



Introdução

Vamos mergulhar nas funcionalidades que tornam o FastAPI tão poderoso e amigável para desenvolvedores.

Introdução

O que é o FastAPI e por que ele é tão popular?

No universo do desenvolvimento web com Python, o FastAPI emergiu como uma das ferramentas mais queridas e eficientes para a criação de APIs (Interfaces de Programação de Aplicações). Se você busca alta performance, facilidade de uso e um desenvolvimento ágil, o FastAPI é a escolha certa.

Este ebook foi pensado para você, que deseja compreender de forma clara e objetiva o que é o FastAPI, suas principais funcionalidades e, o mais importante, como utilizá-lo na prática com exemplos do mundo real.

O FastAPI é um framework web moderno, de alto desempenho, para construir APIs com Python 3.7+ baseado nos type hints (dicas de tipo) padrões do Python. A "mágica" por trás de sua velocidade reside na sua base: ele é construído sobre o Starlette, para a parte web, e o Pydantic, para a validação de dados.

A grande sacada do FastAPI é que ele foi construído sobre duas bases muito sólidas:

- Starlette: Para a parte de alta performance e suporte a operações assíncronas. Isso significa que sua API pode lidar com muitas requisições ao mesmo tempo, de forma muito eficiente.
- Pydantic: Para a validação de dados de forma a



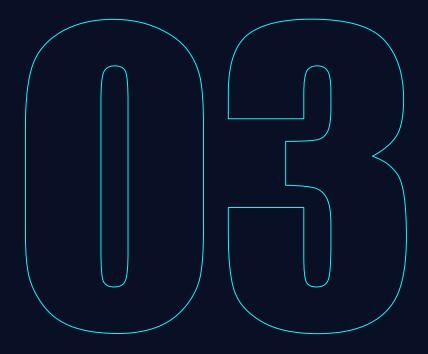
Principais Funcionalidades

Vamos mergulhar nas funcionalidades que tornam o FastAPI tão poderoso e amigável para desenvolvedores.

Principais Funcionalidades

O que ele é capaz de fazer?

- 1. Operações de Rota Simples e Intuitivas: Criar "rotas" ou "endpoints" (os diferentes caminhos da sua API) é extremamente simples. Você utiliza decoradores (@app.get(), @app.post(), etc.) para associar uma função a um método HTTP e a um caminho.
- 2. Validação de Dados com Pydantic: O FastAPI utiliza a biblioteca Pydantic para definir modelos de dados. Isso significa que você pode declarar como seus dados de entrada e saída devem ser, e o FastAPI garante que eles sigam esse formato. Se um dado inválido for enviado, ele automaticamente retorna um erro claro e descritivo.
- **3. Documentação Interativa e Automática:** Essa é, sem dúvidas, uma das funcionalidades mais aclamadas. Ao criar suas rotas e modelos de dados, o FastAPI gera automaticamente uma documentação interativa. Basta acessar /docs no navegador para interagir com sua API, testar os endpoints e ver os modelos de dados esperados.
- **4. Suporte a Programação Assíncrona:** O FastAPI é construído para ser assíncrono por padrão, utilizando async e await. Isso permite que sua aplicação lide com múltiplas requisições simultaneamente de forma muito eficiente, ideal para aplicações que dependem de operações de I/O (entrada e saída), como consultas a bancos de dados e chamadas a outras APIs.
- **5. Injeção de Dependências:** Um sistema poderoso e fácil de usar que permite gerenciar dependências de forma organizada. Isso é útil para, por exemplo, gerenciar conexões com banco de dados, autenticação de usuários e configurações.



Pilares do Funcionamento

O FastAPI funciona combinando três pilares tecnológicos de forma inteligente, transformando o código Python que você escreve em uma API web extremamente rápida, confiável e bem documentada.

Pilares do Funcionamento

Como a Mágica acontece?

1. O Motor: Performance Extrema (ASGI + Starlette)

A velocidade do FastAPI vem de sua fundação. Ele não lida diretamente com a web, ele delega isso para:

ASGI (Asynchronous Server Gateway Interface): É o padrão moderno para servidores Python. Ele é **assíncrono**, o que significa que o servidor (Uvicorn) não fica travado esperando por tarefas lentas, como uma consulta ao banco de dados. Ele pode gerenciar centenas de requisições simultaneamente, otimizando o tempo de espera.

Starlette: É um microframework web levíssimo e ultra-rápido que serve como o "chassi" do FastAPI. Ele cuida de todo o trabalho pesado de roteamento, tratamento de requisições e respostas, WebSockets, etc.

2. O Cérebro: Inteligência e Confiabilidade (Pydantic + Type Hints)

Esta é a parte que torna o desenvolvimento tão fácil e seguro.

Python Type Hints (Dicas de Tipo): São as anotações que você usa no seu código, como def minha_funcao(nome: str, idade: int).

Pydantic: É uma biblioteca que usa essas dicas de tipo em tempo de execução para validar, converter e documentar seus dados.

Quando o FastAPI vê nome: str, ele usa o Pydantic para garantir que o dado recebido é realmente uma string. Se não for, ele automaticamente rejeita a requisição com um erro claro, antes mesmo de seu código ser executado.

Pilares do Funcionamento

Como a Mágica acontece?

3. O Resultado: Automação Total (Injeção de Dependências e OpenAPI)

A combinação dos pilares anteriores permite automatizar tarefas complexas.

Injeção de Dependências: O FastAPI analisa os parâmetros da sua função (como Depends(...)) e cuida de fornecer o que é necessário (ex: uma sessão de banco de dados), promovendo a reutilização de código.

Documentação OpenAPI: Ao ler suas rotas, os type hints e os modelos Pydantic, o FastAPI tem toda a informação que precisa para gerar um "mapa" completo da sua API (um esquema OpenAPI). Esse mapa é usado para criar as páginas de documentação interativas em /docs e /redoc automaticamente.



Ciclo de Vida de Requisições

Essencialmente, o FastAPI age como um guarda de segurança inteligente e um assistente prestativo na porta de entrada da sua lógica de negócio, garantindo que apenas dados válidos entrem e automatizando todo o trabalho repetitivo.

Ciclo de vida de Requisições

Processamento do código

Imagine que um cliente envia uma requisição POST /produtos com dados JSON.

- 1. Chegada: A requisição chega ao servidor ASGI (Uvicorn).
- **2. Roteamento:** O Starlette identifica a rota (/produtos) e a função correspondente.
- **3. Validação (A Mágica):** O FastAPI inspeciona a função def criar_produto(produto: ProdutoModel):.
 - Ele vê produto: ProdutoModel.
 - Usa o Pydantic para pegar o JSON da requisição, validar campo por campo contra o modelo ProdutoModel e convertê-lo em um objeto Python.
 - Se a validação falhar: O processo para e uma resposta de erro 422 (com detalhes do que está errado) é retornada imediatamente.
- **4. Execução:** Se a validação for bem-sucedida, sua função criar_produto é finalmente executada, recebendo um objeto produto já limpo, validado e pronto para uso.
- **5. Resposta:** Sua função retorna um resultado. O FastAPI o converte para JSON e o envia de volta ao cliente.



Mão na Massa

Como utilizar o FastAPI? Vamos criar nossa primeira API com FastAPI. É mais simples do que você imagina!

Mão na Massa!

Como utilizar o FastAPI

1. Instalação: Primeiro, você precisa instalar o FastAPI e um servidor ASGI para rodar nossa aplicação, como o Uvicorn.

```
pip install fastapi uvicorn[standard]
```

2. Primeiro arquivo: Crie um arquivo chamado main.py e adicione o seguinte código

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def ler_raiz():
    return {"message": "Olá, Mundo!"}
```

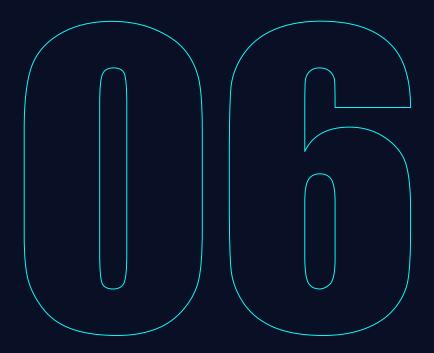
3. Executando a API: No seu terminal, execute o seguinte comando

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

- main: se refere ao arquivo main.py.
- app: o objeto FastAPI que criamos dentro do arquivo.
- --reload: faz com que o servidor reinicie automaticamente após qualquer alteração no código.

Agora, abra seu navegador e acesse http://127.0.0.1:8000. Você verá a mensagem {"message":"Olá, Mundo!"}.

Parabéns, você acabou de criar sua primeira API com FastAPI!



Nesta seção, vamos explorar exemplos de códigos em contextos reais, explicando cada parte para um entendimento completo.

Exemplos Reais

Exemplo 1: API para um Blog Simples

Vamos criar uma API para um blog que permite listar todas as postagens e obter uma postagem específica pelo seu ID.

```
# main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Optional

# --- Modelos de Dados (Pydantic) ---
# Define a estrutura de uma postagem do blog
class Postagem(BaseModel):
    id: int
    titulo: str
    conteudo: str
    autor: Optional[str] = None # Campo opcional
```

- 1º: Importamos as principais classes a serem utilizadas para trabalharmos com o FastAPI.
- **2º:** class Postagem(BaseModel): Aqui definimos a "cara" de uma postagem. Todo objeto Postagem deve ter um id (inteiro), um titulo (string) e um conteudo (string). O campo autor é opcional. O FastAPI usará isso para validar os dados de saída.

Exemplos Reais

3º: Criando uma lista de dicionários para simular um banco de dados.

```
# --- Instância do FastAPI ---
app = FastAPI()
```

4º: Inicializado o FastAPI na variável "app"

Exemplos Reais

```
# --- Rotas (Endpoints) ---

# Rota para listar todas as postagens
@app.get("/postagens", response_model=List[Postagem])
def get_todas_postagens():
    """
    Retorna uma lista de todas as postagens do blog.
    """
    return db_postagens
```

5º: Este decorador cria um endpoint GET no caminho /postagens.

- response_model=List[Postagem]: informa ao FastAPI que a resposta será uma lista (List) de objetos que seguem o modelo Postagem. Isso ajuda na documentação e na validação dos dados de saída.

Exemplos Reais

6º: Cria um endpoint GET que aceita um "parâmetro de caminho". O valor passado em {postagem_id} na URL será recebido pela função.

- def get_postagem_por_id(postagem_id: int): A anotação de tipo postagem_id: int diz ao FastAPI para converter o parâmetro da URL para um inteiro e validá-lo. Se alguém tentar passar um texto, receberá um erro automático.
- raise HTTPException(...): Se o loop for terminar e a postagem não for encontrada, lançamos uma exceção HTTP com o status 404 e uma mensagem de erro.

Agradecimentos

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, e diagramado por humano. O passo a passo se encontra no meu Github

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizado uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.



https://github.com/hrades/prompts-recipe-to-create-a-ebook

