Documentação da API Neoway

Este documento explicita com exemplos, como utilizar os recursos disponíveis na REST API de candidatos aprovados no vestibular, bem como do método de raspagem dos dados dos mesmos.

Destaco que esta API tem como objetivo atender ao desafio técnico da Neoway, podendo ser utilizada para fins de estudo, sendo desenvolvida em Python e utilizando as tecnologias Flask-RESTful e SQLAlchemy ORM, para as requisições e interações com o banco de dados SQL, bem como o Beautiful Soup para raspagem de dados, e envio dos mesmos para a API por requisições, permitindo, desta forma, a independência destes componentes.

Os testes utilizam o Pytest cobrindo os serviços da raspagem de dados.

1. Configurações Iniciais:

Clonar o diretório da API através do link:
https://github.com/wagnerberna/Desafio_Neoway
Acessar o diretório da API e executar os seguintes comandos no terminal:
Instalar dependências:

- python -m pip install -r requirements.txt

Iniciar API:

Execute no terminal o comando:

- python app.py

o serviço vai iniciar conforme imagem:

```
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Iniciar Raspagem dos dados:

Execute no terminal o comando passando ou não os parâmetros permitidos:

- python -c "import start; start.scraping( )"

- python -c "import start; start.scraping(start_page, stop_page, sleep_time_page)"

Descrição dos parâmetros:
nome_instancia.scrape_url
O método de raspagem dos dados permite definir os seguintes parâmetros.
start_page: número da página em que será iniciada a coleta.
stop_page: número da página final da coleta.
sleep_time_page: tempo de espera em segundos entre a coleta de cada página, para evitar que o servidor identifique as requisições como um ataque DDoS, e faça o bloqueio do IP.
Nenhum parâmetro é obrigatório, por padrão, ela vai da primeira à última página sem tempo de espera entre requisições.

2. Métodos de requisições:

Consultar Todos Candidatos

Requisição para listar todos os candidatos do sistema:

Method	URL
`GET`	`/candidates`

Como resposta, obtém-se uma lista de todos candidatos no formato JSON:

Status	Response Body
`200 OK`	`{ "candidates": [{ "candidate_id": 1, "name": "john smith", "score": 99.77, "cpf": "17842211711", "valid_cpf": false }, { "candidate_id": 2, "name": "michelle banks", "score": 73.66, "cpf": "01234685744", "valid_cpf": true }, { "candidate_id": 3, "name": "edward reid", "score": 91.02, "cpf": "01234758644", "valid_cpf": true }] }`

Consultar Candidato

Requisição para visualizar os dados de um candidato pelo seu ID.
Faz-se um GET de **`/candidate/{candidate_id}`**

Method	URL
`GET`	`/candidate/1`

Como resposta, obtém-se um JSON com os dados do candidato requisitado.

Status	Response Body

200 OK	{
	"candidate_id": 1,
	"name": "john smith",
	"score": 99.77,
	"cpf": "17842211711",
	"valid_cpf": false
	}

Caso a pesquisa seja realizada com um ID inexistente, retorna o status de erro com a resposta que o mesmo não foi encontrado.

Status	Response Body

404 Not Found	{
	"message": "candidate not found."
	}

Cadastrar Candidato

As validações e limpeza dos dados são realizadas no momento da raspagem dos dados e não pela API, pois esta já recebe os mesmos prontos.
Exemplo de requisição para cadastrar um novo candidato no formato JSON.

Method	URL
POST	/register
Header	
Content-Type	application/json
Request Body	

{
"name": "clark kent",
"score": 95.55,
"cpf": "77852171060",
"valid_cpf":true
}

Como resposta, obtém-se um JSON com os dados e ID do candidato criado, junto com o status 201 confirmando que a operação obteve sucesso.

Status	Response Body

201 Created	{
	"candidate_id": 5,
	"name": "clark kent",
	"score": 95.55,
	"cpf": "77852171060",
	"valid_cpf": true
	}

Caso o CPF do candidato já exista, retorna o status 409 de erro com a resposta que o mesmo já existe.

Status	Response Body

409 Conflict	{
	"message": "candidate cpf alredy exists"
	}

Atualização dos Dados do Candidato

Exemplo de Requisição para atualizar os dados de um candidato pelo ID: **`/candidate/{candidate_id}`**

Method	URL
PUT	`/candidate/5`
Header	
Content-Type	application/json
Request Body	

{
"name": "clark kent",
"score": 75.55,
"cpf": "77852171060",
"valid_cpf":true
}	

Como resposta, obtém-se um JSON com os dados atualizados e o ID do candidato, junto com o status 200 confirmando que a operação obteve sucesso.

Status	Response Body

200 OK	<pre>{
	  "candidate_id": 5,
	  "name": "clark kent",
	  "score": 75.55,
	  "cpf": "77852171060",
	  "valid_cpf": true
	}</pre>

Deletar um Candidato

Exemplo de Requisição para deletar um candidato pelo ID: **`/candidate/{candidate_id}`**

Method	URL
DELETE	`/candidate/5`
Header	
`Content-Type`	`application/json`

Exemplos de respostas: No primeiro item temos a mensagem de sucesso ao deletar um candidato existente. No segundo, ao tentar deletar o mesmo candidato ou outro inexistente, obtém-se o erro e status 404 informando que não foi encontrado.

Status	Response Body

200 OK	<pre>{
	  "message": "candidate deleted."
	}</pre>

404 Not Found	<pre>{
	  "message": "candidate not found."
	}</pre>

3. Testes:

Os testes estão localizados no diretório tests, sendo executados pelo comando no terminal:

- pytest

Obtendo o seguinte retorno: