

Introdução ao Uso de API

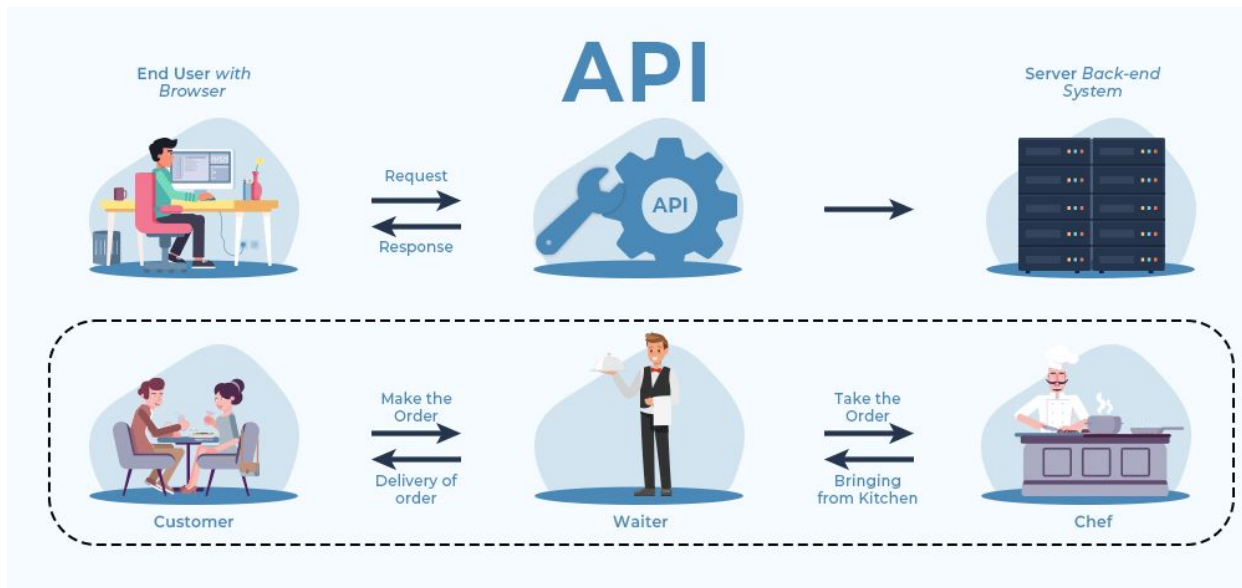
Profº Msc. Jeffson Celeiro Sousa
Doutorando em Ciência da Computação - UFPa

Roteiro da Aula

- Conceitos de API
- Conceitos de REST
- Verbos
- Respostas
- Filtros

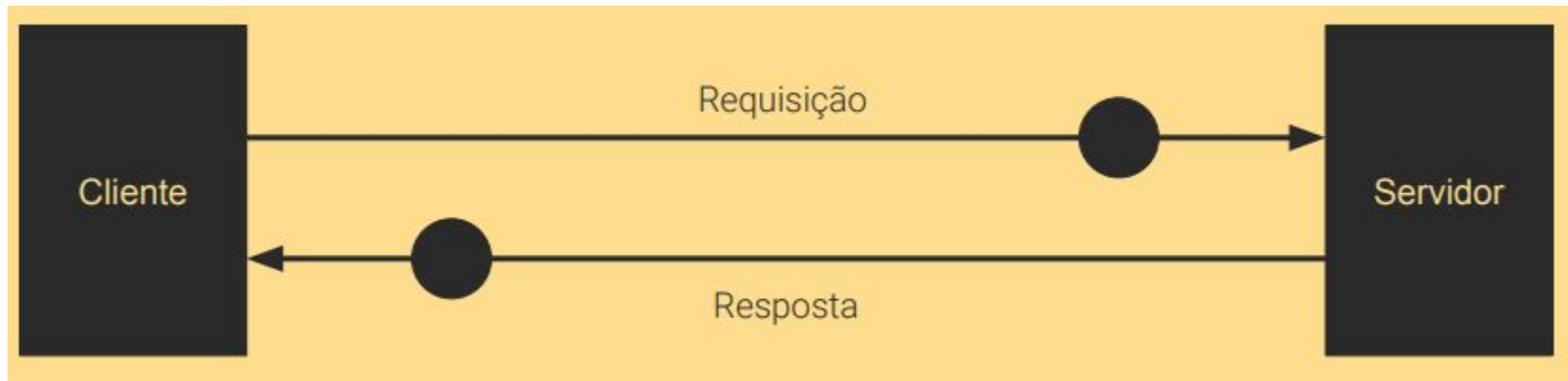
O que é uma API?

- Uma API (Application Programming Interface) é um conjunto de definições e protocolos usados no desenvolvimento e na integração de aplicações



O que é uma API?

- Uma API é um conjunto de definições e protocolos usados no desenvolvimento e na integração de aplicações.

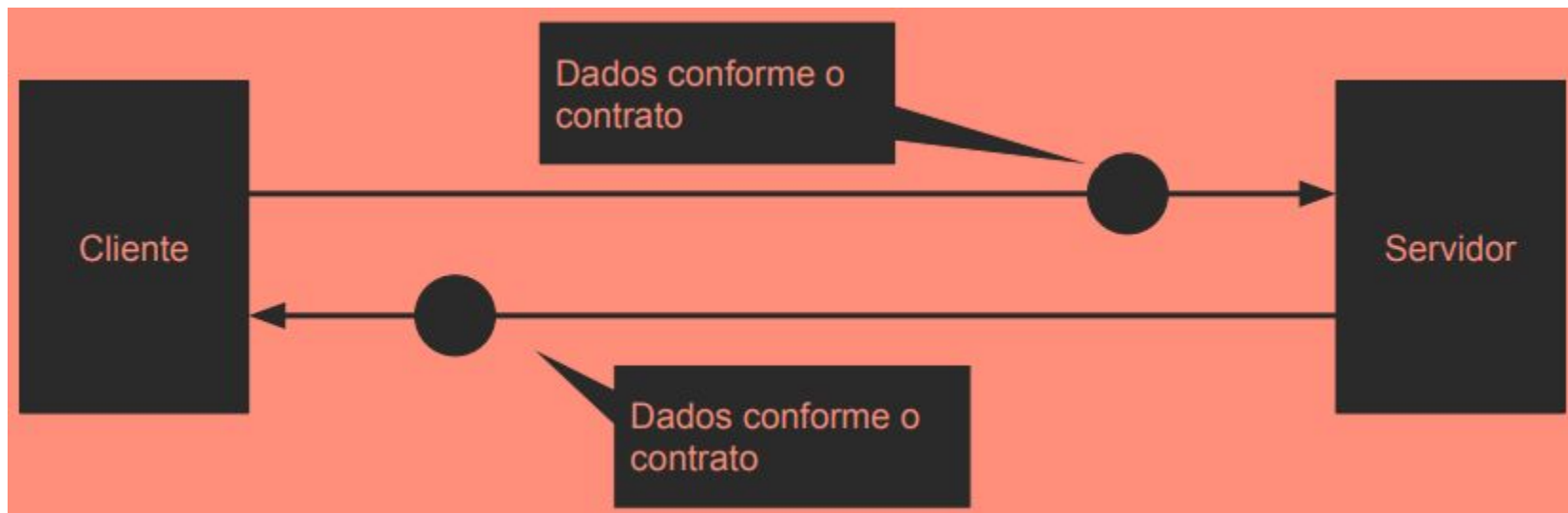


Um contrato

- Basicamente uma api é **contrato** de como os dados serão enviados e quais respostas vamos obter quando um dado específico for solicitado.

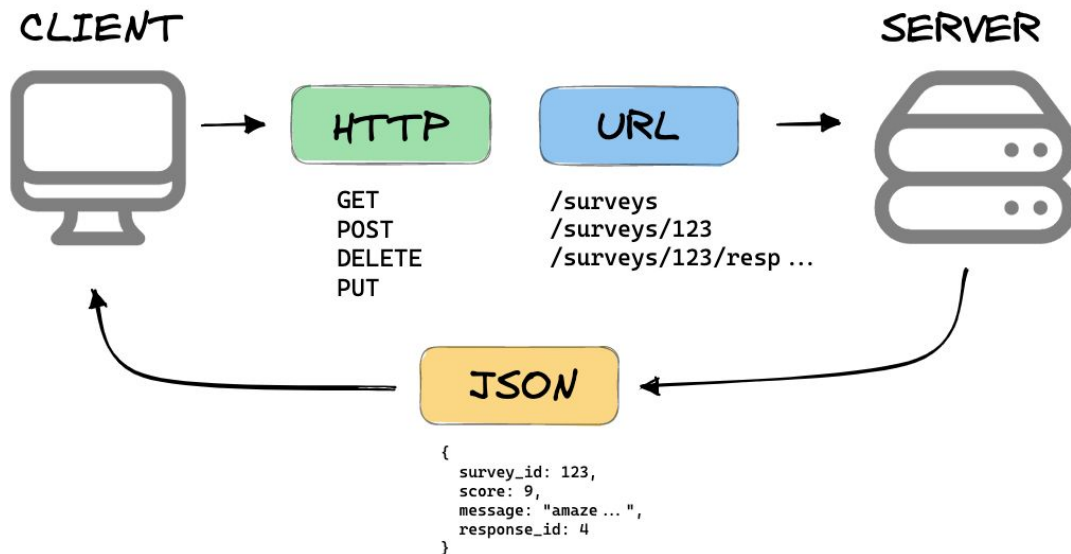
Um contrato

- Basicamente uma api é **contrato** de como os dados serão enviados e quais respostas vamos obter quando um dado específico for solicitado.



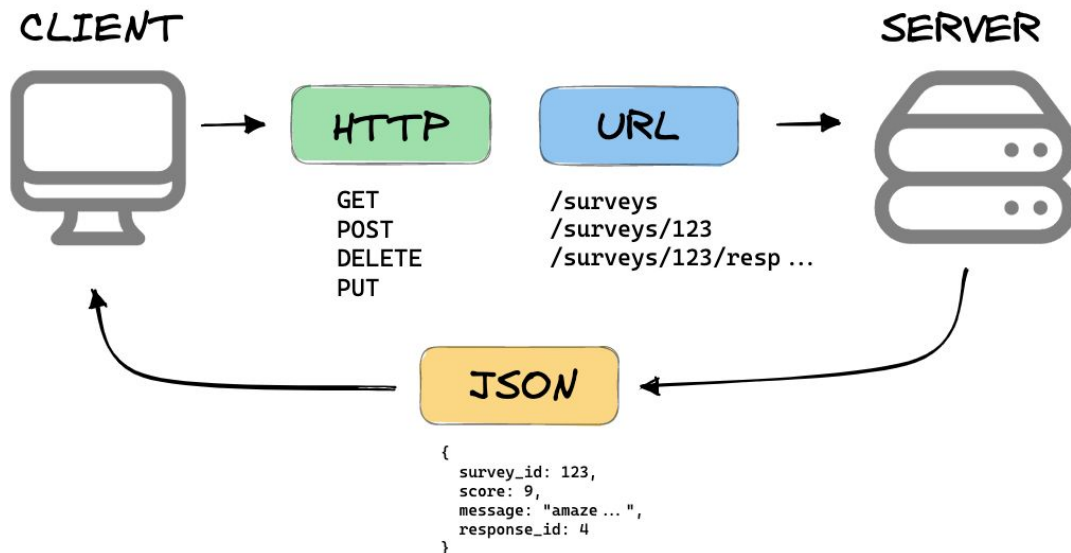
REST

- **REST** (Representational State Transfer) é um conjunto de "regras" e limitações para que a API não seja complexa de mais.

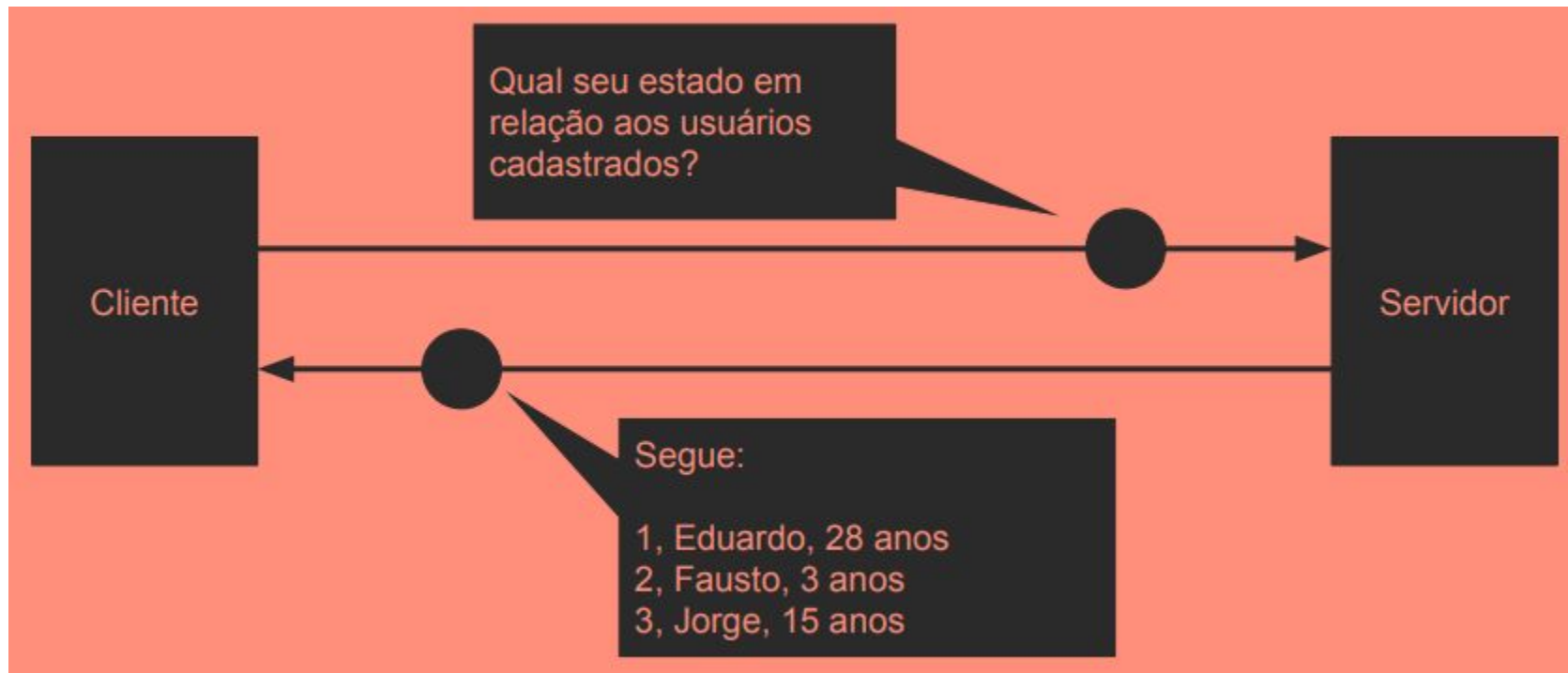


REST

- Basicamente o que pedimos a uma API é o seu **estado** [State] e ela nos retorna uma **Representação** [Representational] dele em uma **Transferência** [Transfer] de dados.



REST



Que ferramentas podemos usar em Python?



Flask



Falcon



CherryPy



AIOHTTP



Tornado

Que ferramentas podemos usar em Python?



Flask



Falcon



CherryPy



AIOHTTP



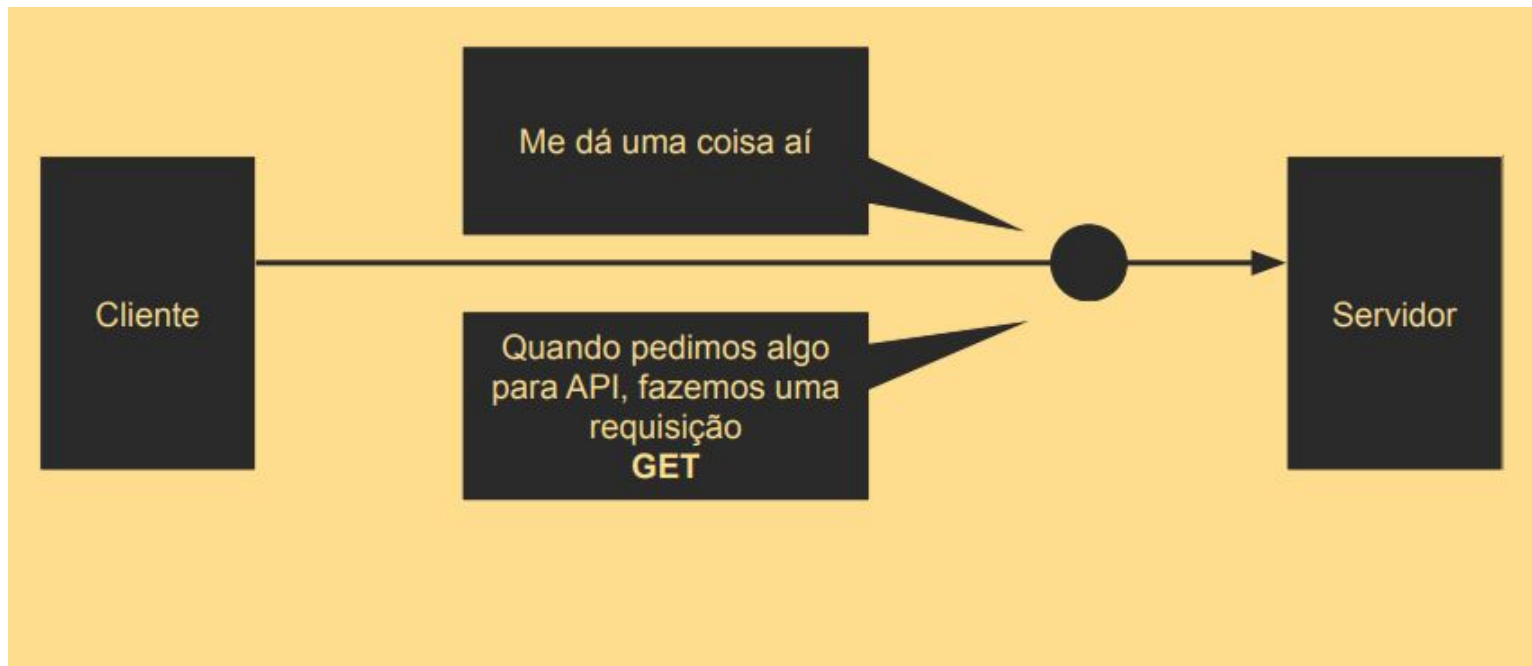
Tornado

Vamos instalar o Flask

- `pip install flask`

Verbos

- Como dizer o que eu quero?



Nosso primeiro exemplo

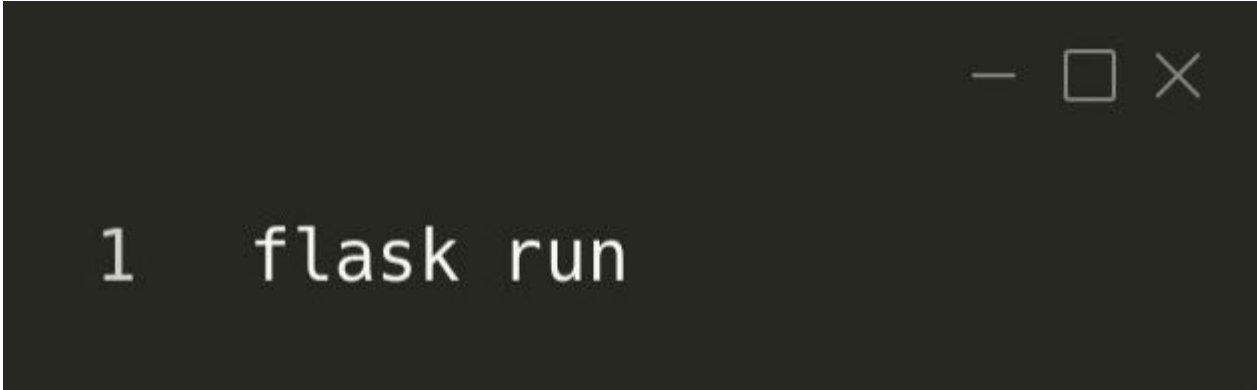
```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.get('/recurso')
6  def respondo_estado():
7      return 'OK'
8
9  app.run()
```

Nosso primeiro exemplo

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.get('/recurso')
6  def respondo_estado():
7      return 'OK'
8
9  app.run( )
```

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.get('/recurso')
6  def respondo_estado():
7      return 'OK'
8
9  app.run( )
```

Tá, OK! Mas como rodar esse código?



```
1 flask run
```

A dark-themed terminal window with a title bar containing standard window controls (minimize, maximize, close). The text '1 flask run' is displayed in a monospaced font.

Instalando um cliente

- `pip install httpie`

Como conversar com ele?

- Nosso servidor, tem um endereço.
- **(http://localhost)**
- Tem uma porta (5000)
- E precisamos acessar um recurso ou como costumamos chamar, endpoint

http://localhost/recurso

0 comando com httpie

A dark-themed terminal window with standard window controls (minimize, maximize, close) in the top right corner. The text 'http get http://localhost:5000' is displayed in a monospaced font, with 'get' highlighted in cyan.

```
http get http://localhost:5000
```

Contrato

- Lembra que falamos antes sobre contrato? Existe um formato padrão para especificação que é o **OpenAPI**.
- Que é como dizemos o que deve ser enviado API e o que será respondido quando um pedido for feito
 - `pip install flask-pydantic-spec`

Contrato

- `from flask import Flask, jsonify`
- **Importa o Flask**, o microframework usado para criar a aplicação web.
- `jsonify` é uma função que **transforma dicionários Python em respostas JSON válidas**, muito útil em APIs REST.

Contrato

- `from flask_pydantic_spec import FlaskPydanticSpec, Response`
- Importa a biblioteca **Flask Pydantic Spec**, que serve para gerar automaticamente documentação da API (Swagger/OpenAPI) com base em classes Pydantic.
- **FlaskPydanticSpec** é a classe principal para configurar a documentação.
- `Response` é usada para especificar o formato e tipo da resposta (saída da API), com base em modelos Pydantic.

Contrato

- `from pydantic import BaseModel`
- Importa o **BaseModel**, que é a classe base do Pydantic usada para definir modelos de dados validados.
- Usamos esses modelos para validar tanto requisições de entrada (input) quanto respostas da API (output).

Contrato

- `app = Flask(__name__)`
- Cria a aplicação Flask. A variável `app` será usada para registrar as rotas da API.
- `__name__` é uma variável especial do Python, e nesse contexto serve para o Flask saber onde está o ponto de entrada da aplicação.

Contrato

- `spec = FlaskPydanticSpec('flask', title='Impulsona Wyden')`
- Inicializa o objeto responsável por gerar a **documentação da API**.
- "flask" é o modo de operação.
- `title='Impulsona Wyden'` define o título da documentação Swagger que será gerada automaticamente.

Contrato

- `spec.register(app)`
- Conecta o **spec** à aplicação Flask.
- A partir daqui, as rotas da aplicação que forem decoradas com **@spec.validate()** serão incluídas na documentação.

Contrato

- `class Pessoa(BaseModel):`
- `nome: str`
- Cria um modelo de resposta usando Pydantic.
- Aqui estamos dizendo que a resposta da API `/pessoas` será um JSON com um campo chamado `nome` que deve ser uma string.
 - Exemplo válido: `{ "nome": "Jeffson Sousa" }`

Contrato

- `@app.get('/pessoas')`
- Define a rota da API /pessoas que responde a requisições do tipo GET.
- O Flask executará a função logo abaixo quando o endpoint for acessado.

Contrato

- `@spec.validate(resp=Response(HTTP_200=PessoaResponse))`
- Adiciona validação e documentação automática para essa rota.
- `resp=Response(HTTP_200=PessoaResponse)` diz que o endpoint retorna uma resposta HTTP 200 com o modelo `PessoaResponse`.

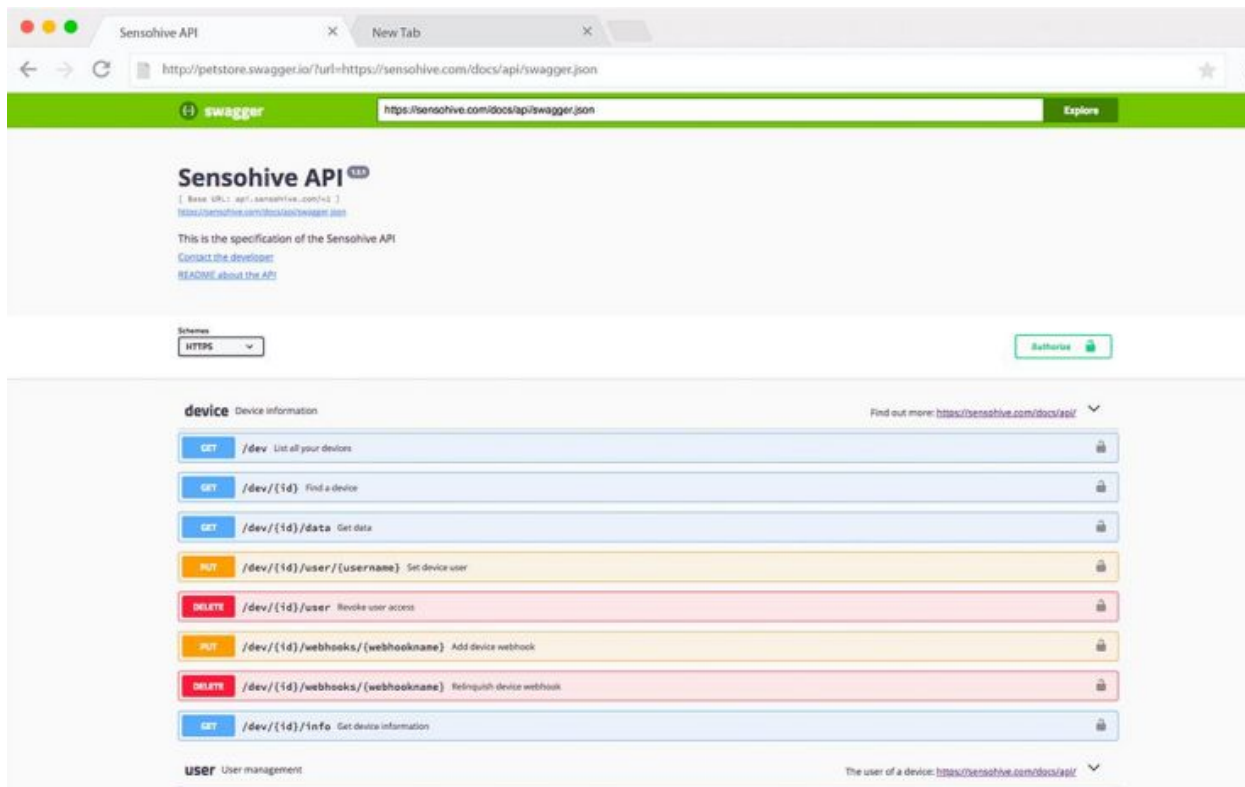
Contrato

- `def buscar_pessoas():`
- `return jsonify(nome='Jeffson Sousa')`
- Função que será executada quando o usuário fizer uma requisição GET para /pessoas.
- Retorna um JSON com o nome "Jeffson Sousa".

Contrato

- `if __name__ == '__main__':`
- `app.run()`
- Ponto de entrada do programa.
- Garante que a aplicação só será executada se esse script for rodado diretamente (não importado como módulo).
- `app.run()` inicia o servidor web Flask local (por padrão na porta 5000).

Contrato



FACI
wyden

Contrato

- `from flask import Flask, jsonify, request`
- `from flask_pydantic_spec import FlaskPydanticSpec, Request, Response`
- `from pydantic import BaseModel`

- `app = Flask(__name__)`
- `spec = FlaskPydanticSpec('flask', title='Impulsona Wyden')`
- `spec.register(app)`

Contrato

- class PessoaInput(BaseModel):
 - nome: str
 - idade: int
- class PessoaOutput(BaseModel):
 - mensagem: str
 - PessoaInput: define o formato esperado do corpo da requisição POST.
 - Espera receber um JSON com nome (string) e idade (inteiro).
 - PessoaOutput: define a resposta da API, com uma mensagem.

Contrato

- `@app.post('/pessoas')`
- `@spec.validate(body=Request(PessoaInput), resp=Response(HTTP_200=PessoaOutput))`
- `def criar_pessoa():`
- `dados = request.context.body # Aqui já está validado com base no modelo PessoaInput`
- `nome = dados.nome`
- `idade = dados.idade`
- `return jsonify(mensagem=f'Pessoa {nome}, com {idade} anos, criada com sucesso!')`

Contrato

- `@app.post('/pessoas')`: define a rota POST /pessoas.
- `@spec.validate(...)`:
- `body=Request(PessoaInput)`: diz que a entrada será validada usando o modelo `PessoaInput`.
- `resp=Response(...)`: define o modelo de resposta esperada (`PessoaOutput`).
- `request.context.body`: é onde o Flask Pydantic Spec armazena o corpo da requisição já validado.
- `jsonify(...)`: retorna a resposta como JSON.

Introdução ao Uso de API (Banco de Dados)

Profº Msc. Jeffson Celeiro Sousa
Doutorando em Ciência da Computação - UFPa

Representação

- Como esses dados devem ser representados?
- Existem vários formatos para a representação desses dados. O formato mais comum atualmente é **json**.
- Embora a representação do dado possa ser feita usando **XML**, e também **YAML**

Representação

- Como esses dados devem ser representados?
- Existem vários formatos para a representação desses dados. O formato mais comum atualmente é **json**.
- Embora a representação do dado possa ser feita usando **XML**, e também **YAML**

```
{  
  "field": "value",  
  "chave": "valor",  
  "nome": "Eduardo"  
}
```

Representação

- Como esses dados devem ser representados?
- Existem vários formatos para a representação desses dados. O formato mais comum atualmente é **json**.
- Embora a representação do dado possa ser feita usando **XML**, e também **YAML**

```
1  from pydantic import BaseModel
2
3
4  class Pessoa(BaseModel):
5      id: Optional[int]
6      nome: str
7      idade: int
```


Usando TinyDB com Flask

- O **TinyDB** é um banco de dados **NoSQL** em formato **JSON**. Ele é ótimo para aprender como funciona o armazenamento de dados de forma persistente, sem precisar instalar MySQL, PostgreSQL, etc.
- Etapa 1 — Instalar as dependências
 - `pip install tinydb`

Usando TinyDB com Flask

```
from flask import Flask, jsonify, request
from flask_pydantic_spec import FlaskPydanticSpec, Request, Response
from pydantic import BaseModel
from tinydb import TinyDB, Query

# Inicialização do Flask
app = Flask(__name__)
spec = FlaskPydanticSpec('flask', title='Impulsona Wyden')
spec.register(app)

# Inicialização do banco TinyDB
db = TinyDB('db.json') # Cria o arquivo db.json para armazenar os dados
pessoas_table = db.table('pessoas') # Cria e acessa uma "tabela" chamada pessoas
```

Usando TinyDB com Flask

- Etapa 1 — Instalar as dependências
 - `pip install tinydb`

Voltando aos Verbos

- O protocolo HTTP é a base usada por trás das APIs REST e as “**chamadas**” são feitas partindo de “**tipos**” de requisições. Vamos conversar sobre as mais populares, mas ao total são 9 verbos.

Voltando aos Verbos

- O protocolo HTTP é a base usada por trás das APIs REST e as “**chamadas**” são feitas partindo de “**tipos**” de requisições. Vamos conversar sobre as mais populares, mas ao total são 9 verbos.

GET	Pedir os recursos, dados. Ler o que está lá no servidor
POST	Criar um recurso, inserir um dado / registro
PUT	Alterar um registro (mandando todos os campos)
DELETE	Deletar um recurso / registro / dado

Respostas

- Para cada requisição no servidor, independentemente do verbo de “**chamada**” retorna alguns códigos de resposta. Eles são divididos em 5 grupos. Vamos conversar só sobre os mais recorrentes

Respostas

- Para cada requisição no servidor, independentemente do verbo de **“chamada”** retorna alguns códigos de resposta. Eles são divididos em 5 grupos. Vamos conversar só sobre os mais recorrentes

200	Códigos de sucesso
201	Criado
204	Sem conteúdo (usado para deletar com sucesso, por exemplo)
400	Códigos de erro
404	Não encontrado
500	Erro no servidor

Introdução ao Uso de API

Profº Msc. Jeffson Celeiro Sousa
Doutorando em Ciência da Computação - UFPa