

Construindo e consumindo uma API com Python

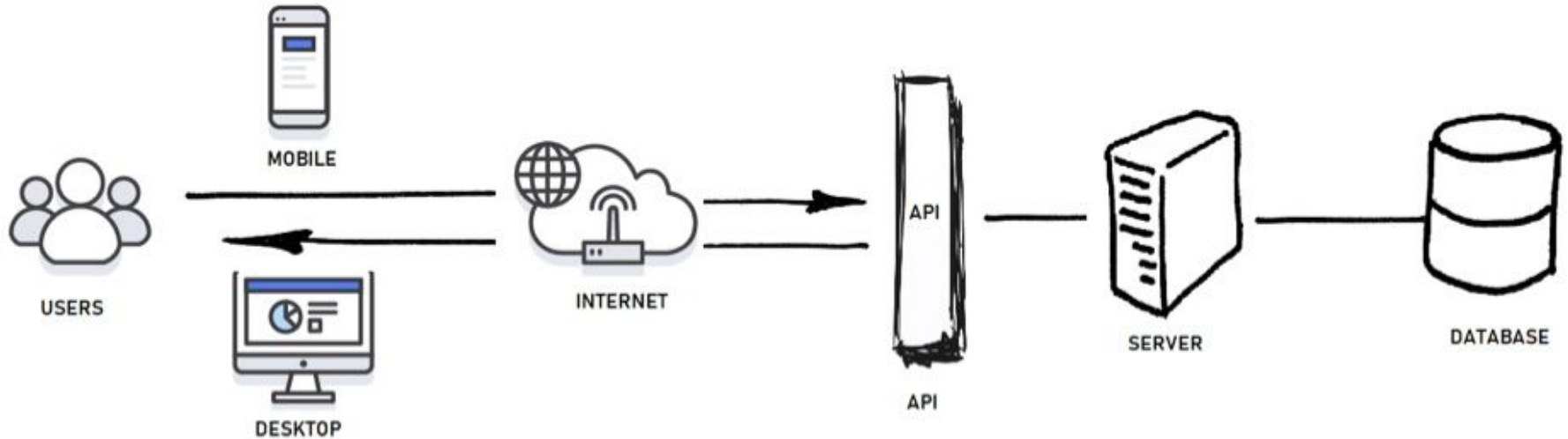
Prof. Ms. Luiz Niero

Agenda

- Fundamentação Teórica
- Prática
 - Instalação do Python Flask
 - Criação de página Web simples
 - Criação de Apis
 - Integração da página web com a Api

Fundamentação Teórica

Arquitetura de um sistema Web



Fonte: <https://hackernoon.com/the-basics-of-designing-an-api-architecture>

Definição

API: Application Programming Interface (Interface de Programação de Aplicativos)

- Conjunto de regras e protocolos que permitem que diferentes softwares se comuniquem entre si
- Define os métodos e formatos de dados que podem ser solicitados
- Largamente utilizadas na web

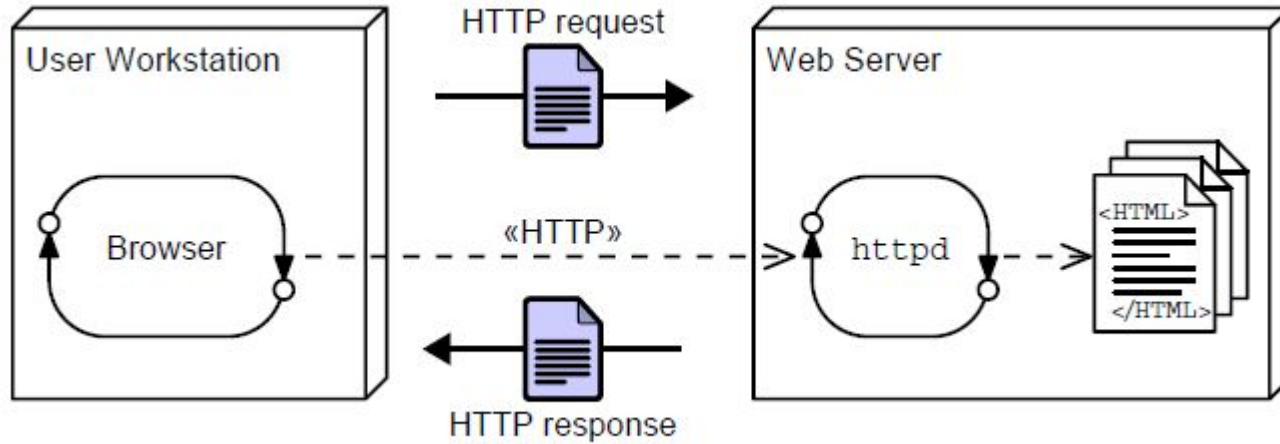
Exemplos:

Apis do Facebook, Whatsapp, Pagseguro, Paypal, Google Maps, Uber, AWS, OpenAI

Mensagens HTTP

- A web funciona através da troca de mensagens HTTP (ou HTTPS)
- Uma API recebe uma mensagem com tipo, conteúdo, parâmetros e retorna outra mensagem.
- Tipos de Mensagem: GET, POST, PUT, DELETE, entre outros.

Na Web, tudo é uma mensagem HTTP/HTTPS



curl.semac.cc

Por que utilizar uma API?

- Para sua própria arquitetura
 - Maior segurança
 - Multiplataforma
 - Arquitetura de microserviços
- Para integrar com terceiros
 - Utilizar soluções já construídas
 - Serviços: AWS, chatGPT, Google Maps
 - Produtos: MercadoLivre, Shopfy

Ferramentas

- Linguagem de Programação: Python v3.11
- Framework: Flask (micro-framework)
- IDE: VsCode, Pycharm, outro.

Livro Flask: https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf

Site oficial: <https://flask.palletsprojects.com/en/2.3.x/>

Prática

Instalando o ambiente

- Crie um diretório onde deseja criar o projeto. Ex: curso_api
- Crie um ambiente virtual python neste diretório.

```
python -m venv api-venv
```

- Inicie o ambiente virtual:

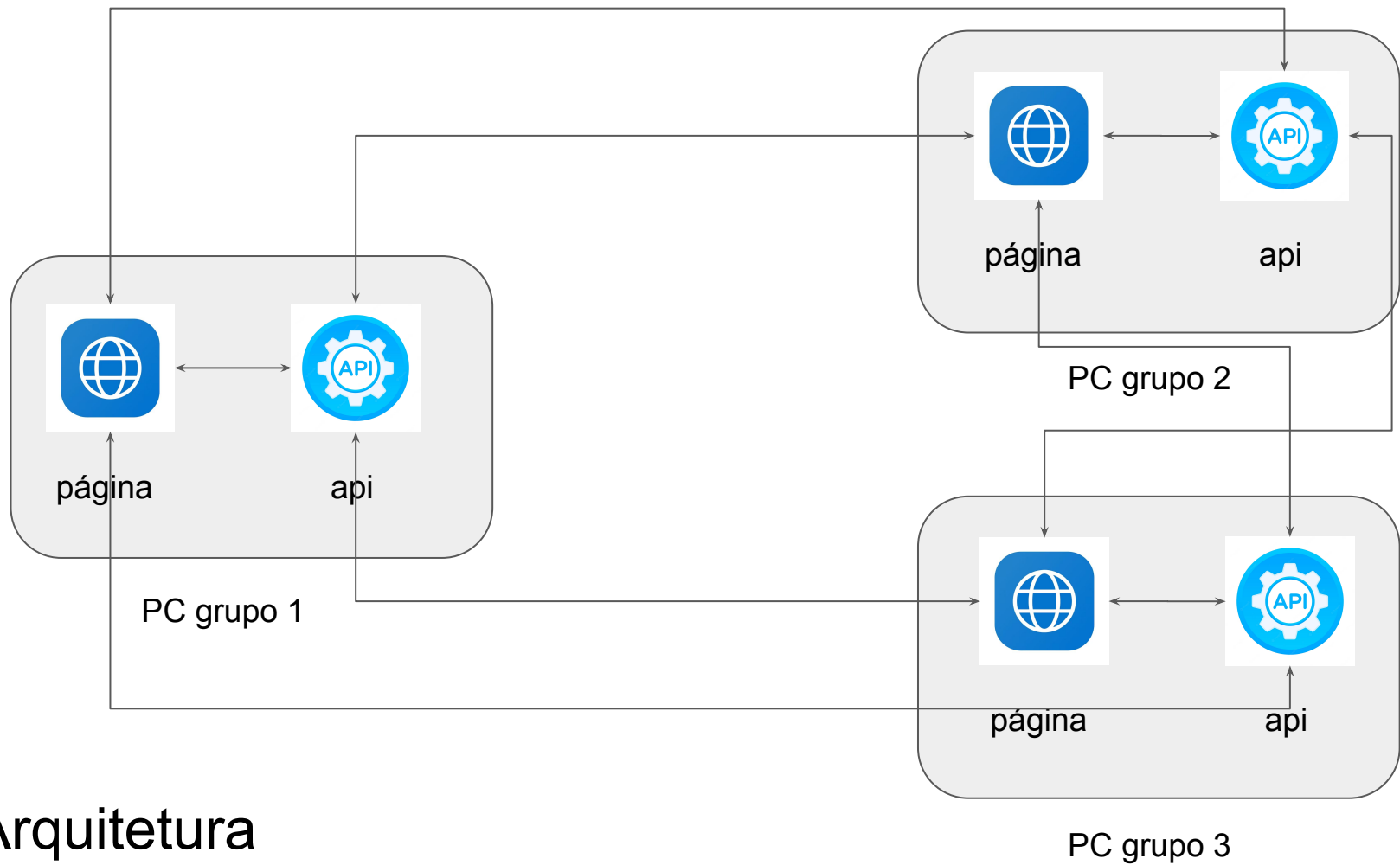
```
./api-venv/scripts/activate
```

- Instale o Flask

```
pip install flask
```

O que nós criaremos?

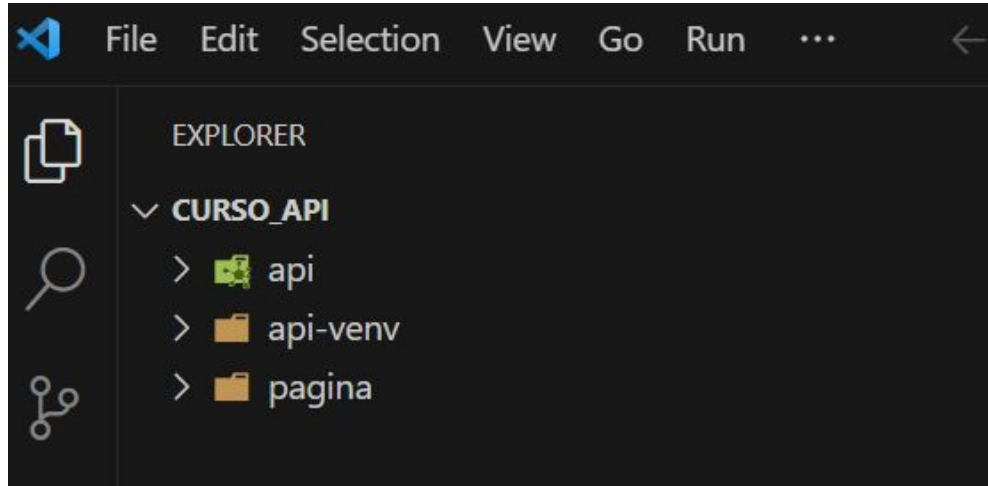
- Uma página WEB simples com 3 listas:
 - Lugares que quero conhecer
 - Livros que quero ler
 - Disciplinas que odiei fazer
- Um endpoint de API com alguma dessas listas
- Conectar nossa página a:
 - Nossa própria API
 - API de mais 2 colegas



Passo 1: Criando a página da Web

Crie 2 novos diretórios:

- pagina (sem acento)
- api



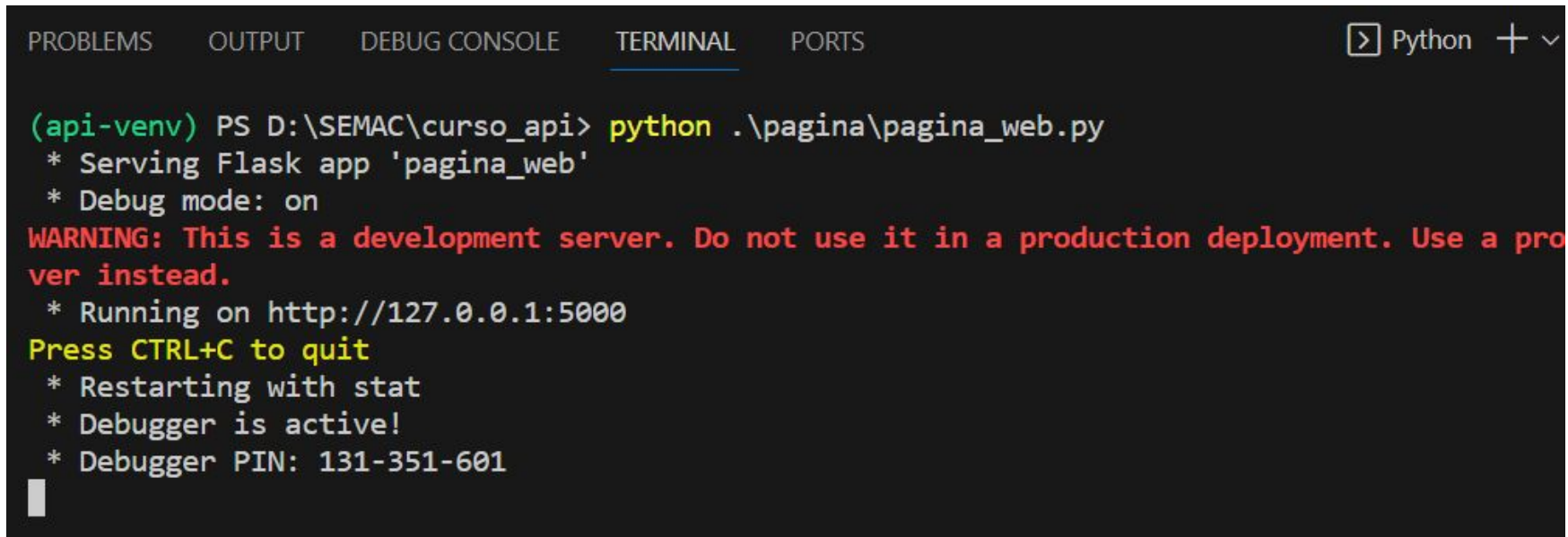
Passo 1: Criando a página da Web

- Na pasta 'pagina', crie um arquivo 'pagina_web.py' e insira o conteúdo:

```
pagina_web.py X
pagina > pagina_web.py
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6
7  def index():
8      return '<h1>Hello Semac!</h1>'
9
10
11 if __name__ == '__main__':
12     app.run(debug=True)
13
```

Passo 2: Executando o servidor web

- Abra um terminal no diretório 'pagina'
- Garanta que o ambiente virtual api-venv está selecionado
- Execute o .py gerado



The screenshot shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. On the right side of the terminal, there is a dropdown menu showing 'Python' with a plus and minus icon. The terminal content shows the command `python .\pagina\pagina_web.py` being executed in the `(api-venv)` environment. The output includes messages about serving the Flask app, debug mode being on, a warning about using a development server, the server running on `http://127.0.0.1:5000`, and instructions to press `CTRL+C` to quit. It also shows the server restarting with `stat`, the debugger being active, and a debugger PIN of `131-351-601`.

```
(api-venv) PS D:\SEMAC\curso_api> python .\pagina\pagina_web.py
* Serving Flask app 'pagina_web'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a pro
ver instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 131-351-601
```


Passo 3: Criando um HTML básico

```
@app.route('/')

def index():
    return '''
        <title> Mini curso de API com Python Flask </title>
        <body>
        <h1> Hello, SEMAC 2023 </h1>
        <ul>
            <li><a href=/lugares> Lugares</a></li>
            <li><a href=/livros> Livros</a></li>
            <li><a href=/disciplinas> Disciplinas</a></li>
        </ul>
        <br/>
        <a href=/> Voltar </a>
        </body>
    '''
```

Passo 4: Criando as páginas Lugares, Livros e Disciplinas

```
@app.route('/lugares')
def lugares():
    return '''
        <title> Mini curso de API com Python Flask </title>
        <body>
        <h1> Lugares </h1>
        <ul>
            <li>Lugar 1</li>
            <li>Lugar 2</li>
            <li>Lugar 3</li>
        </ul>
        <br/>
        <a href='/'> Voltar </a>
        </body>
    '''
```

Passo 5:

- No diretório 'pagina', criem o arquivo 'dados.py'.
- Em 'dados.py', Vamos criar métodos que irão retornar 3 Lugares, 3 Livros e 3 disciplinas quaisquer. Vamos chamar esses métodos para popular as nossas páginas.

pagina > dados.py > ...

```
1 class Dados:
2     def __init__(self) -> None:
3         self.Disciplinas = ['Álgebra Linear', 'Cálculo 3', 'Física']
4         self.Lugares = ['Chile', 'Marrocos', 'Tailândia']
5         self.Livros = ['A Cabana', '1984', 'Agua para elefantes']
6
7     def retorna_disciplinas(self):
8         return self.Disciplinas
9
10    def retorna_lugares(self):
11        return self.Lugares
12
13    def retorna_livros(self):
14        return self.Livros
15
16
17 if __name__ == '__main__':
18     dados = Dados()
19     disciplinas = dados.retorna_disciplinas()
20     print(disciplinas)
21
22     lugares = dados.retorna_lugares()
23     print(lugares)
24
25     livros = dados.retorna_livros()
26     print(livros)
27
```

Passo 6:

- No nosso pagina_web.py, em vez da lista ser estática, vamos chamá-la de dados.py. Para isso, importe a classe dados.
- Vamos substituir 'Lugar 1, Lugar 2, etc' pelos elementos retornados pelas listas.

```
pagina_web.py X dados.py
pagina > pagina_web.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4  import dados
5  endpoint = dados.Dados()
6
```

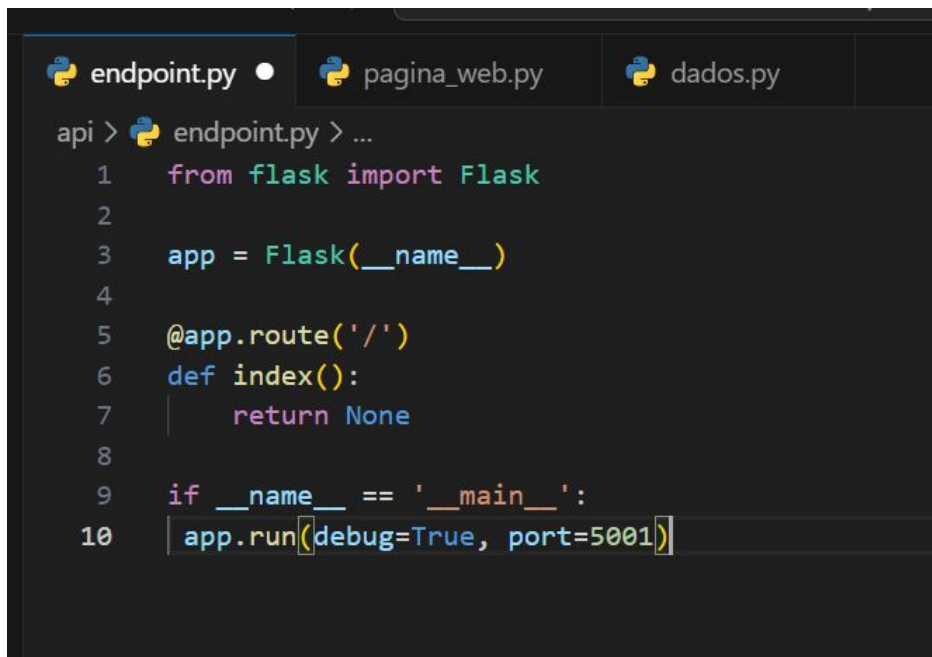
```
@app.route('/lugares')
def lugares():
    lista_lugares = endpoint.retorna_lugares()
    return f'''
        <title> Mini curso de API com Python Flask </title>
        <body>
        <h1> Lugares </h1>
        <ul>
            <li>{lista_lugares[0]}</li>
            <li>{lista_lugares[1]}</li>
            <li>{lista_lugares[2]}</li>
        </ul>
        <br/>
        <a href='/'> Voltar </a>
        </body>
    '''
```

O que nós criaremos?

- Uma página WEB simples com 3 listas:
 - Lugares que quero conhecer
 - Livros que quero ler
 - Disciplinas que odiei fazer
- Um endpoint de API com alguma dessas listas
- Conectar nossa página a:
 - Nossa própria API
 - API de mais 2 colegas

Passo 7

No diretório 'api', que criamos no início, crie o arquivo 'endpoint.py', com a estrutura básica do Flask.



```
api > endpoint.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return None
8
9  if __name__ == '__main__':
10     app.run(debug=True, port=5001)
```

Passo 8

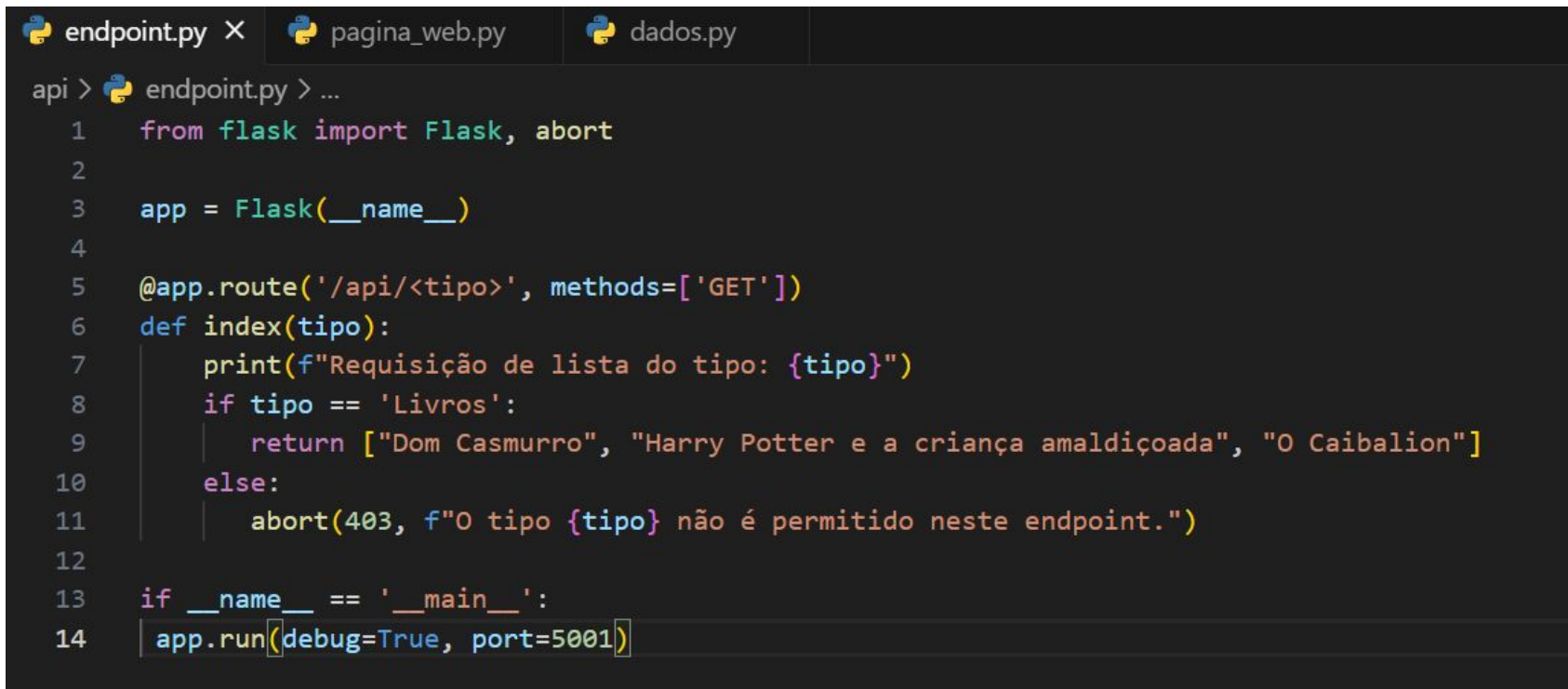
Vamos receber o Tipo de lista desejada pela requisição na própria URL e explicitar que esse método recebe apenas requisições do tipo GET.

```
@app.route('/api/<tipo>', methods=['GET'])
def index(tipo):
    print(f"Requisição de lista do tipo: {tipo}")
    if tipo == 'Livros':
        return ["Dom Casmurro", "Harry Potter e a criança amaldiçoada", "O Caibalion"]
    else:
        return []
```

Inicie o Flask no endpoint.py e utilize o CURL neste endereço.

Passo 9

Fiquemos mais seletivos. Se a requisição não for do tipo que desejamos (escolham uma das três), então retornaremos um erro!

A screenshot of a code editor with three tabs: 'endpoint.py', 'pagina_web.py', and 'dados.py'. The 'endpoint.py' tab is active, showing Python code for a Flask application. The code defines a route for '/api/<tipo>' with the 'GET' method. The 'index' function prints a message and returns a list of book titles if the type is 'Livros', otherwise it aborts with a 403 status and an error message. The code is as follows:

```
api > endpoint.py > ...
1  from flask import Flask, abort
2
3  app = Flask(__name__)
4
5  @app.route('/api/<tipo>', methods=['GET'])
6  def index(tipo):
7      print(f"Requisição de lista do tipo: {tipo}")
8      if tipo == 'Livros':
9          return ["Dom Casmurro", "Harry Potter e a criança amaldiçoada", "O Caibalion"]
10     else:
11         abort(403, f"O tipo {tipo} não é permitido neste endpoint.")
12
13 if __name__ == '__main__':
14     app.run(debug=True, port=5001)
```

O que nós criaremos?

- Uma página WEB simples com 3 listas:
 - Lugares que quero conhecer
 - Livros que quero ler
 - Disciplinas que odiei fazer
- Um endpoint de API com alguma dessas listas
- Conectar nossa página a:
 - Nossa própria API
 - API de mais 2 colegas

Passo 10

Em dados.py, vamos alterar o nosso método para consultar uma API.

- Biblioteca 'requests' (**pip install requests**)
- método = requests.get(url)

```
# UTILIZA API
def obtem_dados_api(self, url):
    try:
        resposta = requests.get(url)
        if resposta.status_code == 200:
            return resposta.json()
    except Exception as e:
        print(f"Ocorreu um erro ao tentar se conectar á API URL {url}")
        print(f"Erro {e}")
    return ['', '', '']
```

Passo 11

Em dados.py, vamos tentar consultar nossa própria API.

Ps: Lembre-se de garantir que o endpoint.py está em execução!

```
self.url_disciplinas = 'http://localhost:5001/api/disciplinas'  
self.url_lugares = 'http://localhost:5001/api/lugares'  
self.url_livros = 'http://localhost:5001/api/livros'
```

```
if __name__ == '__main__':  
    dados = Dados()  
    livros_api = dados.obtem_dados_api(dados.url_livros)  
    print(livros_api)
```

```
36
37 if __name__ == '__main__':
38     dados = Dados()
39     livros_api = dados.obtem_dados_api(dados.url_livros)
40     print(livros_api)
41
42     lugares_api = dados.obtem_dados_api(dados.url_lugares)
43     print(lugares_api)
44
45     disciplinas_api = dados.obtem_dados_api(dados.url_disciplinas)
46     print(disciplinas_api)
47
48     #disciplinas = dados.retorna_disciplinas()
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
● (api-venv) PS D:\SEMAC\curso_api> python .\pagina\dados.py
['Dom Casmurro', 'Harry Potter e a criança amaldiçoada', 'O Caibalion']
['', '', '']
['', '', '']
○ (api-venv) PS D:\SEMAC\curso_api> █
```

Passo 12

Ps: Lembre-se de garantir que o endpoint.py está em execução!

```
@app.route('/livros')
def livros():
    #lista_livros = endpoint.retorna_livros()
    lista_livros = endpoint.obtem_dados_api(endpoint.url_livros)
    return f'''
        <title> Mini curso de API com Python Flask </title>
        <body>
        <h1> Livros </h1>
        <ul>
            <li>{lista_livros[0]}</li>
            <li>{lista_livros[1]}</li>
            <li>{lista_livros[2]}</li>
        </ul>
        <br/>
        <a href='/'> Voltar </a>
        </body>
    '''
```

Passo 13

Suba novamente pagina.web.py e verifique as páginas.

Mini curso de API com Pyth

← → ↻ 🏠 ⓘ localhost

Lugares

⋮

[Voltar](#)

Mini curso de API com Pyth

← → ↻ 🏠 ⓘ localhost

Disciplinas

⋮

[Voltar](#)

Mini curso de API com Pyth

Flask Web

← → ↻ 🏠 ⓘ localhost:5000/livros

Livros

- Dom Casmurro
- Harry Potter e a criança amaldiçoada
- O Caibalion

[Voltar](#)

O que nós criaremos?

- Uma página WEB simples com 3 listas:
 - Lugares que quero conhecer
 - Livros que quero ler
 - Disciplinas que odiei fazer
- Um endpoint de API com alguma dessas listas
- Conectar nossa página a:
 - Nossa própria API
 - API de mais 2 colegas

Passo 14

- Compartilhem entre vocês os IPs de quem criou APIs para:
 - Livros
 - Disciplinas
 - Lugares
- Alterem o endereço de vocês pelo de um grupo que tenha feito APIs que vocês não fizeram em dados.py:

```
self.url_disciplinas = '<URL_DA_API_DO_COLEGA>/api/disciplinas'  
self.url_lugares = '<URL_DA_API_DE_OUTRO_COLEGA>/api/lugares'  
self.url_livros = 'http://localhost:5001/api/livros'
```

Passo 15

- Deixe sua API disponível

- No endpoint.py, adicione seu IP ao app.run:

```
13  if __name__ == '__main__':  
14      app.run(debug=True, host='0.0.0.0', port=5001)
```

- Se houver algum Firewall, insira uma regra de exclusão.
 - Se o firewall for da rede, não conseguiremos nos conectar
 - Se o usuário do windows não for admin, também não conseguiremos.
 - Segurança do Windows
 - Firewall e Proteção de Rede
 - Configurações Avançadas
 - Liberar porta de entrada: 5001

Considerações Finais

- Nem sempre as informações da mensagem estarão na URL
- Geralmente as APIs estão protegidas por Autenticação
 - Métodos de autenticação registram cookies no navegador
 - Autenticação funciona em uma camada de 'middleware'
- Existem outros frameworks tão bom ou melhores que o Flask
 - Python (Django)
 - .NET
 - Node.js
- Uma página HTTPS não deve se conectar à uma API HTTP
 - Conteúdo Misto
 - Alguns frameworks bloqueiam por padrão
 - “Engana” o usuário final

Obrigado!