Metodologia

Há diversas formas de realizar a coleta de dados de um site. Tal ato é chamado de Data Scraping/Web Crawling. Para isso são utilizadas bibliotecas que forneçam ferramentas e drivers para controle de Navegadores. As principais abordagens que se mostraram viáveis ao analizar como funciona a arquitetura do alvo em questão (JusBrasil) foram:

1. Navegar pelas páginas e pegar os dados disponibilizados diretamente no HTML

   Essa abordagem tem a vantagem de ser de implementação relativamente simples

2. Conseguir os dados a partir da interceptação das requisções e respostas realizadas pela páginas vindas da URL "https://www.jusbrasil.com.br/polaris-processos/graphql"

   Essa abordagem tem a vantagem dos dados serem serializados em JSON pelo Back-End da plataforma e, portanto esterão normalizados e serão fáceis de utilizar. Por outro lado, possui uma implementação mais complicada

Eu escolhi utilizar a primeira abordagem pois a biblioteca Python utilizada para interceptar as requisições do browser (SeleniumWire) é detectada pela tecnologia anti-bots da CloudFlare (CDN Responsável por servir as páginas estáticas). Com um pouco de pesquisa percebi que a biblioteca Pupeteer (NodeJS) possui um suporte muito bom para essa técnica, no entanto, por conta da minha maior familiaridade com python escolhi a primeira alternativa

Ferramentas

Linguagens de Programação

- Python - Por conta alta produtividade fornecida pela linguagem e das bibliotecas de automação com grandes comunidades

Bibliotecas

- Selenium - Uma biblioteca utilizada para automação web, permite selecionar e manipular elementos HTML
- Undetected Chrome - Uma distribuição de um navegador baseado em Chromium que possui menos rastros da automação e por isso não pode ser detectada pela tecnologia anit-bots da CloudFlare)

- CSV - Biblioteca nativa do python utilizada para armazenar registros em um arquivo CSV
- Pandocs - Utilizado para gerar a documentação a partir do arquivo com extensão .ipynb (Python Notebooks)

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import undetected_chromedriver as uc
import csv
```

Conseguindo os links das empresas

Conseguindo os links

Dentro da plataforma do JusBrasil uma empresa pode ser referenciada com diversos nomes variando coisas como pontuação, caracteres maísculos e minúsculos, etc. Para conseguir os links que listam os processos com todos esses nomes essa função recebe o link que leva à página de pesquisa de uma empresa e a acessa. Depois disso são selecionados todos os elementos de navegação (tag) que levam às páginas dos processos. Pegamos as propriedades que contém os links para os quais esses elementos iriam redirecionar e armazenamos em um vetor que será retornado ao final da função

Paginação

O principal desafio ao escrever o código dessa função é lidar com a forma como os links são separados por páginas, felizmente é possível avançar na páginação apenas adicionando o parâmetro "&p=numero_da_pagina" à URL, para isso foi necessário utilizar um laço de repetição ENQUANTO, que a cada iteração aumentara em 1 o número, se na página não houver nenhum link para uma página de empresa o laço será interrompido e a lista com as urls obtidas será retornada

```
def conseguir_links_empresa(empresa, driver):

    i = 1

    LINKS_FINAL = []

    while True:
        driver.get(f"{empresa}&p={i}")

        driver.execute_script("window.scrollTo(0,
document.body.scrollHeight)")
```

```
        elems = driver.find_elements(By.XPATH,
'//a[@class="EntitySnippet-anchor"]')

        links = [elem.get_attribute('href') for elem in elems]

        if len(links) == 0:
            break
        else:
            for link in links:
                LINKS_FINAL.append(link)
            i += 1

    return LINKS_FINAL
```

Conseguindo as informações dos processos

A função recebe os links das empresas (gerado pela função anterior) e acessa cada um deles pegando as informações dos processos disponíveis nas páginas, as compilando em um dicionário e adicionando esse dicionário a uma lista.

Carregamento assíncrono das informações

O principal desafio ao recolher as informações dessa página é o carregamento assíncrono na páginas, ou seja, os elementos são carregados conforme a página é rolada para baixo. A minha solução para esse problema é pegar o contador de processos no início da página rolar a página para baixo utilizando Selenium até que o número de elementos presente na página deixe de ser menor que o número indicado no contador. Depois disso todos os dados podem ser compilados em dicionários e adicionados à lista.

```
def conseguir_informacoes_processos(links_empresa, driver):

    lista_info_processos = []

    for link_empresa in links_empresa:

        driver.get(link_empresa)

        elementos_contador = driver.find_elements(By.XPATH,
'//strong[@class="LawsuitCounter"]/span')

        elems = []
```

```
        if len(elementos_contador) > 0:

            contador =
int(elementos_contador[0].get_attribute('innerText').replace(".", ""))

            if contador > 1000:
                contador = 1000

            elems = driver.find_elements(By.CSS_SELECTOR,
'LawsuitList-item')

            while len(elems) < contador:

                print(f"Contador: {contador} | Elementos:
{len(elems)}")

                driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")

                elems = driver.find_elements(By.CSS_SELECTOR,
'.LawsuitList-item')

                driver.implicitly_wait(2)


        for elem in elems:

            try:
                link = "Indisponível"

                if len(elem.find_elements(By.CSS_SELECTOR,
'.LawsuitCardPersonPage-title--link')) > 0:
                    if elem.find_element(By.CSS_SELECTOR,
'.LawsuitCardPersonPage-title--link').get_attribute('href') != None:
                        link = elem.find_element(By.CSS_SELECTOR,
'.LawsuitCardPersonPage-title--link').get_attribute('href')

                numero_processo = elem.find_element(By.CSS_SELECTOR,
'.LawsuitCardPersonPage-title--link .LawsuitCardPersonPage-header-
processNumber').get_attribute('innerText')
```

```
                tribunal_localidade = "Indisponível"

                if len(elem.find_elements(By.CSS_SELECTOR,
'p.LawsuitCardPersonPage-body-row-item-text[role="body-court"]')) > 0:
                    tribunal_localidade =
elem.find_element(By.CSS_SELECTOR, 'p.LawsuitCardPersonPage-body-row-
item-text[role="body-court"]').get_attribute('innerText')

                if len(elem.find_elements(By.CSS_SELECTOR,
'p.LawsuitCardPersonPage-body-row-item-text[role="body-kind"]')) > 0:
                    procedimento = elem.find_elements(By.CSS_SELECTOR,
'p.LawsuitCardPersonPage-body-row-item-text[role="body-kind"]')
[0].get_attribute('innerText')
                else:
                    procedimento = ""

                partes = elem.find_element(By.CSS_SELECTOR,
'strong.LawsuitCardPersonPage-header-
processInvolved').get_attribute('innerText')
                # número do processo, tribunal, localidade, UF, classe
ou procedimento e partes envolvidas.

                localidade = 'Indisponível'
                tribunal = 'Indisponível'

                if '·' in  tribunal_localidade:
                    tribunal = tribunal_localidade.split('·')[0]
                    localidade = tribunal_localidade.split('·')[1]
                else:
                    tribunal = tribunal_localidade

                uf = ""

                if tribunal != "Indisponível":
                    uf = tribunal.strip()[-2:]

                processo = {
                    "url": link,
                    "numero_processo" : numero_processo,
                    "tribunal" : tribunal,
                    "localidade" : localidade,
                    "procedimento" : procedimento,
                    "tribunal" : tribunal,
```

```
                    "uf" : uf,
                    "partes" : partes.split('x') if partes != None
else "Indisponível"
                }

                print(processo)

                lista_info_processos.append(processo)

            except Exception:
                print(f"Erro: {Exception}")
                pass

            if len(lista_info_processos) >= 1000:
                return lista_info_processos

    return lista_info_processos
```

Salvando o resultado das consultas em arquivos

Essa função recebe o nome do arquivo a ser salvo e a lista de dicionários que será transformada num arquivo CSV, depois disso são pegas as chaves do dicionário e colocadas na primeira linha do arquivo, então são pegos os registros e dispostos em linhas correspondentes às suas respectivas colunas.

```
def salvar_arquivo_csv(nome_arquivo, lista):

    chaves = lista[0].keys()

    with open(f'{nome_arquivo}', 'w', newline='') as arquivo_saida:
        dict_writer = csv.DictWriter(arquivo_saida, chaves)
        dict_writer.writeheader()
        dict_writer.writerows(lista)
```

Efetivamente realizando o processo de Scraping

Para realizar o processo de Scraping basta instanciar o driver do navegador, chamar a função que busca os links, passar seu resultado para a função que busca as informações e depois passar esse resultado para a função que salva os arquivos, dessa forma é possível compilar as informações disponíveis na plataforma em um arquivo.

```
driver = uc.Chrome(headless=False,use_subprocess=False)
```

```
links_empresa_puma =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=Puma+do+Brasil+Ltda.', driver)
info_processos_puma =
conseguir_informacoes_processos(links_empresa_puma, driver)
salvar_arquivo_csv('ds_puma.csv', info_processos_puma)

links_empresa_adidas =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=Adidas+do+Brasil+Ltda.', driver)
info_processos_adidas =
conseguir_informacoes_processos(links_empresa_adidas, driver)
salvar_arquivo_csv('ds_adidas.csv', info_processos_adidas)

links_empresa_nike =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=Nike+do+Brasil+Comercio+e+Participações+Ltda',
driver)
info_processos_nike =
conseguir_informacoes_processos(links_empresa_nike, driver)
salvar_arquivo_csv('ds_nike.csv', info_processos_nike)

links_empresa_asics =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=Asics+Brasil%2C+Distribui%C3%A7%C3%A3o+e+Com
%C3%A9rcio+de+Artigos+Esportivos+Ltda', driver)
info_processos_asics =
conseguir_informacoes_processos(links_empresa_asics, driver)
salvar_arquivo_csv('ds_asics.csv', info_processos_asics)

links_empresa_under_armour =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=Under+Armour+Brasil+Com%C3%A9rcio+e+Distribui
%C3%A7%C3%A3o+de+Artigos+Esportivos+Ltda', driver)
info_processos_under_armour =
conseguir_informacoes_processos(links_empresa_under_armour, driver)
salvar_arquivo_csv('ds_under_armour.csv', info_processos_under_armour)

links_empresa_reebok =
conseguir_links_empresa('https://www.jusbrasil.com.br/consulta-
processual/busca?q=reebok+produtos+esportivos+ltda.', driver)
info_processos_rebook =
```

```
conseguir_informacoes_processos(links_empresa_reebok, driver)
salvar_arquivo_csv('ds_rebook.csv', info_processos_rebook)

driver.quit()
```