

Introdução aos Frameworks Web Python



Presented By

João Victor



Flask
web development,
one drop at a time



django



FastAPI

Sobre mim



João Victor da Costa Gomes

Sou estudante de graduação de Ciência da Computação e tenho muito interesse na linguagem Python devido sua simplicidade, praticidade e pelo amplo suporte oferecido pela comunidade. Tenho utilizado essa linguagem para:

SEGMENTAÇÃO DE IMAGENS

REDUÇÃO DE DIMENSIONALIDADE DE DADOS

DESENVOLVIMENTO DE APLICAÇÕES WEB



joao-victor-costa-gomes



jvcgomes14@gmail.com



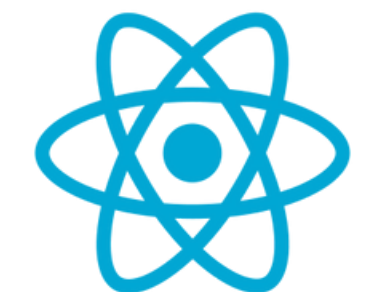
joaovictorcgomes

O que um framework?



Um framework é um **conjunto estruturado de ferramentas e componentes reutilizáveis** para garantir um desenvolvimento eficiente e organizado.

Eles trazem funcionalidades já determinadas para agilizar o processo e **evitar que as pessoas tenham que reescrever essas funções frequentemente.**



Framework web



Um framework web é um tipo específico de framework que ajuda a **construir aplicações na web**, fornecendo ferramentas e convenções para lidar com aspectos essenciais desse tipo de aplicação.



Rotas e URLs (gerenciamento de endpoints)

Banco de dados (ORMs para manipulação de dados)

Autenticação de usuários

Gerenciamento de sessões e cookies

Renderização de páginas HTML (como templates)

Frameworks Web Python mais famosos



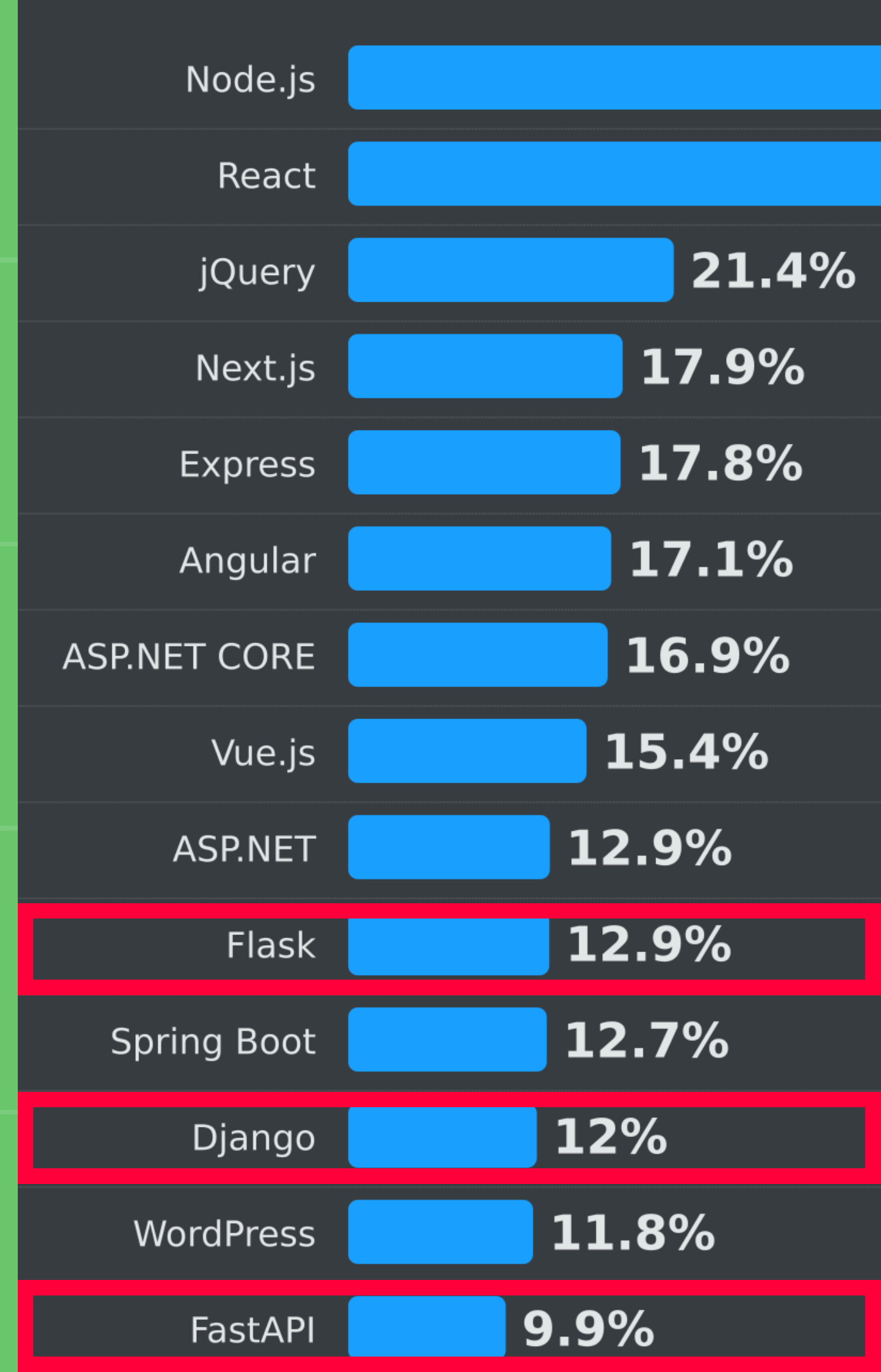
Flask

web development,
one drop at a time

django



FastAPI



Tópicos de cada framework



Breve apresentação

Instalação

Estrutura de Desenvolvimento

Criação da página inicial

Criação de templates

Conexão de arquivos estáticos

Flask - Introdução

O Flask é um **microframework** web para Python que foi projetado para ser **simples**, **leve** e **flexível**, com a intenção de permitir ao desenvolvedor a escolha de como organizar a aplicação, sem impor muitas restrições.

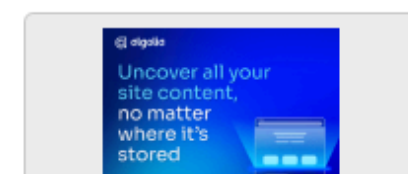
Project Links

[Donate](#)
[PyPI Releases](#)
[Source Code](#)
[Issue Tracker](#)
[Chat](#)

Contents

[Welcome to Flask](#)
[User's Guide](#)
[API Reference](#)
[Additional Notes](#)

Quick search

Flask

Welcome to Flask's documentation. Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.

Get started with [Installation](#) and then get an overview with the [Quickstart](#). There is also a more detailed [Tutorial](#) that shows how to create a small but complete application with Flask. Common patterns are described in the [Patterns for Flask](#) section. The rest of the docs describe each component of Flask in detail, with a full reference in the [API](#) section.

Flask depends on the [Werkzeug](#) WSGI toolkit, the [Jinja](#) template engine, and the [Click](#) CLI toolkit. Be sure to check their documentation as well as Flask's when looking for information.

User's Guide

Ele **deixa várias responsabilidades para o desenvolvedor**, oferecendo grande flexibilidade, mas também exigindo que o desenvolvedor escolha e adicione algumas funcionalidades extras.

Flask - Instalação

Maneira mais fácil com o PIP, instalador de pacotes Python



```
pip install Flask
```


Flask - Primeira página

Só com essas poucas linhas de código em um arquivo chamado app.py, e executando ele, já temos uma aplicação simples rodando.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(debug=True)
```



← → ↻ ⓘ 127.0.0.1:5000

⌵ | k Chicago Crime k The Complete

Hello, World!

Flask - Diferentes rotas

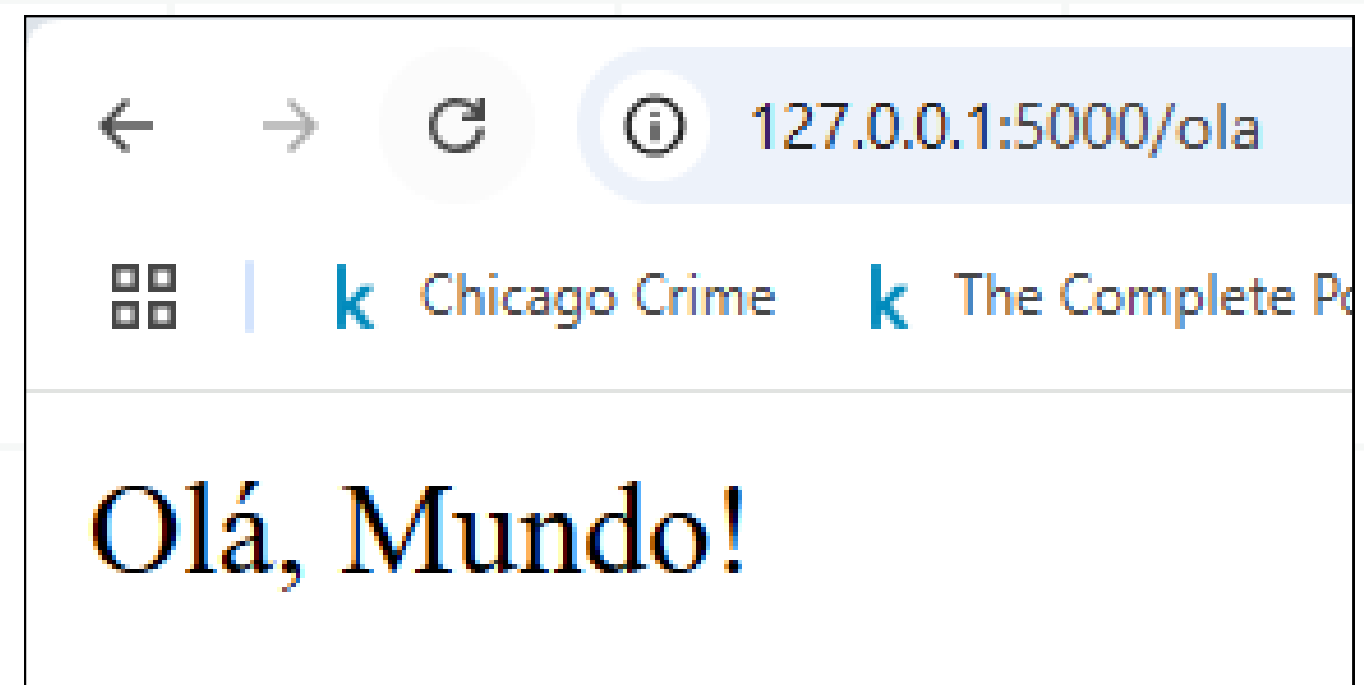
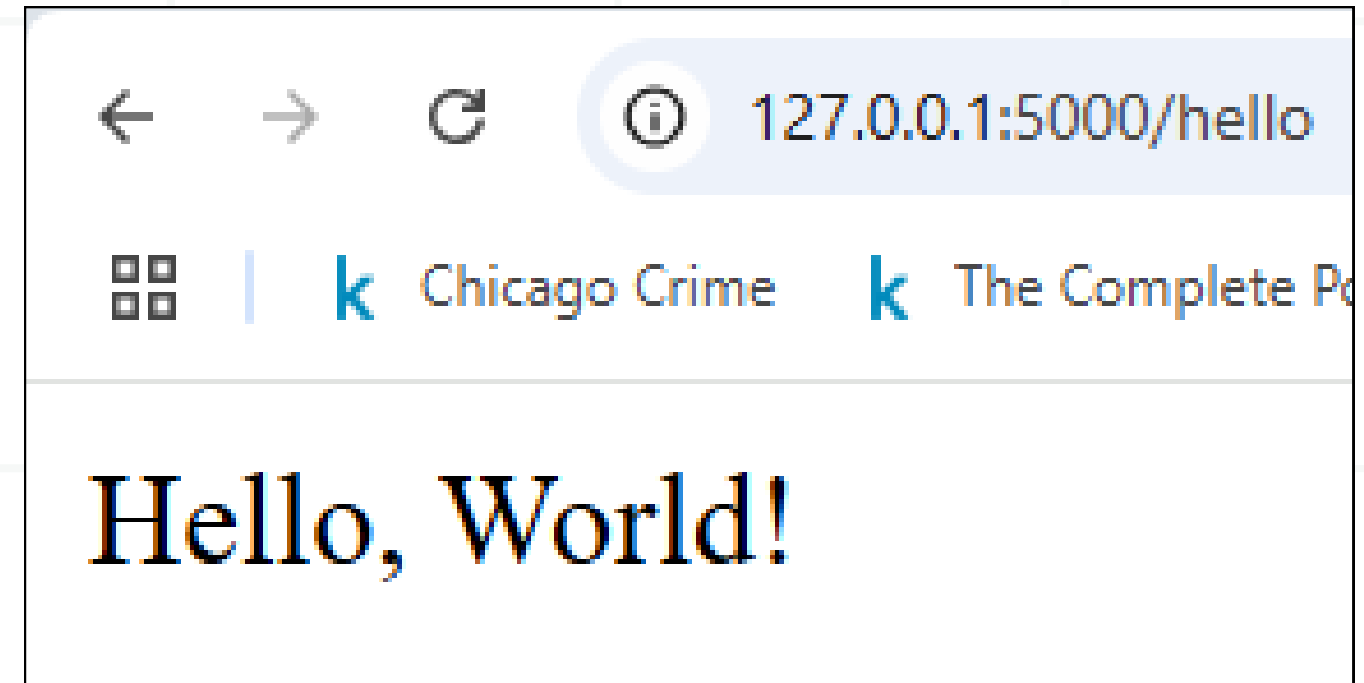
```
from flask import Flask

app = Flask(__name__)

@app.route('/hello')
def hello():
    return "Hello, World!"

@app.route('/ola')
def ola():
    return "Olá, Mundo!"

if __name__ == "__main__":
    app.run(debug=True)
```



Flask - Adicionando templates

Através função `render_template()` importada do flask. Mas primeiro devemos criar um template `.html` dentro do diretório `templates/`

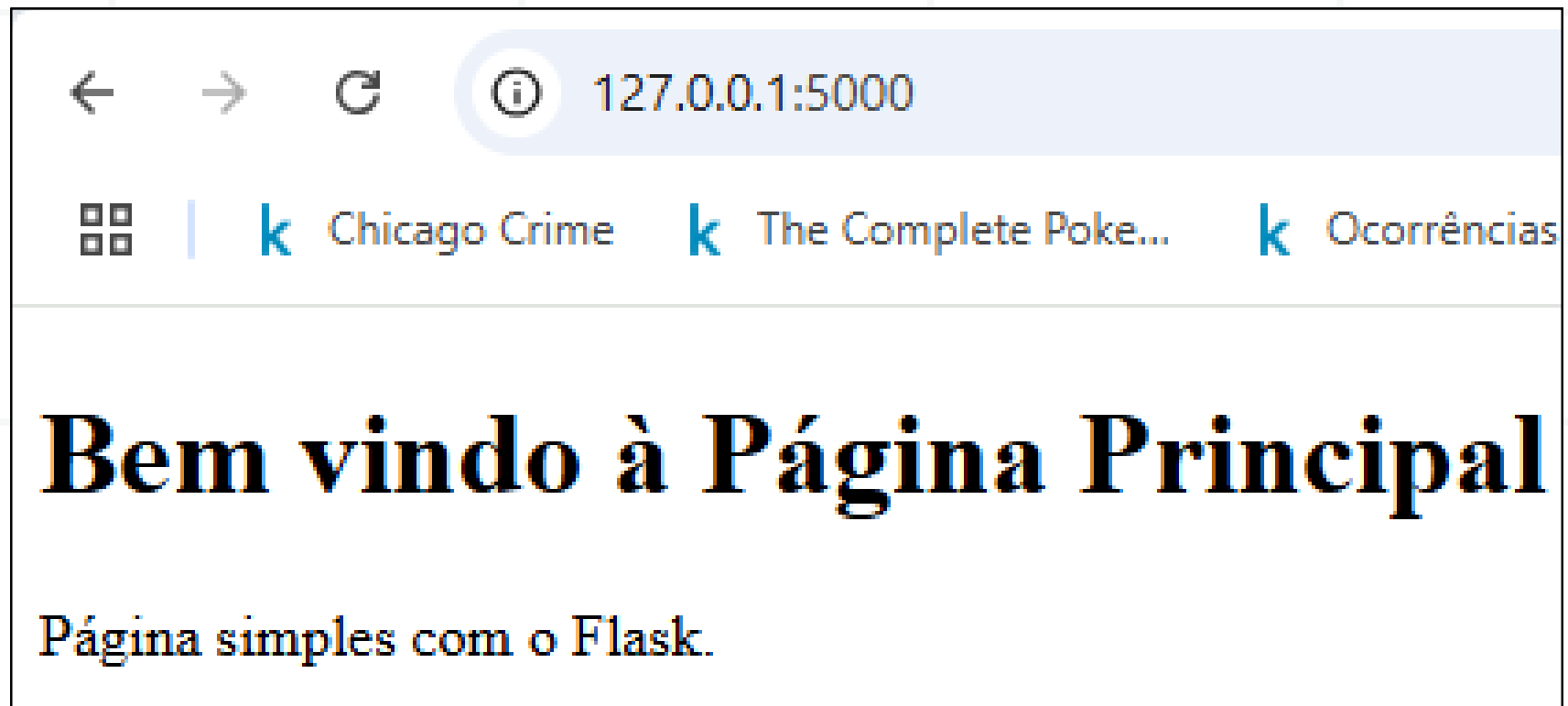
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Página Principal</title>
</head>
<body>
  <h1>Bem vindo à Página Principal</h1>
  <p>Página simples com o Flask.</p>
</body>
</html>
```

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

if __name__ == "__main__":
    app.run(debug=True)
```



Flask - Conectando arquivos estáticos

Criamos os diretórios "static/css/" e "static/js/" e neles colocamos arquivos para teste.

```
flask > static > js > JS script.js
1  alert("Bem vindo à Página Principal!")
```

```
flask > static > css > # style.css > body
1  body {
2      color: red;
3  }
```

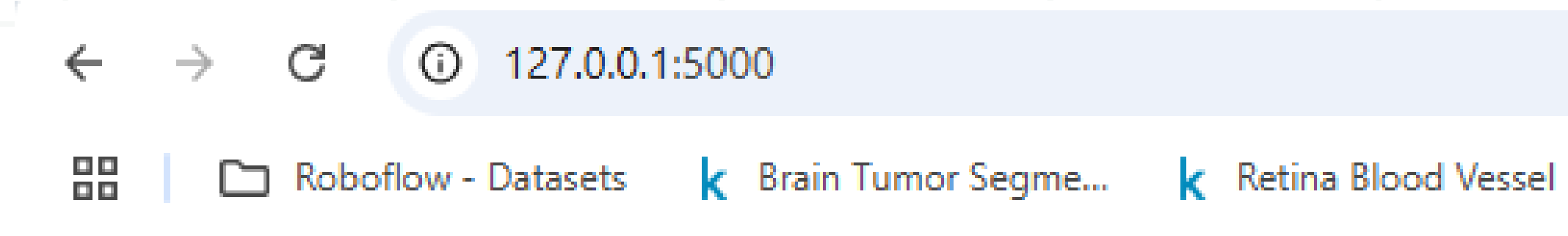
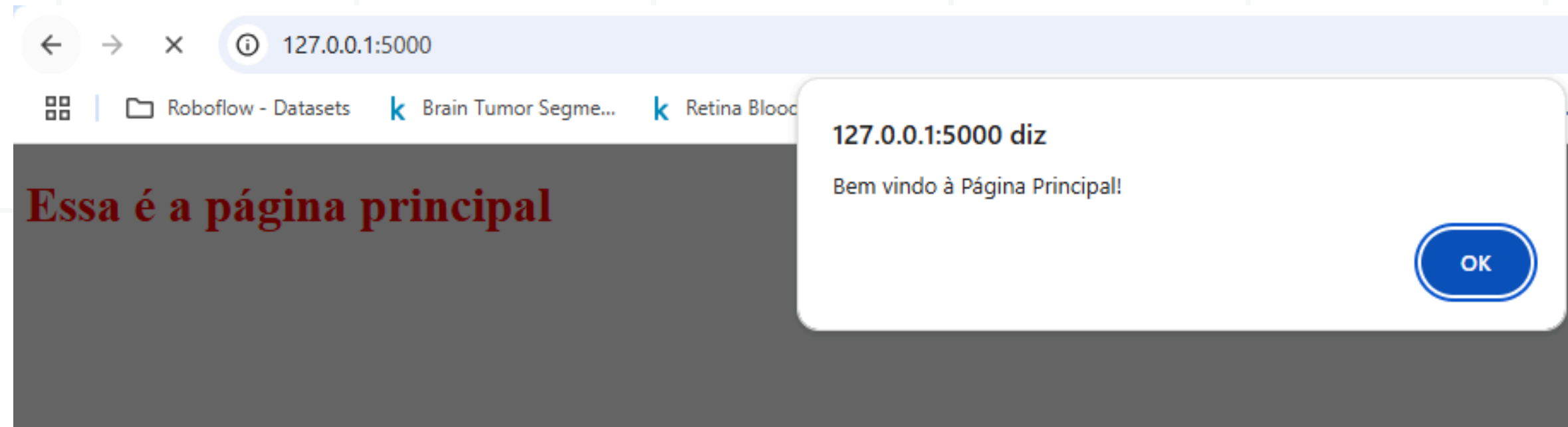
Conectamos eles no template através da função "{{url_for('static', filename='')}}"

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
  <h1>Essa é a página principal</h1>

  <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

Flask - Conectando arquivos estáticos

Resultado:



Essa é a página principal

Flask - Conexão com banco de dados



```
pip install Flask-SQLAlchemy
```

Através do Flask-SQLAlchemy, para interação com o banco de dados SQL através da linguagem Python.

Configuramos para inicializar o banco de dados com nossa aplicação Flask e executamos ela. Isso criará um banco de dados SQLite, mas sem nenhum dado e nenhuma configuração de tabela.

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Configuração do banco de dados
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db' # Banco de dados local SQLite
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False # Para evitar avisos desnecessários

# Inicializa o banco de dados
db = SQLAlchemy(app)
```

```
if __name__ == "__main__":

    # Cria o banco de dados e
    with app.app_context():
        db.create_all()

    app.run(debug=True)
```

Flask - Conexão com banco de dados

Para adicionarmos uma tabela, nos criamos um model. Nele definimos o nome da tabela que queremos criar e quais campos estarão presentes nela.

```
# Inicializa o banco de dados
db = SQLAlchemy(app)

# Define o modelo (tabela) User
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    nome = db.Column(db.String(100), nullable=False)

    def __repr__(self):
        return f'<User {self.nome}>'
```

Em seguida rodamos a aplicação novamente. Com isso, teremos um banco de dados SQLite com a tabela User configurada, mas ainda sem nenhum dado.

Flask - Conexão com banco de dados

Para acelerar as coisas, podemos só rodar esse código em um arquivo separado. Já que o intuito é mostrar esses dados no template.

```
from app import db, User, app

with app.app_context():
    usuario1 = User(nome='João Victor')
    usuario2 = User(nome='Ana Maria')
    usuario3 = User(nome='Pedro Augusto')

    db.session.add(usuario1)
    db.session.add(usuario2)
    db.session.add(usuario3)

    db.session.commit()

print(f'Usuários adicionados com sucesso')
```

Flask - Mostrar dados no template

Para mostrar os dados no template, você tem que fazer uma query no seu banco de dados SQLite e “jogar” no template como contexto.

```
@app.route('/')
def home():
    usuarios = User.query.all()
    return render_template('index.html', usuarios=usuarios)
```

Depois só passar esse contexto no .html utilizando a linguagem de templates do Jinja2

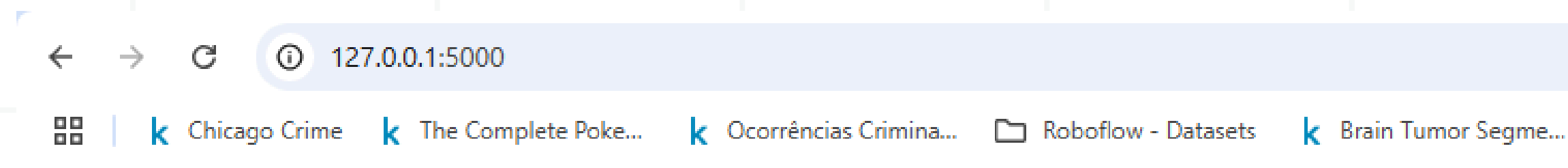
```
<body>
  <h1>Bem vindo à Página Principal</h1>
  <p>Página simples com o Flask.</p>

  <h2>Usuários Cadastrados:</h2>
  <p>{{ usuarios }}</p>

  <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
```

Flask - Mostrar dados no template

Resultado:



Bem vindo à Página Principal

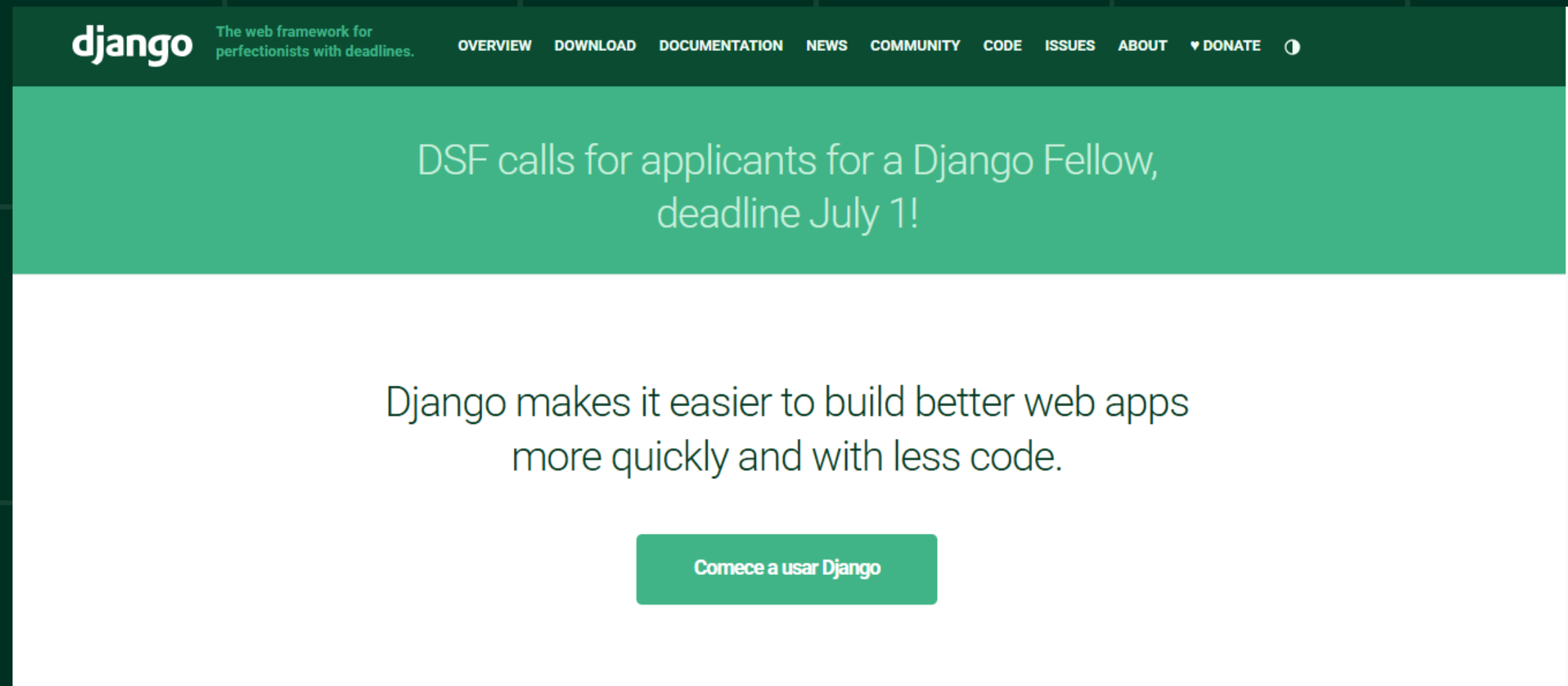
Página simples com o Flask.

Usuários Cadastrados:

[<User João Victor>, <User Ana Maria>, <User Pedro Augusto>]

Django - Introdução

“Django é um framework web Python de alto nível que **incentiva o desenvolvimento rápido** e um design limpo e pragmático. **Ele cuida de grande parte da complexidade do desenvolvimento web**, para que você possa se concentrar em escrever seu aplicativo sem precisar reinventar a roda.”



Django - Instalação

Instalação com o gerenciador de pacotes PIP



```
pip install django
```

Django - Instalação

CRIANDO UM PROJETO



```
django-admin startproject meu_projeto
```

OU ASSIM CASO VOCÊ QUEIRA CRIAR O PROJETO EM UM DIRETÓRIO ESPECÍFICO E NÃO EM UM NOVO



```
django-admin startproject meu_projeto .
```

Django - Instalação

CRIANDO UM APLICATIVO



```
django-admin startapp meu_app
```


Django - Instalação

PROJETO

É a estrutura principal que contém todas as configurações do Django, como configurações globais, URLs, e as definições do banco de dados. **Ele pode conter múltiplas aplicações.**

X

APLICATIVO

É uma parte do projeto que implementa funcionalidades específicas, como autenticação de usuários, gerenciamento de produtos, etc.

Django - Primeira página


Plugamos nosso aplicativo na lista de aplicativos instalados no `meu_projeto/settings.py`


```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    # Nosso aplicativo  
    'meu_app',  
]
```

Django - Primeira página

Criar o diretório templates/
dentro do diretório de nosso
aplicativo e criar nosso
arquivo .html.

Criar uma view para esse
template no arquivo views.py

```
django > meu_app > templates > index.html >  html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Página Principal</title>
7  </head>
8  <body>
9      <h1>Página Principal</h1>
10     <p>Seja bem vindo!</p>
11 </body>
12 </html>
```

```
django > meu_app >  views.py > ...
1  from django.shortcuts import render
2
3  def hello(request):
4      return render(request, 'index.html')
5
```

Django - Primeira página

Importar sua view no arquivo `urls.py` do diretório de seu projeto e definir uma URL para ela.

```
from django.contrib import admin
from django.urls import path
from meu_app.views import hello

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', hello, name='hello'),
]
```

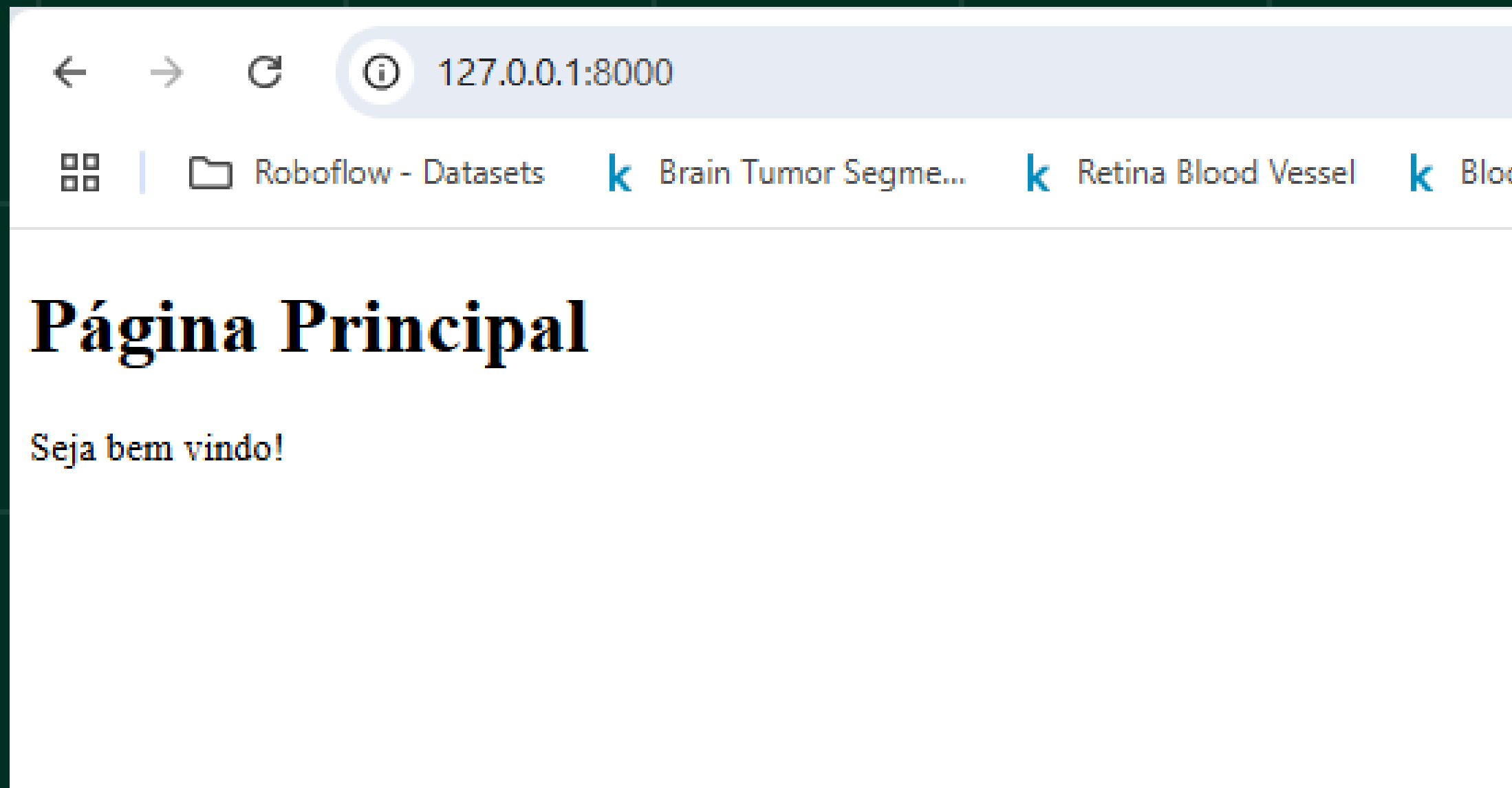
Rodar o projeto com o seguinte comando:



```
python manage.py runserver
```

Django - Primeira página

Resultado:



Django - Conectando arquivos estáticos

Certifique-se que no settings.py existe o trecho... `STATIC_URL = 'static/'`

E em seguida crie os diretórios static/css/ e static/js/ dentro do seu diretório de aplicativo e coloque seus arquivos .css e .js dentro deles.

```
USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.2

DEFAULT_AUTO_FIELD = 'django.db.models.
```

```
django > meu_app > static > js > JS script.js
1 alert("Você chegou na página principal!")
```

```
django > meu_app > static > css > # style.css > body
1 body {
2     color: red;
3 }
```

Django - Conectando arquivos estáticos

Em seguida você adiciona `{% load static %}` no topo do seu template e conecta os arquivos estáticos seguindo esse modelo:

```
<link rel="stylesheet" href="{% static 'css/style.css' %}">
<script src="{% static 'js/script.js' %}"></script>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Página Principal</title>

  <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
  <h1>Página Principal</h1>
  <p>Seja bem vindo!</p>

  <script src="{% static 'js/script.js' %}"></script>
</body>
</html>
```

Só rodar o projeto novamente com o “python manage.py runserver”, acessar a rota raiz e você terá um resultado semelhante ao criado com o Flask

Django - Conectando ao banco de dados

Por padrão, já vem configurado para criar um banco de dados SQLite quando você rodar a aplicação.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

Você pode criar novas tabelas para ele através do arquivo `models.py` dos aplicativos

```
from django.db import models  
  
class Usuario(models.Model):  
    nome = models.CharField(max_length=100)  
  
    def __str__(self):  
        return f'{self.nome}'
```

Para inserir essa tabela (e outras tabelas padrão do Django) basta rodar os seguintes comandos:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Django - Conectando ao banco de dados

Se você quer interagir com o banco de dados na tela do admin. Você deve criar um superusuário rodando o seguinte comando



```
python manage.py createsuperuser
```

E configurar seu modelo para aparecer na tela do admin em admin.py

```
from django.contrib import admin

from .models import Usuario

@admin.register(Usuario)
class UsuarioAdmin(admin.ModelAdmin):
    list_display = ['nome']
```

Depois só rodar o projeto novamente e acessar a rota **/admin** e **inserir as credenciais cadastradas com o superuser** que você criou.

Django - Mostar dados no template

Basta fazer uma consulta simples dentro da sua view e em seguida passar para o template como um dicionário.

```
from django.shortcuts import render
from .models import Usuario

def hello(request):

    usuarios = Usuario.objects.all()

    return render(request, 'index.html', {'usuarios':usuarios})
```

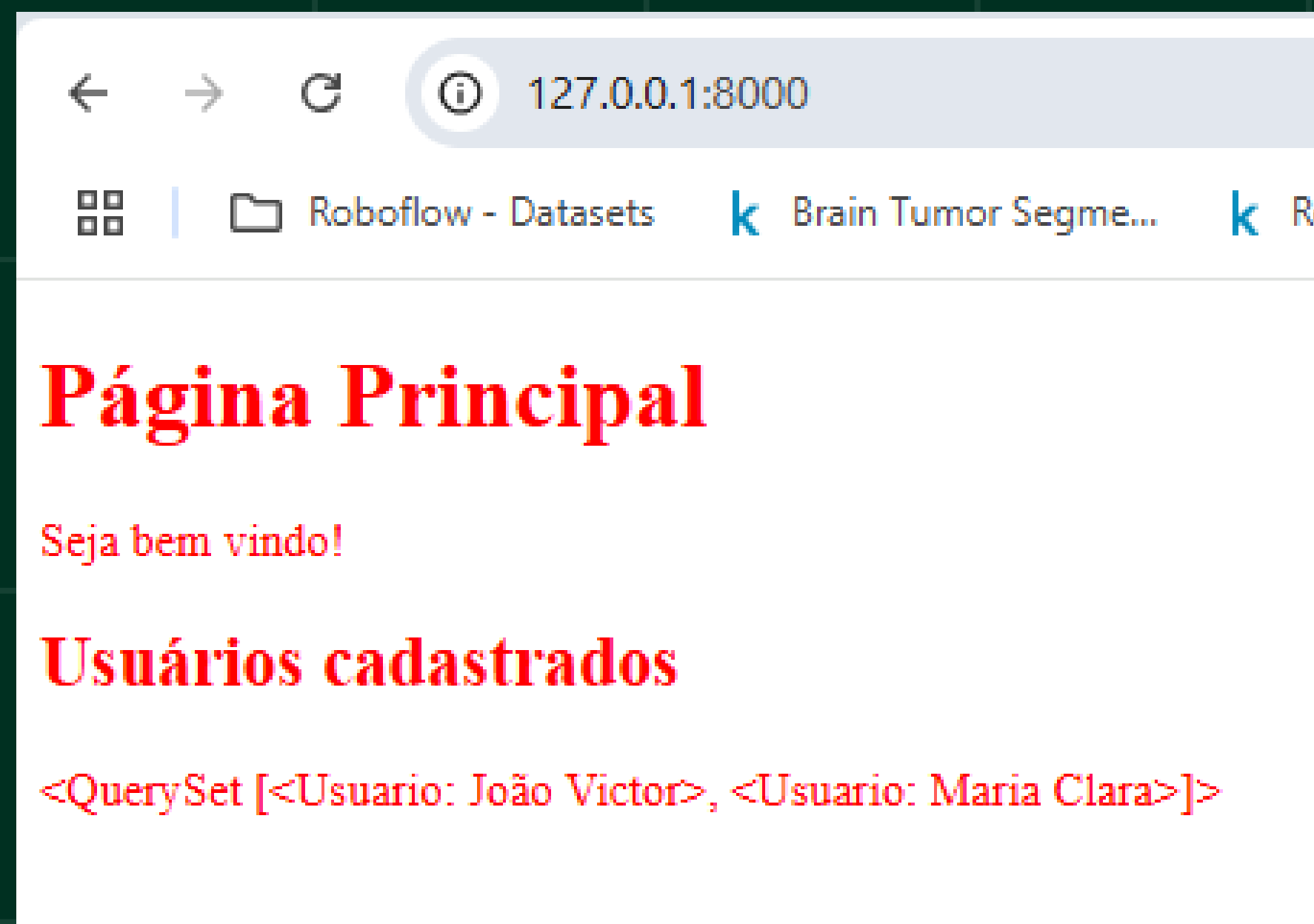
Mostrar esses dados no template é igualmente simples, semelhante a linguagem de template do Jinja2, mas essa é própria do Django.

```
<h1>Página Principal</h1>
<p>Seja bem vindo!</p>

<h2>Usuários cadastrados</h2>
<p>{{ usuarios }}</p>
```

Django - Mostar dados no template

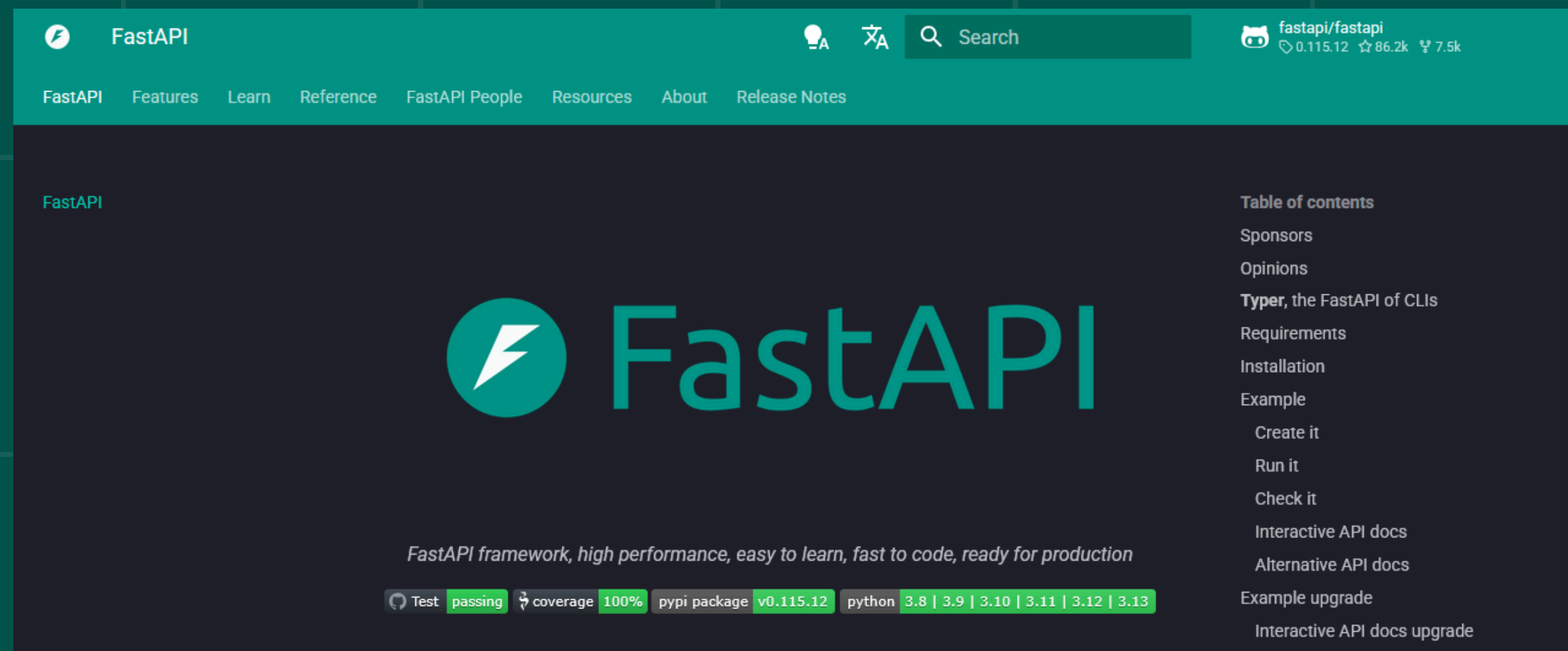
Resultado:



FastAPI - Introdução

FastAPI é um framework web moderno, rápido (de alto desempenho) para criar APIs com base em tipos padrão do Python.

- **Veloz** (feito em cima do Starlette e Pydantic)
- **Rápido de codar** (funções e estruturas intuitivas)
- **Evita bugs** (possui mecanismos de validação)
- **Facilidade de aprender** (menos tempo lendo documentação)



FastAPI - Instalação

Para criação da API



```
pip install fastapi
```

Para rodar em um servidor ASGI local



```
pip install uvicorn
```

FastAPI - Primeira rota (GET)

```
from fastapi import FastAPI

app = FastAPI()

@app.get('/')
def home():
    return 'Nosso primeira rota!'
```

Rodando nossa API em um servidor ASGI local



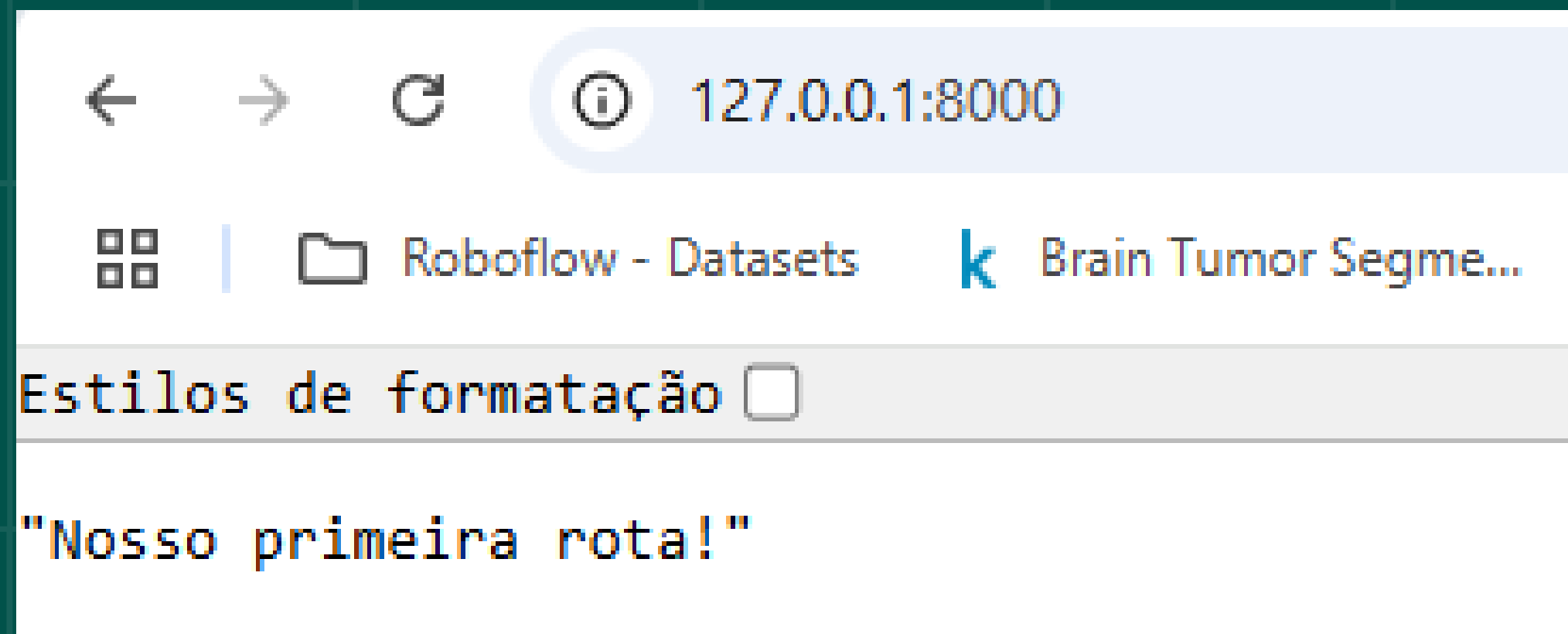
```
uvicorn main:app --reload
```

`main` = nome do nosso arquivo
`app` = nome da nossa API
`--reload` = sempre recarrega o servidor quando detecta alguma mudança no código

FastAPI - Primeira rota (GET)

Resultado:

```
(venv) PS C:\Users\Dell\Desktop\palestra_grupy\fastapi> uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\Dell\\Desktop\\palestra_grupy\\fastapi']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [13464] using StatReload
INFO: Started server process [13836]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:51425 - "GET / HTTP/1.1" 200 OK
```



FastAPI - Rota que devolve dados da aplicação (GET)

Como seria meio trabalhoso configurar um banco de dados do zero só para fazer as demonstrações, montamos um dicionário Python para simular um.

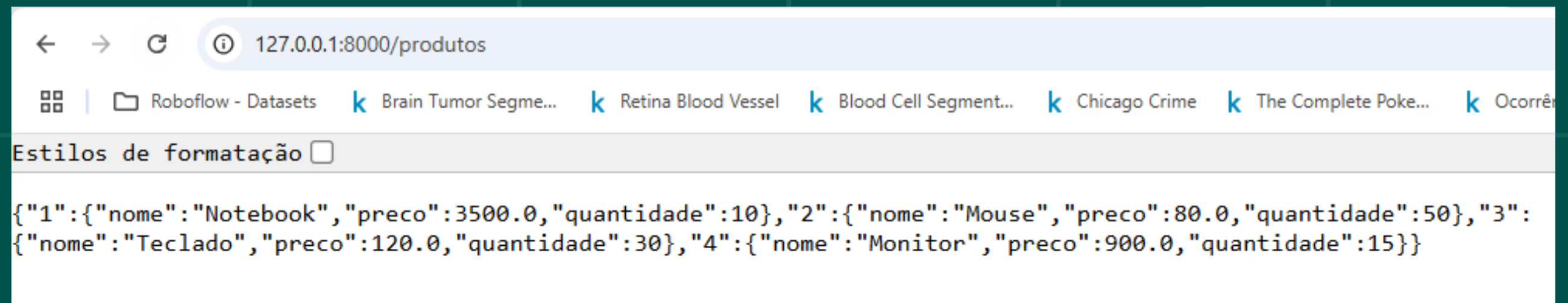
```
from fastapi import FastAPI

app = FastAPI()

produtos = {
    1: {"nome": "Notebook", "preco": 3500.00, "quantidade": 10},
    2: {"nome": "Mouse", "preco": 80.00, "quantidade": 50},
    3: {"nome": "Teclado", "preco": 120.00, "quantidade": 30},
    4: {"nome": "Monitor", "preco": 900.00, "quantidade": 15}
}

@app.get('/')
def home():
    return 'Essa é a rota principal!'

@app.get('/produtos')
def listar_produtos():
    return produtos
```



FastAPI - Rota docs/

O próprio FastAPI já cria automaticamente uma página de documentação das rotas e de testes delas. Basta acessar a URL **/docs**.

FastAPI 0.1.0 OAS 3.1
[/openapi.json](#)

default ^

GET

/produtos Listar Produtos



GET

/ Home



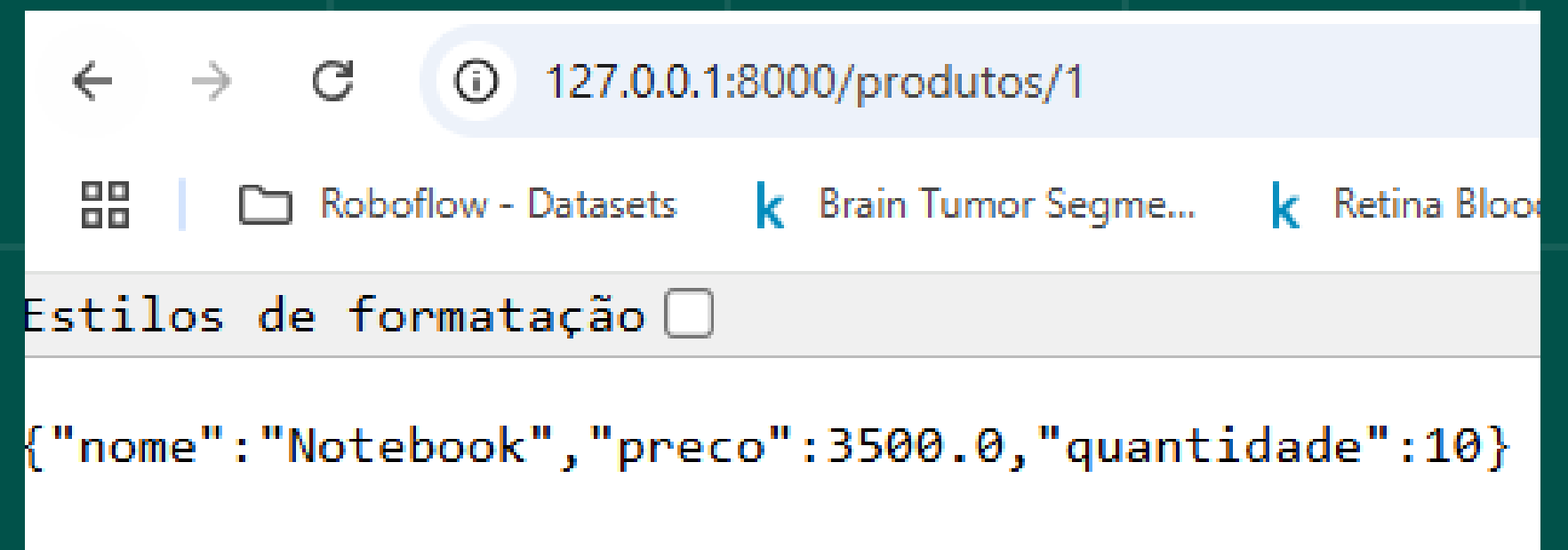
FastAPI - Rota GET que recebe um parâmetro

Criamos a rota `/produtos/{produto_id}` para devolver as informações de um produto específico com base no ID que fornecemos. E deixamos claro para o FastAPI que deve ser um valor inteiro.

```
produtos = {
    1: {"nome": "Notebook", "preco": 3500.00, "quantidade": 10},
    2: {"nome": "Mouse", "preco": 80.00, "quantidade": 50},
    3: {"nome": "Teclado", "preco": 120.00, "quantidade": 30},
    4: {"nome": "Monitor", "preco": 900.00, "quantidade": 15}
}

@app.get('/produtos')
def listar_produtos():
    return produtos

# RECEBE O PARÂMETRO ID PARA DEVOLVER UM PRODUTO ESPECÍFICO
@app.get("/produtos/{produto_id}")
def obter_produto(produto_id: int):
    return produtos.get(produto_id)
```



FastAPI - Rota com parâmetro no docs/

Podemos inclusive testar essa rota que recebe um parâmetro na página docs/ e passar qual parâmetro quisermos.

FastAPI 0.1.0 OAS 3.1
[/openapi.json](#)

default ^

GET /produtos/{produto_id} Obter Produto ^

Parameters

Try it out

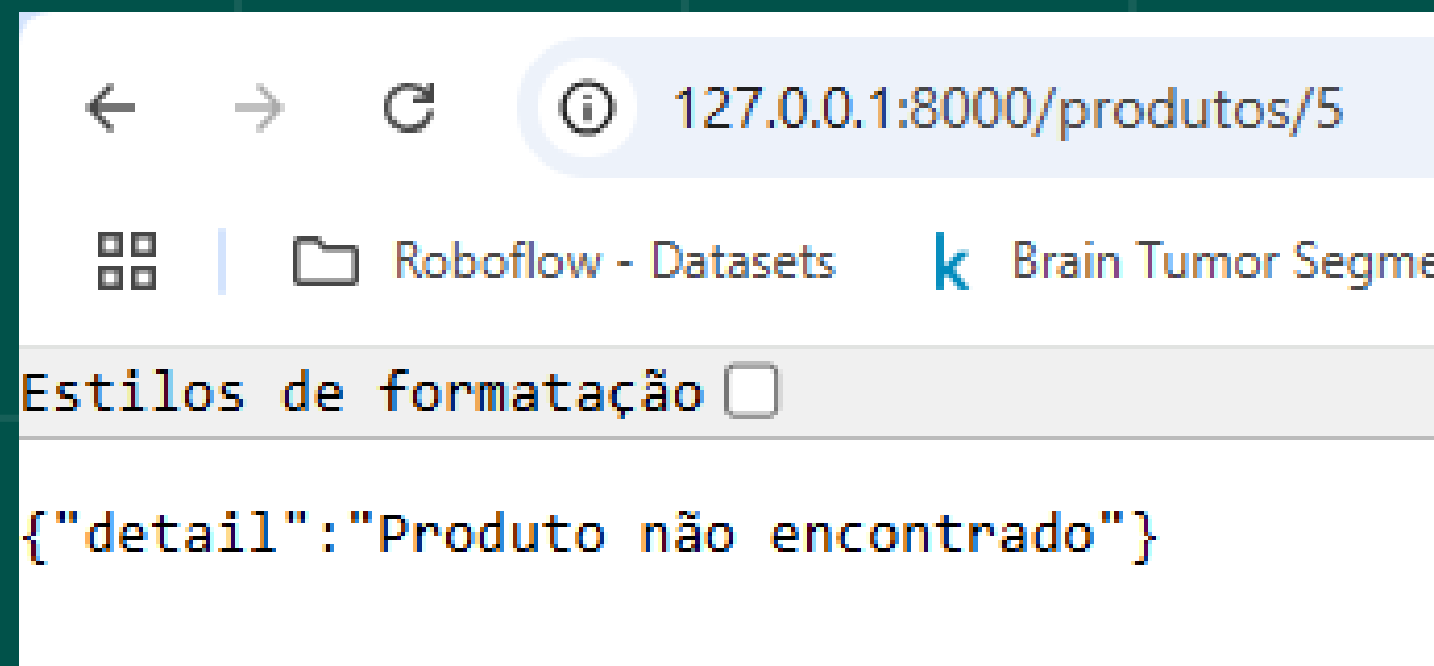
Name	Description
produto_id * required integer (path)	<input type="text" value="produto_id"/>

FastAPI - Exceptions

Se o usuário passar o ID de um produto que não existe, nos podemos criar uma exception para exibir uma mensagem que indique isso. Para não aparecer apenas um "null" para o usuário.

```
from fastapi import HTTPException

@app.get("/produtos/{produto_id}")
def obter_produto(produto_id: int):
    if produto_id not in produtos:
        raise HTTPException(status_code=404, detail="Produto não encontrado")
    return produtos[produto_id]
```



FastAPI - Método POST

Função para adicionar um novo produto através do método POST da API

```
@app.post("/produtos", status_code=201)
def criar_produto(nome: str, preco: float, quantidade: int):

    novo_id = max(produtos.keys()) + 1

    produtos[novo_id] = {
        "nome": nome,
        "preco": preco,
        "quantidade": quantidade,
    }

    return {"msg": "Produto criado", "id": novo_id, "dados": produtos[novo_id]}
```

Podemos testar ele na página /docs

GET

/produtos Listar Produtos

POST

/produtos Criar Produto

FastAPI - Método POST

Resultado dos testes:

POST **/produtos** Criar Produto

Parameters

Name	Description
nome * required string (query)	<input type="text" value="PS5"/>
preco * required number (query)	<input type="text" value="3000.0"/>
quantidade * required integer (query)	<input type="text" value="50"/>

Code

Details

201

Response body

```
{
  "msg": "Produto criado",
  "id": 5,
  "dados": {
    "nome": "PS5",
    "preco": 3000,
    "quantidade": 50
  }
}
```

Response headers

```
content-length: 85
content-type: application/json
date: Fri, 13 Jun 2025 13:55:36 GMT
server: uvicorn
```

```
{
  "1": {
    "nome": "Notebook",
    "preco": 3500.0,
    "quantidade": 10
  },
  "2": {
    "nome": "Mouse",
    "preco": 80.0,
    "quantidade": 50
  },
  "3": {
    "nome": "Teclado",
    "preco": 120.0,
    "quantidade": 30
  },
  "4": {
    "nome": "Monitor",
    "preco": 900.0,
    "quantidade": 15
  },
  "5": {
    "nome": "PS5",
    "preco": 3000.0,
    "quantidade": 50
  }
}
```


FastAPI - Método DELETE

Função para deletar um produto utilizando o método DELETE da API e o ID do produto como parâmetro

```
# ROTA PARA DELETAR UM PRODUTO
@app.delete("/produtos/{produto_id}")
def deletar_produto(produto_id: int):

    if produto_id not in produtos:
        raise HTTPException(status_code=404, detail="Produto não encontrado")

    produto_removido = produtos.pop(produto_id)

    return {
        "msg": "Produto removido com sucesso",
        "id": produto_id,
        "dados": produto_removido
    }
```

POST /produtos Criar Produto

GET /produtos/{produto_id} Obter Produto

DELETE /produtos/{produto_id} Deletar Produto

FastAPI - Método DELETE

Name	Description
produto_id * required integer (path)	<input type="text" value="1"/>
<button>Execute</button>	

Code	Details
200	<div>Response body<pre>{ "msg": "Produto removido com sucesso", "id": 1, "dados": { "nome": "Notebook", "preco": 3500, "quantidade": 10 }}</pre></div> <div>Response headers<pre>content-length: 104content-type: application/jsondate: Fri, 13 Jun 2025 14:05:43 GMTserver: uvicorn</pre></div>

```
{"2":{"nome":"Mouse","preco":80.0,"quantidade":50},"3":{"nome":"Teclado","preco":120.0,"quantidade":30},"4":{"nome":"Monitor","preco":900.0,"quantidade":15}}
```

FastAPI - Método PUT

Para finalizar o CRUD, vamos criar uma função que utiliza o método PUT da API para atualizar algum informação do nosso dicionário.

```
# ROTA PARA ATUALIZAR INFORMAÇÕES DE UM PRODUTO
@app.put("/produtos/{produto_id}")
def atualizar_produto(produto_id: int, nome: str, preco: float, quantidade: int):

    if produto_id not in produtos:
        raise HTTPException(status_code=404, detail="Produto não encontrado")

    produtos[produto_id] = {
        "nome": nome,
        "preco": preco,
        "quantidade": quantidade
    }

    return {
        "msg": "Produto atualizado com sucesso",
        "id": produto_id,
        "dados": produtos[produto_id]
    }
```

GET

/produtos/{produto_id} Obter Produto

DELETE

/produtos/{produto_id} Deletar Produto

PUT

/produtos/{produto_id} Atualizar Produto

FastAPI - Método PUT

Aqui estou atualizando as informações do produto de ID=1 "Notebook", mais especificadamente seu preço e sua quantidade.

Parameters	
Name	Description
produto_id * required integer (path)	<input type="text" value="1"/>
nome * required string (query)	<input type="text" value="Notebook"/>
preco * required number (query)	<input type="text" value="10000"/>
quantidade * required integer (query)	<input type="text" value="100"/>

```
{  
  "msg": "Produto atualizado com sucesso",  
  "id": 1,  
  "dados": {  
    "nome": "Notebook",  
    "preco": 10000,  
    "quantidade": 100  
  }  
}
```

← → ↻ ⓘ

127.0.0.1:8000/produtos/1

⌵

Roboflow - Datasets

k Brain Tumor Segme...

k Retina Blood Ves

Estilos de formatação ☐

```
{"nome": "Notebook", "preco": 10000.0, "quantidade": 100}
```



Simples, minimalista,
flexível e muito
personalizável

Curva de aprendizado baixa

Por se tratar de um
microframework, muita da
implementação fica a
critério do desenvolvedor

Microserviços, APIs
pequenas, projetos simples



Muitas ferramentas
embutidas que ajudam a
criar rapidamente uma
aplicação robusta

Curva de aprendizado média

Segue a filosofia MTV
(Model-Template-View)

Aplicações web completas,
projetos grandes, projetos
focados em produtividade
e rapidez



Projetado para ser o mais
rápido possível no
desenvolvimento de APIs

Curva de aprendizado
baixa, mas necessário
saber algumas convenções

Documentação automática das
rotas e validação
automática de dados

APIs rápidas e assíncronas
e microsserviços de alto
desempenho



**Obrigado pela
atenção!**



Dúvidas?