

Crear API Python MySQL



Ayúdame a crear una API en Python para extraer registros de la tabla "Curso" para una base de datos MySQL



Para crear una API en Python que extraiga registros de la tabla "Curso" en una base de datos MySQL, podemos utilizar el framework Flask para construir la API, junto con `mysql-connector-python` para interactuar con la base de datos MySQL.

A continuación te muestro un ejemplo básico de cómo hacerlo:

1. Instalación de Dependencias

Primero, necesitas instalar las dependencias necesarias:

```
bash
```

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

2. Configuración de la Base de Datos

Antes de empezar, debes configurar los parámetros de conexión a tu base de datos MySQL. Guarda esta configuración en el archivo Python.

3. Creación de la API con Flask

```
python
```

```
from flask import Flask, jsonify
import mysql.connector
from mysql.connector import Error

app = Flask(__name__)

# Configuración de la base de datos
def create_connection():
    connection = None
    try:
        connection = mysql.connector.connect(
            host="localhost", # Cambia esto si usas un host diferente
            user="tu_usuario", # Cambia esto por tu usuario de MySQL
            password="tu_contraseña", # Cambia esto por tu contraseña de MySQL
            database="nombre_base_datos" # Cambia esto por el nombre de tu base de datos
        )
    except Error as e:
        print(f"Error al conectar a MySQL: {e}")

    return connection

# Ruta para obtener todos los cursos
```

```
@app.route('/cursos', methods=['GET'])
def obtener_cursos():
    connection = create_connection()
    cursor = connection.cursor(dictionary=True)

    cursor.execute("SELECT * FROM Curso")
    cursos = cursor.fetchall()

    cursor.close()
    connection.close()

    return jsonify(cursos)

if __name__ == "__main__":
    app.run(debug=True)
```

4. Ejecución de la API

Guarda el código anterior en un archivo llamado `app.py` y ejecútalo con el siguiente comando:

```
bash
```

```
python app.py
```

La API estará disponible en `http://127.0.0.1:5000/cursos`. Puedes acceder a esta URL en tu navegador o usar herramientas como Postman o curl para probarla.

5. Personalización y Mejora

Puedes agregar más rutas, manejar errores de forma más detallada, e implementar autenticación, paginación, entre otras características, según tus necesidades.

¿Hay algún detalle específico que te gustaría ajustar o agregar a esta API?



Ahora ayudame a generar el código de una API para que por medio del método GET consulte un registro específico de la tabla curso



Para consultar un registro específico de la tabla "Curso" usando el método GET, puedes extender la API que creamos previamente añadiendo una nueva ruta que acepte un parámetro (como un ID de curso) y devuelva la información correspondiente.

1. Extensión de la API con una Nueva Ruta

Aquí tienes un ejemplo del código actualizado:

```
python
```

```
from flask import Flask, jsonify, request
import mysql.connector
from mysql.connector import Error
```

```

app = Flask(__name__)

# Configuración de la base de datos
def create_connection():
    connection = None
    try:
        connection = mysql.connector.connect(
            host="localhost", # Cambia esto si usas un host diferente
            user="tu_usuario", # Cambia esto por tu usuario de MySQL
            password="tu_contraseña", # Cambia esto por tu contraseña de MySQL
            database="nombre_base_datos" # Cambia esto por el nombre de tu base de datos
        )
    except Error as e:
        print(f"Error al conectar a MySQL: {e}")

    return connection

# Ruta para obtener todos los cursos
@app.route('/cursos', methods=['GET'])
def obtener_cursos():
    connection = create_connection()
    cursor = connection.cursor(dictionary=True)

    cursor.execute("SELECT * FROM Curso")
    cursos = cursor.fetchall()

    cursor.close()
    connection.close()

    return jsonify(cursos)

# Ruta para obtener un curso específico por ID
@app.route('/cursos/<int:curso_id>', methods=['GET'])
def obtener_curso_por_id(curso_id):
    connection = create_connection()
    cursor = connection.cursor(dictionary=True)

    cursor.execute("SELECT * FROM Curso WHERE id = %s", (curso_id,))
    curso = cursor.fetchone()

    cursor.close()
    connection.close()

    if curso:
        return jsonify(curso)
    else:
        return jsonify({"error": "Curso no encontrado"}), 404

if __name__ == "__main__":
    app.run(debug=True)

```

2. Explicación del Código

- **Ruta** `/cursos/<int:curso_id>`: Esta ruta acepta un parámetro `curso_id` en la URL, que corresponde al identificador del curso en la tabla "Curso".
- `cursor.execute("SELECT * FROM Curso WHERE id = %s", (curso_id,))`: Esta línea ejecuta una consulta SQL para obtener el curso específico cuyo `id` coincida con el `curso_id` proporcionado en la URL.
- `cursor.fetchone()`: Devuelve el primer (y en este caso, único) registro que coincide con la consulta. Si no se encuentra ningún curso con ese ID, `curso` será `None`.
- `if curso:`: Si el curso es encontrado, lo devuelve en formato JSON; si no, devuelve un mensaje de error con un código de estado 404.

3. Ejecución de la API

Al igual que antes, guarda el código en `app.py` y ejecútalo:

```
bash
```

```
python app.py
```

Para obtener un curso específico, realiza una solicitud GET a

`http://127.0.0.1:5000/cursos/{curso_id}`, reemplazando `{curso_id}` con el ID del curso que deseas consultar.

Por ejemplo:

```
arduino
```

```
http://127.0.0.1:5000/cursos/1
```

Este comando devolverá el curso con `id` igual a 1.

4. Consideraciones Adicionales

Puedes agregar validaciones adicionales, mejorar el manejo de errores, y extender la funcionalidad según las necesidades de tu aplicación.

¿Necesitas alguna otra funcionalidad o detalle específico en la API?

ChatGPT puede cometer errores. Considera verificar la información importante.