



API RESTful (Representational State Transfer):

Una API RESTful es un conjunto de reglas y convenciones que se utiliza para crear y gestionar servicios web que se comportan de una manera específica. En lugar de describir una tecnología o protocolo en particular, REST es una arquitectura que utiliza el protocolo HTTP para realizar operaciones en recursos, que pueden ser datos, servicios o información en un servidor.

Métodos en una API RESTful:

POST (Crear): Este método se utiliza para crear un nuevo recurso en el servidor. Por ejemplo, al enviar una solicitud POST a una API de redes sociales, puedes crear un nuevo mensaje.

GET (Leer): El método GET se utiliza para recuperar información de un recurso existente en el servidor. Cuando visitas una página web en tu navegador, estás realizando una solicitud GET para obtener la página.

PUT (Actualizar): Se usa para actualizar un recurso existente en el servidor. Si deseas modificar la información de un perfil de usuario en una aplicación, puedes utilizar una solicitud PUT.

DELETE (Eliminar): Como su nombre indica, este método se utiliza para eliminar un recurso en el servidor. Si deseas eliminar un comentario en una publicación en línea, puedes hacerlo mediante una solicitud DELETE.



Flask es un microframework web escrito en Python. Es "micro" en el sentido de que proporciona las herramientas esenciales para construir aplicaciones web, pero es lo suficientemente flexible como para que puedas agregar las bibliotecas y extensiones que necesites según tus requerimientos. Flask se utiliza comúnmente para crear aplicaciones web ligeras y rápidas, incluidas las API RESTful.

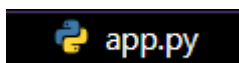
Trabajo Práctico:

Crearemos una API web simple para gestionar información de personas en una base de datos MySQL utilizando Flask y SQLAlchemy. Puedes realizar operaciones como agregar, listar, buscar, modificar y eliminar personas a través de las rutas definidas en la API.

- Instalaremos librerías que usaremos con los siguientes comandos:

```
pip install Flask
pip install Flask-SQLAlchemy mysql-connector-python
```

- A continuación, crea un archivo llamado app.py con el siguiente contenido:



1. Importaciones:

Flask: Importa la clase Flask de Flask, que se utiliza para crear y configurar la aplicación web.

request: Importa el objeto request, que se utiliza para acceder a los datos enviados en las solicitudes HTTP.

jsonify: Importa la función jsonify, que se utiliza para convertir objetos Python en respuestas JSON:

```
from flask import Flask, request, jsonify
```

SQLAlchemy: Importa la clase SQLAlchemy de Flask-SQLAlchemy, que se utiliza para interactuar con la base de datos.

```
from flask_sqlalchemy import SQLAlchemy
```

2. Creación de la aplicación Flask:

Se crea una instancia de la clase Flask y se asigna a la variable app.

```
app = Flask(__name__)
```

Se configura la base de datos MySQL mediante la línea

app.config['SQLALCHEMY_DATABASE_URI'], donde se especifica la ubicación de la base de datos.

```
app.config['SQLALCHEMY_DATABASE_URI'] =  
'mysql+mysqlconnector://root:@localhost/persona'  
db = SQLAlchemy(app)
```

3. Definición del modelo de datos:

Se define la clase Persona que hereda de db.Model, que representa una tabla en la base de datos.

```
class Persona(db.Model):
```

Se definen las columnas de la tabla, como id, nombre, apellido, email y dni.

El método __init__ se utiliza para inicializar una instancia de la clase Persona con los datos proporcionados.

```
id = db.Column(db.Integer, primary_key=True)  
nombre = db.Column(db.String(80), nullable=False)  
apellido = db.Column(db.String(80), nullable=False)  
email = db.Column(db.String(120), unique=True, nullable=False)  
dni = db.Column(db.String(15), unique=True, nullable=False)  
  
def __init__(self, nombre, apellido, email, dni):  
    self.nombre = nombre  
    self.apellido = apellido  
    self.email = email  
    self.dni = dni
```

4. Definición de rutas y funciones:

agregar_persona(): Esta función maneja las solicitudes POST en la ruta '/personas'. Crea una nueva instancia de Persona con los datos proporcionados en la solicitud JSON, la agrega a la base de datos y la guarda.

```
# Agregar  
@app.route('/personas', methods=['POST'])  
def agregar_persona():  
    data = request.get_json()  
    nueva_persona = Persona(  
        nombre=data['nombre'], apellido=data['apellido'],  
email=data['email'], dni=data['dni'])  
    db.session.add(nueva_persona)  
    db.session.commit()  
    return jsonify({'mensaje': 'Persona agregada exitosamente'}), 201
```

obtener_personas(): Esta función maneja las solicitudes GET en la ruta '/personas'. Obtiene todas las personas de la base de datos y las devuelve como una lista de objetos JSON.

```
# Listar
@app.route('/personas', methods=['GET'])
def obtener_personas():
    personas = Persona.query.all()
    personas_json = [{'nombre': persona.nombre, 'apellido':
persona.apellido,
                        'email': persona.email, 'dni': persona.dni} for
persona in personas]
    return jsonify(personas_json), 200
```

buscar_persona(): Esta función maneja las solicitudes POST en la ruta '/personas/buscar'. Busca una persona por su número de identificación (DNI) en la base de datos y la devuelve como un objeto JSON si se encuentra.

```
# Buscar
@app.route('/personas/buscar', methods=['POST'])
def buscar_persona():
    data = request.get_json()
    dni = data.get('dni')
    if dni:
        persona = Persona.query.filter_by(dni=dni).first()
        if persona:
            persona_json = {'nombre': persona.nombre, 'apellido':
persona.apellido,
                            'email': persona.email, 'dni': persona.dni}
            return jsonify(persona_json), 200
    return jsonify({'mensaje': 'Persona no encontrada'}), 404
```

modificar_persona(): Esta función maneja las solicitudes PUT en la ruta '/personas'. Modifica los datos de una persona existente en la base de datos según el DNI proporcionado en la solicitud JSON.

```
# Modificar
@app.route('/personas', methods=['PUT'])
def modificar_persona():
    data = request.get_json()
    dni = data.get('dni')
    if dni:
        persona = Persona.query.filter_by(dni=dni).first()
        if persona:
            persona.nombre = data.get('nombre', persona.nombre)
            persona.apellido = data.get('apellido', persona.apellido)
            persona.email = data.get('email', persona.email)
```

```

        db.session.commit()

        return jsonify({'mensaje': 'Persona modificada
exitosamente'}), 200

    return jsonify({'mensaje': 'Persona no encontrada'}), 404

```

borrar_persona(): Esta función maneja las solicitudes DELETE en la ruta '/personas/<dni>'. Elimina a una persona de la base de datos según su DNI.

```

# Eliminar
@app.route('/personas/<dni>', methods=['DELETE'])
def borrar_persona(dni):
    persona = Persona.query.filter_by(dni=dni).first()
    if persona:
        db.session.delete(persona)
        db.session.commit()
        return jsonify({'mensaje': 'Persona eliminada exitosamente'}),
200
    return jsonify({'mensaje': 'Persona no encontrada'}), 404

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)

```

5. Inicialización de la base de datos:

Se utiliza with app.app_context() para asegurarse de que las operaciones relacionadas con la base de datos se realicen en el contexto de la aplicación Flask.

```

if __name__ == '__main__':
    with app.app_context():

```

Se llama a db.create_all() para crear las tablas en la base de datos si no existen.

```

    db.create_all()

```

6. Ejecución de la aplicación:

La aplicación Flask se ejecuta con app.run(debug=True) en el caso de que este archivo se ejecute directamente (no se importe como un módulo).

```

app.run(debug=True)

```

Código completo del ejercicio:

```

from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

```

```

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+mysqlconnector://root:@localhost/persona'
db = SQLAlchemy(app)

class Persona(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(80), nullable=False)
    apellido = db.Column(db.String(80), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    dni = db.Column(db.String(15), unique=True, nullable=False)

    def __init__(self, nombre, apellido, email, dni):
        self.nombre = nombre
        self.apellido = apellido
        self.email = email
        self.dni = dni

# Agregar
@app.route('/personas', methods=['POST'])
def agregar_persona():
    data = request.get_json()
    nueva_persona = Persona(
        nombre=data['nombre'], apellido=data['apellido'],
email=data['email'], dni=data['dni'])
    db.session.add(nueva_persona)
    db.session.commit()
    return jsonify({'mensaje': 'Persona agregada exitosamente'}), 201

# Listar
@app.route('/personas', methods=['GET'])
def obtener_personas():
    personas = Persona.query.all()
    personas_json = [{'nombre': persona.nombre, 'apellido':
persona.apellido,
                    'email': persona.email, 'dni': persona.dni} for
persona in personas]
    return jsonify(personas_json), 200

# Buscar

```

```

@app.route('/personas/buscar', methods=['POST'])
def buscar_persona():
    data = request.get_json()
    dni = data.get('dni')
    if dni:
        persona = Persona.query.filter_by(dni=dni).first()
        if persona:
            persona_json = {'nombre': persona.nombre, 'apellido':
persona.apellido,
                            'email': persona.email, 'dni': persona.dni}
            return jsonify(persona_json), 200
        return jsonify({'mensaje': 'Persona no encontrada'}), 404

# Modificar
@app.route('/personas', methods=['PUT'])
def modificar_persona():
    data = request.get_json()
    dni = data.get('dni')
    if dni:
        persona = Persona.query.filter_by(dni=dni).first()
        if persona:
            persona.nombre = data.get('nombre', persona.nombre)
            persona.apellido = data.get('apellido', persona.apellido)
            persona.email = data.get('email', persona.email)
            db.session.commit()
            return jsonify({'mensaje': 'Persona modificada
exitosamente'}), 200
        return jsonify({'mensaje': 'Persona no encontrada'}), 404

# Eliminar
@app.route('/personas/<dni>', methods=['DELETE'])
def borrar_persona(dni):
    persona = Persona.query.filter_by(dni=dni).first()
    if persona:
        db.session.delete(persona)
        db.session.commit()
        return jsonify({'mensaje': 'Persona eliminada exitosamente'}),
200
    return jsonify({'mensaje': 'Persona no encontrada'}), 404

if __name__ == '__main__':
    with app.app_context():

```

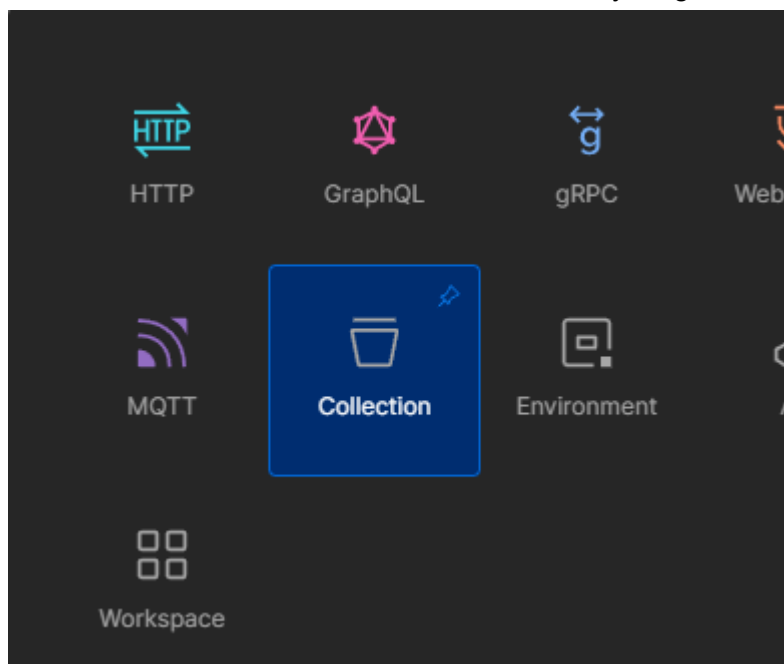
```
db.create_all()  
app.run(debug=True)
```

- Ejecuta la aplicación con `python app.py`




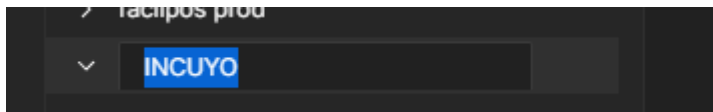
Postman es una herramienta de desarrollo y pruebas de API que se utiliza para simplificar el proceso de probar y depurar servicios web y API. Es una aplicación de escritorio que proporciona una interfaz de usuario intuitiva para enviar solicitudes HTTP a API y recibir respuestas, lo que facilita la interacción con servicios web y la verificación de su funcionalidad.

- Vamos a usar esta herramienta para crear, modificar, listar, y eliminar personas de nuestra base de datos.
1. Creamos una collection, vamos a new y elegimos "Collection"



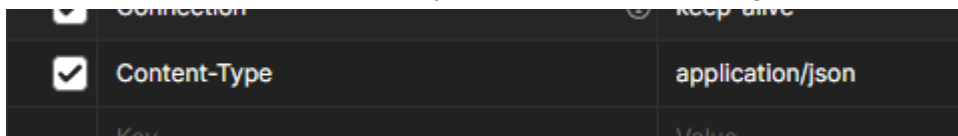
2. Cambiamos en nombre de la nueva collection, tocamos en los 3 puntitos

 y elegimos rename, y la llamamos INCUYO (o como más guste)



3. Ponemos en Add a request, y para cada uno de los métodos ponemos lo siguiente.

- a. vamos a  y al final colocamos lo siguiente:



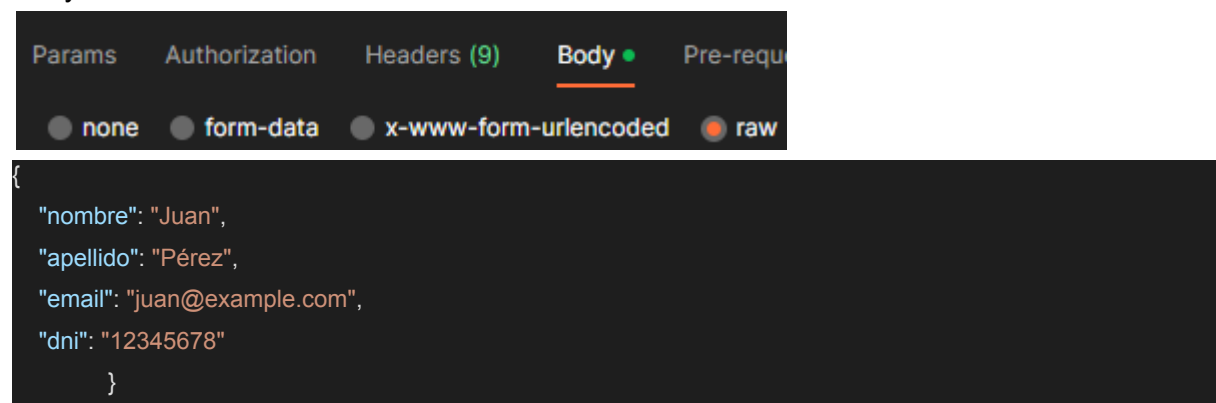
- b. Después vamos a body y configuramos según requiera la función.

- Nombre: Agregar una Persona

url: <http://localhost:5000/personas>

método: POST

Body/raw



- Nombre: Obtener todas las Personas

url: <http://localhost:5000/personas>

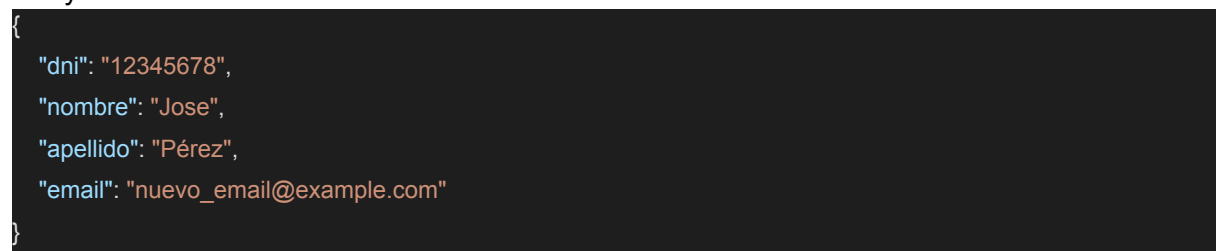
método: GET

- Nombre: Modificar una Persona

url: <http://localhost:5000/personas>

método: PUT

Body/raw



- Nombre: Buscar una Persona

url:<http://localhost:5000/personas/buscar>

método: POST

Body/raw

```
{  
  "dni": "12345678"  
}
```

- Nombre: Borrar una Persona

url:<http://localhost:5000/personas/borrar>

método: DELETE

Body/raw

```
{  
  "dni": "12345678"  
}
```