

Machine Learning Engineer

Python para ML e IA

Desenvolvimento de API com FastAPI

Leonardo Pena

/ Seja muito bem vindo



OBJETIVO

Apresentar o framework, criar a estrutura inicial e configurar autenticação básica



COMPARATIVO

Já vimos Flask, agora vamos ver porque FastAPI tem vantagens únicas



EXEMPLO

Desenvolveremos na prática uma aplicação

Objetivo dessa primeira parte



Passo 1

Entender o
que é FastAPI
e seus
principais
diferenciais



Passo 2

Criar um
arquivo
principal
(main.py) e
instanciar app
= FastAPI(...)



Passo
3

Configurar
autenticação
HTTP Basic
com
HTTPBasic e
Depends



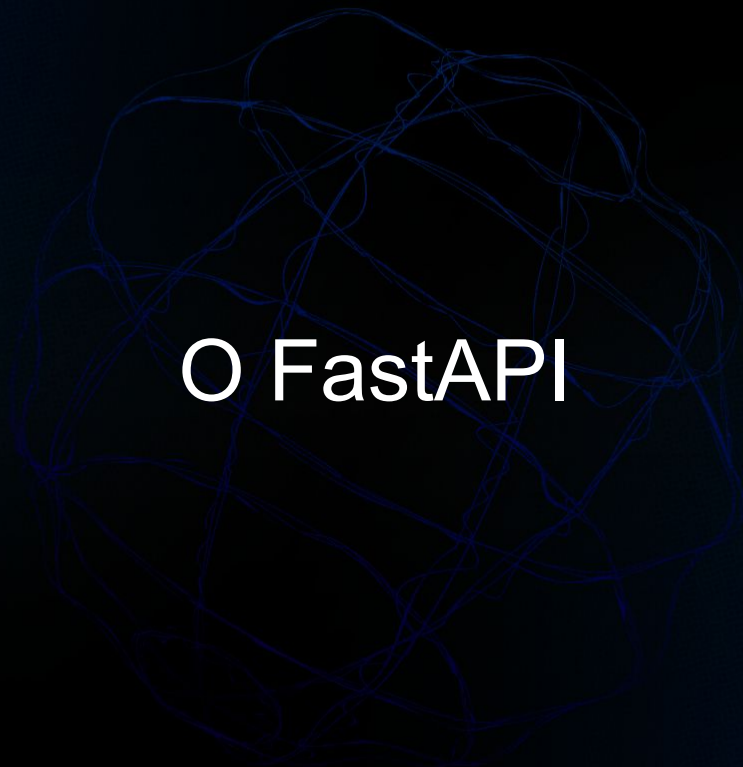
Passo 4

Criar rota
/hello
protegida
por
BasicAuth



Passo
5

Demonstrar
teste inicial
via uvicorn
main:app
--reload



O FastAPI



/ O que é?



- Framework moderno para construção de APIs em Python
- Baseado em tipagem (type hints) do Python 3.6+
- Altamente performático (usa Starlette e Uvicorn)
- **Documentação automática** (Swagger UI /docs e Redoc /redoc)
- Ganho de produtividade e facilidade de manutenção

/ Principais diferenças do Flask



- Performance próxima a frameworks async como Node.js
- Autogeração de docs interativas (Swagger UI)
- Validação automática de tipos via Pydantic
- Simples de configurar rotas e dependências
- Facilidade de migração para quem vem do Flask

A decorative graphic consisting of numerous thin, light gray wavy lines that flow from the bottom left towards the right, creating a sense of movement and depth.

/ Vamos começar
nosso exemplo, mas
antes

- Configure o ambiente virtual
- Instale *fastapi*
- Instale o *uvicorn*

/

Criando nosso main.py

```
main.py > ...  
1  from fastapi import FastAPI  
2  
3  app = FastAPI(  
4      title="My FastAPI API",  
5      version="1.0.0",  
6      description="API de Exemplo com FastAPI"  
7  )  
8  
9  @app.get("/")  
10 async def home():  
11     return "Hello, FastAPI!"  
12
```

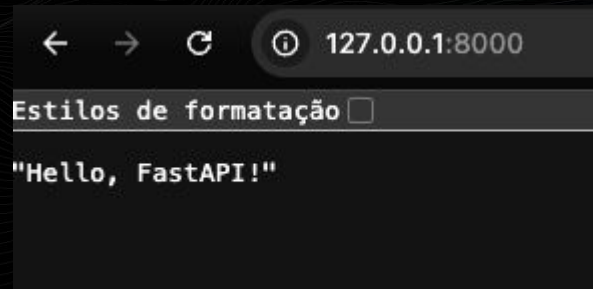

- `title`, `version` e `description` aparecem na doc /docs
- `app` é nossa aplicação principal, similar a `Flask(__name__)` no Flask
- Rotas declaradas via decorators `@app.get()`, `@app.post()`, etc.
- Suporte a `async/await`, o que aproveita recursos assíncronos do Python

```
main.py > ...  
1  from fastapi import FastAPI  
2  
3  app = FastAPI(  
4      title="My FastAPI API",  
5      version="1.0.0",  
6      description="API de Exemplo com FastAPI"  
7  )  
8  
9  @app.get("/")  
10 async def home():  
11     return "Hello, FastAPI!"  
12
```

/

E aí é só rodar o servidor


```
(venv) leonardopena@MacBook-Pro-de-Leonardo aula05-new % uvicorn main:app --reload  
INFO: Will watch for changes in these directories: ['/Users/leonardopena/Documents/aula05-new']  
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
INFO: Started reload process [33763] using Starlette
```



/

A documentação é gerada automaticamente!



A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.

/ Pronto, fizemos nosso primeiro
Hello, FastAPI!



/ Configuração de Autenticação

- Usaremos *HTTPBasic* do *fastapi.security* para BasicAuth
- Precisamos de um “banco” de usuários, mesmo que seja um dicionário simples
- Função *verify_password* para comparar credenciais e retornar username ou *HTTPException*
- Uso de *Depends* para injetar a lógica de verificação na rota
- Em produção, usar HTTPS para proteger credenciais

/ Declarando dicionário Users e Security

```
users = {  
    "user1": "password1",  
    "user2": "password2"  
}  
  
security = HTTPBasic()
```

- users simula um “banco” de credenciais
- security = HTTPBasic() instancia o esquema BasicAuth
- Futuramente, poderíamos implementar hashing de senhas
- Em projetos reais, dados devem vir de um DB
- Objetivo: demonstrar o conceito, sem complexidade no backend

/

Função

verify_password

```
main.py > ...
1  from fastapi import FastAPI
2  from fastapi import Depends, HTTPException, status
3  from fastapi.security import HTTPBasic, HTTPBasicCredentials
4
5  app = FastAPI(
6      title="My FastAPI API",
7      version="1.0.0",
8      description="API de Exemplo com FastAPI"
9  )
10
11  # Banco de dados de usuários em memória para autenticação
12  users = {
13      "user1": "password1", # Usuário 1
14      "user2": "password2" # Usuário 2
15  }
16
17  security = HTTPBasic()
18
19
20  def verify_password(credentials: HTTPBasicCredentials = Depends(security)):
21      username = credentials.username
22      password = credentials.password
23      if username in users and users[username] == password:
24          return username
25      raise HTTPException(
26          status_code=status.HTTP_401_UNAUTHORIZED,
27          detail="Credenciais inválidas",
28          headers={"WWW-Authenticate": "Basic"},
29      )
30
31  @app.get("/")
32  async def home():
33      return "Hello, FastAPI!"
34
```

Rota /hello protegida por Auth

```
35 @app.get("/hello")
36 ✓ async def hello(username: str = Depends(verify_password)):
37     return {"message": f"Hello, {username}!"}
38
```

Fazer login

http://127.0.0.1:8000

Nome de usuário

Senha

← → ↻ ⓘ 127.0.0.1:8000/hello

Estilos de formatação ☐

{"message": "Hello, user1!"}

/

Rota protegida na documentação

My FastAPI API 1.0.0 OAS 3.1

/openapi.json

API de Exemplo com FastAPI

Authorize



default



GET

/ Home

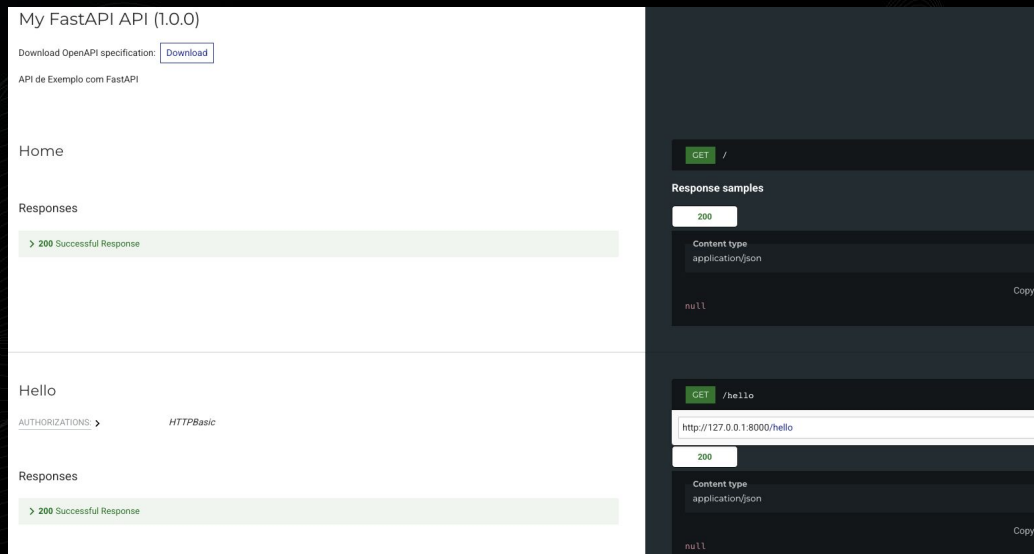


GET

/hello Hello



Outra documentação gerada é a /redoc



A decorative background consisting of numerous thin, wavy, light gray lines that create a sense of movement and depth, primarily concentrated on the left side of the slide.

/ Boas Práticas em Autenticação Básica

- Em produção, use HTTPS sempre (evitar credenciais em texto puro)
- Considere hashing de senhas no DB, mesmo que simples (bcrypt, passlib)
- Para APIs externas ou públicas, preferir OAuth2/Bearer Token (mais seguro)
- Limitar tentativas (Rate Limiting) para evitar brute force
- Logging de acessos pode ajudar a detectar uso indevido

/ Comparação Flask



Flask

- Flask é minimalista, poucas convenções
- FastAPI traz convenções sobre tipagem e doc integradas
- Em Flask, não há doc automática out-of-the-box (precisamos flasgger, etc.)
- Em FastAPI, dependências e tipagem são nativos
- Ambos convivem bem, escolha depende do estilo do time e requisitos do projeto

POSTECH

FIAP + alura