

# Créer une API RESTFUL à l'aide de Python

## (FLASK RETFUL CRUD API) et MySQL

PROJECT : API pour une station météo connectée Meteous v1.0

Réalisé par : Ismail Mouyahada

Piloter par :

Pr. Benoît Berenger | | Pr. Julien Toutain

## Introduction

L'API RESTFUL est un concept avancé de développement Web qui est implémenté sur le serveur et utilisé avec la méthode HTTP ( GET/ POST/ PUT/ DELETE ) pour gérer les données. Les méthodes et ressources de requête HTTP sont identifiées à l'aide d'URL et gérées en conséquence.

Si vous envisagez d'implémenter l'API Flask RESTFUL pour récolter les données d'une station météo, Nous allons vous guider dans cette documentation comprendre et apprendrez à créer une API Flask RESTFUL à l'aide de MYSQL et Postman pour débbugger et tester les requêtes.

Nous couvrirons cette documentation avec un exemple en direct avec des méthodes C.R.U.D HTTP telles que (GET/ POST/ PUT/ DELETE) pour implémenter l'API Python RESTFUL avec des opérations CRUD.

## Prérequis

Pour pouvoir manipuler l'API que nous allons créer par la suite, et Nous espérons que vous avez installé Python sur votre Windows ou Linux avec Python en l'occurrence Flask et ses packages. Ici, nous utilisons la version Python 3.10. Nous importer dans notre projet les modules et packages de [flask](#) et [mysql](#) .

Pour créer notre API RESTFUL en utilisant Python (Flask) et MySQL. Nous avons un répertoire de projet [Dossier : API] et les principaux fichiers sont :

[./config.py](#)

Nous allons procéder par la suite à coder les configurations nécessaires pour se connecter à notre base de données

[./app.py](#)

Qui va contenir le module instance Flask l'API Flask.

[./api\\_executer.py](#)

C'est ce fichier qui va contenir la plupart notre code, Il est aussi le fichier à exécuter à l'aide de python pour activer ou désactiver l'API, mais aussi pour ajouter des éléments sur la base de données

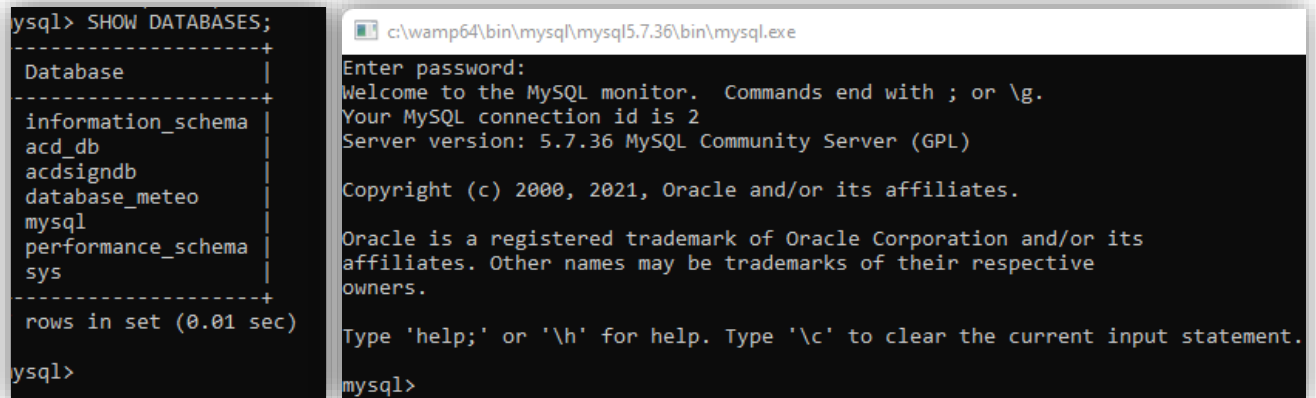
## Étape 1 : Créer une table de base de données MySQL

Comme nous allons implémenter l'API RESTful pour effectuer des opérations CRUD, nous allons donc créer des tables dans notre base de données MySQL et pour cela, nous avons le choix entre créer une base de données

en utilisant la ligne de commande de votre choix d'OS, ou à l'aide d'un gestionnaire de DB comme [phpmyadmin](#), [adminer](#) ..etc, ou générer notre base de données en utilisant python lui-même.

Pour tester notre API, veuillez télécharger le fichier [./database\\_meteo.sql](#)

Ensuite l'importer depuis votre gestionnaire de DB ou utiliser le terminal considérant que vous avez déjà un MySQL ou mariaDB installer sur votre machine



The image shows two terminal windows side-by-side. The left window shows the output of the 'SHOW DATABASES;' command in MySQL, listing databases like information\_schema, acd\_db, acdsigndb, database\_meteo, mysql, performance\_schema, and sys. The right window shows the MySQL command prompt with a password prompt and a welcome message, indicating the server version is 5.7.36 MySQL Community Server (GPL).

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| acd_db          |
| acdsigndb       |
| database_meteo  |
| mysql           |
| performance_schema |
| sys             |
+-----+
rows in set (0.01 sec)

mysql>
```

```
c:\wamp64\bin\mysql\mysql5.7.36\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.36 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

## Étape 2 : Importer le module Flask

Comme nous gérons la fonctionnalité de l'API REST à l'aide de Flask et MySQL, nous aurons donc besoin des deux modules. Le module Flask fonctionne comme un framework Web tandis que le module MySQL nécessite d'établir une connexion avec la base de données MySQL. Nous allons donc créer le app.py script Python, importer le module flask et créer l'instance flask à utiliser avec MySQL module.

```
from flask import Flask # Pour avoir accès au module Flask
from flask_cors import CORS, cross_origin # Gérer le cross-origin

app = Flask(__name__) # Déclarer l'instance Flask
CORS(app) # Englober l'instance dans le cross-origin
```

## Etape 3 : Créer une connexion MySQL

Nous allons créer config.py un fichier Python pour initialiser les détails de connexion à la base de données MySQL afin d'établir une connexion avec la base de données MySQL. Nous allons importer le **app** script pour gérer la connexion à la base de données MySQL avec le module Flask .

De l'application d'importation de l'application

```
from app import app # Importer le fichier d'instance Flask app.py
from config import mysql # importer le fichier de la connexion
import pymysql # importer le package pour gérer la connexion MySQL
from flask import jsonify # importer le package de JSON depuis flask
```

```
from flask import request # importer le package des requêtes CRUD depuis Flask
```

## Étape 4 : Créer une opération CRUD d'API REST

Nous allons créer un `api_executer.py` script et importer les modules **app** et **config**. Nous nous connecterons à la base de données MySQL et implémenterons les opérations CRUD en définissant tous les URL REST. Ici, nous avons utilisé POST la méthode HTTP pour ajouter de nouveaux enregistrements à la base de données. Nous avons utilisé GET la méthode HTTP pour obtenir tous les enregistrements ou un enregistrement individuel ou globale et utilisé PUT la méthode HTTP pour la mise à jour des enregistrements. DELETE Méthode HTTP également implémentée pour supprimer l'enregistrement. S'il n'y a pas d'enregistrement trouvé, il a défini la méthode 404 pour gérer l'erreur introuvable.

```
from app import app # Importer le fichier d'instance Flask app.py
from config import mysql # importer le fichier de la connexion
import pymysql # importer le package pour gérer la connexion
MySQL

from flask import jsonify # importer le package de JSON depuis flask
from flask import request # importer le package des requêtes CRUD
depuis Flask

# Add a new data into the database rows [table name : donnees]
@app.route('/api/v1/ajouter/', methods=['POST'])
def ajouter_donnees():

    _json = request.json
    _fahrenheit = _json['fahrenheit']
    _temperature = _json['temperature']
    _humidity = _json['humidite']
    _capteur = _json['capture']

    if _temperature and _humidity and _capteur and _fahrenheit and request.method == 'POST':
        sqlQuery = "INSERT INTO donnees( temperature, humidite, capture,fahrenheit) VALUES(%s, %s, %s, %s)"
        bindData_add_don = ( _temperature, _humidity, _capteur,_fahrenheit)
        conn = mysql.connect()
        cursor_add_don = conn.cursor()
        cursor_add_don.execute(sqlQuery, bindData_add_don)
        conn.commit()
        response_add_don = jsonify('les données ont été enregistrées avec succès !')
        response_add_don.status_code = 200
        return response_add_don
    else:
        return not_found()

# fetch all rows from database concerning the table utilisateurs [table name : donnees]
@app.route('/api/v1/donnees/')
def lire_donnees():
    try:
        conn = mysql.connect()
        cursor_read_don = conn.cursor(pymysql.cursors.DictCursor)
        cursor_read_don.execute("SELECT id, fahrenheit, temperature, humidite, date, capture FROM donnees")
        empRows = cursor_read_don.fetchall()
        response_read_don = jsonify(empRows)
```

```

        response_read_don.status_code = 200
    return response_read_don
except Exception as e:
    print(e)
finally:
    cursor_read_don.close()
    conn.close()

@app.route('/api/v1/donnees/humidite/<int:id>')
def fitlrer_Humidite(id):
    try:
        conn = mysql.connect()
        cursor_hum = conn.cursor(pymysql.cursors.DictCursor)
        cursor_hum.execute("SELECT id, humidite, date, capture FROM donnees
WHERE id =%s", id)
        empRow = cursor_hum.fetchone()
        response_filt_hum = jsonify(empRow)
        response_filt_hum.status_code = 200
        return response_filt_hum
    except Exception as e:
        print(e)
    finally:
        cursor_hum.close()
        conn.close()

@app.route('/api/v1/donnees/temperature/<int:id>')
def filtrer_temperature(id):
    try:
        conn = mysql.connect()
        cursor_temp = conn.cursor(pymysql.cursors.DictCursor)
        cursor_temp.execute("SELECT id, fahrenheit, temperature, date,
capture FROM donnees WHERE id =%s", id)
        empRow = cursor_temp.fetchone()
        response_filt_temp = jsonify(empRow)
        response_filt_temp.status_code = 200
        return response_filt_temp
    except Exception as e:
        print(e)
    finally:
        cursor_temp.close()
        conn.close()

@app.route('/api/v1/modifier/', methods=['PUT'])
def modifier_donnees():
    try:
        _json = request.json
        _fahrenheit = _json['fahrenheit']
        _id = _json['id']
        _temperature = _json['temperature']
        _humidity = _json['humidite']
        _date = _json['date']
        _capteur = _json['capture']
        if _id and _fahrenheit and _temperature and _humidity and _date and
        _capteur and request.method == 'PUT':
            sqlQuery = "UPDATE donnees SET fahrenheit=%s, temperature=%s,
humidite=%s, date=%s, capture=%s WHERE id=%s"
            bindData_mod_don = (_id, _fahrenheit, _temperature, _humidity,
            date, _capteur)

```

```

        conn = mysql.connect()
        cursor_mod_don = conn.cursor()
        cursor_mod_don.execute(sqlQuery, bindData_mod_don)
        conn.commit()
        response_mod_don = jsonify('les données ont été modifié avec
succès')
        response_mod_don.status_code = 200
        return response_mod_don
    else:
        return not_found()
except Exception as e: \
    print(e)
finally:
    cursor_mod_don.close()
    conn.close()

@app.route('/api/v1/supprimer/<int:id>', methods=['DELETE'])
def supprimer_donnees(id):
    try:
        conn = mysql.connect()
        cursor_sup_don = conn.cursor()
        cursor_sup_don.execute("DELETE FROM donnees WHERE id=%s", id)
        conn.commit()
        response_sup_don = jsonify('Données ont été supprimées avec
succès!')
        response_sup_don.status_code = 200
        return response_sup_don
    except Exception as e:
        print(e)
    finally:
        cursor_sup_don.close()
        conn.close()

@app.errorhandler(404)
def not_found(error=None):
    message = {
        'status': 404,
        'message': 'Enregistrement introuvable mais l\'api marche : ' +
request.url,
    }
    response_error = jsonify(message)
    response_error.status_code = 404
    return response_error

if __name__ == "__main__":
    # app.run(host='0.0.0.0', port=5000) # Si vous voulez personnaliser
    app.run()

```

## Étape 5 : exécuter l'application

Maintenant, nous allons aller dans le répertoire du projet API et exécuter la commande python **api\_executer.py** et le serveur démarrera sur le port par défaut 5000. Nous allons maintenant utiliser Postman pour exécuter notre API Python RESTFUL avec les méthodes (POST, GET, PUT ou DELETE) pour le tester.

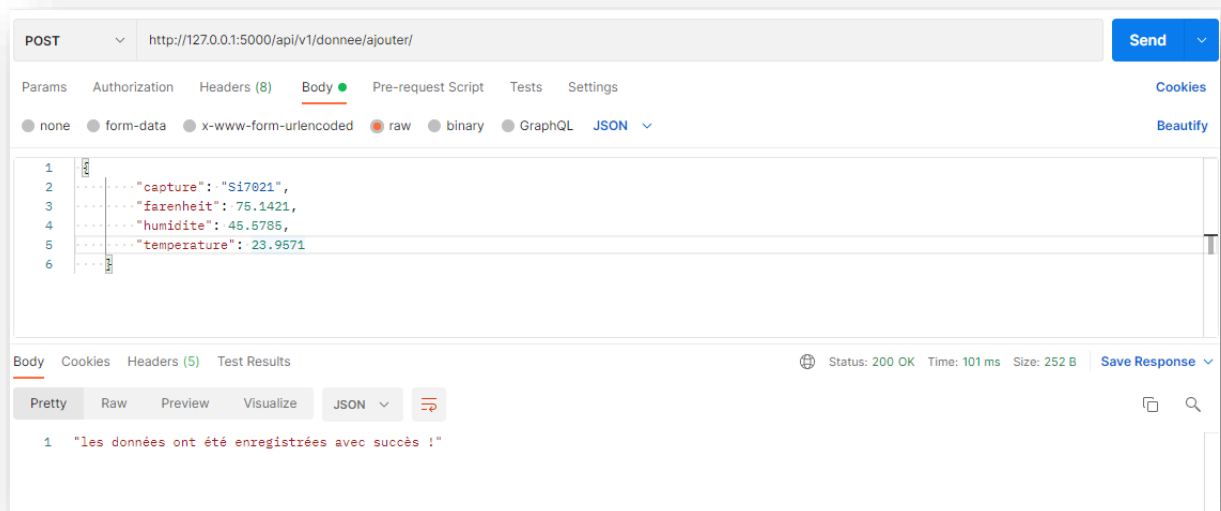
Pour exécuter depuis linux :

```
# chmod u+x ./chemin_vers_fichier/api_executer.py
# sudo python3 api_executer.py
Pour exécuter depuis windows:
# Pour windows utiliser : python ./chemin vers fichier/api_executer.py
Utiliser pycharm pour ceux qui préfèrent les Gui
```

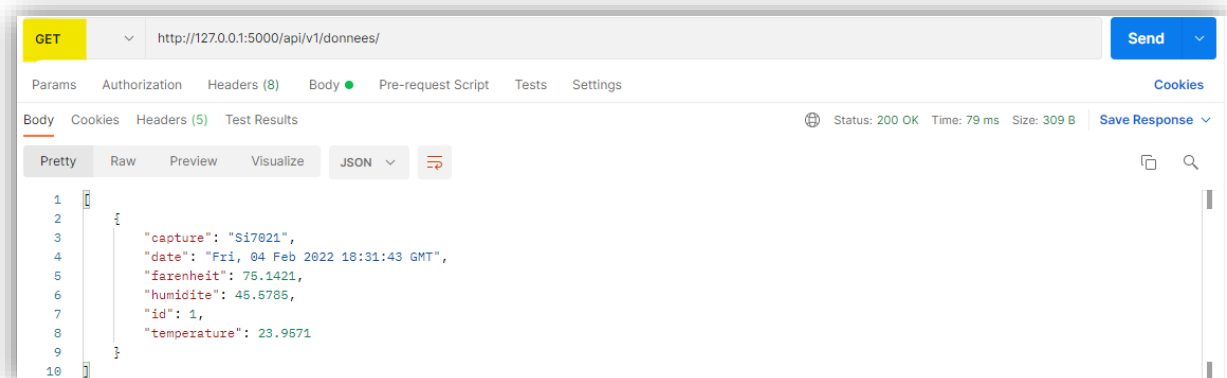
Nous exécuterons l'URL ci-dessous avec la GET méthode HTTP pour obtenir tous les données de la station météo et afficher ces données au format JSON.

Nous obtiendrons l'enregistrement des données dans les données JSON avec l'identifiant 1 en utilisant l'URL ci-dessous avec la GET méthode HTTP.

[http : //127.0.0.1:5000/api/v1/donnee/ajouter/](http://127.0.0.1:5000/api/v1/donnee/ajouter/)



La réponse sera des données JSON :

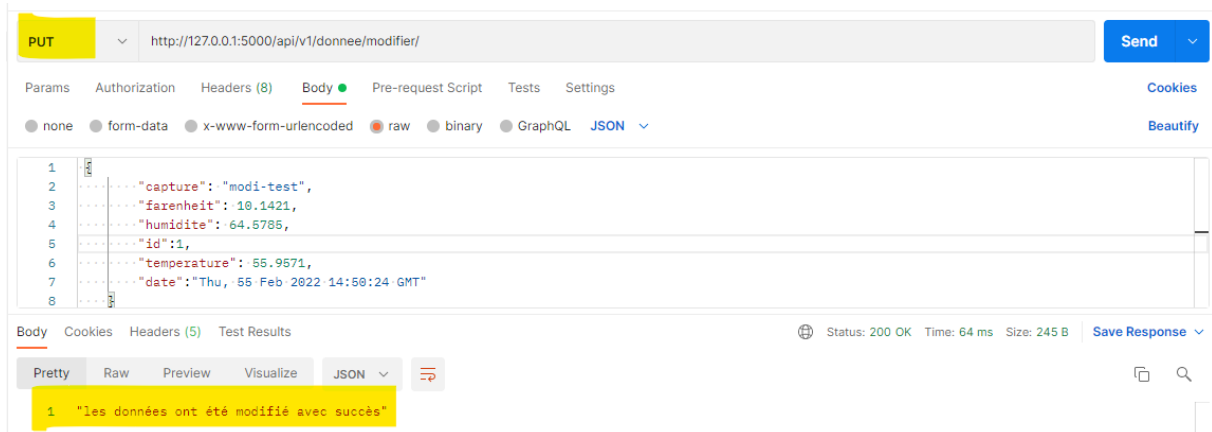


La réponse de l'employé de données JSON ajoutera un message.

Nous mettrons à jour l'enregistrement de l'existant avec le numéro 1 à l'aide de la PUT méthode HTTP.

<http://127.0.0.1:5000/donnee/modifier/>

Le corps de la requête sera le suivant :

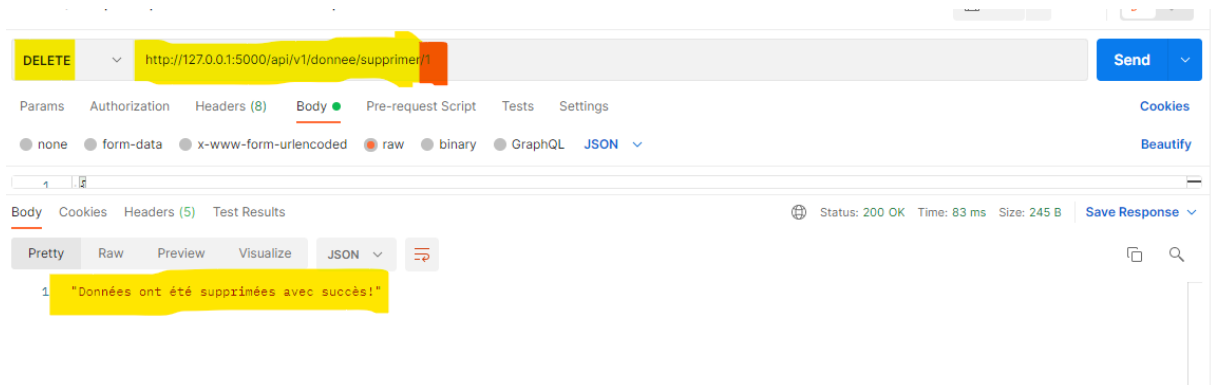


La réponse sera un message d'enregistrement des données

Nous supprimerons l'employé existant avec l'identifiant #1 en utilisant la DELETE méthode HTTP.

<http://127.0.0.1:5000/donnee/delete/1>

La réponse sera le message de suppression de l'enregistrement des données.



Retrouver la liste de toutes les requêtes possible ici.

<http://127.0.0.1:5000/api/v1/utilisateurs/>

<http://127.0.0.1:5000/api/v1/utilisateur/ajouter/>

<http://127.0.0.1:5000/api/v1/utilisateur/modifier/>

[http://127.0.0.1:5000/api/v1/utilisateur/supprimer/\\${id}](http://127.0.0.1:5000/api/v1/utilisateur/supprimer/${id})

<http://127.0.0.1:5000/api/v1/donnees/>



-----  
http://127.0.0.1:5000/api/v1/donnees/humidite/{id}

-----  
http://127.0.0.1:5000/api/v1/donnees/temperature/{id}

-----  
http://127.0.0.1:5000/api/v1/donnee/ajouter/

-----  
http://127.0.0.1:5000/api/v1/donnee/modifier

-----  
http://127.0.0.1:5000/api/v1/donnee/supprimer/\${id}

-----  
http://127.0.0.1:5000/api/v1/capteurs/

-----  
http://127.0.0.1:5000/api/v1/capteur/ajouter/

-----  
http://127.0.0.1:5000/api/v1/capteur/modifier

-----27.0.0.1-----  
http://127.0.0.1:5000/api/v1/capteur/supprimer/\${id}

Vous avez terminé notre documentations sur l'API Python RESTFUIL en utilisant Flask et MySQL avec des exemples sur ma machine. Vous pouvez ensuite l'implémenter dans votre projet en fonction de vos besoins. Si vous avez des questions, vous pouvez soumettre me les envoyer via ma boîte mail : [ismail.mouyahada@viacesi.fr](mailto:ismail.mouyahada@viacesi.fr)

**Projet réalisé le : 04-02-2022**