Construindo um Aplicativo em Python usando o FastAPI

FastAPI é um framework leve e rápido para criar interfaces de programação de aplicativos modernos usando o Python 3.6 ou superior. Neste tutorial, explicaremos o básico para construir um aplicativo com FastAPI e mostraremos a você porque ele foi nomeado como um dos melhores frameworks de código aberto de 2021.

Quando você estiver pronto para desenvolver seus próprios aplicativos FastAPI, você não terá que procurar muito para encontrar um lugar para hospedá-los. Os serviços de Hospedagem de Aplicativos e Hospedagem de Banco de Dados da Kinsta fornecem uma plataforma como um serviço baseado em Python.

Aprenderemos o básico primeiro.

Vantagens do FastAPI

Abaixo estão algumas das vantagens que a framework FastAPI traz para um projeto.

- **Velocidade:** Como o nome indica, FastAPI é uma framework muito rápida. Sua velocidade é comparável com o Go e Node.js, que são geralmente consideradas uma das opções mais rápidas para a construção de APIs.
- **Fácil de aprender e codificar:** FastAPI já descobriu quase tudo que você precisará para fazer uma API pronta para produção. Como um desenvolvedor que usa FastAPI, você não precisa codificar tudo do zero. Com apenas algumas linhas de código, você pode ter uma API RESTful pronta para a implantação.
- **Documentação abrangente:** FastAPI usa os padrões de documentação OpenAPI, assim a documentação pode ser gerada de forma dinâmica. Esta documentação fornece informações detalhadas sobre os endpoints, respostas, parâmetros e códigos de retorno do FastAPI.
- **APIs com menos bugs:** FastAPI suporta validação de dados personalizada, o que permite aos desenvolvedores construir APIs com menos bugs. Os desenvolvedores de FastAPI se orgulham de que o framework resulta em menos bugs induzidos por humanos – até 40% menos.
- **Dicas de digitação:** O módulo types foi introduzido no Python 3.5. Isso permite que você declare o `type` de uma variável. Quando o tipo de uma variável é declarado, as IDEs conseguem fornecer melhor suporte e prever erros com mais precisão.

Como começar com o FastAPI

Para seguir este tutorial e começar com FastAPI, você precisará realizar algumas coisas primeiro.

Certifique-se de ter um editor de texto/IDE de um programador, como o Visual Studio Code. Outras opções incluem Sublime Text e Expresso.

É comum que aplicativos e instâncias do Python sejam executados em ambientes virtuais. Os ambientes virtuais permitem que diferentes conjuntos de pacotes e configurações sejam executados simultaneamente, evitando conflitos causados ??por versões de pacotes incompatíveis.

Para criar um ambiente virtual, abra seu terminal e execute este comando:

```
$ python3 -m venv env
```

Você também precisará ativar o ambiente virtual. O comando para fazer isso vai variar dependendo do sistema operacional e do shell que você está usando. Aqui estão alguns exemplos de ativação da CLI para vários ambientes:

```
# On Unix or MacOS (bash shell):
/path/to/venv/bin/activate

# On Unix or MacOS (csh shell):
/path/to/venv/bin/activate.csh

# On Unix or MacOS (fish shell):
/path/to/venv/bin/activate.fish
```

```
# On Windows (command prompt):
pathtovenvScriptsactivate.bat

# On Windows (PowerShell):
pathtovenvScriptsActivate.ps1
```

(Algumas IDEs Python-aware também podem ser configuradas para ativar o ambiente virtual atual.)

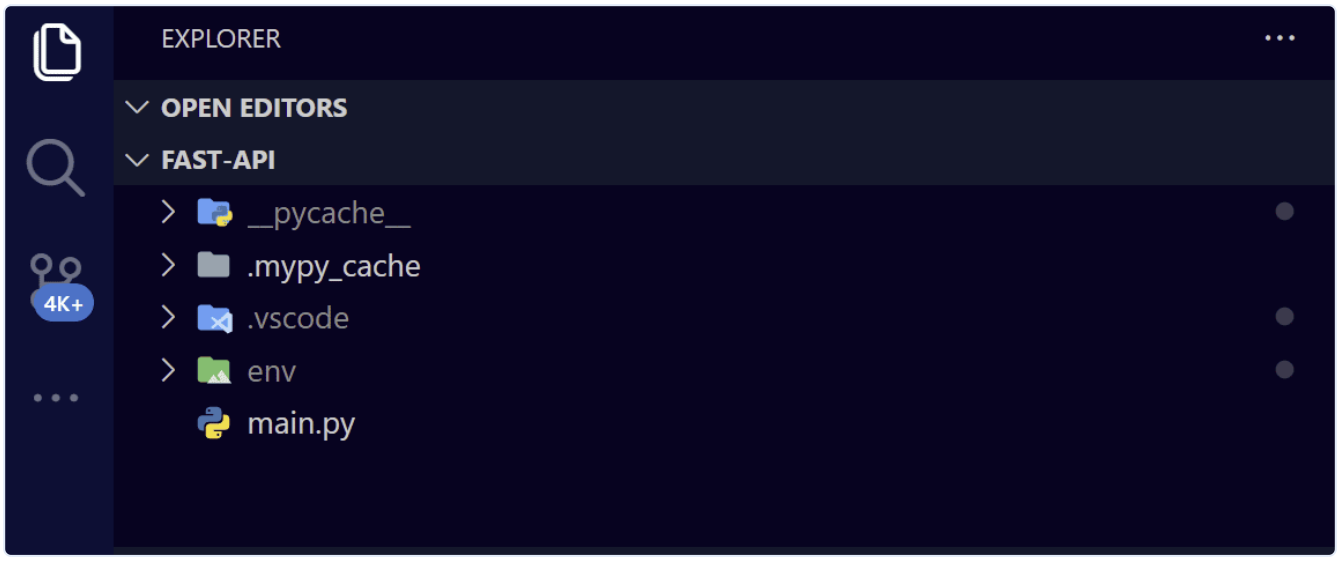
Agora, instale o FastAPI:

```
$ pip3 install fastapi
```

FastAPI é um framework para construir APIs, mas para testar suas APIs você precisará de um servidor web local. O Uvicorn é um servidor web ASGI (Asynchronous Server Gateway Interface) para Python, ótimo para desenvolvimento. Para instalar o Uvicorn, execute este comando:

```
$ pip3 install "uvicorn[standard]"
```

Após a instalação bem-sucedida, crie um arquivo chamado **main.py** no diretório de trabalho do seu projeto. Este arquivo será o ponto de entrada do seu aplicativo.



— Visualização de um projeto FastAPI básico em uma IDE.

Um exemplo rápido de FastAPI

Você testará sua instalação FastAPI configurando rapidamente um endpoint de exemplo. Em seu arquivo **main.py**, cole no seguinte código e depois salve o arquivo:

```
# main.py
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
async def root():
 return {"greeting":"Hello world"}
```

O snippet acima cria um endpoint básico do FastAPI. Abaixo está um resumo do que cada linha faz:

- `from fastapi import FastAPI`: A funcionalidade para sua API é fornecida pela classe FastAPI Python.

- `app = FastAPI()`: Isso cria uma instância FastAPI.
- `@app.get("/")`: Este é um decorador python que especifica ao FastAPI que a função abaixo dele é responsável pelo tratamento dos pedidos.
- `@app.get("/")`: Este é um decorador que especifica a rota. Isso cria um método `GET` na rota do site. O resultado é então retornado pela função wrapped.
- Outras operações possíveis, usadas para comunicar incluem `@app.post()`, `@app.put()`, `@app.delete()`, `@app.options()`, `@app.head()`, `@app.patch()`, e `@app.trace()`.

No diretório de arquivos, execute o seguinte comando em seu terminal para iniciar o servidor API:

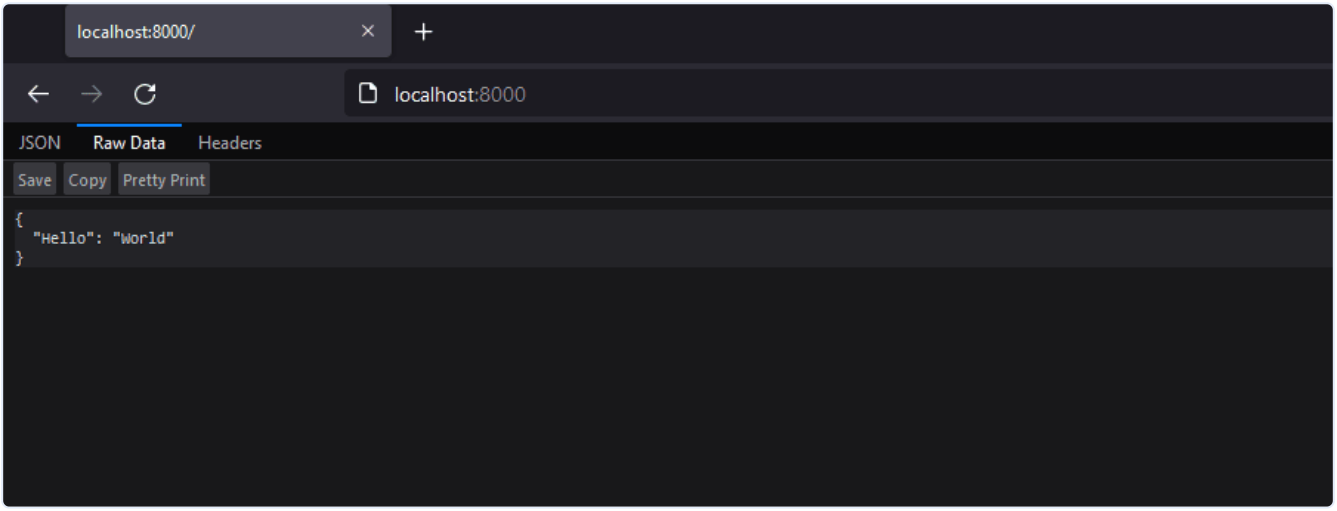
```
$ uvicorn main:app --reload
```

Neste comando, `main` é o nome do seu módulo. O objeto `app` é uma instância do seu aplicativo, importado para o servidor ASGI. A flag `--reload` diz ao servidor para recarregar automaticamente quando você realizar qualquer mudança.

Você deve ver algo assim em seu terminal:

```
 $ uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['D:\WEB DEV\Eu
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to qu
INFO: Started reloader process [26888] using WatchFiles
INFO: Started server process [14956]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Em seu navegador, navegue até `http://localhost:8000` para confirmar que sua API está funcionando. Você deve ver o “Hello”: “World” como um objeto JSON na página. Isso ilustra como é fácil criar uma API com FastAPI. Tudo que você tinha que fazer era definir uma rota e retornar seu dicionário Python, como visto na linha seis do snippet acima.



— Aplicativo FastAPI Hello World em um navegador web.

Usando sugestões Type

Se você usa Python, você está acostumado a anotar variáveis com tipos de dados básicos como `int`, `str`, `float`, e `bool`. No entanto, a partir da versão 3.9 do Python, foram introduzidas estruturas de dados avançadas. Isso permite que você trabalhe com estruturas de dados como `dictionaries`, `tuples`, e `lists`. Com as dicas de Type do FastAPI, você pode estruturar o schema de seus dados usando modelos pydantic e então, usar os modelos pydantic para digitar a dica e se beneficiar da validação de dados que é fornecida.

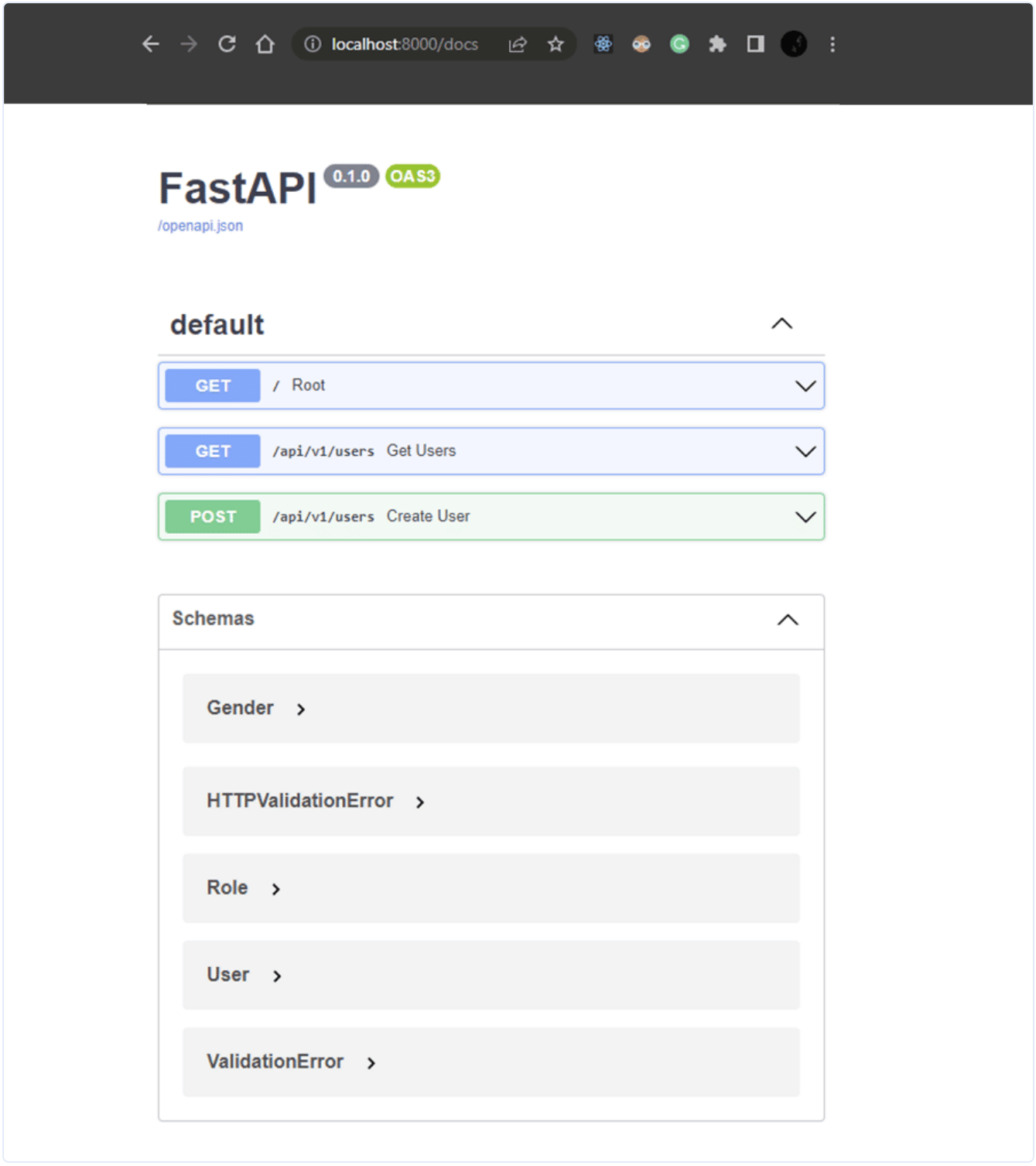
No exemplo abaixo, o uso de dicas de Type em Python é demonstrado com uma simples calculadora de preço de refeição, `calculate_meal_fee`:

```
def calculate_meal_fee(beef_price: int, meal_price: int) -> int:
 total_price: int = beef_price + meal_price
 return total_price
print("Calculated meal fee", calculate_meal_fee(75, 19))
```

Note que as dicas de Type não mudam como seu código funciona.

Documentação da API interativa FastAPI

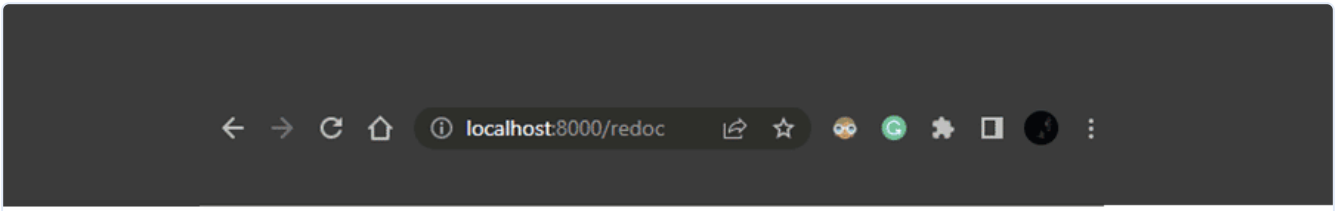
FastAPI usa a Swagger UI para fornecer documentação API interativa automática. Para acessá-la, navegue até `http://localhost:8000/docs` e você verá uma tela com todos os seus endpoints, métodos e schemas.



— Documentação da Swagger UI para FastAPI.

Esta documentação API automática, baseada em navegador, é fornecida pela FastAPI, e você não precisa fazer mais nada para tirar proveito dela.

Uma documentação API alternativa baseada em navegador, também fornecida pela FastAPI, é a Redoc. Para acessar a Redoc, navegue até `http://localhost:8000/redoc`, onde lhe será apresentada uma lista de seus endpoints, os métodos e suas respectivas respostas.



FastAPI (0.1.0)

Download OpenAPI specification: Download

Root

Responses

> 200 Successful Response

Get Users

— Documentação da Redoc para FastAPI.

Configurando rotas no FastAPI

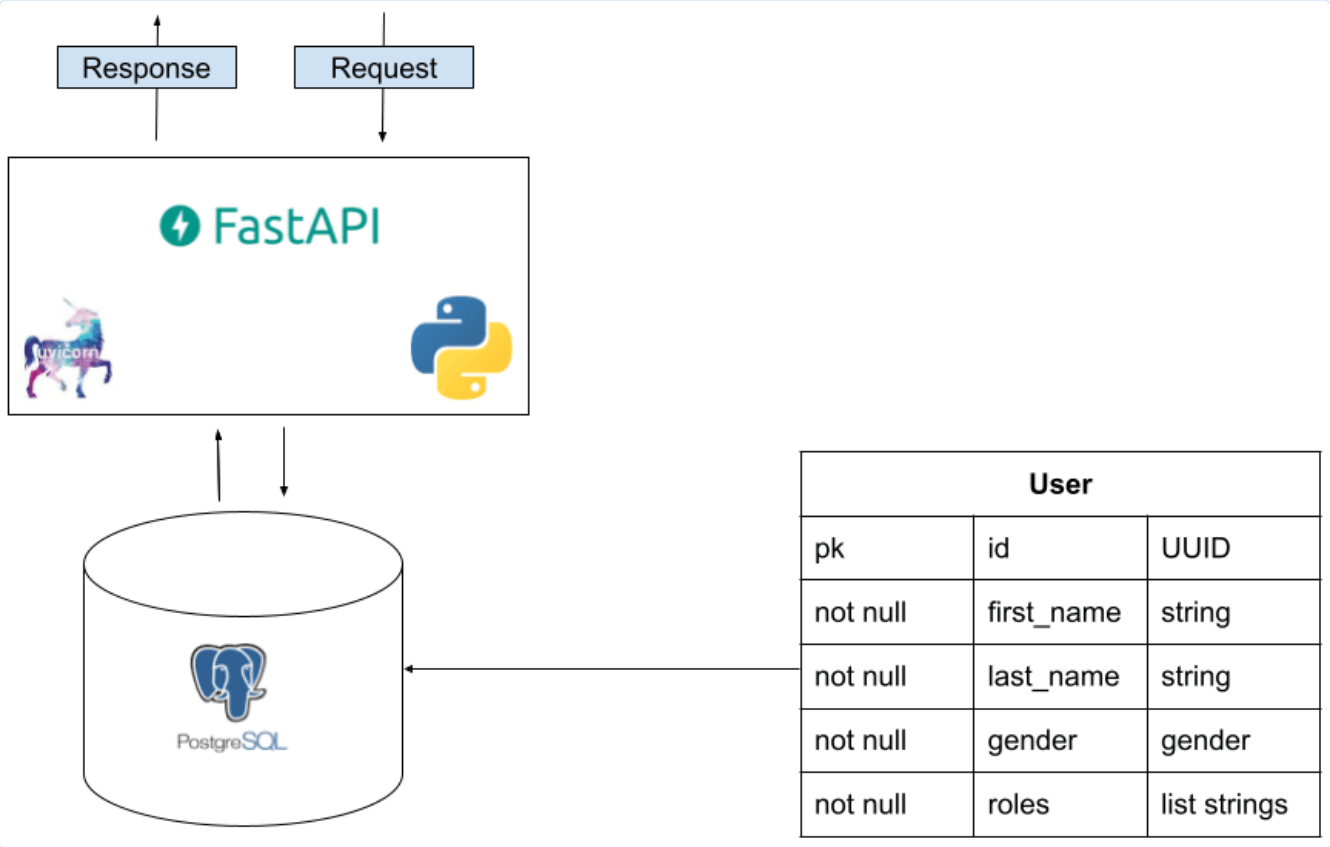
O decorador `@app` permite especificar o método de rota, como `@app.get` ou `@app.post`, e suporta `GET`, `POST`, `PUT`, e `DELETE`, bem como as opções menos comuns, `HEAD`, `PATCH`, e `TRACE`.

Construindo seu aplicativo com FastAPI

Este tutorial mostra como criar um aplicativo CRUD usando FastAPI. O aplicativo pode:

- Criar um usuário
- Ler o registro do banco de dados de um usuário
- Atualizar um usuário existente
- Excluir um usuário em particular

Para executar estas operações CRUD, você criará métodos que expõem os endpoints da API. O resultado será um banco de dados in-memory que pode armazenar uma lista de usuários.



User		
pk	id	UUID
not null	first_name	string
not null	last_name	string
not null	gender	gender
not null	roles	list strings

— Estrutura da tabela do banco de dados para exemplos CRUD

Você usará a biblioteca pydantic para realizar validação de dados e gerenciamento de configurações usando anotações do Type Python. Para os propósitos deste tutorial, você irá declarar a forma dos seus dados como classes com atributos.

Este tutorial utilizará o banco de dados in-memory. Isso é para você começar rapidamente a usar o FastAPI para construir suas APIs. Entretanto, para a produção, você pode recorrer a qualquer banco de dados da sua escolha, como PostgreSQL, MySQL, SQLite, ou mesmo a Oracle.

Construindo o aplicativo

Você começará criando seu modelo de usuário. O modelo de usuário terá os seguintes atributos:

- `id`: Um Identificador Único Universal (UUID)
- `first_name`: O primeiro nome do usuário
- `last_name`: O sobrenome do usuário
- `gender`: O gênero do usuário
- `roles`: Uma lista contendo funções `admin` e `user`

Comece criando um arquivo chamado **model.py** em seu diretório de trabalho, depois cole o seguinte código no **model.py** para criar seu modelo:

```
# models.py
from typing import List, Optional
from uuid import UUID, uuid4
from pydantic import BaseModel
from enum import Enum
from pydantic import BaseModel
class Gender(str, Enum):
 male = "male"
 female = "female"
class Role(str, Enum):
 admin = "admin"
 user = "user"
class User(BaseModel):
 id: Optional[UUID] = uuid4()
 first_name: str
 last_name: str
 gender: Gender
 roles: List[Role]
```

No código acima:

- Sua classe `User` estende `BaseModel`, sendo então importada de `pydantic`.
- Você definiu os atributos do usuário, como discutido acima.

O próximo passo é criar o seu banco de dados. Substitua o conteúdo do seu arquivo **main.py** com o seguinte código:

```
# main.py
from typing import List
from uuid import uuid4
from fastapi import FastAPI
from models import Gender, Role, User
app = FastAPI()
db: List[User] = [
 User(
 id=uuid4(),
 first_name="John",
 last_name="Doe",
 gender=Gender.male,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
 first_name="Jane",
 last_name="Doe",
 gender=Gender.female,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
 first_name="James",
 last_name="Gabriel",
 gender=Gender.male,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
 first_name="Eunit",
 last_name="Eunit",
```

```
    gender=Gender.male,
    roles=[Role.admin, Role.user],
    ),
   ]
```

Em seu arquivo **main.py:**

- Você inicializou `db` com um tipo de `List`, e passou no modelo `User`
- Você criou um banco de dados in-memory com quatro usuários, cada um com os atributos necessários, tais como `first_name`, `last_name`, `gender`, e `roles`. Ao usuário `Eunit` são atribuídas as funções de `admin` e `user`, enquanto aos outros três usuários é atribuída apenas a função de `user`.

Leitura de registros do banco de dados

Você configurou com sucesso seu banco de dados in-memory e o preencheu com usuários, então o próximo passo é configurar um endpoint que irá retornar uma lista de todos os usuários. É aqui que entra o FastAPI.

Em seu arquivo **main.py**, cole o seguinte código logo abaixo do seu endpoint `Hello World`:

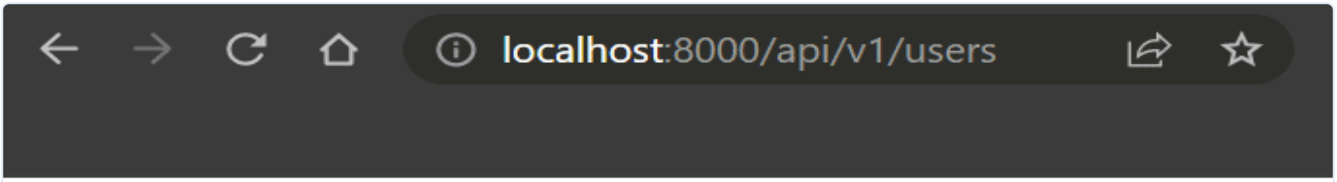
```
# main.py
 @app.get("/api/v1/users")
 async def get_users():
 return db
```

Este código define o endpoint `/api/v1/users`, e cria uma função assimétrica, `get_users`, que retorna todo o conteúdo do banco de dados, `db`.

Salve seu arquivo, e você pode testar seu endpoint de usuário. Execute o seguinte comando em seu terminal para iniciar o servidor da API:

```
$ uvicorn main:app --reload
```

Em seu navegador, navegue até `http://localhost:8000/api/v1/users`. Isso deve retornar uma lista de todos os seus usuários, como visto abaixo:




```
[
  {
    "id": "6f212352-ac0e-403b-8f1e-c7998b8bc77c",
    "first_name": "John",
    "last_name": "Doe",
    "gender": "male",
    "roles": [
      "user"
    ]
  },
  {
    "id": "5b287e59-8cf9-4e6a-bbf9-60f67ba4412a",
    "first_name": "Jane",
    "last_name": "Doe",
    "gender": "female",
    "roles": [
      "user"
    ]
  },
  {
    "id": "f1602a18-7b02-4af2-a9a6-09fc81fd39cd",
    "first_name": "James",
    "last_name": "Gabriel",
    "gender": "male",
    "roles": [
      "user"
    ]
  },
  {
    "id": "527acbf1-c9fb-4a10-b5ff-ff7e207ad92a",
    "first_name": "Eunit",
    "last_name": "Eunit",
    "gender": "male",
    "roles": [
```

— Dados de usuário recuperados pelo banco de dados FastAPI read request.

Neste estágio, seu arquivo **main.py** será parecido com este:

```
# main.py
from typing import List
from uuid import uuid4
from fastapi import FastAPI
from models import Gender, Role, User
app = FastAPI()
db: List[User] = [
 User(
 id=uuid4(),
 first_name="John",
 last_name="Doe",
 gender=Gender.male,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
 first_name="Jane",
 last_name="Doe",
 gender=Gender.female,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
 first_name="James",
 last_name="Gabriel",
 gender=Gender.male,
 roles=[Role.user],
 ),
 User(
 id=uuid4(),
```

```
    first_name="Eunit",
    last_name="Eunit",
    gender=Gender.male,
    roles=[Role.admin, Role.user],
    ),
   ]
   @app.get("/")
   async def root():
    return {"Hello": "World",}
   @app.get("/api/v1/users")
   async def get_users():
    return db
```

Criando registros do banco de dados

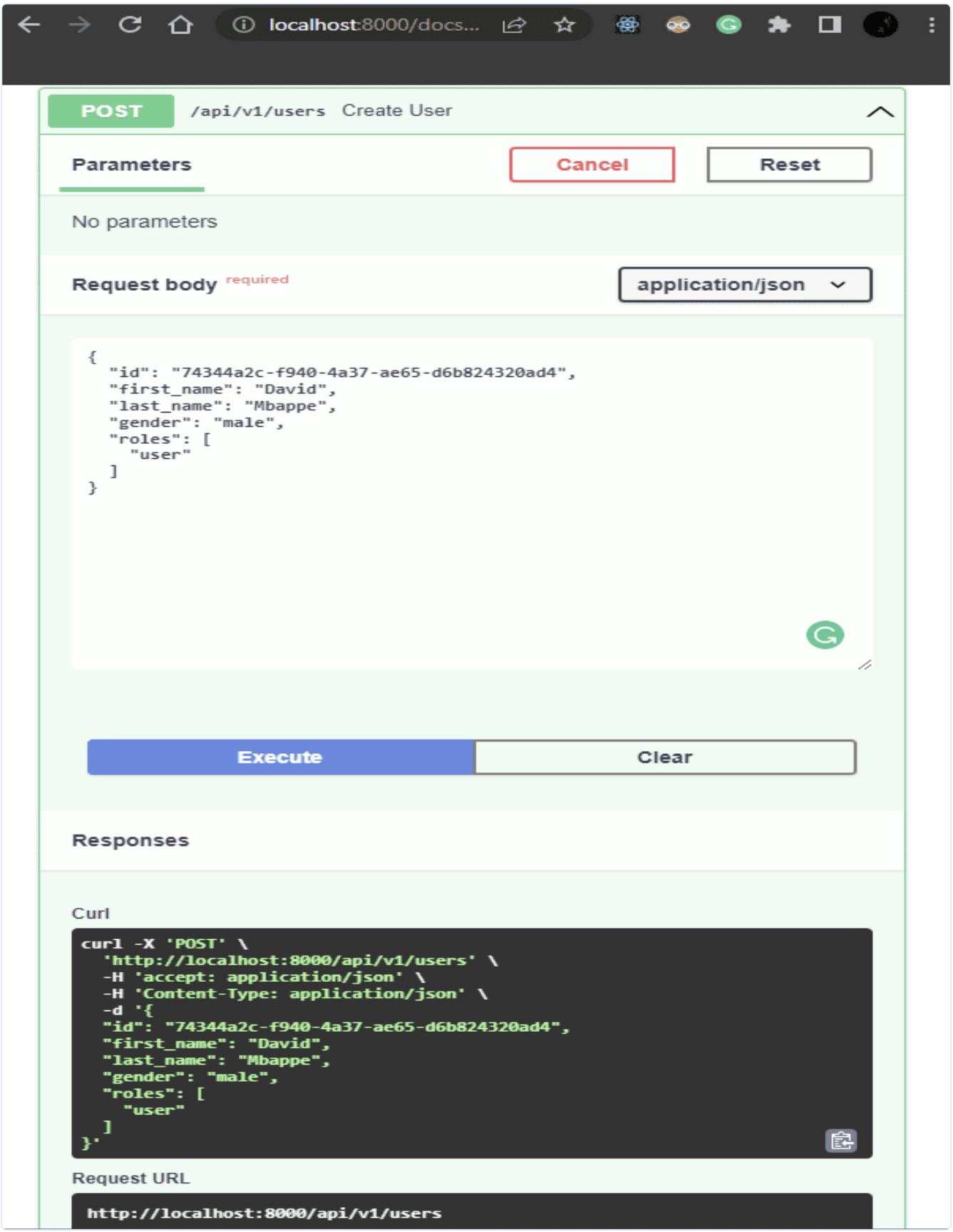
O próximo passo é criar um endpoint para criar um novo usuário no seu banco de dados. Cole o seguinte snippet em seu arquivo **main.py**

```
   # main.py
   @app.post("/api/v1/users")
   async def create_user(user: User):
    db.append(user)
    return {"id": user.id}
```

Neste snippet, você definiu o endpoint para enviar um novo usuário e recorreu ao decorador `@app.post` para criar um método `POST`.

Você também criou a função `create_user`, que aceita `user` do modelo `User`, e anexou (adicionado) o recém-criado `user` ao banco de dados, db. Finalmente, o endpoint retorna um objeto JSON do usuário recém-criado `id`.

Você terá que usar a documentação API automática fornecida pela FastAPI para testar seu endpoint, como visto acima. Isso é porque você não pode fazer uma solicitação de publicação usando o navegador da web. Navegue até `http://localhost:8000/docs` para testar usando a documentação fornecida pela SwaggerUI.


localhost:8000/docs...
POST
/api/v1/users
Create User
Parameters
Cancel
Reset
No parameters
Request body
required
application/json
{
"id": "74344a2c-f940-4a37-ae65-d6b824320ad4",
"first_name": "David",
"last_name": "Mbappe",
"gender": "male",
"roles": [
"user"
]
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8000/api/v1/users' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"id": "74344a2c-f940-4a37-ae65-d6b824320ad4",
"first_name": "David",
"last_name": "Mbappe",
"gender": "male",
"roles": [
"user"
]
}'
Request URL
http://localhost:8000/api/v1/users

— Parâmetros para uma solicitação FastAPI POST.

Excluindo registros do banco de dados

Já que você está construindo um aplicativo CRUD, seu aplicativo precisará ter a habilidade de excluir um recurso especificado. Para este tutorial, você irá criar um endpoint para excluir um usuário.

Cole o seguinte código em seu arquivo **main.py:**

```
# main.py
from uuid import UUID
from fastapi HTTPException
@app.delete("/api/v1/users/{id}")
async def delete_user(id: UUID):
for user in db:
 if user.id == id:
 db.remove(user)
 return
raise HTTPException(
 status_code=404, detail=f"Delete user failed, id {id} not found."
 )
```

Aqui está uma descrição linha por linha de como o código funciona:

- `@app.delete("/api/v1/users/{id}")`: O usuário criou o endpoint de exclusão usando o decorador `@app.delete()`. O caminho ainda é `/api/v1/users/{id}`, mas ele recupera o `id`, sendo uma variável de caminho correspondente ao id do usuário.
- `async def delete_user(id: UUID):`: Cria a função `delete_user`, que recupera o `id` a partir da URL.

- `for user in db:`: Isto diz ao aplicativo para percorrer os usuários no banco de dados, e verificar se o `id` passou corresponde a um usuário no banco de dados.
- `db.remove(user)`: Se o `id` corresponder a um usuário, o usuário será excluído; caso contrário, será gerado um `HTTPException` com um código de status 404.


localhost:8000/doc...
DELETE
/api/v1/users/{id} Delete User
Parameters
Cancel
Name
Description
id * required
string($uuid)
(path)
18d3f517-5008-46fb-9519-e2d5669b1dd3
Execute
Clear
Responses
Curl
curl -X 'DELETE' \
'http://localhost:8000/api/v1/users/18d3f517-5008-46fb-9519-e2d5669b1dd3'
-H 'accept: application/json'
Request URL
http://localhost:8000/api/v1/users/18d3f517-5008-46fb-9519-e2d5669b1dd3
Server response
Code
Details
200
Response body
null
Download
Response headers
content-length: 4
content-type: application/json
date: Tue,30 Aug 2022 02:50:20 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links

— Parâmetros para uma solicitação FastAPI DELETE.

Atualizando registros do banco de dados

Você criará um endpoint para atualizar os detalhes de um usuário. Os detalhes que podem ser atualizados incluem os seguintes parâmetros: `first_name`, `last_name`, e `roles`.

Em seu arquivo **model.py**, cole o seguinte código sob seu modelo `User`, ou seja, após a classe `User(BaseModel):`:

```
# models.py
class UpdateUser(BaseModel):
first_name: Optional[str]
last_name: Optional[str]
roles: Optional[List[Role]]
```

Neste snippet, a classe `UpdateUser` estende `BaseModel`. Você então define os parâmetros atualizáveis do usuário, tais como `first_name`, `last_name`, e `roles`, para serem opcionais.

Agora você criará um endpoint para atualizar os detalhes de um determinado usuário. Em seu arquivo **main.py**, cole o seguinte código após a classe `@app.delete`:

```
# main.py
@app.put("/api/v1/users/{id}")
async def update_user(user_update: UpdateUser, id: UUID):
 for user in db:
 if user.id == id:
```

```
    if user_update.first_name is not None:
    user.first_name = user_update.first_name
    if user_update.last_name is not None:
    user.last_name = user_update.last_name
    if user_update.roles is not None:
    user.roles = user_update.roles
    return user.id
    raise HTTPException(status_code=404, detail=f"Could not find use
```

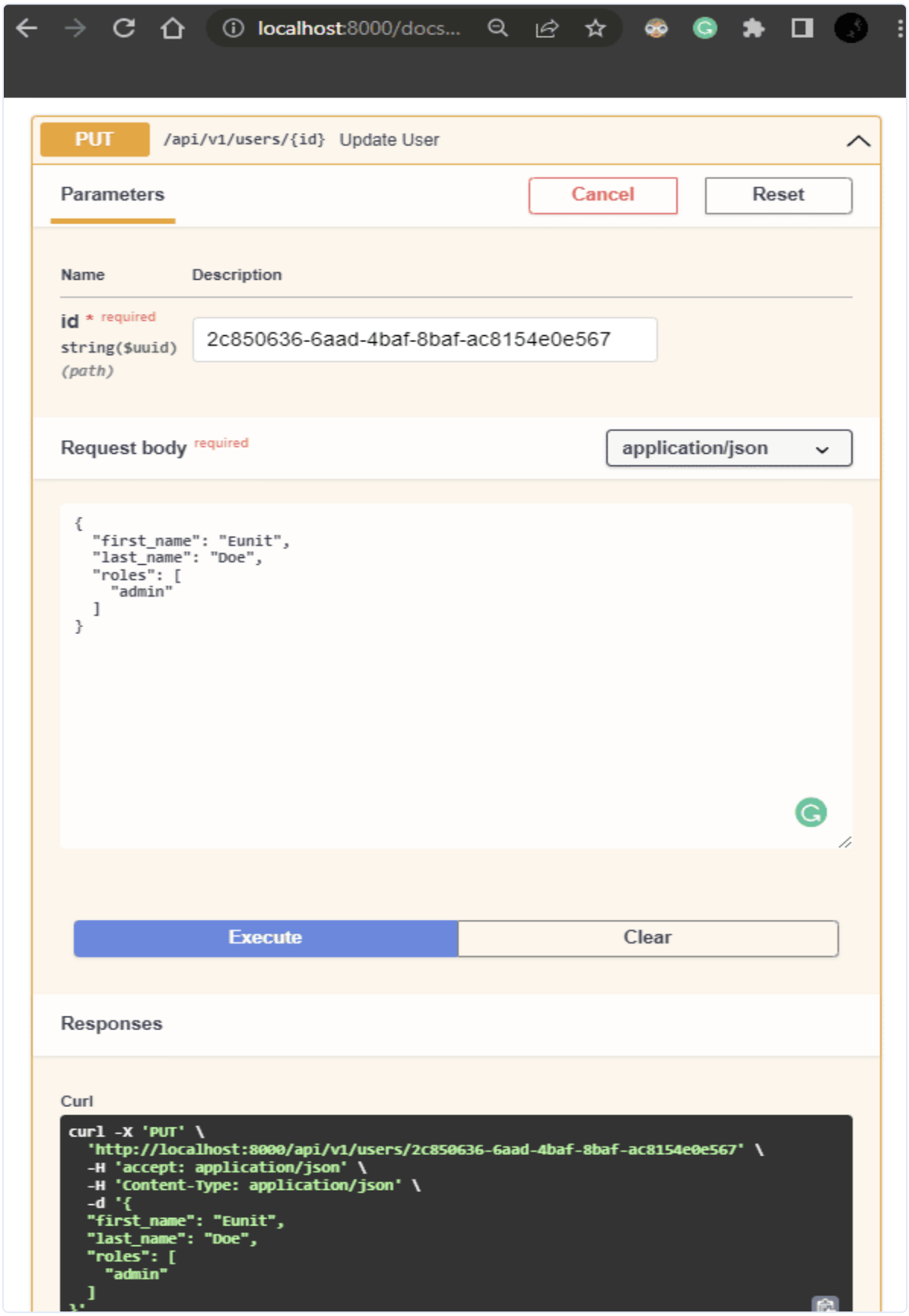
No código acima, você fez o seguinte:

- Criou o endpoint de atualização `@app.put("/api/v1/users/{id}")`. Ele tem um parâmetro variável `id` que corresponde ao id do usuário.
- Criou um método chamado `update_user`, que inclui a classe `UpdateUser` e `id`.
- Usou um loop `for` para verificar se o usuário associado com o `id` está no banco de dados.
- Verificou se algum dos parâmetros do usuário é `is not None` (não é nulo). Se algum parâmetro, como `first_name`, `last_name`, ou `roles`, não é nulo, então ele é atualizado.
- Se a operação for bem-sucedida, o id do usuário é retornado.
- Caso o usuário não foi localizado, uma exceção `HTTPException` com um código de status 404 e uma mensagem `Could not find user with id: {id}` é criado.

Para testar este endpoint, certifique-se de que seu servidor Uvicorn está rodando. Se ele não estiver rodando, digite este comando:

```
uvicorn main:app --reload
```

Abaixo está uma imagem da tela do teste.


localhost:8000/docs...
PUT
/api/v1/users/{id} Update User
Parameters
Cancel
Reset
Name
Description
id * required
string($uuid)
(path)
2c850636-6aad-4baf-8baf-ac8154e0e567
Request body required
application/json
{
"first_name": "Eunit",
"last_name": "Doe",
"roles": [
"admin"
]
}
Execute
Clear
Responses
Curl
curl -X 'PUT' \
'http://localhost:8000/api/v1/users/2c850636-6aad-4baf-8baf-ac8154e0e567' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"first_name": "Eunit",
"last_name": "Doe",
"roles": [
"admin"
]
}'

— Parâmetros para uma solicitação de atualização FastAPI

Resumo

Neste tutorial, você explorou sobre o framework FastAPI usando Python e viu por si mesmo a velocidade com que um aplicativo FastAPI pode ser instalado e executado. Você aprendeu como construir endpoints CRUD API usando a framework – criando, lendo, atualizando e apagando registros do banco de dados.

Se você deseja levar o desenvolvimento do seu aplicativo da web para o próximo nível, certifique-se de verificar a plataforma Kinsta para Hospedagem de Aplicativos e Hospedagem de Banco de Dados. Como FastAPI, é muito simples.