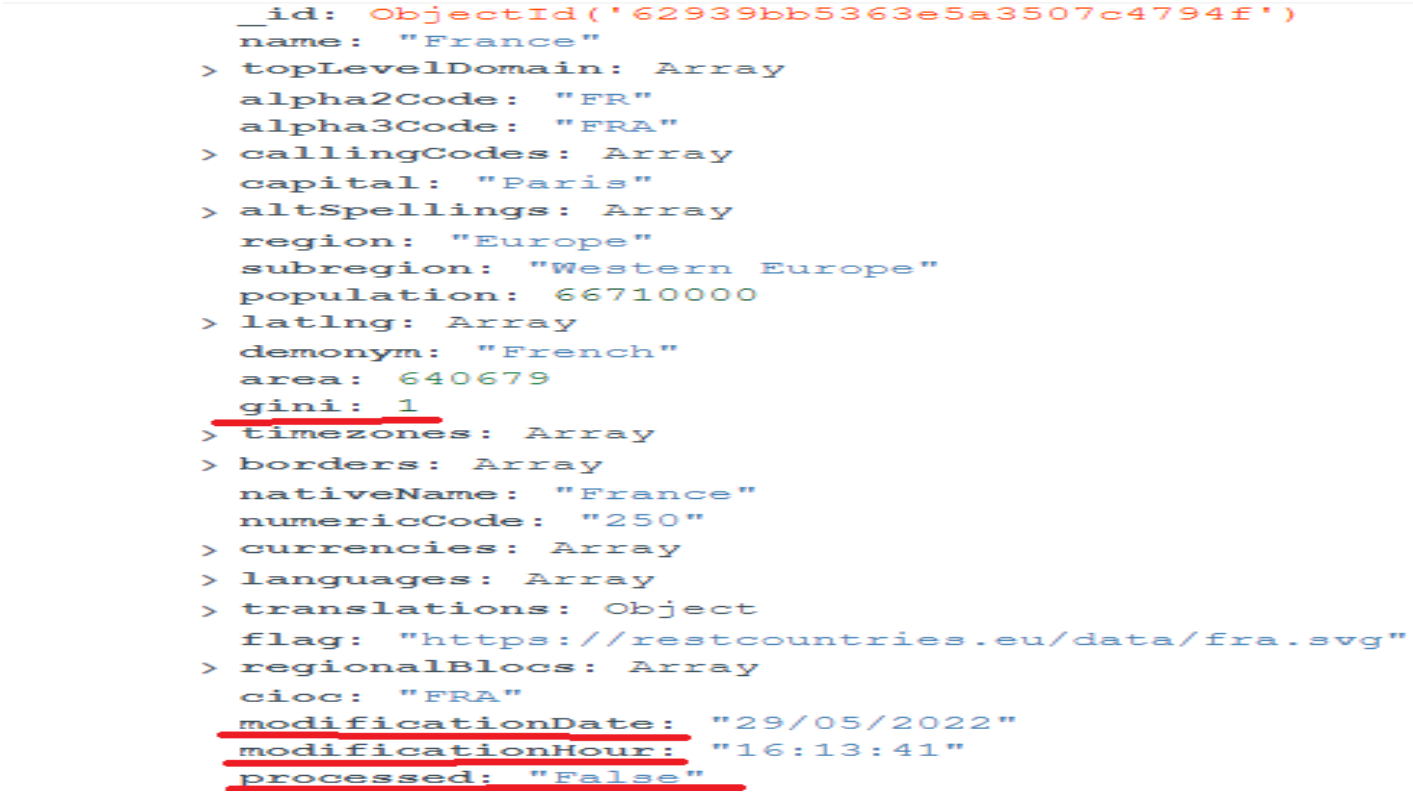
@app.route('/country/update/<field>', **methods**=['PUT'])

Cette URI permet la mise à jour d'un champ ***<field>*** concernant un document de la collection countries, le champ doit au préalable exister dans le document. Comme nous l'avons dit plus haut, seules les valeurs des champs : ***« name », « capital », « population », « area » et « gini »*** seront ouvert à la modification.

Exemple : Modifions l'indice de répartition des revenus de la population Française.







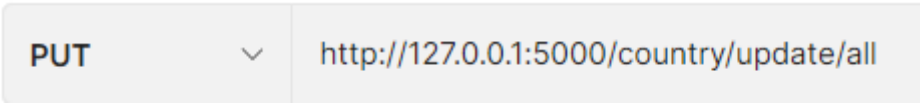
@app.route('/country/update/<field1>/ <field2>', **methods**=['PUT'])

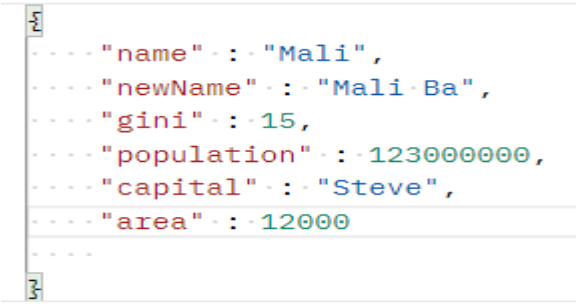
Celle-ci permet comme la précédente de modifier un document de la collection, cependant là nous permettons à l'utilisateur d'en modifier deux simultanément ***<field1>*** et ***<field2>***. Il faut aussi noter ni cette URI ni la précédente ne permet de modifier le nom d'un pays de la collection, c'est-à-dire modifier le champ ***« name »*** d'un document, seule l'URI si dessous le permet.

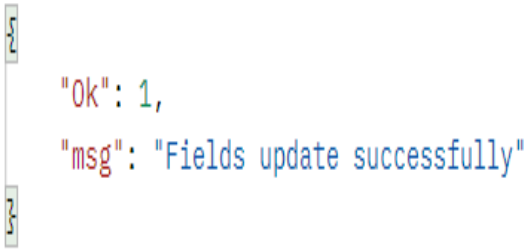
@app.route('/country/update/all', **methods**=['PUT'])

Cette dernière requête de modification permet à tout utilisateur de mettre à jour tous les champs dont la modification des valeurs est permise à savoir : ***« name », « capital », « population », « area » et « gini ».***

Exemple : Mettons à jour les champs concernant le Mali.



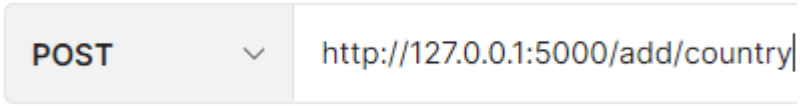


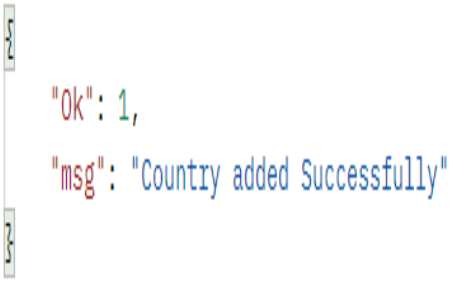


Les requêtes de création

@app.route('/add/country', **methods**=['POST'])

Enfin cette requête permet d'ajouter un nouveau pays. Il faut aussi préciser que toutes les requêtes de modifications ou d'ajout de documents stockent les documents modifiés ou à ajouter dans la collection ***« update ».*** Puis l'administrateur de la **db** se chargera de vérifier la véracité des modifications effectuées ou des pays ajoutés par les utilisateurs avant leurs prises en compte dans la collection principale ***« countries ».***





Ainsi donc présenté les différentes requêtes de notre API, vous trouverez en pièce jointe le code du projet très bien commmenté. Les commentaires sont en anglais, pas parce que nous avons copié le code mais tout simplement pour l'importante place qu'occupe cette langue dans le domaine des télécommunications.

FIN