

REST API

Uma introdução à REST API com Flask

| Santana de Parnaíba | Johnny Matos | <https://github.com/DS-Johnny>  johnny.datascietist@gmail.com |
| --- | --- | --- |

**O que é uma API?**

Uma API (Application Programming Interface) ou “Interface de Programação de Aplicações” é um conjunto de regras que permitem que diferentes sistemas se comuniquem entre si.

Imagine que uma API seja uma ponte que conecta dois lugares diferentes. Quando um sistema precisa de informações de outro, ele usa a API para enviar um pedido e receber uma resposta.

Um exemplo prático:

Suponha que você use um aplicativo de clima no seu celular. Esse aplicativo não armazena todas as informações sobre o clima armazenadas nele. Em vez disso, ele faz uma requisição a uma API de previsão do tempo, que retorna os dados necessários, como temperatura e condições climáticas.

Como uma API funciona?

* Você (usuário ou sistema) faz um pedido (requisição)
* A API leva esse pedido para um sistema (servidor ou banco de dados)
* O sistema processa o pedido e devolve uma resposta (os dados ou ação solicitada)

Então o que é uma API REST?

REST (Representational State Transfer ou “Transferência de estado representacional”) é uma arquitetura de software que define como uma API deve funcionar e que tem sido muito utilizada na comunicação entre sistemas de computadores na internet atualmente.

Ela não é um protocolo ou tecnologia específica, mas sim um conjunto de princípios de design que guiam a comunicação entre sistemas em redes, como a internet.

Uma característica importante da REST API é a utilização de métodos HTTP como:

* “GET” - Para buscar informações
* “POST” - Para criar novos registros
* “PUT” - Para atualizar dados existentes
* “DELETE” - Para remover registros

Neste material, iremos abordar uma maneira simples para criar sua primeira REST API com Python utilizando o framework web Flask.

**Preparando o Ambiente**

Imagine que você é um desenvolvedor Python e seu gerente pediu que você criasse uma REST API para gerenciar o registro de alunos de uma escola. Para isso, utilizaremos algumas tecnologias essenciais para o desenvolvimento da nossa API:

Tecnologias utilizadas:

* Python 3 - Uma linguagem de programação de alto nível, amplamente utilizada para desenvolvimento web, automação, análise de dados e inteligência artificial. Sua sintaxe simples e extensa biblioteca padrão tornam o desenvolvimento mais rápido e eficiente.
* Flask - Um microframework para desenvolvimento web em Python. Ele é leve, flexível e ideal para criar APIs REST, pois permite construir aplicações com poucos recursos iniciais, adicionando funcionalidades conforme necessário.
* Sqlite3 - Um banco de dados relacional leve e incorporado, que não requer um servidor separado para operar. Ele é uma excelente escolha para aplicações de pequeno e médio porte, sendo ideal para projetos de aprendizado e prototipagem.
* Postman - Uma ferramenta utilizada para testar APIs. Com ela, podemos enviar requisições HTTP para nossa API, visualizar respostas e depurar possíveis erros sem precisar escrever código adicional para testes.

Primeiros Passos

Criar um ambiente virtual

Para garantir que nosso projeto fique isolado e não tenha conflitos de dependências com outros projetos, vamos criar um **ambiente virtual Python**. Isso facilita a instalação de pacotes específicos sem afetar o sistema como um todo.

(Neste passo a passo, consideraremos que o sistema operacional utilizado é Windows)

1- Abra o CMD no diretório do seu projeto e execute o seguinte comando:

> *py -m venv env*

Isso criará uma pasta chamada “env”, que será o seu ambiente virtual Python para esse projeto.

2- Ative o ambiente virtual com o comando:

> *env\scripts\activate*

Se tudo deu certo, o nome do ambiente virtual aparecerá entre parênteses antes do caminho do diretório, indicando que ele está ativo. Exemplo:

> *(env) C:\Users\Professor\Desktop\SENAI - Santana de Parnaíba\Aula API*

3- Agora só falta o Flask, o framework que utilizaremos para criar nossa API:

> *py -m pip install flask*

Após a instalação do Flask, seu ambiente estará pronto! Agora basta abrir o diretório do projeto na sua IDE de preferência. Para este guia, usaremos o VSCode como exemplo.

**Criando e testando as rotas.**

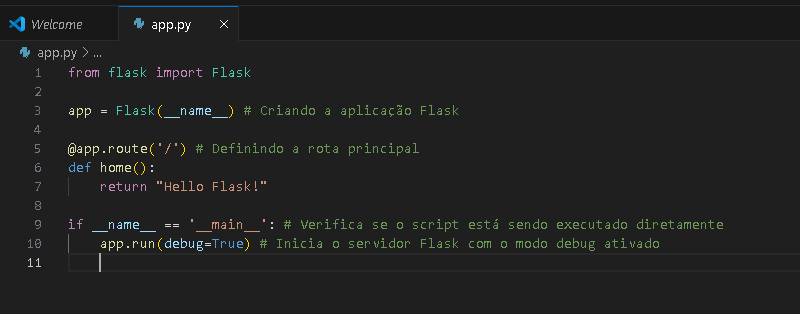
O que são rotas?

As rotas (routes) são responsáveis por definir quais URLs a aplicação deve responder e qual função deve ser executada quando essas URLs forem acessadas. No Flask, usamos o decorador *@app.route()* para definir cada rota.

Vamos botar a mão na massa!

Primeiro, crie um arquivo chamado app.py no diretório do seu projeto.

Agora vamos criar nossa primeira aplicação Flask. Digite no seu script Python o seguinte código e depois salve o arquivo.



Explicação:

Na linha 1 estamos importando a classe Flask da biblioteca flask, repare que a biblioteca se escreve com inicial minúscula e a classe se escreve com inicial maiúscula.

Na linha 3, *app = Flask(\_\_name\_\_)*, estamos criando uma instância da classe Flask, que representa nossa aplicação. No momentos não vamos nos aprofundar em relação ao parâmetro “\_\_name\_\_”, mas ele é importante para que o flask indentifique corretamente os arquivos da aplicação.

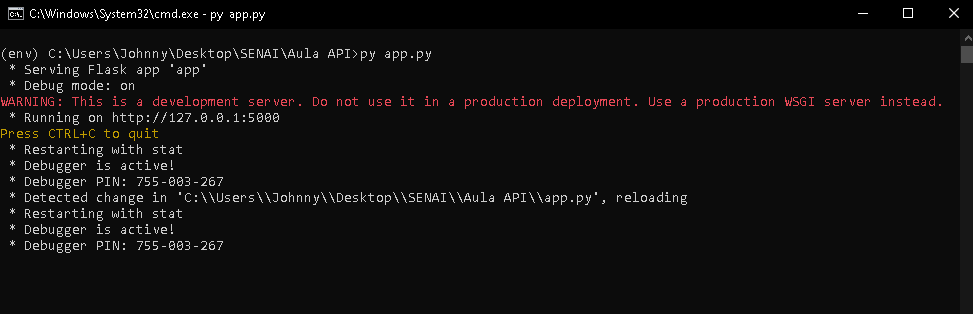
Logo após, na linha 5, nós utilizamos um decorador de rota para definir a home do nosso “site”.O decorador deve ser utilizado em conjunto com a função que ira dizer qual ação deve acontecer quando a URL for acessada. Neste caso, a função “home()” irá retornar a mensagem “Hello Flask”

E por último mas não menos importante, criamos uma condição para verificar se o seu arquivo “app.py” está sendo executado diretamente, e caso esteja, o comando app.run(debug=True) inicia o servidor Flask localmente com o modo de depuração ativado.

Agora vamos testar. No CMD digite:

> *py app.py*

*Isso irá executar a aplicação e se tudo der certo você vai ver algo do tipo:*



Esta imagem mostra que um servidor local de desenvolvimento foi iniciado em <http://127.0.0.1:5000>.

Além disso o “Debugger” está ativo, isso fará com que o servidor reinicie automaticamente a cada alteração que fizermos em nosso código.

Agora você pode acessar o endereço <http://127.0.0.1:5000> em seu navegador e você verá a mensagem “Hello Flask!” na tela.



**Definindo as Rotas da nossa API**

Para resolver nosso problema, criaremos um sistema do tipo CRUD (Create, Read, Update, Delete). Cada uma dessas operações terá uma rota específica na API.

Para isso, utilizaremos os métodos HTTP correspondentes:

POST → Criar um novo registro

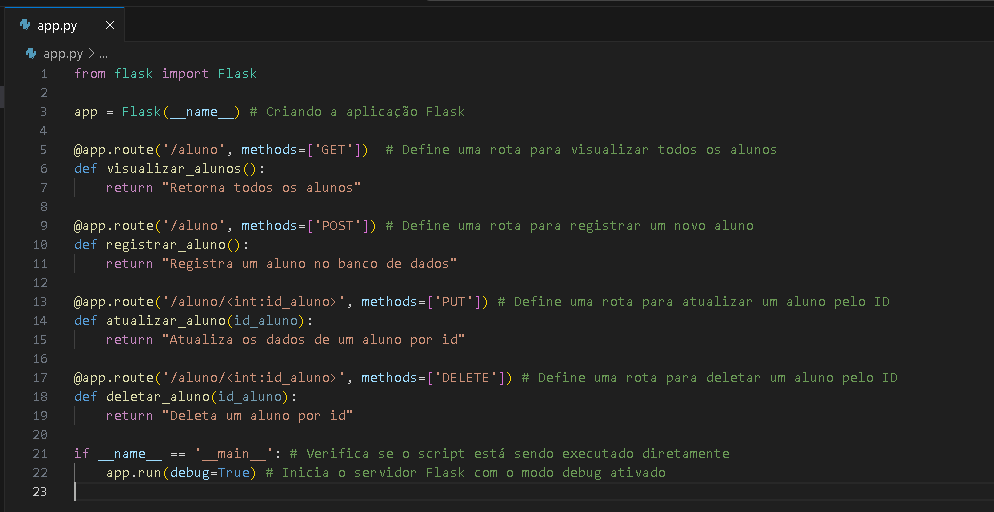
GET → Ler dados existentes

PUT → Atualizar um registro

DELETE → Remover um registro

Como o navegador só permite acessar URLs via GET, utilizaremos o Postman, um software que nos permite testar todos os métodos HTTP da API de forma simples e eficiente. Você pode acessar o Postman diretamente pelo navegador <https://www.postman.com/> ou então baixar o app desktop. O que for melhor para você.

Vamos definir algumas rotas agora, atualize seu script de acordo com a imagem a seguir:



Explicação

Este código define as rotas da API para gerenciar alunos, seguindo o modelo CRUD.

GET /aluno → Retorna a lista de alunos.

POST /aluno → Registra um novo aluno no banco de dados.

PUT /aluno/<id> → Atualiza os dados de um aluno específico.

DELETE /aluno/<id> → Remove um aluno pelo ID.

O parâmetro methods define quais métodos HTTP a rota aceita.

Por padrão, o Flask trata todas as rotas como GET, mas ao passar methods=['POST'], por exemplo, garantimos que a rota aceitará apenas requisições POST.

Nas rotas PUT e DELETE, usamos /<int:id\_aluno> na URL. Isso significa que:

id\_aluno é uma variável dinâmica da URL.

O <int:> indica que o Flask deve converter o valor recebido para inteiro (int).

O Flask passa esse valor como parâmetro da função, permitindo que possamos usá-lo dentro da lógica da API.

Exemplo:

Se acessarmos /aluno/5 em uma requisição PUT, o Flask executará atualizar\_aluno(5).

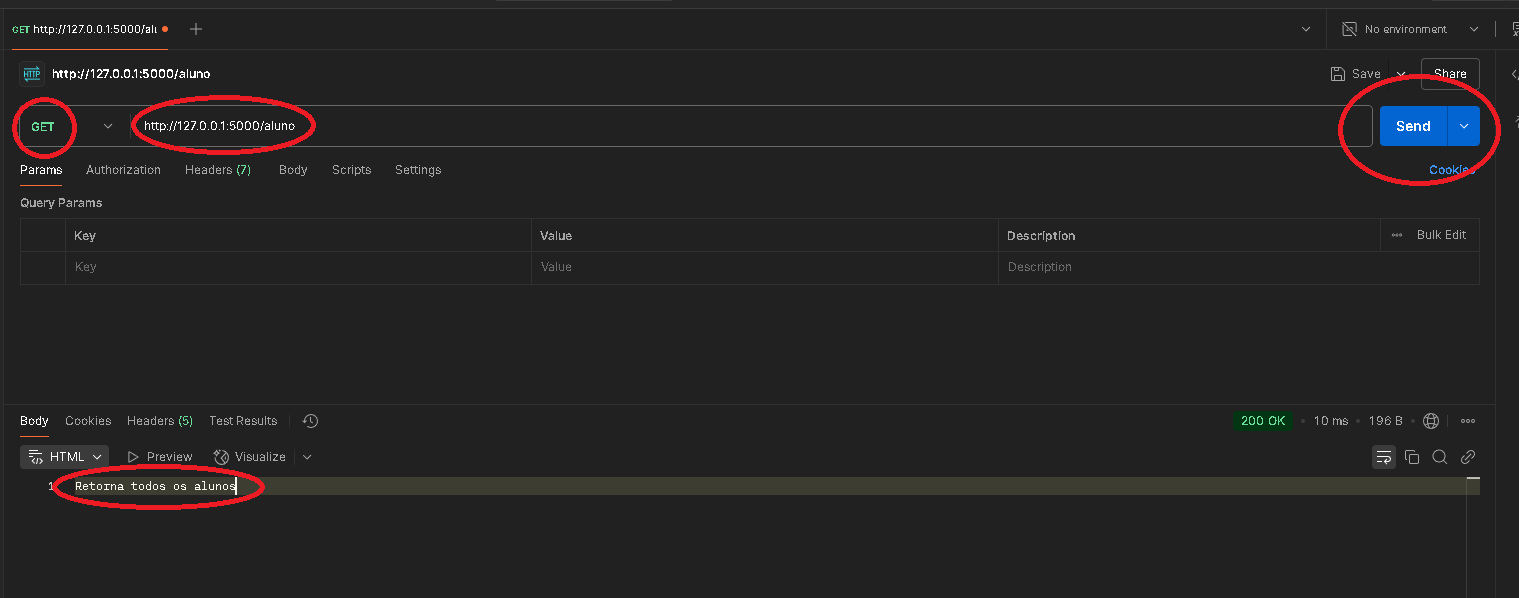
Isso permite manipular alunos específicos de forma dinâmica!

**Testando as rotas com Postman**

Agora é hora de testar as operações da API usando o Postman.

Este guia foca na criação da API, então, se você não estiver familiarizado com o Postman, há diversos tutoriais simples no YouTube que podem te ajudar!

Testando a rota GET:



O que você deve fazer:

1. Digitar a URL: <http://127.0.0.1:5000/aluno> na barra de endereço
2. Selecionar o método desejado: “GET”
3. Clicar em enviar(Send)

Se tudo estiver correto, a mensagem “Retorna todos os alunos será exibida no terminal.  
 Após isso você deve testar as outras rotas uma por vez desta maneira:

“POST” - <http://127.0.0.1:5000/aluno>

“PUT” - [http://127.0.0.1:5000/aluno/1](http://127.0.0.1:5000/aluno)

“Delete” - [http://127.0.0.1:5000/aluno/1](http://127.0.0.1:5000/aluno)

E as mensagens devem ser respectivamente:

"Registra um aluno no banco de dados"

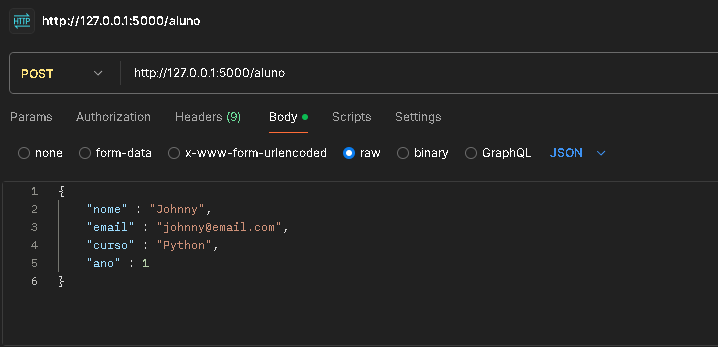
"Atualiza os dados de um aluno por id"

"Deleta um aluno por id"

Se as mensagens que apareceram no terminal do Postman correspondem às mensagens definidas nas funções de cada rota, estamos no caminho certo.

Então agora vamos para o próximo passo, recuperar informações digitadas no postman utilizando o método “request” do Flask.

1. No postman, vá até a guia “Body” e depois marque a opção “raw”.

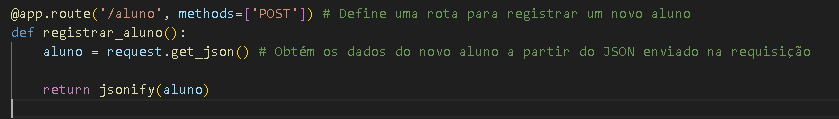


Desta maneira poderemos postar as informações dos alunos no formato Json antes de registrá-las no banco de dados. Não esqueça de copiar o dicionário conforme mostrado na imagem.

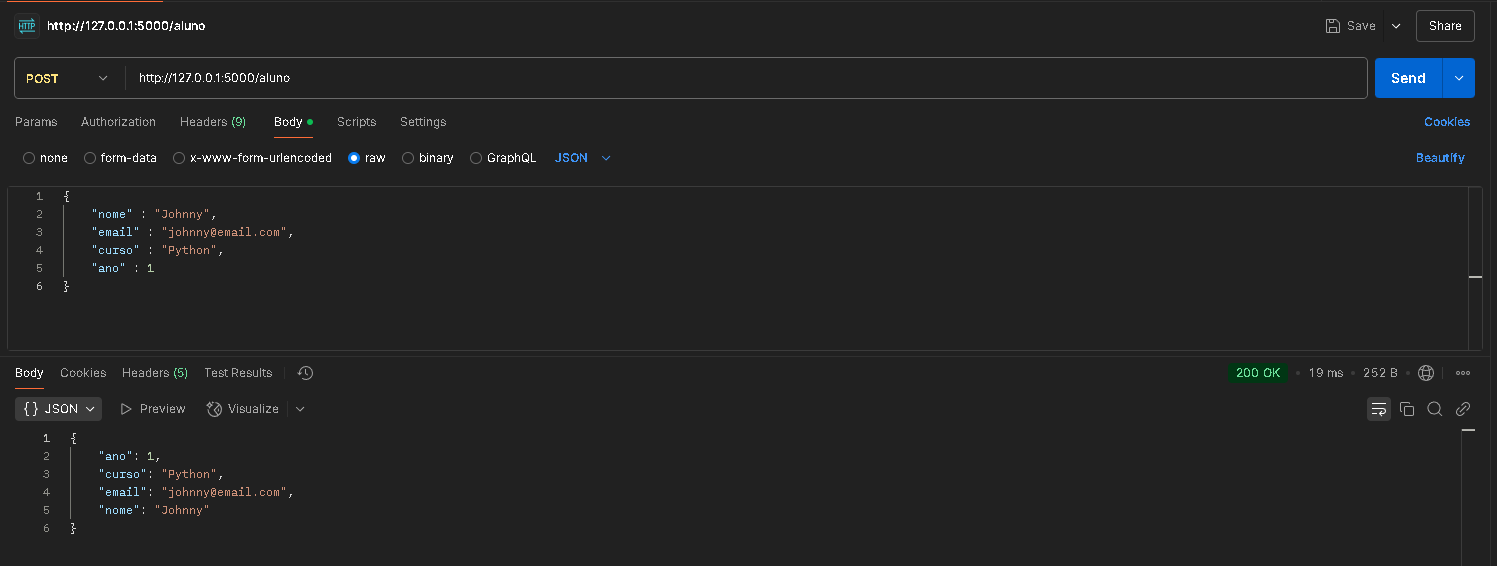
1. Atualize a linha de imports no seu script:



1. Atualize a função “registrar\_aluno()”:



Teste no Postman:



O que está acontecendo aqui é:

Com o método request.get\_jason() do Flask nós conseguimos obter os dados inseridos no Postman. E com o método Jsonify conseguimos retornar as mesmas informações para o terminal do Postman. Nossa comunicação está começando a funcionar.

Se tudo ocorreu como mostrado nos exemplos, é hora de seguir para o próximo passo. Criar o banco de dados!

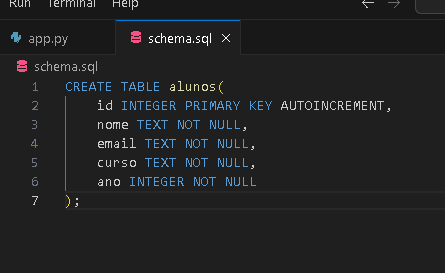
**Criando e configurando Banco de Dados**

Como o foco deste guia é Python, não vou entrar em muitos detalhes sobre SQL, e como o projeto é uma API simples, não será necessário para que possamos concluir nosso objetivo.

Em primeiro lugar vamos precisar definir um schema para criar um arquivo chamado “alunos.db” que irá ser o nosso banco de dados. Nele teremos uma tabela simples onde registraremos nossos alunos.

Vamos lá:

1. Crie um arquivo chamado “schema.sql” no seu diretório do projeto.
2. Copie o script SQL que irá criar a nossa tabela e salve o arquivo:



1. No CMD, crie o nosso arquivo de banco de dados utilizando o comando:

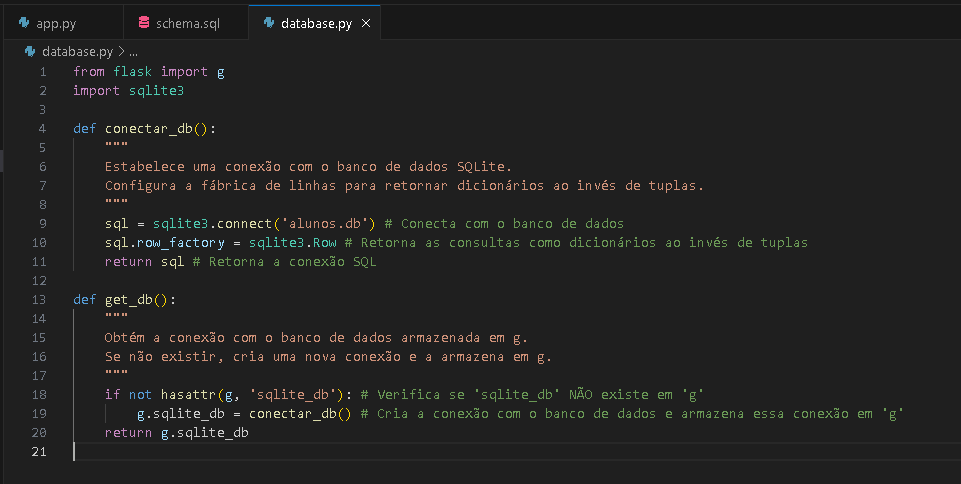
> *sqlite3 alunos.db < schema.sql*

Isso irá gerar um arquivo de banco de dados chamado “alunos.db”

1. Agora vamos criar um novo arquivo python chamado “database.py”

Nele vamos criar duas funções que vão auxiliar na utilização do SQL em nossa API.

Copie o código:



Este script é um módulo python que iremos importar dentro do script “app.py”, ele permite criar a conexão

Com o banco de dados e também armazena esta conexão em ‘g’ que é uma variável global do Flask.

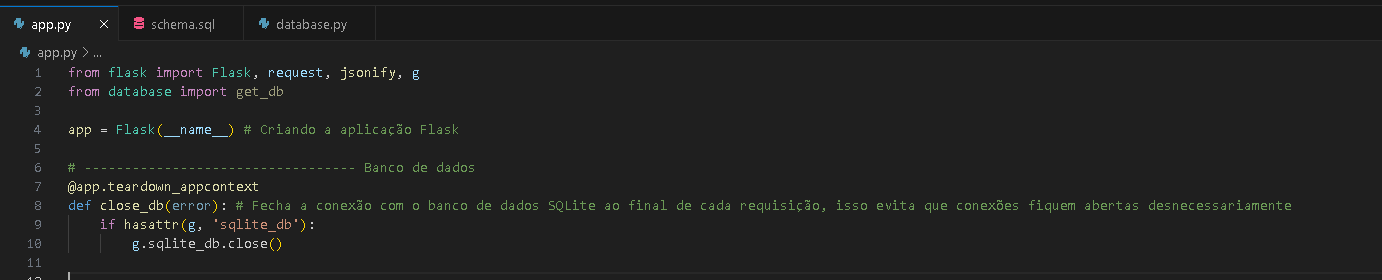
No Flask, o objeto g é uma variável global que permite armazenar dados que devem estar disponíveis durante toda a requisição, mas não necessariamente entre requisições.

Então na função get\_db() verificamos se já existe uma conexão com o banco de dados estabelecida e armazenada em ‘g’ usando “hasattr(g, ‘sqlite\_db’). Se não existir, criamos esta nova conexão e armazenamos a mesma em g utilizando “g.sqlite\_db = conectar\_db() e retornamos esta conexão com o banco de dados armazenada em g.

O uso do objeto g aqui permite que a conexão com o banco de dados seja espabelecida apenas uma vez por requisição, e reutilizada em diferentes partes da API. Isso ajuda a melhorar a performance e reduzir o overhead de criar e fechar conexões com o banco de dados.

1. Agora vamos voltar para o script app.py

Vamos adaptar o código novamente, importe ‘g’, importe a função get\_db do módulo database e crie esta nova função decorada:



Esta parte é muito importante, aqui estamos criando uma função que irá fechar a conexão com o banco de dados automaticamente sempre após o final de cada requisição HTTP graças ao decorador “@app.teardown\_appcontext”

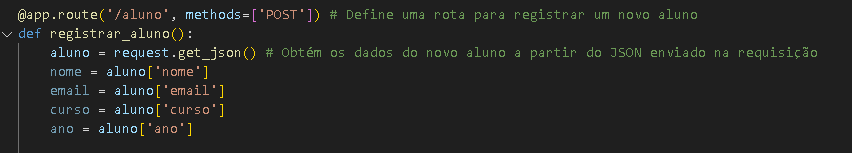
Isso vai impedir que a conexão com o banco de dados permaneça sempre aberta e atrapalhe a performance da nossa API.

Após essas alterações, estamos prontos para começar a trabalhar em cada uma das rotas.

**Adicionando um novo aluno no banco de Dados**

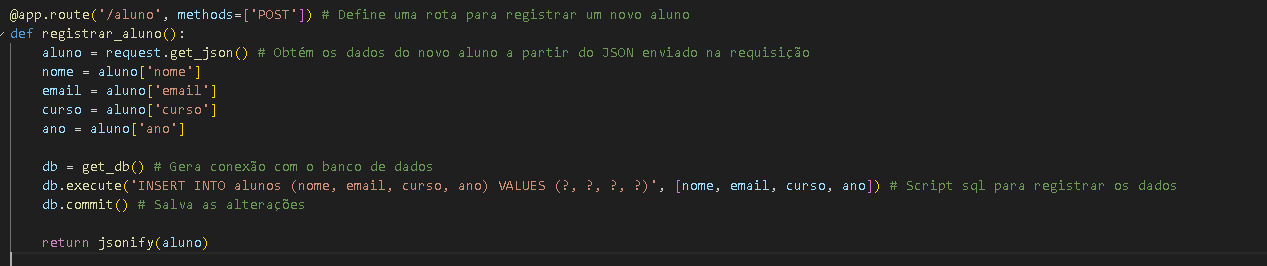
Agora chegou o momento de começarmos a interagir com o banco de dados.

Vamos adaptar função da rota que definimos o método “POST” para que ela registre as informações obtidas no formato Json diretamente no banco de dados. Se lembra do momento que utilizamos o método request.get\_json()? Ele retornou as informações no formato Json que é semelhante à um dicionário, ou seja, podemos conseguir cada dado relacionado ao aluno de uma maneira bem simples:

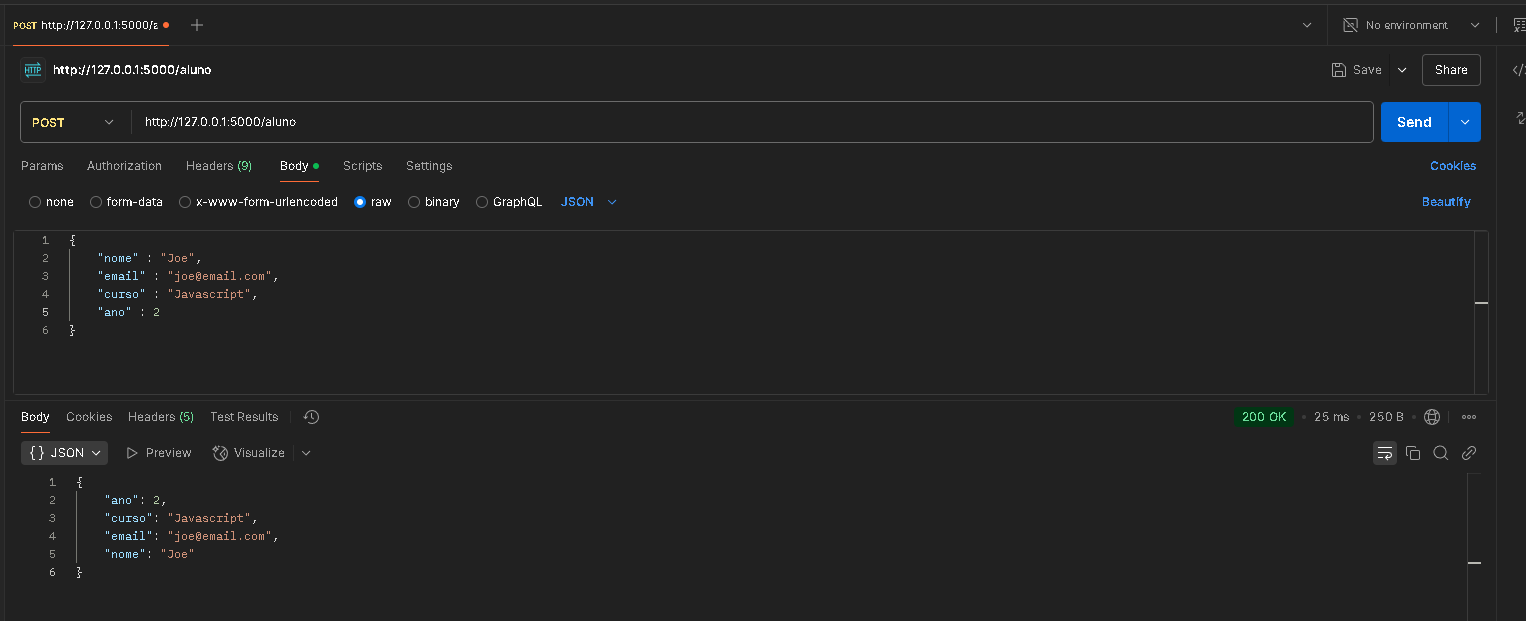


Desta maneira poderemos inserir essas informações dentro de um script SQL para registrá-las no banco de dados.

Assim:

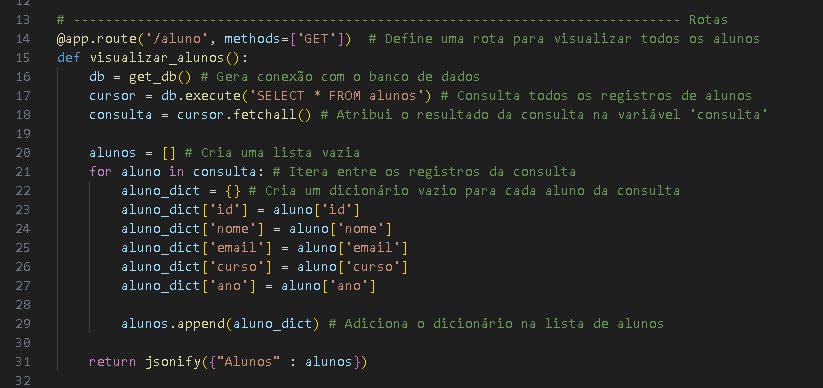


Se o código estiver correto, você agora pode registrar um novo aluno pelo Postman. Assim que clicar em enviar, os dados do aluno serão salvos no banco de dados.



**Consultando o banco de dados**

Fazer uma consulta SQL é muito simples, agora vamos adaptar a rota onde definimos o método “GET”.

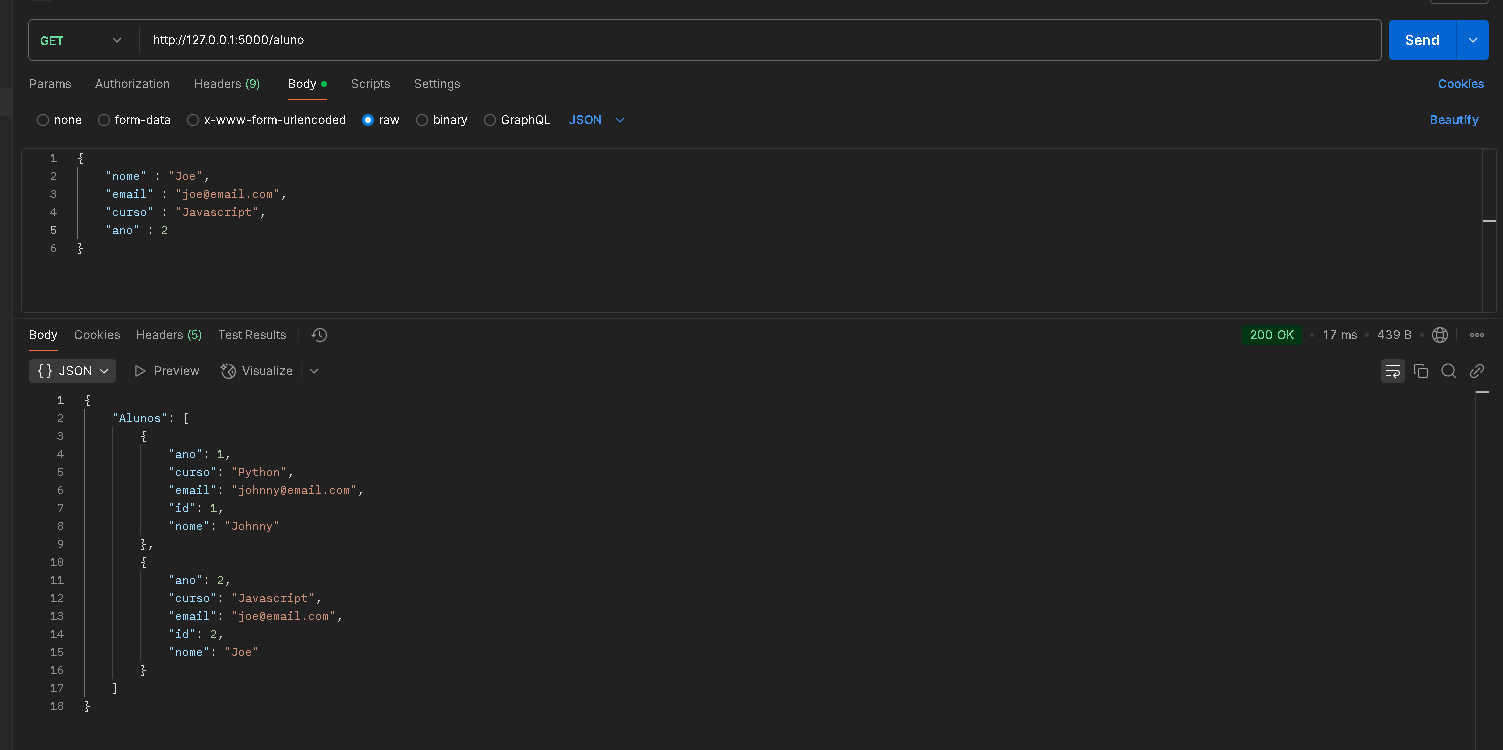


Aqui o processo é um tanto diferente do anterior.

Primeiro definimos a conexão com o banco de dados e depois definimos um cursor executando a consulta SQL que retorna todos os registros da tabela alunos.

Esta consulta resultará em uma lista de objetos “sqlite.Row” e para que possamos exibila no formato Json, precisaremos transformá-la em uma lista de dicionários.

Com um loop ‘for’ simples conseguimos acessar cada registro dos alunos e adaptar o resultado da consulta em uma lista de dicionários, e assim tornando mais fácil de exibir o resultado com o jsonify.



Agora já é possível consultar todos os registros.

**Atualizando os dados de um aluno**

Agora chegou a hora de utilizar a rota definida com o método “PUT”.

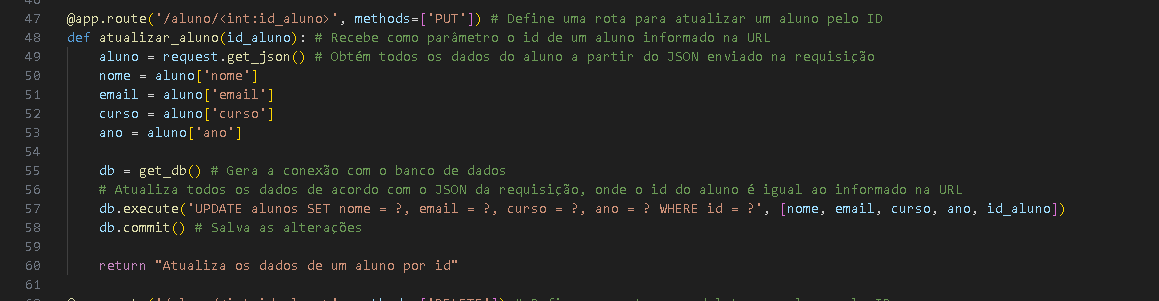
Este método exige que você envie todas as informações registradas, mesmo aquelas que não mudaram, para que ele funcione.

A variável definida na URL '/aluno/<int:id\_aluno>’ permitirá encontrar um aluno específico atravez de seu id.

Por exemplo, se você deseja atualizar o cadastro do aluno com id 3, o fim da URL será: ‘/aluno/3’

A função atualizar\_aluno() irá receber o id como atributo e assim poderemos utilizar essa informação dentro da consulta SQL.

O código adaptado ficará assim:



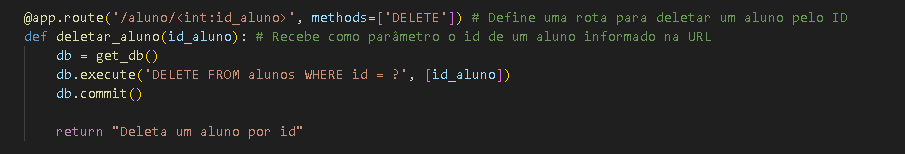
Em primeiro lugar obtemos os dados por JSON, do mesmo modo que foi feito quando registramos os alunos.

Depois executamos o comando SQL que atualizará todos os dados no banco de dados onde o id do aluno é igual ao id informado na URL no momento da requisição.

**Deletando um aluno**

E finalmente chegamos na última operação, deletar um registro de um aluno. É tão simples quanto as outras etapas, e desta vez novamente iremos utilizar o id de um aluno específico.

O código ficará assim:



Esta função apenas recebe como parâmetro o id do aluno e executa o comando SQL que deleta o registro onde o id do aluno é igual ao passado como parâmetro.

**Missão cumprida!**

Espero que tenham gostado da aula!

**PROJETO**: <https://github.com/DS-Johnny/API_de_Gerenciamento_de_Alunos>