# Web Scraper do Website JusBrasil pela biblioteca Puppeteer

Josué Teodoro Moreira

**Meridiano**
**2023**

# Índice de conteúdo

# Introdução

Assim que recebi o desafio, minha primeira ideia foi de utilizar uma API disponibilizada pela própria JusBrasil. Tal API existe, de fato, e se chama Digesto. Infelizmente, a Digesto é paga e requere diversos processos burocráticos que interferem com as regras definidas pelo desafio.

Dessa maneira, troquei minha atenção em seguida para o Web Scraping por meio da biblioteca Puppeteer. Na verdade, não pensei primeiramente na biblioteca Puppeteer, mas sim na Crawly (Elixir), depois na Selenium (Python), e por fim decidi continuar com a Puppeteer (Javascript/Typescript), já que durante minha entrevista foi citado que a LISS utilizava da tecnologia/linguagem Javascript.

## Web Scraping e processamento de dados

O web scraping é a prática de extrair informações por meio da leitura e processamento do conteúdo do híper texto (HTML) de um website. Nesse caso em específico, se trata da automatização do processo de entrar no website da JusBrasil, e utilizá-lo como um ser humano normal, porém controlado por um “robô”, enquanto extrai as informações presentes.

## Dos dados

Existem diversas maneiras de processar o conteúdo de um website e filtrar somente o necessário. No caso específico do desafio, percebi que o site da JusBrasil tem o conteúdo de estilização (CSS) muito bem estruturado, então seria a melhor maneira de separar os dados presentes:

### Páginas de processos

Antes de extrair os dados dos processos em si, percebi que diversos processos dão nomes diferentes para a mesma empresa. Na prática, isso cria vários resultados de pesquisa diferentes para uma única empresa.

Ante isso, julguei necessário extrair e agregar todas as páginas que levavam a processos dispersos:

- Paginação: `.Pagination`
- Botão de próximo:
  - Desativado: `li.pagination-item > a[aria-label=”Próximo”]`
  - Ativado: `li.pagination-item > a[aria-label=”Próximo”]`
- Lista de resultados: `.EntitySnippet-item`
- Url da página de processos únicos: `.EntitySnippet-header > .EntitySnippet-header-info > .EntitySnippet-anchor-wrapper > a`

### Processos

Como dito anteriormente, após extrair e agregar os links das páginas de processos dispersos, posso finalmente entrar nesses links e agregar os processos.

- Quantidade de processos: `.navbar-link.active > h2.navinline > span`
- Lista de processos: `a.LawsuitCardPersonPage`
- Link do processo: `a.LawsuitCardPersonPage-title—link`
- Número do processo: `span.LawsuitCardPersonPage-header-processNumber`
- Corte do processo: `p.LawsuitCardPersonPage-body-row-item-text[role="body-court"]`
- Procedimento/Classe do processo: `p.LawsuitCardPersonPage-body-row-item-text[role="body-kind"]`
- Partes do processo: `strong.LawsuitCardPersonPage-header-processInvolved`

# Desenvolvimento

Assim que separei os “selectors” que iria utilizar na extração dos dados, passei à programação em si. Criei o projeto NodeJS pel Yarn com versionamento Git e transpilação para Typescript, e fiz upload no Github: https://github.com/J0sueTM/jusbrasil_scraper. A documentação do código a seguir possui hyperlinks que direcionam ao github, inclusive à linha exata que estou citando.

## Engenharia de Software

Como estilo de arquitetura, apesar de eu prefirir pessoalmente modelos hexagonais e/ou programação funcional, julguei propício para esse software um modelo orientado a objeto, por meio de abstrações, herança, encapsulamento, etc, com armazenamento de dados e estado por meio de uma state machine (máquina de estado).

### Da State Machine

A máquina de estado é um conceito inicialmente matemático, mas bastante presente na engenharia de software. Uma máquina de estado é um agregado de informações que tem um valor dependente do momento; ou seja, no início da execução da aplicação, o estado pode ser 2, mas três operações a seguir, o estado pode ser 244.

Na minha aplicação, o estado se refere aos dados após cada execução de uma Spider (explicada a seguir). Está definida neste arquivo, baseados nas interfaces (DTOs) definidas neste arquivo.

Superficialmente, a state machine nesta aplicação é uma interface com os dados globalmente acessíveis, guardando informações geradas após os processamentos adjuntos das Spiders e dos Middlewares.

## Das Classes

Classes, no âmbito da programação orientada a objeto, é uma representação abstrata de um conceito real, que possui propriedades reais, as quais são igualmente abstraídas na programação. A seguir segue as que foram inseridas por mim no Software.

### *Spider*

Na terminologia do Web Scraping, uma spider é a implementação de um "robô" que realiza ambas as operações de simulação de um ser humano, e a extração de dados. Baseado nisso, criei a classe base Spider.

A classe Spider é uma classe abstrata, ou seja, não existe com o intuito de realizar operações, mas para ser base na construção de classes mais complexas.

- Atributos
  - **baseUrl**: O link base. Ex: www.jusbrasil.com.br/busca-processual.
  - **urls**: Uma lista com os links a serem processados.
  - **name**: Um nome de identificação.
  - **cacheFilePath**: O arquivo onde os dados extraídos serão salvos.
  - **middlewares**: Uma lista com os middlewares que processarão os dados.
- Métodos
  - **crawl()**: Executa a extração dos dados.
  - **saveCompany()**: Salva os dados apenas da última empresa que extraiu.
  - **save()**: Salva todos os dados de todas as empresas extraídas.
  - **load()**: Caso a operação já tenha sido executada antes, e um cachê foi encontrado, carrega os dados já salvos na memória, salvando tempo e processamento.
  - **shouldLoad()**: Verifica se existe um cachê da Spider em questão.
  - **run()**: Executa os processos definidos pela Spider já implementada.

### *ProcPagesSpider*

Essa é a primeira Spider executada. ProcPagesSpider é responsável por extrair os links das páginas dispersas que referem à mesma empresa. O motivo, como explicado anteriormente, é que devido aos diferentes nomes utilizados nos múltiplos processos, acabaram espalhados e separados em páginas dispersas.

A classe ProcPagesSpider herda as características da classe abstrata Spider, porém implementa os métodos especificamente:

**Implementação do método crawl()**

```
for await (const [idx, url] of this.urls.entries()) {
  await appState.page.goto(url);

  let curCompany: string = companies[idx];
```

Iteração de todos os urls de pesquisas para cada empresa.

```
let pageUrls = await appState.page.$$eval(
  ".EntitySnippet-item",
  (elms) => {
    return elms.map((elm) => {
      const selector: string =
        ".EntitySnippet-header > .EntitySnippet-header-info > .EntitySnippet-anchor-wrapper > a";

      return elm.querySelector(selector)?.getAttribute("href");
    });
  }
);
curPages = curPages.concat(pageUrls);
```

Lê todos os urls de páginas de processos encontrados e os agrega na array curPages.

```
let disNextButton = await appState.page.$$eval(
  disNextBtnSlc,
  (elms) => elms
);
let pagination = await appState.page.$$eval(
  paginationSlc,
  (elms) => elms
);
if (disNextButton.length >= 1 || pagination.length <= 0) {
  finished = true;

  break;
}
```

Verifica se existe um botão de próximo. Se existe, continua o loop. Caso não, significa que terminamos de ler os urls de páginas de processos e podemos passar para a próxima empresa.

```
await appState.page.waitForSelector(actNextBtnSlc, { visible: true });
await appState.page.click(actNextBtnSlc);
}
```

Caso o botão de Próximo exista, ele é clicado, levando-nos para a próxima página.

```
  appState.procPages.set(curCompany, curPages);
  this.middlewares[0].run();
  this.saveCompany(curCompany, curPages);
}
```

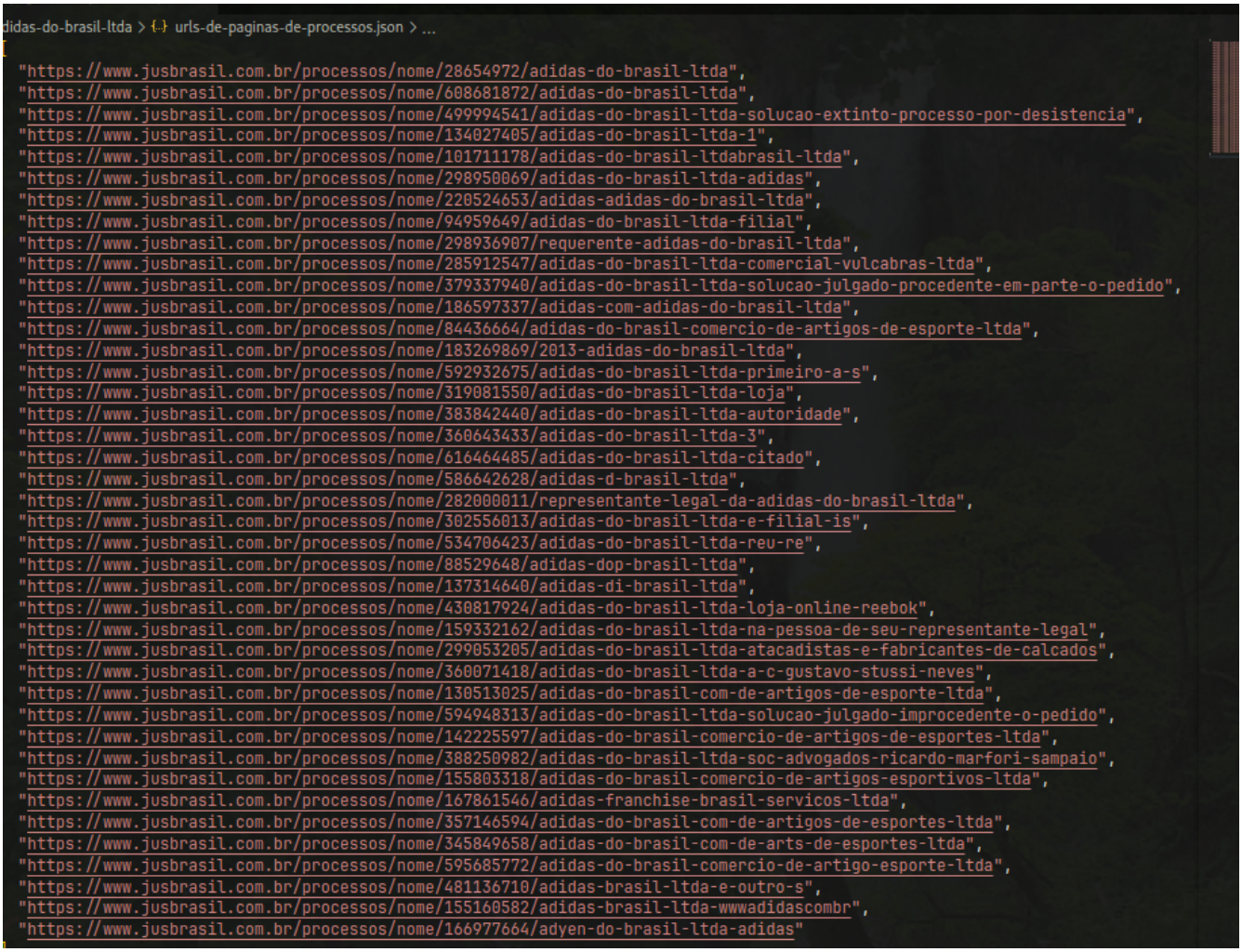
Assim que chega no fim da paginação, roda o middleware ProcPagesCleaner, o qual será explicado posteriormente, e salva os dados em cachê.

**Implementação dos métodos save() e load()**

```
  protected save() {
    // formatting would be nicer with:
    // const rawData = JSON.stringify(Object.fromEntries(this.dispProcUrls))
    // but I couldn't reverse it.

    const rawData = JSON.stringify(Array.from(appState.procPages.entries()));
    fs.writeFileSync(this.cacheFilePath, rawData);
  }

  protected load() {
    const rawData = fs.readFileSync(this.cacheFilePath, "utf8");
    appState.procPages = new Map(JSON.parse(rawData));
  }
}
```

Os métodos em si são simples, e apenas parseiam os dados para JSON e salva em cachê, e vice-versa.

Após finalizado, o dado final é salvo desta maneira:



## *ProcsSpider*

Esta é a segunda Spider a ser executada. ProcsSpider é responsável por utilizar dos links processados pela ProcPagesSpider, e finalmente extrair os dados dos processos em si.

**Implementação do método crawl()**

```
for await (const [company, page] of appState.procPages.entries()) {
  console.info(`Extraindo processos de ${company}...`);

  const progressBar = new cliProgress.MultiBar(
    {
      clearOnComplete: false,
      hideCursor: true,
      format: "{bar} >> {pagename} << {value}/{total}",
    },
    cliProgress.Presets.shades_classic
  );
```

Inicia-se iterando as páginas de processos extraídas pela ProcPagesSpider. Desta vez, como algumas das extrações demoravam muito tempo, adicionei um barra de progresso, que indica quantas processos e páginas de processos foram extraídos, quantos faltam, e de qual empresa os processos estão sendo lidos.

```
  await appState.page.goto(url);

  let isPageUnavailable = await appState.page.evaluate(() => {
    const elms = Array.from(document.querySelectorAll("*"));
    return elms.some((elm) =>
      elm.textContent?.includes("Página não encontrada")
    );
  });

  if (isPageUnavailable) {
    continue;
  }
```

Assim que o robô entra no website, é verificado se não ocorreu nenhum erro. Caso sim, a operação na página atual é encerrada, e a próxima é carregada.

```
const procCountSlc: string =
  ".navbar-link.active > h2.navinline > span";
let totalPageProcCount: number = await appState.page.$eval(
  procCountSlc,
  (elm) => {
    let rawNum: string = elm.textContent?.replace(/\D/g, '');

    const defaultThreshold: number = 1000;
    return Math.min(+rawNum, defaultThreshold);
  }
);
const pageBar = progressBar.create(totalPageProcCount, 0);
```

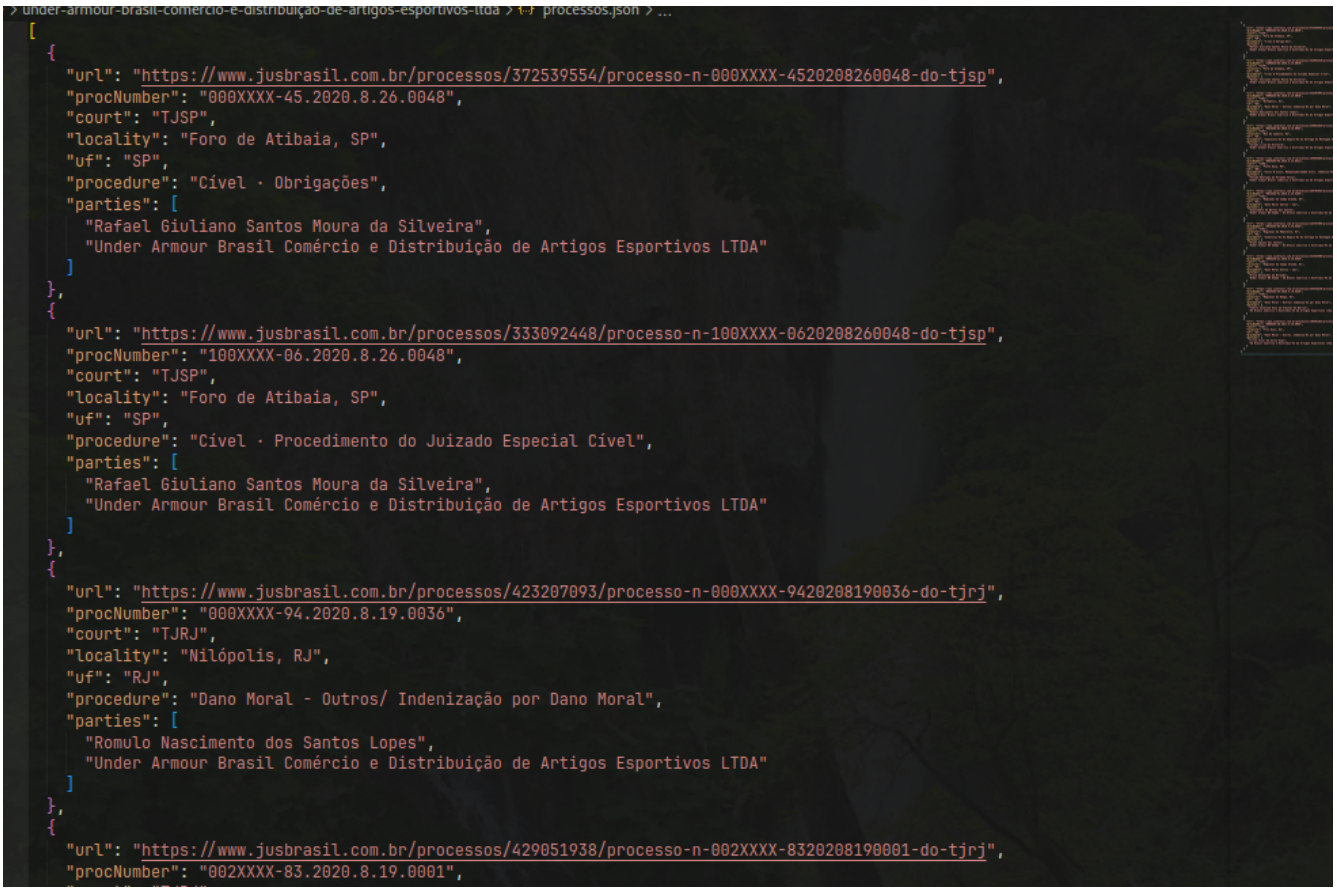
A seguir, o indicador da quantidade de processos naquela página é parseado e lido.

```
  const procUrl: string =
    elm.querySelector(procUrlSlc)?.getAttribute("href") ??
    "Indisponível";
  const procNum: string =
    elm
      .querySelector(procNumSlc)
      ?.textContent.replace("Processo nº ", "") ?? "Indisponível";
  const procCourtAndLocality: string[] = elm
    .querySelector(procCourtAndLocalitySlc)
    ?.textContent?.split(" · ") ?? [
    "Indisponível",
    "Indisponível",
  ];
  const procProcedure: string =
    elm.querySelector(procProcedureSlc)?.textContent ??
    "Indisponível";
  const procParties: string[] = elm
    .querySelector(procPartiesSlc)
    ?.textContent?.split("x")

  const procCourt: string = procCourtAndLocality[0];
  var procLocality: string = procCourtAndLocality[1];
  var procUF: string = procCourt?.slice(-2) ?? "Indisponível";
  if (procCourtAndLocality[1] !== "Indisponível") {
    procLocality = procCourtAndLocality[1] ?? "Indisponível";
    procUF = procLocality?.split(",")[1]?.slice(1) ?? "Indisponível";
  }

  // clean parties names' blank spaces
  procParties[0] =
    procParties[0] === undefined
      ? "Indisponível"
      : procParties[0]?.slice(0, -1);

  procParties[1] =
    procParties[1] === undefined
      ? "Indisponível"
      : procParties[1]?.slice(1);
```

Após tudo isso, os dados são finalmente extraídos.

```
    curPageProcCount = curPageProcs.length;
    pageBar.update(curPageProcCount, { pagename: `Página ${idx + 1}` });
  }

  this.savePage(company, idx + 1, curPageProcs);

  procs = procs.concat(curPageProcs);

  progressBar.remove(pageBar);
  compBar.update(idx + 1, { pagename: company.split(" ")[0] });
}

progressBar.stop();

console.info(`Total extraído: ${procs.length}\n`);

appState.procs.set(company, procs);

this.saveCompany(company, procs);
```

Em seguida, os processos lidos são adicionados à lista de processos da empresa atual, processados pelo middleware ProcParserMiddleware (que será explicado posteriormente), e por fim salvos em um arquivo JSON, como na imagem a seguir:



## *Middleware*

Um middleware é um processo menor adicionado à uma sequência já existente de processos maiores; No caso específico desse Software, os processos maiores são as Spiders, que executam a extração dos dados, enquanto os processos menores, os middlewares, executam a limpeza, parseamento dos dados, etc.

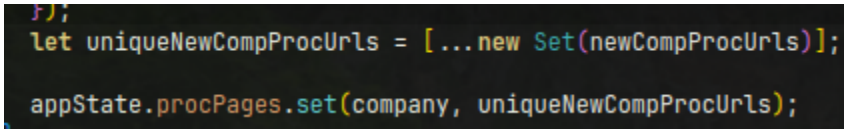
Assim como a classe Spider, a classe Middleware é uma classe abstrata que não possui nenhum atributo, mas apenas um método: run(), que executa o processo do middleware em si.

## *ProcPagesCleanerMiddleware*

A classe ProgPagesCleanerMiddleware herda as propriedados da classe Middleware, e implementa o método run(), realizando duas operações:

```
private blockedStrs: string[] = [
  "adyen",
  "petroleo",
  "petrleo",
  "petrolio",
  "system",
  "combustiveis",
  "engenharia",
];

public run() {
  for (const [company, urlsPage] of appState.procPages.entries()) {
    let newCompProcUrls = urlsPage.filter((url: string) => {
      return !this.blockedStrs.some((blockedStr: string) =>
        url.includes(blockedStr)
      );
    });
```

Remove quaisquer urls que não sejam estritamente das empresas em questão. Isso acontece devido a conflitos de nomes de empresas, como é o caso entre a Puma do Brasil (calçados) e a Puma do Brasil (petróleo).



```
});
let uniqueNewCompProcUrls = [...new Set(newCompProcUrls)];

appState.procPages.set(company, uniqueNewCompProcUrls);
```

Por fim, remove duplicatas e troca a lista antiga pela nova lista de urls de páginas de processos.

### *ProcParserMiddleware*

A classe ProcParserMiddleware também herda as propriedades da classe Middleware, e implementa o método run() parseando os processos adquiridos de cada empresa, e salvando-os em suas pastas finais.

```
public run() {
  try {
    const parser = new Parser(this.opts);

    for (const [company, procs] of appState.procs) {
      let newProcs = procs.map((proc) => {
        if (proc.procNumber.includes("Processo") || proc.procNumber.includes("Indisponível")) {
          proc.procNumber = proc.url?.split("processo-n-")[1]?.slice(0, 21)?.replaceAll('-', '.');
        }

        return proc;
      })
      appState.procs.set(company, newProcs)

      const parsedCsv = parser.parse(newProcs);

      fs.writeFileSync(`cache/${company.replaceAll(" ", "-")}/processos.csv`, parsedCsv);
    }
  } catch (err) {
    console.error(`Não foi possível gerar o CSV :: ${err}`);
  }
}
```

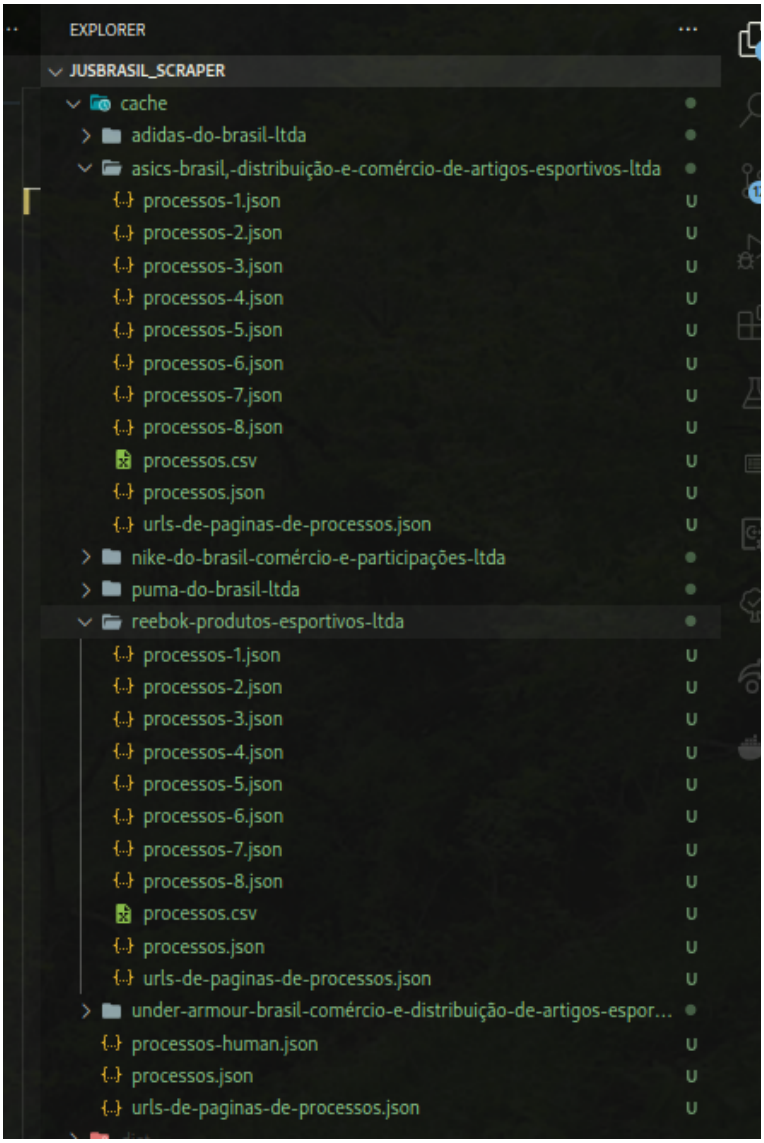
### *Scraper*

A class Scraper é a que controla todas as operações, incluindo as Spiders, os Middlewares e o driver do Puppeteer.

- Atributos
  - **baseUrl**: O url base.
  - **spiders**: Uma lista com instâncias das spiders que serão executadas.
- Métodos
  - **createCache()**: Cria as pastas necessárias para salvar os dados extraídos.
  - **openBrowser()**: Abre o navegador para realizar o Web Scraping.
  - **setSpiders()**: Setta as spiders a serem executadas.
  - **scrape()**: Inicia o processo de scraping.

## Da execução

Com os blocos básicos do Software detalhados, a inicialização e execução é realizada neste arquivo. Após executado, os arquivos são colocados em cachê, e prontos para análise.



## Considerações finais

Apesar de tudo, o desenvolvimento do Software foi um processo divertido e de altíssima aprendizagem. Eu nunca havia utilizado o puppeteer, e consegui produzir uma aplicação aparentemente viável, e que cumpriu seus objetivos de maneira muito melhor do que eu pessoalmente esperava.

Acerca de críticas, existem dois pontos que eu acredito que, se tivesse tempo, melhoraria e adicionaria ao Software: Threading e maior organização de arquitetura de software.

De qualquer forma, foi um prazer participar do desafio de estágio da LISS e espero que a minha solução seja de agrado para a vaga.