



Framework FastAPI, alta performance, fácil de aprender, fácil de codar, pronto para produção

 Test  passing  coverage  100%  pypi package  v0.81.0

Documentação: <https://fastapi.tiangolo.com>

Código fonte: <https://github.com/tiangolo/fastapi>




FastAPI é um moderno e rápido (alta performance) *framework web* para construção de APIs com Python 3.6 ou superior, baseado nos *type hints* padrões do Python.

Os recursos chave são:

- **Rápido:** alta performance, equivalente a **NodeJS** e **Go** (graças ao Starlette e Pydantic). [Um dos frameworks mais rápidos disponíveis](#).
- **Rápido para codar:** Aumenta a velocidade para desenvolver recursos entre 200% a 300%. *
- **Poucos bugs:** Reduz cerca de 40% de erros induzidos por humanos (desenvolvedores). *
- **Intuitivo:** Grande suporte a *IDEs*. [Auto-Complete](#) em todos os lugares. Menos tempo debugando.
- **Fácil:** Projetado para ser fácil de aprender e usar. Menos tempo lendo documentação.
- **Enxuto:** Minimiza duplicação de código. Múltiplos recursos para cada declaração de parâmetro. Menos bugs.
- **Robusto:** Tenha código pronto para produção. E com documentação interativa automática.
- **Baseado em padrões:** Baseado em (e totalmente compatível com) os padrões abertos para APIs: [OpenAPI](#) (anteriormente conhecido como Swagger) e [JSON Schema](#).

* estimativas baseadas em testes realizados com equipe interna de desenvolvimento, construindo aplicações em produção.

Patrocinadores Ouro

 {% if sponsors %} {% for sponsor in sponsors.gold -%}  {% endfor -%} {% for sponsor in sponsors.silver -%}  {% endfor %} {% endif %}

[Outros patrocinadores](#)

Opiniões

"[...] Estou usando **FastAPI** muito esses dias. [...] Estou na verdade planejando utilizar ele em todos os times de **serviços Machine Learning na Microsoft**. Alguns deles estão sendo integrados no core do produto **Windows** e alguns produtos **Office**."

Kabir Khan - **Microsoft** [\(ref\)](#)

"Estou extremamente entusiasmado com o **FastAPI**. É tão divertido!"

Brian Okken - **Python Bytes** podcaster [\(ref\)](#)

"Honestamente, o que você construiu parece super sólido e rebuscado. De muitas formas, eu queria que o **Hug** fosse assim - é realmente inspirador ver alguém que construiu ele."

Timothy Crosley - **criador do Hug** [\(ref\)](#)

"Se você está procurando aprender um **framework moderno** para construir aplicações REST, dê uma olhada no **FastAPI** [...] É rápido, fácil de usar e fácil de aprender [...]"

"Nós trocamos nossas **APIs** por **FastAPI** [...] Acredito que vocês gostarão dele [...]"

Ines Montani - Matthew Honnibal - **fundadores da Explosion AI** - **criadores da spaCy** [\(ref\)](#) - [\(ref\)](#)



"Nós adotamos a biblioteca **FastAPI** para criar um servidor **REST** que possa ser chamado para obter **predições**. [para o Ludwig]"

Piero Molino, Yaroslav Dudin e Sai Sumanth Miryala - **Uber** [\(ref\)](#)

Typer, o FastAPI das interfaces de linhas de comando

Typer

Se você estiver construindo uma aplicação CLI para ser utilizada em um terminal ao invés de uma aplicação web, dê uma olhada no [Typer](#).

Typer é o irmão menor do FastAPI. E seu propósito é ser o **FastAPI das CLIs**.  

Requisitos

Python 3.6+

FastAPI está nos ombros de gigantes:

- [Starlette](#) para as partes web.
- [Pydantic](#) para a parte de dados.

Instalação

```
$ pip install fastapi
```

```
---> 100%
```

Você também precisará de um servidor ASGI para produção, tal como [Uvicorn](#) ou [Hypercorn](#).

```
$ pip install uvicorn[standard]
```

```
---> 100%
```

Exemplo

Crie

- Crie um arquivo `main.py` com:

```
from typing import Optional

from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

► Ou use `async def...`

Rode

Rode o servidor com:

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

```
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [28720]
INFO:      Started server process [28722]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

► Sobre o comando `uvicorn main:app --reload...`

Verifique

Abra seu navegador em <http://127.0.0.1:8000/items/5?q=somequery>.

Você verá a resposta JSON como:

```
{"item_id": 5, "q": "somequery"}
```

Você acabou de criar uma API que:

- Recebe requisições HTTP nas rotas `/` e `/items/{item_id}`.
- Ambas rotas fazem operações `GET` (também conhecido como *métodos* HTTP).
- A rota `/items/{item_id}` tem um *parâmetro de rota* `item_id` que deve ser um `int`.
- A rota `/items/{item_id}` tem um *parâmetro query* `q` `str` opcional.

Documentação Interativa da API

Agora vá para <http://127.0.0.1:8000/docs>.

Você verá a documentação automática interativa da API (fornecida por [Swagger UI](#)):

Fast API 0.1.0 OAS3
/openapi.json

default

GET /items/{item_id} Read Item Get

Parameters

Try it out

Name	Description
item_id * required integer (path)	
q string (query)	

Responses

Code	Description	Links
200	Successful Response	No links
	application/json Controls Accept header.	
422	Validation Error	No links
	application/json	

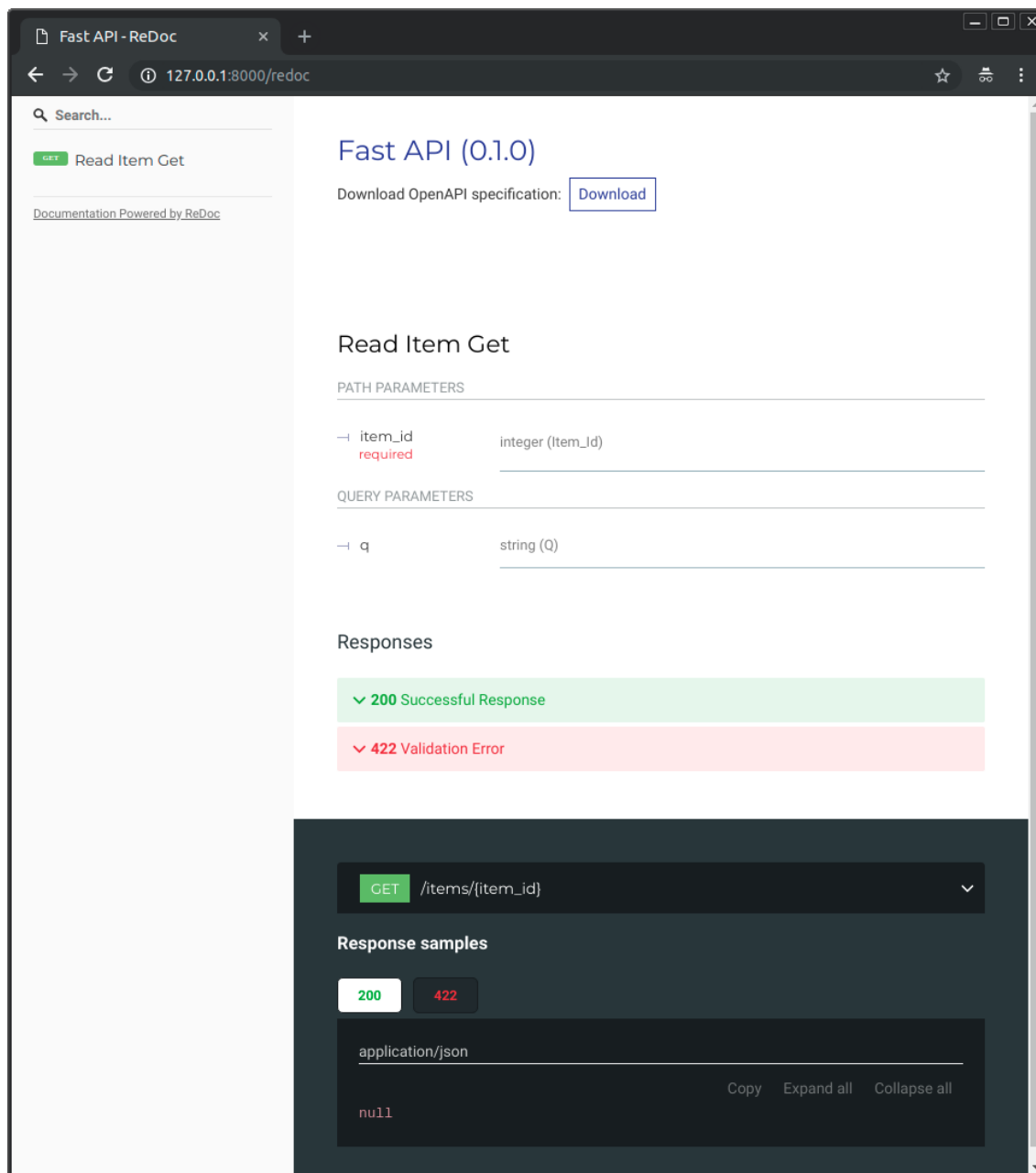
Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string"
      ]
    }
  ]
}
```

Documentação Alternativa da API

E agora, vá para <http://127.0.0.1:8000/redoc>.

Você verá a documentação automática alternativa (fornecida por [ReDoc](#)):



Evoluindo o Exemplo

Agora modifique o arquivo `main.py` para receber um corpo para uma requisição `PUT`.

Declare o corpo utilizando tipos padrão Python, graças ao Pydantic.

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    is_offer: Optional[bool] = None

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}
```

O servidor deverá recarregar automaticamente (porque você adicionou `--reload` ao comando `uvicorn` acima).

Evoluindo a Documentação Interativa da API

Agora vá para <http://127.0.0.1:8000/docs>.

- A documentação interativa da API será automaticamente atualizada, incluindo o novo corpo:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Fast API 0.1.0 OAS3

/openapi.json

default

GET / Read Root Get

GET /items/{item_id} Read Item Get

PUT /items/{item_id} Save Item Put

Parameters Try it out

Name	Description
item_id * required integer (path)	

Request body required application/json

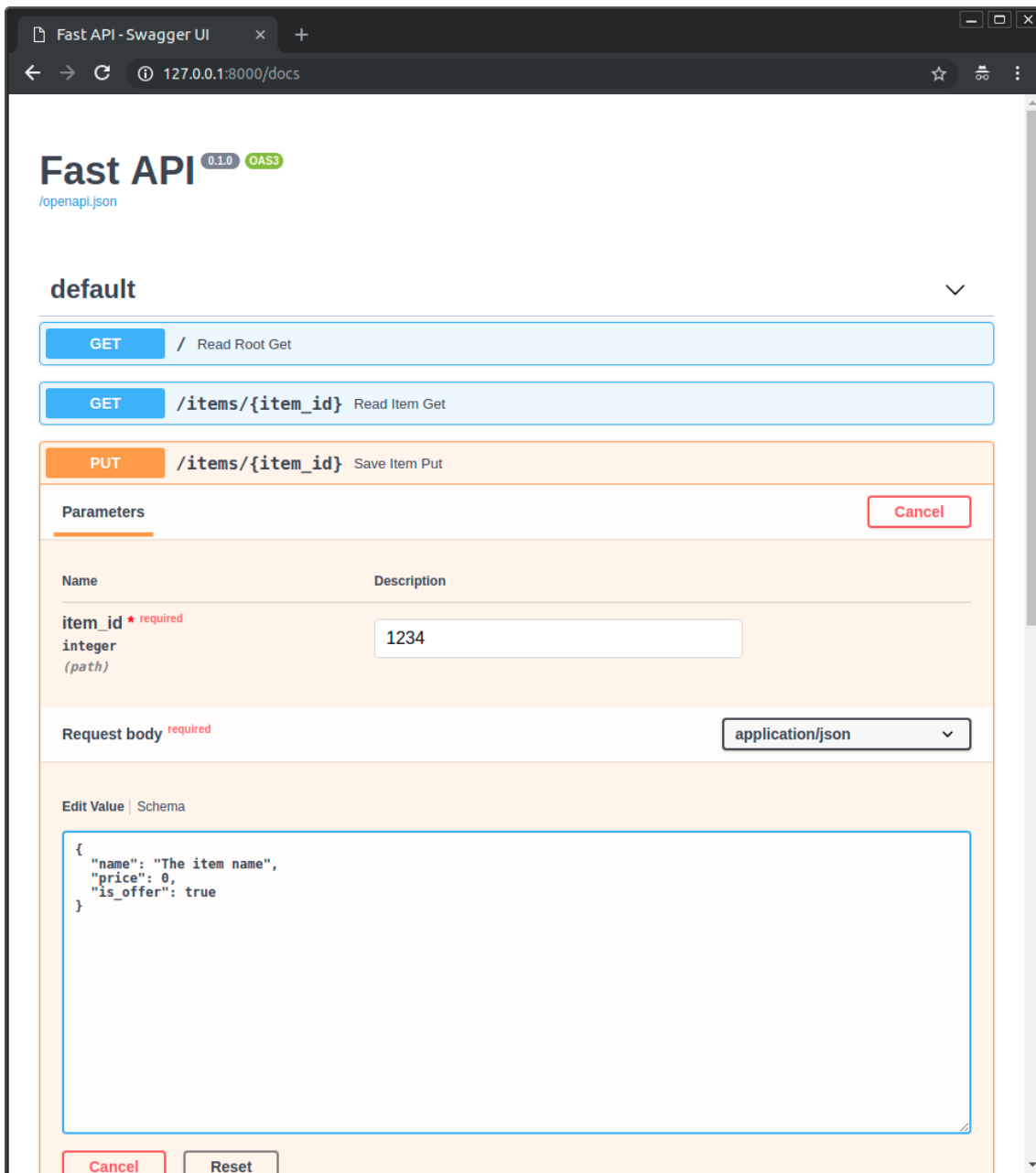
Example Value | Schema

```
{  
  "name": "string",  
  "price": 0,  
  "is_offer": true  
}
```

Responses

Code	Description	Links
200	Successful Response	No links

- Clique no botão "Try it out", ele permiirá que você preencha os parâmetros e interaja diretamente com a API:



- Então clique no botão "Execute", a interface do usuário irá se comunicar com a API, enviar os parâmetros, pegar os resultados e mostrá-los na tela:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Edit Value | Schema

```
{
  "name": "The item name",
  "price": 0,
  "is_offer": true
}
```

Cancel Reset

Execute Clear

Responses

Curl

```
curl -X PUT "http://127.0.0.1:8000/items/1234" -H "accept: application/json" -H "Content-Type: application/json" -d "{\"name\": \"The item name\", \"price\": 0, \"is_offer\": true}"
```

Request URL

```
http://127.0.0.1:8000/items/1234
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "item_name": "The item name", "item_id": 1234 }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 44 content-type: application/json</pre>

Evoluindo a Documentação Alternativa da API

E agora, vá para <http://127.0.0.1:8000/redoc>.

- A documentação alternativa também irá refletir o novo parâmetro da *query* e o corpo:

Fast API - ReDoc

127.0.0.1:8000/redoc#operation/save_item_items__item_id__put

Search...

GET Read Root Get

GET Read Item Get

PUT Save Item Put

Documentation Powered by ReDoc

Save Item Put

PATH PARAMETERS

item_id	integer (Item_Id)
---------	-------------------

REQUEST BODY SCHEMA: application/json

name	string (Name)
price	number (Price)
is_offer	boolean (Is_Offer)

Responses

- ✓ 200 Successful Response
- ✓ 422 Validation Error

PUT /items/{item_id}

Request samples

Payload

application/json

```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```

Copy Expand all Collapse all

Recapitulando

Resumindo, você declara **uma vez** os tipos dos parâmetros, corpo etc. como parâmetros de função.

Você faz com tipos padrão do Python moderno.

Você não terá que aprender uma nova sintaxe, métodos ou classes de uma biblioteca específica etc.

Apenas **Python 3.6+** padrão.

Por exemplo, para um `int` :

```
item_id: int
```

ou para um modelo mais complexo, `Item` :

```
item: Item
```

...e com essa única declaração você tem:

- Suporte ao Editor, incluindo:
 - Completação.
 - Verificação de tipos.
- Validação de dados:
 - Erros automáticos e claros quando o dado é inválido.
 - Validação até para objetos JSON profundamente aninhados.
- Conversão de dados de entrada: vindo da rede para dados e tipos Python. Consegue ler:
 - JSON.
 - Parâmetros de rota.
 - Parâmetros de *query* .
 - *Cookies*.
 - Cabeçalhos.
 - Formulários.
 - Arquivos.
- Conversão de dados de saída de tipos e dados Python para dados de rede (como JSON):
 - Converte tipos Python (`str` , `int` , `float` , `bool` , `list` etc).
 - Objetos `datetime` .
 - Objetos `UUID` .
 - Modelos de Banco de Dados.
 - ...e muito mais.
- Documentação interativa automática da API, incluindo 2 alternativas de interface de usuário:
 - Swagger UI.
 - ReDoc.

Voltando ao código do exemplo anterior, **FastAPI** irá:

- Validar que existe um `item_id` na rota para requisições `GET` e `PUT` .
- Validar que `item_id` é do tipo `int` para requisições `GET` e `PUT` .
 - Se não é validado, o cliente verá um útil, claro erro.
- Verificar se existe um parâmetro de *query* opcional nomeado como `q` (como em `http://127.0.0.1:8000/items/foo?q=somequery`) para requisições `GET` .
 - Como o parâmetro `q` é declarado com `= None` , ele é opcional.
 - Sem o `None` ele poderia ser obrigatório (como o corpo no caso de `PUT`).
- Para requisições `PUT` para `/items/{item_id}` , lerá o corpo como JSON e:
 - Verifica que tem um atributo obrigatório `name` que deve ser `str` .
 - Verifica que tem um atributo obrigatório `price` que deve ser `float` .
 - Verifica que tem an atributo opcional `is_offer` , que deve ser `bool` , se presente.
 - Tudo isso também funciona para objetos JSON profundamente aninhados.
- Converter de e para JSON automaticamente.
- Documentar tudo com OpenAPI, que poderá ser usado por:
 - Sistemas de documentação interativos.

- Sistemas de clientes de geração de código automáticos, para muitas linguagens.
- Fornecer diretamente 2 interfaces *web* de documentação interativa.

Nós arranhamos apenas a superfície, mas você já tem idéia de como tudo funciona.

Experimente mudar a seguinte linha:

```
return {"item_name": item.name, "item_id": item_id}
```

...de:

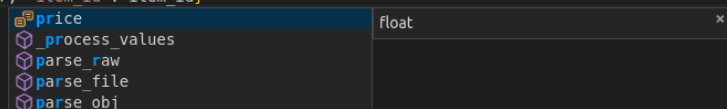
```
... "item_name": item.name ...
```

...para:

```
... "item_price": item.price ...
```

...e veja como seu editor irá auto-completar os atributos e saberá os tipos:

```
1 from fastapi import FastAPI
2 from pydantic import BaseModel
3
4 app = FastAPI()
5
6
7 class Item(BaseModel):
8     name: str
9     price: float
10    is_offer: bool = None
11
12
13 @app.get("/")
14 def read_root():
15     return {"Hello": "World"}
16
17
18 @app.get("/items/{item_id}")
19 def read_item(item_id: int, q: str = None):
20     return {"item_id": item_id, "q": q}
21
22
23 @app.put("/items/{item_id}")
24 def save_item(item_id: int, item: Item):
25     return {"item_name": item.pr, "item_id": item_id}
```



Para um exemplo mais completo incluindo mais recursos, veja [Tutorial - Guia do Usuário](#).

Alerta de Spoiler: o tutorial - guia do usuário inclui:

- Declaração de **parâmetros** de diferentes lugares como: **cabeçalhos**, **cookies**, **campos de formulários** e **arquivos**.
- Como configurar **Limitações de Validação** como `maximum_length` ou `regex`.
- Um poderoso e fácil de usar sistema de **Injeção de Dependência**.
- Segurança e autenticação, incluindo suporte para **OAuth2** com autenticação **JWT tokens** e **HTTP Basic**.

- Técnicas mais avançadas (mas igualmente fáceis) para declaração de **modelos JSON profundamente aninhados** (graças ao Pydantic).
- Muitos recursos extras (graças ao Starlette) como:
 - **WebSockets**
 - **GraphQL**
 - testes extremamente fáceis baseados em `requests` e `pytest`
 - **CORS**
 - **Cookie Sessions**
 - ...e mais.

Performance

Testes de performance da *Independent TechEmpower* mostram aplicações **FastAPI** rodando sob Uvicorn como [um dos frameworks Python mais rápidos disponíveis](#), somente atrás de Starlette e Uvicorn (utilizados internamente pelo FastAPI). (*)

Para entender mais sobre performance, veja a seção [Benchmarks](#).

Dependências opcionais

Usados por Pydantic:

- [ujson](#) - para JSON mais rápido "parsing".
- [email_validator](#) - para validação de email.

Usados por Starlette:

- [requests](#) - Necessário se você quiser utilizar o `TestClient` .
- [aiofiles](#) - Necessário se você quiser utilizar o `FileResponse` ou `StaticFiles` .
- [jinja2](#) - Necessário se você quiser utilizar a configuração padrão de templates.
- [python-multipart](#) - Necessário se você quiser suporte com "parsing" de formulário, com `request.form()` .
- [itsdangerous](#) - Necessário para suporte a `SessionMiddleware` .
- [pyyaml](#) - Necessário para suporte a `SchemaGenerator` da Starlette (você provavelmente não precisará disso com o FastAPI).
- [graphene](#) - Necessário para suporte a `GraphQLApp` .
- [ujson](#) - Necessário se você quer utilizar `UJSONResponse` .

Usados por FastAPI / Starlette:

- [uvicorn](#) - para o servidor que carrega e serve sua aplicação.
- [orjson](#) - Necessário se você quer utilizar `ORJSONResponse` .

Você pode instalar todas essas dependências com `pip install fastapi[all]` .

Licença

Esse projeto é licenciado sob os termos da licença MIT.