API REST avec PYTHON et MONGODB

L'API REST que vous allez créer servira une structure de données représentant des étudiants. Une fois de plus, vous allez utiliser la base Ecole comme exemple et créez une appli web capable de travailler avec ces données. L'un des objectifs d'une API est de découpler les données de l'application qui l'utilise, en masquant les détails d'implémentation des données.

Qu'est-ce que REST ?

Avant de commencer à créer un serveur API, je veux parler un peu de REST. REST c'est un ensemble de conventions tirant parti du protocole HTTP pour fournir un comportement CRUD (Créer, Lire, Mettre à jour et Supprimer) sur des données et des collections de données. Le comportement CRUD correspond bien aux verbes de la méthode de protocole HTTP comme ceci:

action	Méthode HTTP	description
Créer	POST	Créer un nouvel objet
Lire	GET	Lire les informations d'un objet existant
Mettre à Jour	PUT	Mettre à jour les informations sur un objet existant
Supprimer	DELETE	Supprimer un objet

Vous pouvez effectuer ces actions sur une ressource. Il est utile de penser à une ressource comme un objet au quel un identifiant peut s'appliquer: une personne, une commande pour quelque chose, l'adresse d'une personne. Cette idée de ressource s'aligne bien avec une URL (Unique Resource Locator). Une URL doit identifier une ressource unique sur le Web. Avec ces deux idées, nous avons moyen d'entreprendre des actions d'application communes sur quelque chose qui est identifié de manière unique sur le Web.

L'API REST Etudiant

Pour notre exemple de programme, vous allez créer une API REST donnant accès à une collection d'étudiants, stockée dans MONGODB, sur laquelle nous allons appliquer des opérations du CRUD. Voici la conception de l'API :

action	HTTP	URL	description
Créer	POST	/api/etudiants	URL unique pour créer un nouvel étudiant en base. Les données sont à récupérer dans le POST.
Lire tous les étudiants	GET	/api/etudiants	URL unique pour retourner la liste complète de tous les étudiants
Lire un étudiant	GET	/api/etudiants/id	URL unique pour lire les données d'un étudiant unique, identifier par son id.
Mettre à jour	PUT	/api/etudiants/id	URL unique pour modifier les données d'un étudiant unique, identifier par son id.
Effacer	DELETE	/api/etudiants/id	URL unique pour supprimer les données d'un étudiant unique, identifier par son id.

Chaque URL est aussi appelée endpoint.

L'API avec FASTAPI

Pour commencer, créez une classe Etudiant héritant de BaseModel, contenant nom, prénom, âge et classe.

Créez la base MONGODB correspondante.

Une fois que tout est en place, vous allez pouvoir créer une API Web à l'aide de FASTAPI. Mettez en place votre serveur :

```
from fastapi import FastAPI, HTTPException
from data import DataAccess as da

app = FastAPI()

@app.get("/")
def get_index():
    return {"message":"hello"}
```

Testez le serveur avec la commande suivante :

```
uvicorn main:app --reload --host 0.0.0.0 --port 8081
```

Codez les routes (ou endpoints) correspondantes aux URL proposées ci-dessus :

```
@app.get("/api/etudiants/<int:id>")
def get_etudiant(id):
    pass

@app.delete("/api/etudiants/<int:id>")
def del_etudiant(id):
    pass

…
```

Enfin, il faut que votre API retourne un message d'erreur lorsqu'une url ne correspond à aucun endpoint :

```
@app.errorhandler(404)
def page_not_found(e):
    return "erreur", 404
```

Vous devez, évidement, compléter ce code avec des accès à la base MONGODB, pour ajouter, modifier, supprimer ou lire les données. L'idéale serait, évidement, d'écrire tout le code d'accès à la base dans un fichier séparé.
Aucune de vos fonctions ne retourne de page HTML. Elles retournent toutes des informations sous forme JSON.

Application test en FLASK

Vous allez maintenant pouvoir créer une application web qui va consommer votre API. Vous pouvez, dans un autre projet, réutiliser FLASK, pour préparer un site :

- Avec la liste des étudiants en page d'accueil (obtenu à travers l'API),

- Lorsqu'un étudiant est cliqué, toutes les infos le concernant s'affichent sur une nouvelle page,
- Cette nouvelle page propose 2 boutons, un de mise à jour des informations, l'autre de suppression,
- En cas de suppression, il faut retourner à la page d'accueil,
- La mise à jour ouvre un formulaire de modification des données,
- Sur la page d'accueil, il y a, aussi, un lien vers un formulaire de saisie pour ajouter un nouvel étudiant.

Toutes ces fonctionnalités font appel à l'API. Toutes les actions sur la base MONGODB ne peuvent que se faire grâce à l'API. En aucun cas, le code de cet application test ne doit accéder à MONGODB.

Pour consommer votre WEB API, utilisez la bibliothèque **requests**, comme déjà vue pour les connexions d'API. Cette bibliothèque propose toutes les méthodes http dont vous avez besoin :

```
import requests

#envoi d'une requête en get
reponse = requests.get("http://127.0.0.1:5001/api/etudiants")
#envoi d'une requête en post
attributs = {"nom":"doe", "prenom":"john", "age":18, "classe":"cinquième"}
requests.post("http://127.0.0.1:5001/api/etudiants", data= attributs)
#envoi d'une requête en get avec un id
reponse = requests.get("http://127.0.0.1:5001/api/etudiants/"+id)
#envoi d'une requête en delete
requests.delete("http://127.0.0.1:5001/api/etudiants/"+id)
#envoi d'une requête en put
attributs = {"nom":"doe", "prenom":"john", "age":18, "classe":"cinquième"}
requests.put("http://127.0.0.1:5001/api/etudiants/"+id, data=attributs)
```

Page d'accueil

Voici un exemple de page d'accueil avec la liste des étudiants :

Chaque nom d'étudiant est un lien vers une page d'infos le concernant.

Dans l'appli test
La fonction correspondant à la route par défaut envoie une requête l'API WEB sur l'URL demandant la lecture de tous les étudiants, en GET, met sous forme de JSON la réponse et affiche la liste des étudiants sur la page d'accueil.

```
@app.route("/")
def index():
    reponse = requests.get(URL_API)
    content = json.loads(reponse.content.decode("utf-8"))

    return render_template("index.html", etudiants=content)
```

La liste, dans le HTML est généré dynamiquement (exemple à partir d'un JSON):

```
            <ul>
                {% for etudiant in etudiants : %}
                <l> <a href="{{url_for('get_etudiant', id=etudiant['_id'])}}"
id="etudiant">{{etudiant['nom']}} {{etudiant['prenom']}} - {{etudiant['classe'
]}}</a></li>
                {% endfor %}
            </ul>
```

Notez que les *id* des étudiants sont ajoutés à l'URL du lien généré. Ainsi, nous pouvant identifier un étudiant unique à partir de l'URL.

Dans l'API WEB
La fonction requêtée se connecte à la base, lit tous les étudiants en base, se déconnecte et retourne la liste sous forme de JSON, ainsi que le code 200. Voici un exemple, avec la connexion sous forme d'objet :

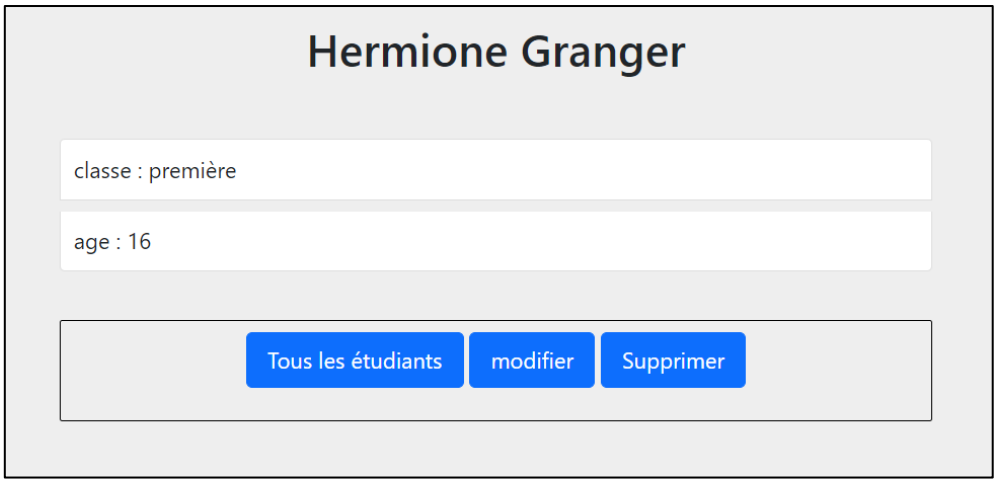
```
@app.route("/api/etudiants", methods=['GET'])
def get_etudiants():
    da.connexion()
    etudiants = da.get_etudiants()
    da.deconnexion()

    return jsonify(etudiants), 200
```

Accès à la base
Il faut tout simplement une fonction qui lit tous les étudiants en base.

```
    @classmethod
    def get_etudiants(cls):
        etudiants = cls.db.etudiants.find({})
        return list(etudiants)
```

Page étudiant

Voici un exemple de page d'infos pour un étudiant :



Dans l'appli test
L'id de l'étudiant à afficher est dans l'URL. Il faut donc la récupérer pour la passer dans l'URL de l'API demandant la lecture d'un étudiant précis. La réponse reçue est mise sous forme de JSON et envoyé à la page HTML pour affichage.

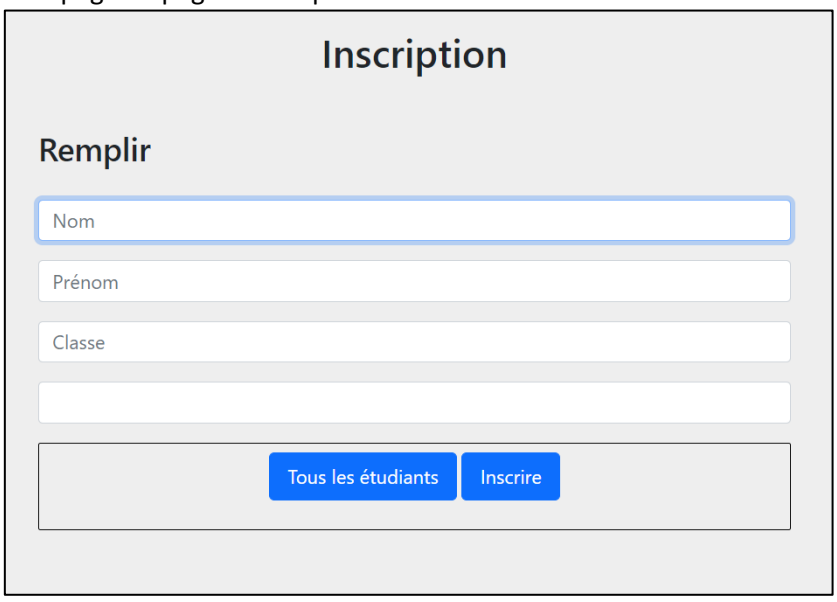
Sur la page HTML, les liens permettant de modifier et de supprimer un étudiant doivent aussi contenir l'id, pour savoir quel étudiant est concerné.

Dans l'API WEB
La fonction requêtée se connecte à la base, cherche l'étudiant correspondant à l'id, se déconnecte et retourne la liste sous forme de JSON, ainsi que le code 200.

Accès à la base
Il faut tout simplement une fonction qui lit le document correspondant à l'id transmis.

Page d'inscription

Voici un exemple de page de page d'inscription :



Dans l'appli test
Une fonction affiche le formulaire.

Une autre fonction, une fois les champs remplis, envoie les infos vers l'API, par méthode POST, à l'url d'ajout d'étudiant. Puis l'utilisateur est redirigé vers la page d'accueil.

Attention, il vous faut 2 URL :

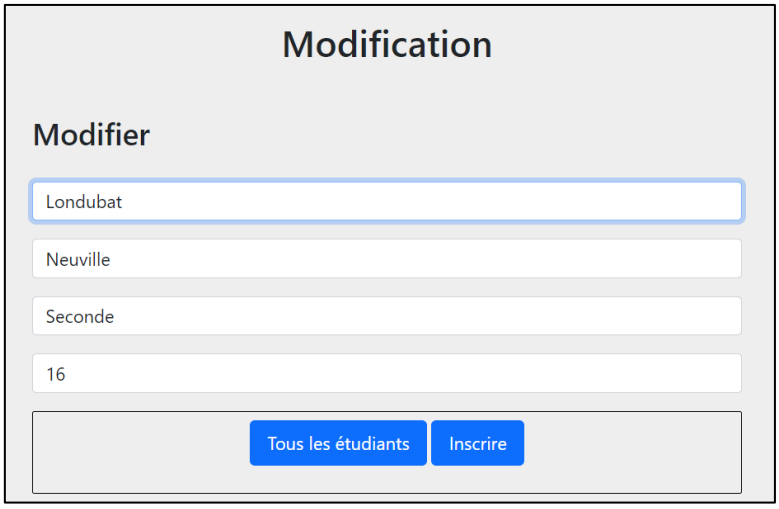
- Une pour afficher la page d'inscription,
- Une autre pour traiter le POST et faire la redirection.

Dans l'API WEB
La fonction récupère toutes les données à ajouter en base, puis, comme d'habitude, fait appel à la base pour se connecter, ajouter un nouveau document, et se déconnecter. Elle renvoie, sous forme JSON, un message OK, ainsi que le code 200.

Accès à la base
Il faut une fonction qui génère un nouvel id (en incrémentant, par exemple, le dernier en base) et créée un nouveau document à partir des données reçues.

Page de modification

Cette page ressemble à celle d'inscription, mais a les champs préremplis avec les données de l'étudiant sélectionné :



Dans l'appli test
Une première fonction doit, tout d'abord, afficher le formulaire ainsi que l'étudiant identifié par son id, passé dans l'URL.
Une deuxième fonction, une fois les champs remplis, envoie toutes les infos vers l'API, par méthode PUT, à l'url de mise à jour. Puis l'utilisateur est redirigé vers la page d'accueil.

Attention, il vous faut 2 URL :

- Une pour afficher la page d'inscription,
- Une autre pour traiter le POST et faire la redirection.

Dans l'API WEB
La fonction récupère toutes les données à ajouter en base, puis, comme d'habitude, fait appel à la base pour se connecter, modifie le document identifié par son id, et se déconnecter. Elle renvoie, sous forme JSON, un message OK, ainsi que le code 200.

Accès à la base
Il faut une fonction qui fasse la mise à jour du bon document (grâce à l'id) à partir des données reçues.

Le lien de suppression

Pas de page, le lien redirige vers la page d'accueil.

Dans l'appli test
L'URL de suppression envoie l'id de l'étudiant à effacer à l'API, par URL, via une méthode DELETE.

Dans l'API WEB
La fonction récupère dans l'URL l'id du document à supprimer, se connecte à la base, supprime le document, et se déconnecte. Elle renvoie, sous forme JSON, un message OK, ainsi que le code 200.

Accès à la base
Il faut une fonction qui supprime un document suivant l'id reçu.