UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

João Paulo Taylor Ienczak Zanette

Extração de dados processuais dos Tribunais de Justiça para auxílio ao Jornalismo Investigativo

Florianópolis

João Paulo Taylor Ienczak Zanette

Extração de dados processuais dos Tribunais de Justiça para auxílio ao Jornalismo Investigativo

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Profa. Dra. Carina Friedrich Dorneles

Florianópolis

RESUMO

Devido ao aumento da disponibilidade de informações decorrente das tecnologias modernas de comunicação, o jornalismo de dados surge com o objetivo de avaliar essas informações, reorganizá-las e humanizá-las ao público. Para dados processuais, porém, há uma carência de acesso organizado a eles por parte dos portais dos Tribunais de Justiça do Brasil. Utilizando técnicas de raspagem de dados, neste trabalho foi construída uma ferramenta, nomeada TJScraper , que extrai os dados de todos os processos do Tribunal de Justiça do Rio de Janeiro. Para otimização dessa extração, foram utilizadas técnicas de programação assíncrona via corrotinas, *caching* dos processos além do aproveitamento do sistema de numeração unificado do Conselho Nacional de Justiça para redução significativa do trabalho necessário para descoberta de processos. A aplicação de tais técnicas se mostrou capaz de tornar o tempo de extração de processos de um ano específico hábil. Ao final, são discutidos alguns dos impactos das decisões de projeto e perspectivas de futuras otimizações e aspectos para se expandir do projeto.

Palavras-chave: jornalismo investigativo, web, tribunal de justiça, jornalismo de dados, consultas processuais.

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO E CONTEXTO

O avanço das tecnologias de comunicação, gerando uma abundância maior de informações bastante notória principalmente com a popularização da internet, traz à área de jornalismo mudanças em seu modo de produção. Com o pauteiro guiando a linha editorial buscando fatos, tendências e fenômenos, é papel do jornalista narrar uma história — ou seja, construir uma matéria — que corresponda a essas pautas e para isso buscar formas (entrevistas com os envolvidos na pauta, documentos, etc.) de responder as 6 perguntas básicas do jornalismo: Quem, O quê, Quando, Onde, Por Quê, e Como (KROLL, 2018). Quando se trata de Jornalismo de Dados[1], esse volume maior de informações fornece a determinados jornalistas acesso a bases de dados que nelas essas perguntas já estão respondidas, mas não há nenhuma **sintetização conclusiva** desses dados e tampouco uma **humanização**[2] (KROLL, 2022). É, por exemplo, o contraste entre tabelas contendo informações sobre condenações, localidades e renda familiar, e uma matéria apontando que o sistema judiciário é mais severo quando a condenação se aplica a pessoas que moram em locais da periferia (AMANDA LEMOS DANIEL E. DE CASTRO, 2018). O trabalho do jornalista de dados é então, em resumo, tratar de humanizar e apresentar esses dados ao público.

Porém ainda que haja abundância de informação, isso não significa que ela esteja acessível de forma conveniente para jornalistas. Uma base de dados expressa em uma planilha, agregada em um CSV ou seguindo alguma implementação de SQL é manipulável por *softwares* e bibliotecas que muitas vezes são Software-Livre (e portanto de livre uso (FSF, 2019)) e permitem a reorganização, filtragem, geração de gráficos estatísticos e outras tarefas conforme a necessidade de quem se propõe a escrever uma matéria jornalística. Já casos como os Tribunais Regionais de Justiça (TJs, para simplificação) brasileiros não fornecem seus dados (nesse caso, dados processuais) em nenhuma dessas formas, em vez disso o que há são resultados parciais (com relação ao todo dos processos registrados no portal do TJ) a partir de buscas pontuais como nome das partes, sentença, informações de advogados envolvidos, etc, ou seja, a informação está esparsa e tentar juntá-la manualmente pode ser humanamente impossível já que há milhões de processos por ano em apenas um TJ como o do Rio de Janeiro (JUSTIÇA DO ESTADO DO RIO DE JANEIRO, 2022). Há uma demanda — elencada em conversa com profissionais da

[1] Também chamado em inglês de *Data-Driven Journalism*.

[2] "Humanização de dados", no jornalismo, é apresentá-los de forma que o público deixe de vê-los como apenas números e passem a perceber que eles se referem a pessoas, a seres humanos.

área de Jornalismo Investigativo — pela sintetização desses dados e por consultas mais abrangentes, como busca pelo assunto do processo, assim um jornalista que atue com processos trabalhistas poderia analisar apenas os que dizem respeito essa área.

Para casos como esse em que um portal de informações não as disponibiliza de forma conveniente, uma forma de contornar essa deficiência é através de raspagem de dados da Web (*Web Scraping* (SOUZA; FILHO; SANTOS, 2021)), ou seja, o uso de ferramentas de software que façam a varredura de páginas da internet buscando e reorganizando dados de interesse de forma que possam ser posteriormente processados (GLEZ-PEÑA *et al.*, 2013; ZHAO, 2017). O uso dessas ferramentas, chamadas de "extratores", para tais finalidades dentro do jornalismo de dados, aliado a outras práticas, caracteriza o "jornalismo investigativo". Para os TJs, porém, não há — ao menos de acesso fácil e público — esse tipo de ferramenta, dificultando o trabalho de jornalistas investigativos na obtenção e análise de processos.

1.2 OBJETIVO E ESCOPO

Visando resolver a demanda por ferramentas de consulta processual mais abrangentes, neste trabalho se propôs a construção de uma ferramenta de raspagem de dados especializada em tribunais de justiça do Brasil de forma que a adição de suporte a um novo tribunal de justiça não acarrete em reescrever significativamente a ferramenta. A ferramenta, nomeada **TJScraper** , para o escopo deste trabalho se limita a apenas processos em primeira instância do Tribunal de Justiça do Estado do Rio de Janeiro (TJ-RJ).

Este trabalho também propõe, para implementação da ferramenta, estratégias para aceleração das consultas que ataquem os seguintes problemas: o acúmulo do tempo de espera entre o envio de requisições e o recebimento de suas respostas, e o retrabalho causado por consultas subsequentes cujos processos que extrairiam se interseccionem com processos que já foram previamente extraídos por uma consulta anterior. Para tal, deve-se ter também a construção da especificação de uma base de dados que considere a possibilidade de exportação para outros formatos.

1.3 MÉTODO DE DESENVOLVIMENTO

O levantamento e avaliação de requisitos deste trabalho foi construído com base no contato direto com dois profissionais da área de jornalismo investigativo através de reuniões ao longo de seu desenvolvimento. Pelo fato do Tribunal de Justiça do Estado do Rio de Janeiro (TJ-RJ) ser o mais informatizado no território brasileiro, ele é utilizado

como ponto de partida da produção da ferramenta TJScraper focando apenas nos seus processos em primeira instância para simplificação.

O funcionamento das páginas, rotas e APIs disponibilizadas pelo portal do TJ-RJ são descobertos via tentativa e erro, em especial para as APIs pela falta de documentação disponível. O portal do TJ-RJ possui uma restrição em que apenas no período das 23h00 às 7h30, o qual neste trabalho será referido como "horário noturno", está disponível para ferramentas de raspagem de dados para acessá-lo livremente. Fora dessa faixa de horário, o acesso é bloqueado por meio de *captchas* (pequenos "quebra-cabeças" que devem ser resolvidos para se obter acesso a uma página e os quais são inviáveis para softwares resolverem). Para contornar o horário noturno e permitir a produção da ferramenta em outros horários, foram criados testes unitários ao longo do desenvolvimento a partir de amostras de respostas de diferentes rotas do TJ-RJ.

2 TRABALHOS CORRELATOS

De outros extratores de Tribunais de Justiça, a (CRAWLY, 2022) é uma API JSON[1] que fornece consultas processuais simples a diversos portais públicos (TJ-RJ incluso), sem focar em jornalismo de dados. As rotas que esta API possui para acesso a dados do portal do TJ-RJ incluem apenas consultas com os mesmos parâmetros/campos de busca dele, tendo os acessos feitos ao subdomínio http://www4.tjrj.jus.br/ (o mesmo apresentado na Seção 3.1.1). Bastante semelhante a esta é a (BRAINY, 2022), também fornecendo dados processuais apenas sabendo informações prévias sobre eles (neste, necessariamente o número do processo), com diferenças na forma de uso. A (CODILO, 2022), também uma API, possui uma intersecção maior com os recursos deste trabalho: alimenta uma base de dados local, fornece *softwares* extratores de dados jurídicos, e permite a consulta de processos. Não é claro em sua apresentação sobre detalhes a respeito de quais dados os extratores são responsáveis por capturar, dificultando um levantamento de seus limites. O que se sabe, porém, é que as fontes de extração são portais diferentes dos utilizados no TJScraper , como o PJe (Processo Judicial Eletrônico) para processos trabalhistas e o Projudi para os estaduais. Essas três ferramentas não são abertas nem de **Software-Livre**[2]. São, inclusive, pagas (a (BRAINY, 2022) através da compra de créditos que são gastos ao longo das consultas). A Tabela 1 resume as diferenças entre as ferramentas descritas.

[1] "API JSON", neste trabalho, se refere a serviços Web que utilizam elementos das requisições HTTP para dar acesso a uma API que responde com objetos JSON.

[2] Um **Software-Livre** é aquele atende a quatro liberdades essenciais (FSF, 2019): a de se executá-lo como desejar, de estudá-lo e modificá-lo, de redistribuí-lo, e de redistribuir cópias de edições modificadas dele.

Característica	**(CRAWLY, 2022)**	**(BRAINY, 2022)**	**(CODILO, 2022)**	**TJScraper**
Tipo	API Web	API Web	API Web	Biblioteca/API Web
Licença	Não especificada	Não especificada	Não especificada	GPL-v3
Parâmetros de busca	Mesmos do TJ-RJ	Número do Processo	Não especificado	Numeração Unificada, Assunto, Ano
Campos dos processos disponibilizados	Todos	Todos	Não especificado	Todos
Restrição de acesso a dados	Token de acesso (pago)	Crédito (pago) por busca	Negociação com a empresa	Nenhuma
Banco de Dados local	Não	Não	Integração com BD pré-existente do cliente	Sim (próprio da ferramenta)
Fonte (TJ-RJ)				
Projudi			✓	
PJe	✓	✓	✓	
Consulta Processual	✓	Não especificado	✓	✓

Tabela 1 – Comparação de características de outras ferramentas com objetivos semelhantes à TJScraper.

3 CONSTRUÇÃO DA FERRAMENTA TJSCRAPER

A ferramenta TJScraper construída neste trabalho é um extrator produzido para ser capaz de extrair com qualidade satisfatória e em tempo hábil os dados de processos dos Tribunais de Justiça (TJs), tendo no momento implementada a extração de dados especificamente do TJ do Rio de Janeiro (TJ-RJ). As estratégias empregadas, portanto, levam em conta que todos os acessos serão ao TJ-RJ, ainda que visando a possibilidade de serem aplicadas às páginas dos TJs dos demais estados. Em resumo, a ferramenta acessa determinadas rotas (URLs) de TJs obtendo dados processuais, armazena-os e possibilita exibi-los ou exportá-los como planilhas.

Conforme critérios próprios deste trabalho, estabelecidos pelo levantamento de requisitos através de reuniões com profissionais do Jornalismo Investigativo com interesse nos dados processuais dos TJs, a qualidade dos dados extraídos será considerada satisfatória se:

- Puderem ser, de maneira direta, exportados para um formato legível para quem não conheça o funcionamento interno da ferramenta. Tal formato pode ser um formato de arquivo como uma planilha XLSX ou uma visualização em tabela em HTML;
- Possuírem campos úteis para análises tanto individuais quanto estatísticas dos processos como nomes das partes, assunto, data de abertura e última movimentação;
- A forma como os dados estão guardados viabilize realizar consultas personalizadas neles, tais como busca por assunto.

A construção do extrator se dá majoritariamente em termos das estratégias elaboradas neste trabalho conforme as necessidades que emergiram ao longo de sua produção: descoberta de processos válidos registrados em um TJ a partir da avaliação dos parâmetros de busca e exploração de padrões na numeração de processos do CNJ; a aquisição dos dados desses processos, dada em termos da correta interpretação das respostas dadas em HTML e JSON para consultas ao TJ-RJ; e a redução do tempo por consulta através de requisições assíncronas e do retrabalho em novas consultas via criação de um banco de dados local[1] para *caching* dos resultados.

3.1 ESTRATÉGIAS DE EXTRAÇÃO

O TJ-RJ possui rotas e subdomínios que dão acesso aos mesmos dados de processos por formatos de apresentação diferentes. Foram elaboradas estratégias para extrair

[1] "Local" dado em termos da máquina em que a ferramenta está executando.

dados da rota que os fornece em formato HTML, conhecida antes da produção efetiva da ferramenta, e da que os fornece em formato JSON, encontrada durante o processo de produção e portanto implementada posteriormente. O formato HTML é apresentado como uma página de consulta para um usuário humano, enquanto o formato JSON é um conjunto de rotas que implementam uma API Web.

Para a produção desta ferramenta não foi encontrada uma forma de se obter todos os processos de maneira direta, como uma lista de números de processo existentes ou uma tabela de registros no TJ-RJ. A estratégia adotada para isso, então, foi descobri-los exaurindo as possibilidades de valores de entrada de algum dos tipos de busca disponíveis nos portais do TJ-RJ. Analisando as dificuldades existentes em cada um dos tipos (Tabela 2), foi escolhido o campo de busca por número do processo por conta do domínio de valores válidos ser limitado e conhecido: números de processos são dados seguindo os formatos numéricos fixos **unificado** e **antigo**.

Tipo de busca	Dificuldades
Por Número	Número alto de combinações possíveis (para cada ano, $10^7 \times \|O\|$ tanto para numeração unificada quanto antiga, em que $\|O\|$ é a quantidade de unidades de origem do TJ em questão).
Por Nome	Inviável. Demanda conhecer todos os nomes existentes em processos.
Por OAB	Sem garantia de cobertura completa.
Por Nome do Advogado	Inviável. Demanda conhecer todos os nomes existentes em processos.
Por CPF / CNPJ	Número alto de combinações possíveis (em um sequenciamento ingênuo, 10^{11} para CPF e 10^{14} para CNPJ).
Por Protocolo	Sem garantia de cobertura completa.
Por Sentença	Sem garantia de cobertura completa.

Tabela 2 – Dificuldades encontradas para se exaurir as possibilidades de valores de entrada para cada um dos campos de busca permitidos pelo TJ-RJ.

O **unificado** (também referenciado como "numeração única") é universal para todos os TJs, padronizado pelo CNJ (JUSTIÇA, 2022, 2008) e segue o padrão "NNNNNNN-DD.AAAA.J.TR.OOOO", em que "NNNNNNN" é um número sequencial e único dado a cada processo, "DD" são os dígitos verificadores validados pelo algoritmo Módulo 97 Base 10 (STANDARDIZATION, 2003), "AAAA" indica o ano de distribuição, "J" identifica o órgão ou segmento do Poder Judiciário conforme a relação enumerada em (JUSTIÇA, 2008) Artigo 1º §4º, "TR" é uma subclassificação do órgão/segmento e indica qual o Tri-

bunal em questão (19 para o TJ-RJ) e "OOOO" é o código de a qual das unidades de origem (específicas para cada TJ) o processo se refere.

O **antigo** é o formato de numeração próprio de uso interno que cada TJ define para si. O do TJ-RJ segue o padrão "AAAA.OOO.NNNNNN-N" em que "AAAA" indica o ano de distribuição, "OOO" é o código da unidade de origem e cada "N" restante é um dígito de 0 a 9. Diferentemente da numeração unificada, os campos com "N" não são designados de forma serial, podendo haver sequências intercaladas de números que não correspondam a processos válidos.

Considerando ambos os formatos de numeração, para se exaurir as entradas possíveis é possível fixar um ano e um TJ (neste caso, o TJ-RJ) — o que também implica em fixar o segmento em 8 (valor para a categoria "Justiça dos Estados e do Distrito Federal e Territórios") — e variar os dígitos para os demais campos. No caso da unidade de origem, é possível se basear em uma lista conhecida (JUSTIÇA DO ESTADO DO RIO DE JANEIRO, 2009), o que reduz o número de entradas a se testar, e por fim os dígitos de verificação podem ser calculados a partir de uma combinação dos demais campos, não interferindo na quantidade de entradas testadas.

Apesar de supostamente o número de entradas a se testar para ambos os tipos de numeração ser o mesmo, a preferência foi dada à **numeração unificada** para descoberta de processos válidos tanto pelo caráter de padronização nacional, abrindo espaço para facilitar a adaptação de seu uso em outros TJs além do TJ-RJ do ponto de vista de implementação da ferramenta, quanto por algumas unidades de origem que possuem seu código de 4 dígitos não terem um espelho com 3 dígitos (JUSTIÇA DO ESTADO DO RIO DE JANEIRO, 2009). Além disso, há também o fator de serialidade do campo "N" na numeração unificada, que permite saber que a partir do primeiro número de processo que não corresponda a um processo existente a busca pode ser finalizada concluindo-se que todos os processos daquele TJ registrados até o momento no sistema foram encontrados.

Os demais métodos de busca, como via nome das partes, possuem uma abrangência de valores muito grande e difícil de se conhecer em sua totalidade ou, nos casos como a busca por Sentença, não são capazes de, mesmo exauridas todas as combinações de valores de entrada, dar acesso a todos os processos registrados (visto que nem todo processo possui uma sentença).

3.1.1 Estratégia inicial: extração HTML

Os resultados de consulta processual do TJ-RJ em HTML são os das requisições feitas ao subdomínio `ww4.tjrj.jus.br`. Em uma URL de visualização de um processo

um válido[2] nesse subdomínio (Figura 1), o parâmetro de URL (*query string*) `numProcesso` identifica o número do processo na numeração antiga. A estratégia de extração dos processos disponibilizados por esse subdomínio é efetuar requisições para a mesma URL variando apenas esse parâmetro com um número de processo diferente utilizando a numeração **antiga**. Cada requisição pode retornar uma página de processo válido, inexistente[3], de numeração inválida[4] ou de Captcha (caso esteja fora do horário noturno).

No caso de páginas de processo válidos, os campos que compõem os dados de um processo estão todos na mesma página bastando então *parsear* seu código HTML. Já as páginas de erro são identificáveis ora pelo seu título e ora pelo conteúdo com textos como "Erro" e "Parâmetro Incorreto" (Figura 2).

Há um porém: ainda que o subdomínio ww4 também permita pela sua interface Web a busca por numeração única, o resultado em caso de um processo válido é o redirecionamento à mesma página de caso a busca fosse feita por numeração antiga, com o valor do parâmetro de URL `numProcesso` também na numeração antiga. Portanto, não bastaria apenas variar o valor desse parâmetro com testes no formato da numeração única. É possível, contudo, aplicar a mesma tática de requisições em duas etapas descrita na Seção 3.1.2.

---

[2] Exemplo de URL de processo válido: `http://www4.tjrj.jus.br/consultaProcessoWebV2/consultaProc.do?numProcesso=2021.004.015548-9`.

[3] Exemplo de URL de processo inexistente: `http://www4.tjrj.jus.br/consultaProcessoWebV2/consultaProc.do?numProcesso=2020.004.015548-9`.

[4] Exemplo de URL de processo inválido: `https://www4.tjrj.jus.br/consultaProcessoWebV2/consultaMov.do?numProcesso=0`.

As informações aqui contidas não produzem efeitos legais.
Somente a publicação no DJERJ oficializa despachos e decisões e estabelece prazos.

Processo Nº 0015712-81.2021.8.19.0004

TJ/RJ - 13/06/2022 19:24:56 - Primeira instância - Distribuído em 24/08/2021

Processo eletrônico - clique aqui para visualizar.

Prioridade - Pessoa Idosa - Lei n º 10.741/03

Prioridade - Pessoa Deficiente - Lei n º 2.988/98

Comarca de São Gonçalo	**1ª Vara Cível** **Cartório da 1ª Vara Cível**
Endereço:	Getúlio Vargas 2512 3º andar
Bairro:	Santa Catarina
Cidade:	São Gonçalo
Ação:	Incapacidade Laborativa Permanente / Auxílio-Acidente (Art. 86) / Benefícios em Espécie
Competência:	Acidentes do Trabalho
Assunto:	Incapacidade Laborativa Permanente / Auxílio-Acidente (Art. 86) / Benefícios em Espécie
Classe:	Procedimento Comum
Autor	FERNANDO GOMES DE ARAUJO
Procurador	PEDRO ALBERTO DO NASCIMENTO
Réu	INSS - INSTITUTO NACIONAL DO SEGURO SOCIAL
Perito	CELSO TAVARES GARCIA
Advogado(s):	RJ053039 - PEDRO ALBERTO DO NASCIMENTO
Tipo do Movimento:	**Envio de Documento Eletrônico**
Data da remessa:	12/05/2022
Processo(s) no Tribunal de Justiça:	Não há.

Para visualizar Petições Pendentes de Análise ou Juntada clique aqui.

Localização na serventia:	Aguardando Manifestação

Figura 1 – Página de resultado da consulta pública para o processo válido de número 2021.003.015548-9 (numeração antiga).

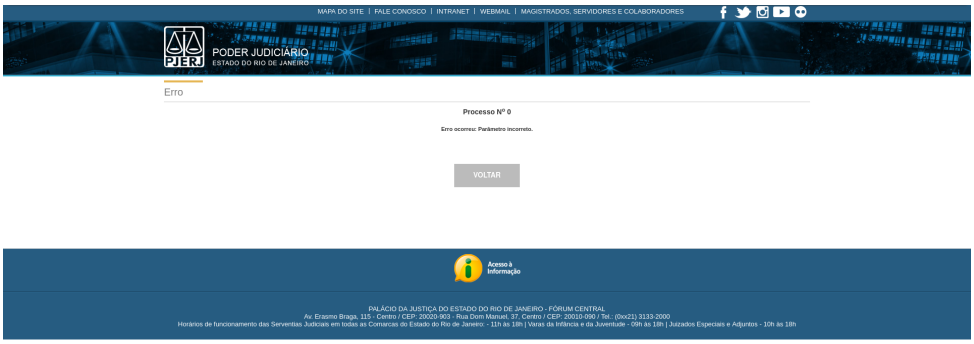

Figura 2 – Página de resultado da consulta pública para um número inválido de processo.

3.1.2 Estratégia alternativa: varredura com API JSON

No subdomínio `ww3.tjrj.jus.br`, há uma página de busca[5] semelhante à oferecida no subdomínio `ww4.tjrj.jus.br`, também fornecendo os dados em HTML. A diferença é que em vez de o servidor responder com o HTML completo e pronto para o lado cliente

[5] Rota da página de consulta: `https://www3.tjrj.jus.br/consultaprocessual/#%2Fconsultapublica%23porNumero=`

renderizar como uma página nova (que é o caso do subdomínio ww4), no subdomínio ww3 ao se acionar o botão de busca o cliente envia uma requisição AJAX para uma API do servidor, que responde com um objeto JSON com os dados do processo e então o HTML é atualizado no lado cliente de forma dinâmica. A estratégia para descoberta e aquisição dos campos então nesse caso passa a ser trivial: a partir do número de um processo, enviar uma requisição para a tal API e, com os dados de um processo em JSON, basta investigar quais os campos retornados, filtrar os de interesse e exportar da forma desejada.

A API do servidor do TJ-RJ possui rotas específicas para consultas pela numeração unificada[6] e pela antiga[7], ambas respondendo com dados de um processo através de requisições POST utilizando parâmetros de URL para especificar os parâmetros de busca. Os tipos de resultados de ambas as rotas encontrados durante a construção desta ferramenta estão expostos na Tabela 3.

Respostas comuns às consultas por numeração antiga e unificada	
Parâmetros de URL no formato de *Query String*	Resposta (JSON)
`tipoProcesso=1&codigoProcesso=0`	`["O processo informado não foi encontrado."]`
`tipoProcesso=1&codigoProcesso=abc`	`["Número do processo inválido."]`
Qualquer (em período noturno)	`{"status": 412, "mensagem": "Erro de validação do Recaptcha. Tente novamente."}`
Respostas: Consulta por numeração antiga	
`tipoProcesso=1&codigoProcesso=2021.001.140006-4`	`{"codProc": "2021.001.140006-4", [...]}`
Respostas: Consulta por numeração unificada	
`tipoProcesso=1&codigoProcesso=0158400-75.2021.8.19.0001`	`[{"codigoCnj": "0158400-75.2021.8.19.0001", [...]}]`

Tabela 3 – Respostas dadas pela API do `ww3.tjrj.jus.br` conforme diferentes valores na *query string*. Objetos com os dados dos processos foram truncados para facilitar a leitura da tabela.

A resposta em formato JSON para busca pela numeração unificada, em caso de um número válido de um processo existente, é uma lista com um objeto para cada instância daquele processo (identificada através do campo `tipoProcesso`) e apenas alguns, mas não todos, os campos úteis. Já a resposta da busca pela numeração antiga é um único objeto com todos os campos úteis do processo.

Conforme os motivos citados anteriormente para preferência pela busca por numeração unificada, ela é utilizada para a descoberta de novos processos, mas sendo incapaz de fornecer todos os dados do processo quando é encontrado um processo existente, uma requisição adicional para a rota da busca por numeração antiga é necessária. Para isso, se aproveita do fato de que os objetos JSON da resposta da numeração unificada possuem

[6] Rota para consulta pela numeração unificada: `https://www3.tjrj.jus.br/consultaprocessual/api/processos/por-numeracao-unica`

[7] Rota para consulta pela numeração antiga: `http://www3.tjrj.jus.br/consultaprocessual/api/processos/por-numero/publica`

um campo `numProcesso` com o número equivalente daquele processo na numeração antiga. Com a resposta dessa requisição adicional, selecionam-se os campos úteis de acordo com a relação exposta na Tabela 4. Alguns dos campos do JSON resultante da busca pela numeração unificada possuem chaves diferentes dos pela antiga, como o `numProcesso` que passa a utilizar a chave `codProc`.

Campo desejado	Chave no objeto JSON	Valor de exemplo
Assunto	`"txtAssunto"`	`"Furto (Art. 155 - CP)"`
Número do Processo (Num. Antiga)	`"codProc"`	`"2021.001.140006-4"`
Número do Processo (Num. Unificada)	`"codCnj"`	`"0158400-75.2021.8.19.0001"`
Data de distribuição	`"dataDis"`	`"14/07/2021"`
Unidade Federativa (UF)	`"uf"`	`"RJ"`
Advogados (Nome, Número da OAB,)	`"advogados"`	`[{"nomeAdv": "DEFENSOR PÚBLICO", "numOab": "0"}, ...]`
Cidade	`"cidade"`	`"Rio de Janeiro"`
Personagens (Autor, Autor do Fato, …)	`"personagens"`	`[{"codPers": "29760922", "nome": "MINISTERIO PUBLICO DO ESTADO DO RIO DE JANEIRO", "codTipPers": "1", "descPers": "Autor", "tipoPolo": "A"}, ...]`
Última Movimentação do Processo	`"ultMovimentoProc"`	`{"codTipAnd":6,"ordem":34,"dtAlt":"23/05/2022","descrMov":" - Documento","dtMovimento":"23/05/2022", ...}`

Tabela 4 – Relação entre campos desejados e qual a sua chave correspondente no objeto JSON retornado pela API de consulta pública do `ww3.tjrj.jus.br`.

3.2 ESTRATÉGIAS DE ACELERAÇÃO DE CONSULTA

Considerando uma análise ingênua de pior caso[8] visando obter todos os processos do TJ-RJ exaurindo as 97×10^7 [9] possibilidades da numeração unificada em um período arbitrário, o tempo médio para se obter 1 processo válido deve ser no máximo $0.89ms$ para menos de 30 dias[10] e $0.21ms$ para menos de 7 dias. Ou seja, para não tornar o uso da ferramenta inviável, foram implementadas estratégias para acelerar o tempo de obtenção de dados de processos visando que não supere muito tais valores.

3.2.1 Requisições assíncronas

O tempo despendido na busca por processos com operações de IO de rede[11], especialmente na espera pela resposta do servidor, cresce consideravelmente com a quantidade de processos buscados (Figura 3). Sendo assim, esta estratégia visa reduzir esse tempo

[8] O "pior caso" é quando o TJ em questão possui 9999999 processos registrados na última unidade de origem testada, assim para cada combinação de números de processo fixando o campo "NNNNNNN", apenas a última tentativa acarreta em um processo existente.

[9] Esse número vem do cálculo descrito na Tabela 2. No TJ-RJ, são 97 unidades de origem para serem testadas para cada um dos 7 dígitos em "NNNNNNN".

[10] Aproveitando as 8h do horário noturno durante 30 dias: $\lceil \frac{30 \times 8 \times 3600s}{97 \times 10^7} \rceil \simeq 0.89ms$

[11] "IO de Rede" é a comunicação com dispositivos de Entrada/Saída (no inglês, *Input/Output*) para comunicação de rede.

aproveitando recursos de programação assíncrona para que apenas o envio das requisições e o tratamento da resposta sejam sequenciais entre si, porém paralelos com relação às operações de IO (via IO não-bloqueante).

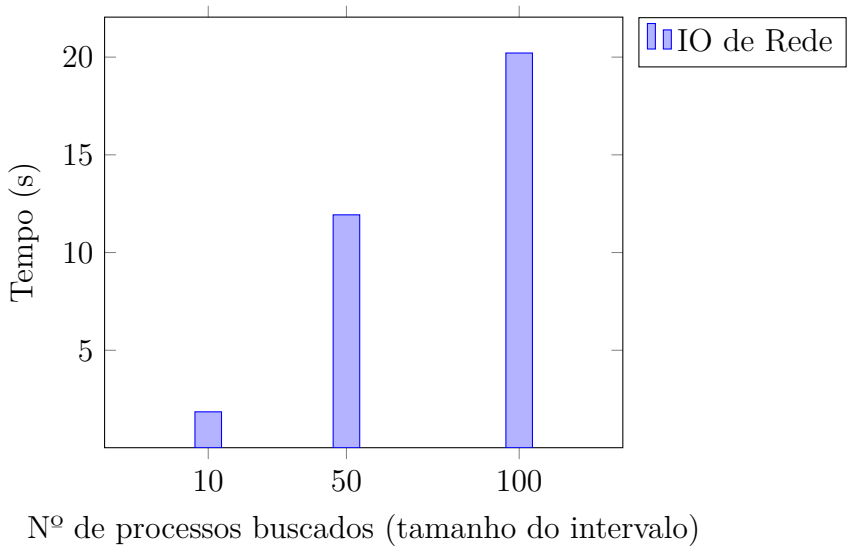

Figura 3 – Relação do tempo despendido com IO de rede conforme o tamanho do intervalo de números de processos buscado. Dados obtidos pelo perfilamento da execução de uma implementação em modelo síncrono das funções de busca da ferramenta.

Para ilustrar o mecanismo de requisições assíncronas, a Figura 4 compara como seriam executadas as operações de envio, espera e tratamento das requisições enviadas ao servidor de um TJ hipotético considerando a obtenção dos dados de 6 processos, enumerados como $P_{1...6}$. A sequência das operações estão em passos de tempo discretos ordenados da esquerda para direita. A linha "Processador" indica em qual operação o processador está trabalhando em um determinado passo de tempo e as linhas $E_{1...6}$ representam uma operação de IO relacionada a uma requisição R_i, como a espera pela resposta do servidor. Para cada processo P_i, R_i representa a operação de preparo e envio da requisição e T_i representa a operação de tratamento da resposta do servidor do TJ a R_i (por exemplo, classificação do processo como válido/inválido/inexistente ou reconhecimento de resposta com *Captcha*).

Nesse cenário hipotético com tempos arbitrários de espera para as requisições, para as requisições síncronas o processador seria obrigado a esperar pela resposta do servidor antes de começar a próxima tarefa para só então efetuar a próxima requisição, enquanto no modelo com requisições assíncronas todas as requisições são disparadas no início e, conforme o servidor responde às requisições, suas respostas já são tratadas imediatamente. A consequência esperada no cenário exposto é em $Tempo = 12$ se concluir o tratamento

de apenas uma única resposta no modelo síncrono, em contraste com o modelo assíncrono em que todas são tratadas nesse mesmo passo de tempo.

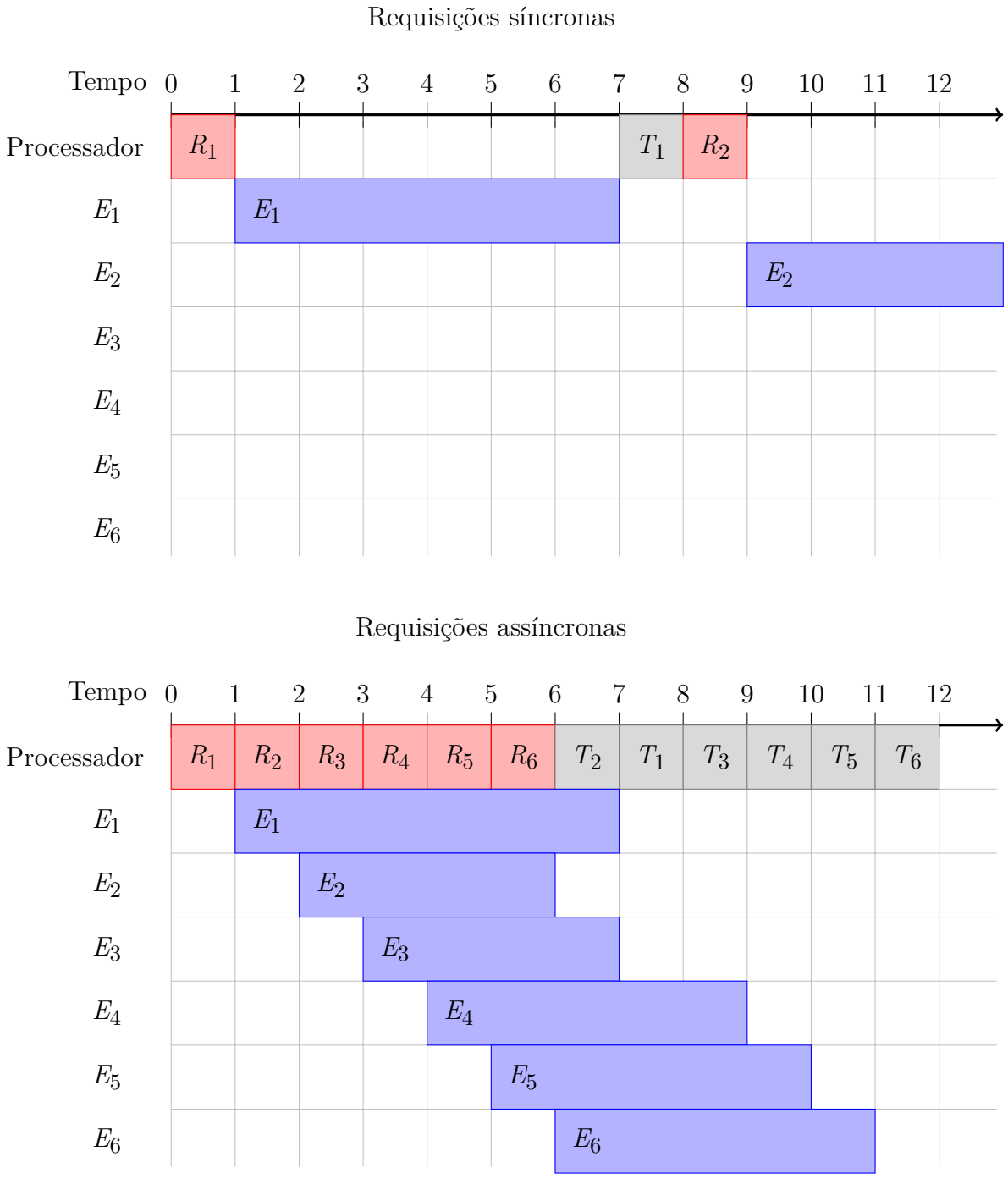

Figura 4 – Comparativo do mecanismo de requisições síncronas e assíncronas em um cenário hipotético.

3.2.2 *Cache* dos resultados

Considerando que diferentes consultas pela ferramenta podem ser feitas, existe um retrabalho quando elas possuem uma intersecção no conjunto de processos que retornariam. A estratégia para atacar esse problema consiste em guardar os processos descobertos em um banco de dados local simples, funcionando como uma *cache* para os processos já conhecidos, visto que uma primeira busca atendida já trará alguns dos processos da(s) próxima(s), não precisando ser buscados novamente por requisições ao servidor do TJ-RJ já que estarão prontos no banco de dados da ferramenta. Há também uma vantagem adicional com relação a descoberta de processos válidos uma vez que, conhecidos todos os processos de um único ano, não é necessário gastar tempo descobrindo quais números de processos correspondem a processos reais.

A *cache* também permite que o processo de aquisição de processos seja elaborado em duas etapas: uma de descoberta de processos e outra de exportação. Um benefício disso se reflete em consultas personalizadas que realizem algum tipo de filtragem dos resultados, uma vez que os processos descobertos em vez de descartados podem ser salvos na *cache* e reutilizado em outra consulta — seja esta filtrada ou não. Outro se dá com o horário de operação da ferramenta: se apenas no horário noturno processos podem ser acessados no TJ-RJ, com a *cache* eles passam a estar disponíveis em qualquer período desde que já tenham sido salvos anteriormente.

4 IMPLEMENTAÇÃO DA FERRAMENTA TJSCRAPER

4.1 ORGANIZAÇÃO GERAL

A ferramenta, cujo código-fonte está disponível em um repositório online na plataforma GitHub (TAYLOR IENCZAK ZANETTE, 2022), foi projetada como uma biblioteca de *software* acompanhada de duas interfaces de usuário[1]: uma aplicação de Interface de Linha de Comando (ILC[2]) e uma aplicação de servidor Web.

A ILC tem como objetivo permitir o uso de boa parte dos diferentes recursos da biblioteca em um terminal para a execução tarefas pontuais, como exibir informações sobre estado atual da *cache*, iniciar um processo de extração, ou mesmo iniciar uma instância de servidor Web da ferramenta para fins de desenvolvimento.

O servidor Web é destinado ao usuário que queira uma interface de simples acesso e uso da ferramenta, primariamente focando na obtenção de dados dos TJs com os recursos que não estejam disponíveis nas páginas oficiais dos TJs (descritos na Seção 1.1), tendo em mente como público alvo jornalistas.

4.2 FERRAMENTAS UTILIZADAS

Visando fomentar a iniciativa e movimento de Software-Livre, tomou-se como exigência própria para a implementação que todas as ferramentas, assim como a próprio TJScraper , fossem Software-Livre por, além de outros fatores, questões de reprodutibilidade e auditoria.

Para a implementação da TJScraper , escolheu-se Python 3.10 como linguagem principal considerando os seguintes critérios técnicos:

1. A premissa de que o gargalo do tempo de busca seria o de IO de Rede (para busca dos processos nos servidores dos TJs) ou de Armazenamento (para buscas na *cache*);

2. Não se tratando de um problema inerentemente atrelado à CPU[3], não seriam prioritárias otimizações agressivas de compilador tanto AOT nem JIT[4];

3. Familiaridade do autor com a linguagem;

[1] Nesta seção, "usuário" refere-se a alguém que não esteja utilizando a ferramenta como uma biblioteca de software, e sim como aplicação.

[2] Comumente referenciado com a sigla em inglês "CLI", de "Command Line Interface".

[3] Tradução livre para o termo em inglês "*CPU-bound*".

[4] "Ahead-Of-Time" — antes da execução do programa — e "Just-In-Time" — durante a execução do programa —, respectivamente.

4. Disponibilidade de recursos de abstração de alto-nível na biblioteca padrão da linguagem ou em bibliotecas externas para as principais operações da ferramenta.

O gerenciamento de dependências e isolamento do ambiente de desenvolvimento foi feito utilizando (POETRY, 2022) visando tanto a reprodutibilidade dos resultados e funcionamento da ferramenta quanto facilitar seu empacotamento e distribuição. A ILC foi implementada via (RAMÍREZ, 2022) pelas garantias de corretude de tipagem. Para a interface Web, foi escolhido (PALLETS, 2022) por conta da simplicidade e familiaridade do autor com a biblioteca.

Das implementações de extração: como a estratégia inicial visa a extração em páginas HTML, (ZYTE, 2022) foi utilizada por resolver essa demanda com recursos que visam facilitar tarefas comuns em raspagem de dados, enquanto a extração via API JSON utilizou-se puramente de recursos nativos da biblioteca padrão de Python em boa parte das operações e a biblioteca (NIKOLAY KIM AND ANDREW SVETLOV, 2022) para as requisições assíncronas. Do armazenamento, a *cache* foi implementada como um banco de dados (SQLITE, 2022). Da exportação, foram utilizados os formatos JSONLines (WOUTER BOLSTERLEE, 2022) como intermediário (visando facilitar a sua manipulação por outras bibliotecas e ser simples de gerar e exportar para outros formatos) e XLSX para visualização e manipulação por usuários via outros softwares que suportem o formato.

4.3 IMPLEMENTAÇÃO DAS ESTRATÉGIAS

Toda consulta por processos por meio da TJScraper corresponde a um processo completo de extração dos dados dos processos dos TJs, sequenciado nos seguintes passos:

1. Normalização dos parâmetros de busca a partir da entrada do usuário para um formato interno padronizado da ferramenta TJScraper . Os parâmetros implementados no momento da publicação deste trabalho são:

    - Intervalo de número dos processos, dados em termos do valor inicial e final para o campo “NNNNNNN” da numeração unificada;
    - Tribunal de Justiça a qual os processos pertencem;
    - Ano de distribuição dos processos;
    - Palavras a serem filtradas. Quando omitido, este parâmetro é ignorado.

2. Pré-filtragem dos dados averiguando quais dos processos no intervalo de busca já estão na *cache*, já aplicando os demais parâmetros de busca. De resultado, dados

previamente salvos são já exportados e um sub-conjunto com os demais números de processos é enviado para o passo do item 3.

3. Chamada da função de extração conforme a estratégia de extração utilizada (HTML ou API JSON);

4. Exportação dos dados advindos do item 3 no formato escolhido pelo usuário suportado pela ferramenta.

As estratégias enunciadas no Capítulo 3 se aplicam aos itens 2 e 3, nos quais são feitas respectivamente as consultas à *cache* e as requisições aos servidores dos TJs.

4.3.1 Extração HTML

Para a extração dos dados em páginas HTML, os passos da extração foram determinados como:

1. Dado um intervalo de números de processos (no formato unificado ou antigo), as URLs correspondentes a cada um dos valores nele são montadas e repassadas para uma *spider*[5] do Scrapy;

2. Cada URL passada irá eventualmente, em alguma ordem, chamar o método `parse` da *spider*, e nele é extraído o conteúdo de uma página seguindo os passos:

    a) Identificar se a página é de um processo inválido, inexistente ou se há a exigência de preenchimento de um *captcha*.

    b) Caso seja um processo válido, buscam-se na página os campos desejados.

Para os passos do item 2, utilizou-se a especificação (W3C, 2022) suportada nativamente pelo Scrapy para a busca dos padrões de resultado e de campos desejados no HTML da página avaliando a estrutura interna da página.

Uma página de resultado da consulta pública do subdomínio ww4 do TJ-RJ para um número de processo válido apresenta os campos dos dados do processo em uma *tag* HTML `<table>` com cada linha visual correspondendo a uma *tag* `tr` composta por duas *tags* `td`: uma para o nome do campo e outra para o valor (Listagem 4.3.1). Assim, um campo "A" pode ser encontrado procurando por uma `td` cujo conteúdo seja "A:", e o valor é o próximo vizinho em relação à hierarquia do HTML da página. A expressão XPath que representa esse padrão é utilizada na função `extract_field(field_text)`

[5] Mecanismo interno do Scrapy que reorganiza, escalona e lança as requisições repassando suas respostas à função de tratamento.

```
<tr>
  <td class="info" valign="top" nowrap="nowrap">Assunto:</td>
  <td valign="top">
    Incapacidade Laborativa Permanente / Auxílio-Acidente (Art.
    ↪  86) / Benefícios em Espécie
  </td>
</tr>
```

Listing 4.3.1 – Código HTML do campo "Assunto:" presente na Figura 1.

(Listagem 4.3.2), que extrai o valor de um campo arbitrário em uma resposta contendo uma página HTML conforme a seguinte ideia:

1. `//`: Define que o item buscado pode estar em qualquer ponto do documento. Uma busca genérica dessa forma evita que pequenas variações na hierarquia HTML da página prejudiquem a busca.

2. `td[text()='TEXTO:']`: Busca e seleciona um item da *tag* `<td>` que tenha como conteúdo exatamente a *string* "TEXTO:". Na função `extract_field`, "TEXTO" é substituído pelo argumento enviado ao parâmetro `field_text`.

3. `/following-sibling::td`: Se foi encontrado o item, então será seleciona o próximo vizinho de mesmo nível hierárquico que seja da *tag* `<td>`.

4. `/text()`: Desse item vizinho, captura-se o conteúdo dele como *string*. Por ser o último item do XPath, então esse será o valor retornado pela busca.

Para a detecção de páginas de erro, as expressões correspondentes e a ação que é realizada em cima delas para verificação do erro estão descritas na Tabela 5.

Erro	Verificação complementar	XPath
Página de *Captcha*	Existência do elemento.	`//*[@id="container_captcha"]`
Número inválido	Valor é exatamente "erro".	`//title/text()`

Tabela 5 – Expressões XPath e qual operação de verificação é realizada em cima de seus resultados para verificação de erro.

4.3.2 Extração de processos via API JSON e requisições assíncronas

A função `download_with_json_api` do módulo `tj_scraper.download` abstrai o processo de extração de dados de processos pela API JSON de TJs que conhecidamente

```python
def extract_field(response: Response, field_text: str) -> str:
    field_xpath =
    ↪  f"//td[text()='{field_text}:']/following-sibling::td/text()"
    return response.xpath(field_xpath).get().strip()
```

Listing 4.3.2 – Código da função responsável pela extração de um campo em uma resposta de uma requisição a uma página de visualização de processo.

a tenham. A implementação dessa função já faz proveito da estratégia de requisições assíncronas descrita na Seção 3.2.1 e tem como parâmetros: uma especificação das combinações de números de processo na numeração unificada que serão testadas, composta pelo intervalo de valores que serão utilizados para preencher o campo "NNNNNNN", o TJ em questão e o ano; o arquivo JSONLines (WOUTER BOLSTERLEE, 2022) de saída que será preenchido com os dados dos processos extraídos; o caminho para a *cache*; a função que será utilizada para filtrar os resultados; e o tamanho dos "lotes" (explicados mais à frente). O TJ passado por parâmetro contém as definições de suas rotas e unidades de origem previamente carregadas a partir de um arquivo TOML (PRESTON-WERNER, 2022). A definição das rotas é dada por uma rota para a numeração unificada e outra para a numeração antiga.

Adentrando na implementação do processo de extração, cuja ideia geral simplificada está resumida no Algoritmo 1, criam-se via expressão geradora[6] todas as requisições que serão enviadas ao servidor do TJ a partir da iteração pela especificação de combinações. Cada iteração gera uma combinação parcial em que se fixa, além dos demais campos previamente fixados na chamada da função[7], um valor para o campo "NNNNNNN" dado sequencialmente até que se chegue ao último valor do intervalo passado por parâmetro, mantendo variáveis apenas os campos de unidade de origem ("OOOO") e dígitos de verificação ("DD").

Em seguida, para cada combinação parcial, varia-se o campo de "OOOO" para cada uma das unidades de origem do TJ em questão, ordenadas do menor para o maior número. A cada variação, monta-se a *string* representante da numeração unificada e, então, uma primeira requisição de teste é enviada à rota correspondente do TJ. Utilizando a Tabela 3 como referência, a resposta do servidor é classificada em um resultado de erro específico ou um processo válido. No caso de um processo válido, uma requisição adicional é enviada para a rota da numeração antiga do TJ para se extrair os campos úteis do processo

[6] Mecanismo de Python para iteração através de avaliação preguiçosa.

[7] Mais claramente, como já são passados um TJ e um ano específico, os campos fixados previamente são "J", "TR" e "AAAA".

conforme a Tabela 4. Para processos com múltiplas instâncias, apenas o da primeira instância é utilizado.

Algoritmo 1: Extração de processos via API JSON (simplificada).

Entrada: $N_i \geq 0, N_f \geq 0, A \geq 0$, um TJ e uma função *Filtro*
Saída: Conjunto de processos P
para cada $N \in [N_i, N_f]$ **faça**
  para cada $O \in UnidadesDeOrigem(TJ)$ **faça**
    $D \leftarrow Modulo97(N, A, 8, Codigo(TJ))$;
    Gere Num_{CNJ} como a numeração unificada preenchida com $(N, D, A, 8, Codigo(TJ), O)$;
    Envie uma requisição R_1 para a rota de numeração unificada de TJ com Num_{CNJ};
    se R_1 *responder com um processo válido* **então**
      Extraia o campo de numeração antiga de R_1 e salve-o em Num_{Antiga};
      Envie uma requisição R_2 para a rota de numeração antiga de TJ com Num_{Antiga};
      se R_2 *responder com sucesso* **então**
        Salve o processo contido em R_2 na *cache* como válido;
        se $Filtro(R_2) = falso$ **então**
          Exporte R_2;
        fim
        Pare o *loop* interno;
      fim
    fim
    Salve o processo contido em R_2 na *cache* como inválido;
  fim
fim

Para realizar o disparo das requisições de forma assíncrona implementando a estratégia descrita na Seção 3.2.1, as funções internas chamadas pela `download_with_json_api` foram definidas segundo o modelo *async/await* da biblioteca padrão de Python. O modelo, nativo de Python 3.5+ baseado em Corrotinas (YURY SELIVANOV, 2015), é responsável pelo caráter de IO não-bloqueante nas requisições e exige, para tal e portanto, que a biblioteca de requisições seja compatível com ele, justificando o uso da AIOHTTP. A Listagem 4.3.3 reproduz, com as devidas simplificações, o código da função responsável pela tentativa de descoberta de um processo a partir de um único número em que o método `aiohttp.ClientSession.post` envia uma requisição POST de forma assíncrona. Foi utilizado um objeto de sessão para reaproveitamento da conexão, visto que a intenção é disparar múltiplas requisições.

```python
async def fetch_process(
    session: aiohttp.ClientSession,
    cnj_number: CNJProcessNumber,
    tj: TJ,
) -> FetchResult:
    cnj_number_str = make_cnj_number_str(cnj_number)

    request_args = TJRequestParams(tipoProcesso="1", codigoProcesso=cnj_number_str)

    # Descoberta do processo
    raw_response: TJResponse
    async with session.post(
        tj.cnj_endpoint,
        json=request_args,
        ssl=False,
    ) as response:
        raw_response = json.loads(await response.text())

    fetch_result = classify(raw_response, cnj_number, tj)

    if isinstance(fetch_result, FetchFailReason):
        return fetch_result

    request_args = TJRequestParams(
        tipoProcesso=str(fetch_result.get("tipoProcesso")),
        codigoProcesso=str(fetch_result.get("numProcesso")),
    )

    # Extração de todos os campos
    async with session.post(
        tj.main_endpoint,
        json=request_args,
        ssl=False,
    ) as response:
        raw_response = json.loads(await response.text())

    return classify(raw_response, cnj_number, tj)
```

Listing 4.3.3 – Reprodução do procedimento de busca por processos de maneira assíncrona.

Como forma de evitar sobrecarga dos servidores dos TJs por um excesso de requisições em um período curto de tempo causado pela implementação assíncrona, o que arriscaria sanções de algum dos servidores ou mesmo consumo excessivo de memória da máquina cliente devido à quantidade de objetos das Corrotinas instanciadas para as requisições, foi implementado um mecanismo limite de requisições simultâneas. O mecanismo funciona iterando pelas combinações em "lotes" de tamanho fixo, em que cada lote dispara um número máximo de requisições e o próximo lote só é executado quando todas do atual tiverem sido tratadas.

É necessário notar pela implementação que a iteração em blocos é sobre as combinações parciais — ou seja, variando pelo valor do campo "NNNNNNN", de forma a ser interpretada como "tentativa de se descobrir se existe um número de processo válido dados determinados valores para o campo "NNNNNNN"". Sendo assim, um semáforo iniciado com o tamanho do lote é utilizado para garantir que as variações internas para o campo "OOOO" das combinações parciais não irão acidentalmente ultrapassar o limite

do tamanho do lote.

Em busca de alguma informação sobre limites de tráfego definidos pelo TJ-RJ, tentou-se procurar se alguma dos caminhos intermediários da rota de consulta processual fornecia acesso a um arquivo `robots.txt` (comumente utilizado para especificar limitações para softwares como indexadores de busca), porém em nenhuma se obteve sucesso. Logo, para a estimava de tamanho de lote se limitou arbitrariamente a no máximo 1000 processos por lote.

4.3.3 Estrutura e mecanismo de *Caching*

A *cache* foi implementada como um instância de banco de dados SQLite (SQLITE, 2022) com uma única tabela "Processos" (reproduzida na Tabela 6), com a descrição dos valores possíveis para a coluna *"cache_state"* descritos na Tabela 7. A chave primária da tabela foi escolhida como o número do processo seguindo a numeração unificada, visto que é uma informação única a cada processo e, convenientemente, é o parâmetro de busca utilizado na extração, facilitando a separação de quais processos serão buscados na *cache* e quais serão requisitados ao servidores dos TJs.

Essa separação é uma operação trivial: para saber se um processo deve ou não ser requisitado aos servidores dos TJs, basta para cada número de processo da lista de números a serem buscados consultar, na *cache*, se não existe uma entrada cuja chave primária se diferencie apenas nos campos "DD" e "OOOO" e que o valor de "cache_state" seja "CACHED".

Processos		
Coluna	Tipo	Descrição
*`id`	text	Número do processo no padrão unificado.
`cache_state`	text	Estado atual do processo na *cache*.
`assunto`	text	Assunto do processo.
`json`	text	Dados do processo no formato JSON espelhando os campos retornados pela API do ww3.

Tabela 6 – Definição da tabela SQLite "Processos" utilizada para *cache*. "*" indica que o campo é uma chave primária.

Um processo sempre é salvo na *cache* quando o servidor do TJ responde à requisição referente a uma consulta por número de processo, com exceção de casos em que a resposta é um erro de captcha. Ou seja, mesmo que durante uma consulta um processo seja filtrado, para referência e aceleração de consultas futuras ele ainda assim é salvo na *cache*. Da

Estado	Descrição
`CACHED`	O número do processo é válido e os dados do processo estão na *cache*.
`INVALID`	O número do processo é inválido (por exemplo, dígitos de validação não conferem pelo algoritmo do TJ).
`NOT_CACHED`	O número do processo é válido porém os dados dele não estão na *cache*. Para uso no software, não é registrado na *cache*.

Tabela 7 – Definição da tabela SQLite "Processos" utilizada para *cache*. "*" indica que o campo é uma chave primária.

mesma forma, combinações que não gerem números de processos existentes também são salvas na *cache* com o estado `INVALID`.

Consultas à *cache* foram implementadas com consultas SQL aproveitando o registro de funções definidas em Python para as operações de filtragem, conforme código reproduzido na Listagem 4.3.4.

```
def restore_json_for_ids(
    cache_path: Path,
    ids: list[CNJProcessNumber],
    filter_function: Callable[[DBProcess], bool],
) -> list[ProcessJSON]:
    if not cache_path.exists():
        raise FileNotFoundError(cache_path)

    def is_id_in_list(id_: str) -> bool:
        return to_cnj_number(id_) in ids

    def custom_filter(number: str, state: CacheState, subject: str, json_str: str) -> bool:
        try:
            process = DBProcess(
                number, cache_state=state, subject=subject, json=json.loads(json_str)
            )
            return filter_function(process)
        except Exception as error:
            print(f"Failed to use custom filter: {error}")
            raise

    with sqlite3.connect(cache_path) as connection:
        connection.create_function("is_in_list", 1, is_id_in_list)
        connection.create_function("custom_filter", 4, custom_filter)
        cursor = connection.cursor()

        return [
            json.loads(item_json)
            for item_json, in cursor.execute(
                "select json from Processos"
                " where is_in_list(id)"
                " and custom_filter(id, cache_state, subject, json)",
            )
        ]

    return []
```

Listing 4.3.4 – Reprodução do código de restauração dos dados de processos da *cache* a partir de filtros personalizados.

5 ANÁLISES EXPERIMENTAIS

Nesta seção estão descritos os resultados obtidos a partir de experimentos visando como objetivo principal averiguar a capacidade da ferramenta de obter e exportar dados de processos de um TJ em tempo viável. Para isso, adotou-se como critérios de análise o **tempo total da consulta** de um intervalo de processos, separados entre tempo de CPU, IO de Rede e IO de Armazenamento, e o **tempo médio** para se descobrir um processo em um intervalo.

5.1 CONFIGURAÇÃO EXPERIMENTAL

Os experimentos foram realizados em uma estação de trabalho com um Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 16GB RAM DDR4 2133MHz e interface de rede 1000Mbps em uma conexão residencial.

Os dados de temporização foram obtidos através da média dos tempos de 3 execuções do perfilamento de uma chamada completa do processo de extração via API JSON do TJ-RJ de um intervalo de processos variando os parâmetros com todas as combinações dos valores especificados na Tabela 8, com o intervalo inicial (15712) escolhido arbitrariamente. Nas combinações com tamanho do lote em 1, o cenário é classificado como Síncrono (Sync), e os demais como Assíncrono (Async).

Parâmetro	Valores
Número inicial do intervalo	15712
Tamanho do intervalo do campo "NNNNNNN" (nº de processos)	10, 50, 100, 1000
Tamanho do lote	1, 10, 100, 500, 1000

Tabela 8 – Valores utilizados para os parâmetros de extração de dados de processos.

5.2 RESULTADOS EXPERIMENTAIS

A Figura 5 mostra um comparativo do tempo decorrido para uma implementação síncrona e uma assíncrona nos diferentes cenários de configuração, com a Tabela 9 expondo o tempo por processo do mesmo comparativo, obtido dividindo-se o tempo total naquela configuração pelo número de processos. Percebe-se por esses dados que passando do cenário síncrono para o assíncrono há uma redução significativa no tempo de IO de Rede (de 66% a 98%) mantendo um tempo de CPU bastante próximo, demonstrando a eficiência do uso de requisições assíncronas.

É possível perceber que o uso de requisições assíncronas traz uma maior redução no tempo médio com uma quantidade maior de processos, uma vez que um conjunto de poucos processos tira pouco proveito da assincronia não substituindo o tempo de ociosidade com o tempo de se disparar mais requisições, enquanto um número maior é capaz de preencher esse tempo.

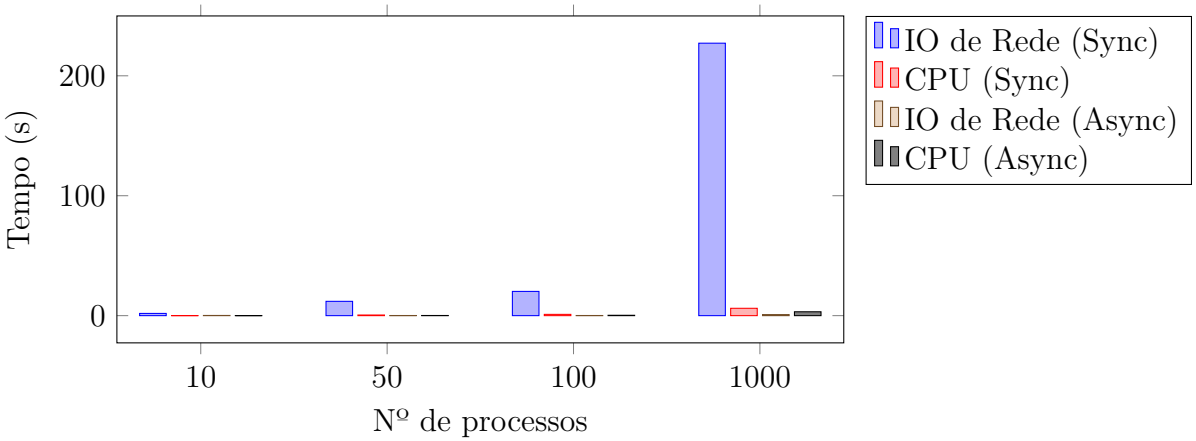

Figura 5 – Comparativo de tempo despendido com IO de Rede e processamento em CPU entre o cenário sem a aplicação de programação assíncrona via IO não-bloqueante (Sync) e com a aplicação (Async).

	Sync		Async	
Nº de processos	IO de Rede	CPU	IO de Rede	CPU
10	0.1850	0.0039	0.0158	0.0020
50	0.2386	0.0104	0.0022	0.0026
100	0.2021	0.0106	0.0012	0.0030
1000	0.2272	0.0061	0.0008	0.0032

Tabela 9 – Tempo médio de busca por processo comparando os cenários Síncrono e Assíncrono.

Quanto à separação das requisições em lotes, as medições de temporização estão expostas na Figura 6. Os resultados para um intervalo 10000 processos estão separados na Figura 7 para melhor visualização. De maneira geral, tamanhos de lote maiores ou iguais que 100 tiveram resultados similares para quase todos os tamanhos de intervalo de processos, tendo como excepcionalidade o intervalo de 10 processos com tamanho do lote em 500. Essa excepcionalidade, porém serve apenas para uma observação comparativa, visto que TJs possuem na ordem de centenas de milhares de processos por ano.

Para analisar essas medidas é preciso levar em conta que, para evitar sanções e sem acesso a um `robots.txt`, não é possível ter sempre o tamanho do lote igual ao do intervalo de processos. Partindo dessa premissa, as medidas expõem que, ao menos

para um intervalo de até 1000 processos com tamanhos de lote maiores ou iguais a 100, não há ganhos significativos em se manter o tamanho de lote tão próximo do tamanho do intervalo, porém para os casos de intervalos de tamanho 1000 e 10000 os lotes de tamanho 500 se demonstraram suficientes e com os melhores resultados.

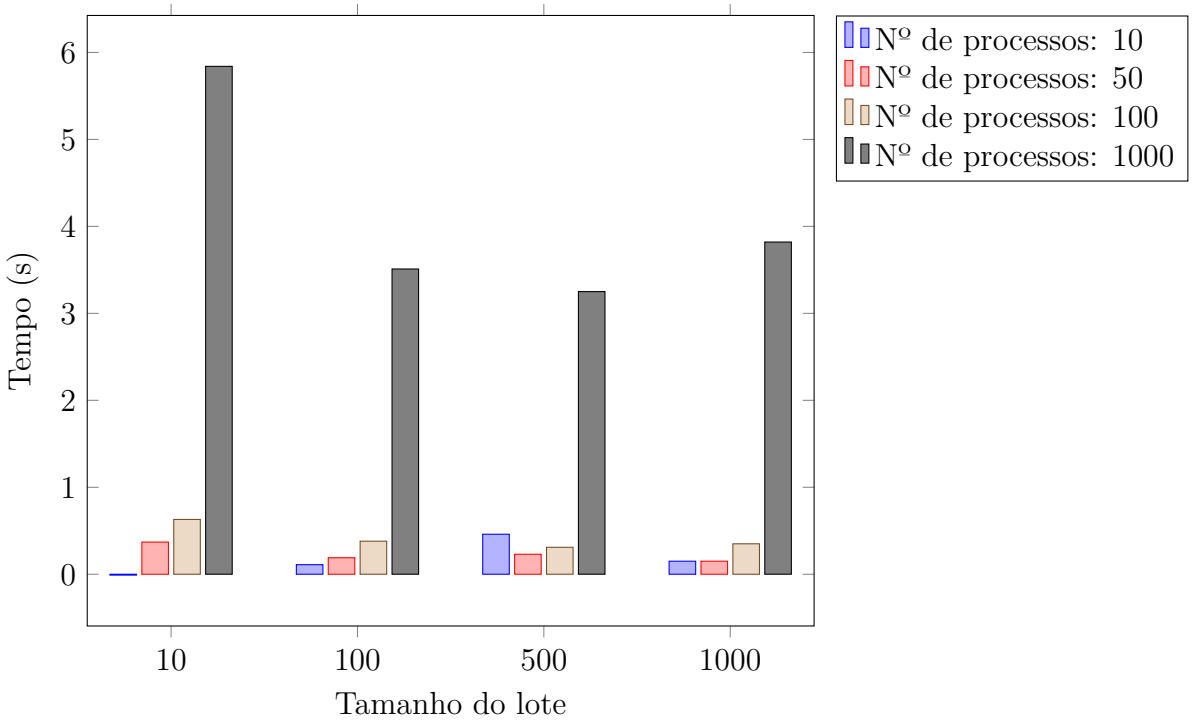

Figura 6 – Comparação do tempo de obtenção de todos os processos de diferentes intervalos conforme tamanho do lote.

Analisando-se o tempo de espera por IO de Rede na Figura 7, dado em uma média aritmética por processo, percebe-se também que um tamanho de lote muito grande (nesse caso, 1000) acarreta em um aumento no tempo de espera, o que deve explicar o fato de que o tempo total de consulta passa a aumentar com relação aos lotes de tamanho 500.

Para medir a eficiência da *cache*, foram separados três intervalos de valores para "NNNNNNN" com intersecção, $I_1 = [15712, 16712]$, $I_2 = [16212, 17212]$ e $I_3 = I_1 \cup I_2$, e 3 cenários diferentes Tabela 10 com os mesmos parâmetros de consulta (com exceção do intervalo de processos). As medidas de eficiência, dadas em termos do tempo total de execução e do tempo gasto com IO de Rede (ambos em segundos) estão expostas na Figura 8. A execução de cada cenário se dava com uma *cache* inicialmente vazia.

No cenário com intersecção completa (cenário C), como esperado, na consulta C_2 o tempo de IO de Rede é zerado visto que todos os processos que C_2 baixaria já foram previamente salvos na *cache* por C_1 e portanto sendo carregados dela em vez da rede. É interessante notar que, com isso, o tempo total da consulta também se reduziu drasticamente, demonstrando que o uso da *cache* foi compensatório. No cenário com

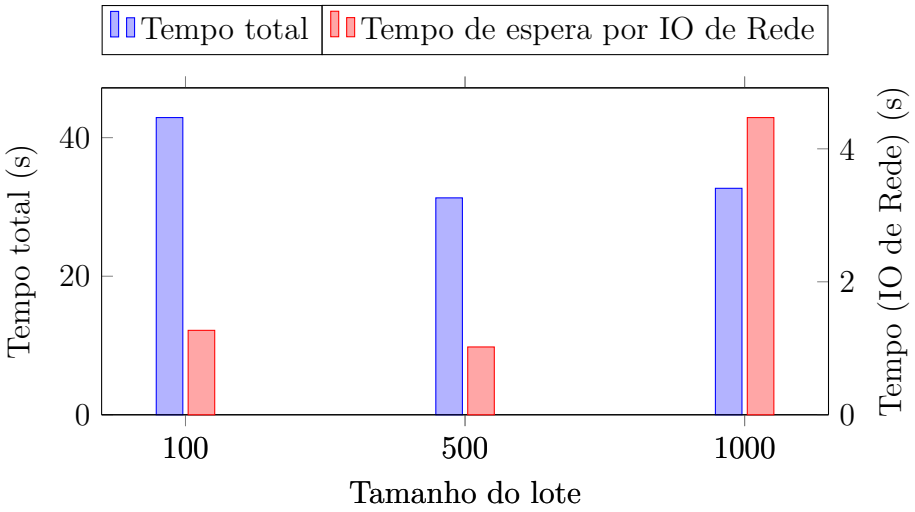

Figura 7 – Comparação do tempo de obtenção de todos os processos de um intervalo de tamanho 10000 conforme tamanho do lote.

intersecção parcial (cenário A), novamente o tempo total de consulta também se reduziu significativamente além do tempo de IO de Rede. Comparando as consultas A_2 e B (que operam sob o mesmo intervalo, I_2), reforçam-se as conclusões analisadas em C: para um mesmo intervalo, a *cache* ainda que parcial foi capaz de reduzir ambos o tempo de IO de rede e o tempo total da consulta, sendo novamente compensatória.

Cenário	Objetivo	Características
A	Medir a eficiência da cache em reduzir parte do retrabalho.	Separado em duas consultas feitas em sequência: A_1 (intervalo I_1) e A_2 (intervalo I_2). A *cache* preenchida por A_1 é reutilizada para A_2.
B	Medir o tempo de A_2 sem *cache*.	Uma única consulta no intervalo I_2.
C	Averiguar se há compensação em utilizar a *cache* para reduzir o tempo de busca em caso de retrabalho.	Separado em duas consultas, C_1 e C_2, no intervalo I_3 reaproveitando a *cache* preenchida por C_1 em C_2.

Tabela 10 – Descrição dos cenários utilizados para medição da eficiência da *cache*.

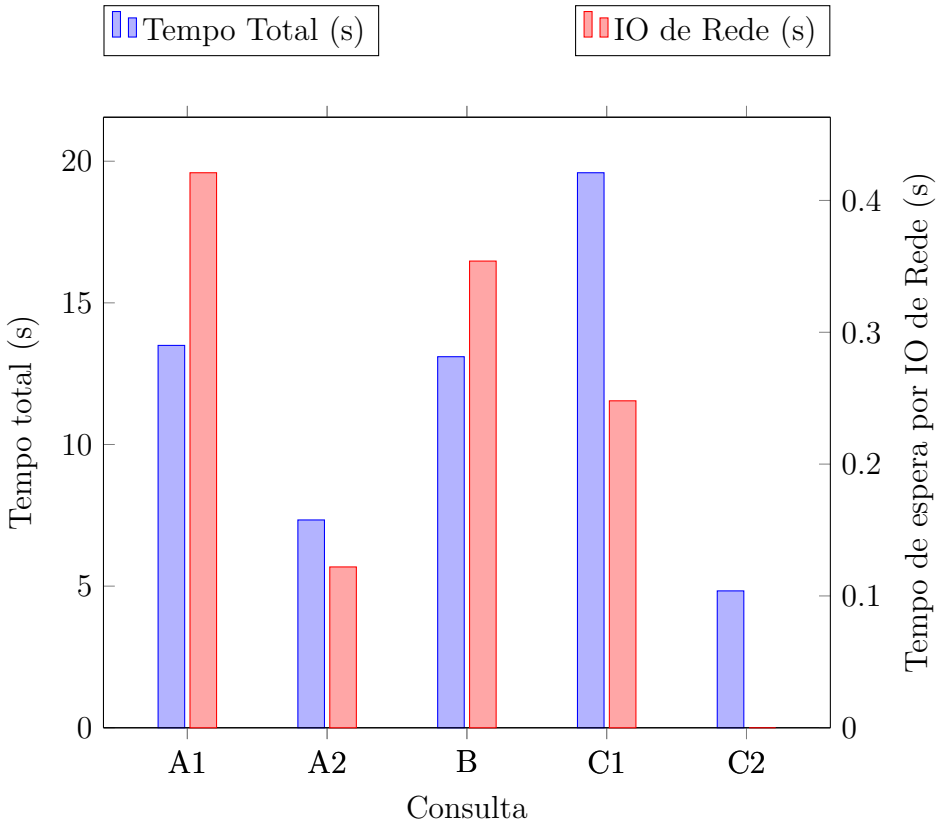

Figura 8 – Comparação do tempo de obtenção de todos os processos em 5 consultas específicas a fim de demonstrar a eficiência do uso de *cache*.

6 CONCLUSÕES E PERSPECTIVAS

Há pelo menos quatro pontos de vista que se permitem ser analisados neste trabalho: do propósito da ferramenta TJScraper , dos resultados obtidos pelos experimentos, das decisões de projeto de implementação dela, e das escolhas de terceiros na modelagem e apresentação de dados da fonte de extração.

No que se refere ao propósito da ferramenta, por ser capaz de obter os dados todos os processos de um Tribunal de Justiça, a partir do momento que se tem esses dados de alguma forma local independente do próprio portal do TJ há já um auxílio para jornalistas investigativos uma que vez esses dados, convertidos para um formato de familiaridade do investigador, podem ser reorganizados da maneira que for mais conveniente para sua análise. Há, nisso, um detalhe curioso de que ainda que o propósito original em essência da ferramenta seja a exportação dos dados em um formato como uma planilha XLSX, o processo de implementação troca a finalidade para a alimentação e gerenciamento correto da *cache* — na prática uma replicação local de um banco de dados —, sendo a exportação é um mero detalhe como um passo adicional desse processo.

Pelos resultados experimentais ficou confirmada a hipótese de que o gargalo do tempo de extração dos dados de processos estaria no tempo de IO (especialmente o de Rede). Atacando esse gargalo, as técnicas de aceleração de consulta aplicadas neste trabalho cumpriram seu objetivo de reduzir significativamente o tempo de consulta, chegando a — considerando apenas IO de Rede — cerca de 4ms por processo. Ainda que não estejam dentro dos 0.78s mencionados no Capítulo 3, se for considerado que processos do TJ-RJ costumam totalizar menos de 2 milhões de processos (CONJUR, 2019; AMAERJ, 2020) e considerando apenas o tempo de IO de Rede é possível extrair dados de todos os processos de um ano do TJ-RJ em menos de 30 dias. Uma observação pertinente é que, não se utilizando de *threads* para a implementação das estratégias (e sim Corrotinas), a complexidade de implementação foi mínima visto que não foram necessários mecanismos de sincronização ou controle de condições de corrida. Dito isso, ainda há espaços para mais otimizações, como o reordenamento das unidades de origem de forma dinâmica ao longo da extração pelas que possuirem mais processos registrados (explorando uma potencial distribuição estatística). Com a presença da *cache*, também, é possível que o trabalho de extração seja feito de forma distribuida e posteriormente unidos os registros de *caches* de diferentes máquinas em uma única.

Do ponto de vista das decisões de projeto de implementação da TJScraper , a estratégia de utilizar a API JSON, por não depender de reconhecimento de padrões textuais e sintáticos para obtenção de dados, se demonstrou mais fácil e eficiente de se implemen-

tar e utilizar. Ainda assim, a implementação da extração HTML servindo como prova de conceito e podendo ser útil para TJs que não possuam uma API JSON — contando apenas com a exibição de páginas HTML — foi mantida na base do código-fonte.

Na questão de ferramentas, vale ressaltar que foram necessários meios não tão convencionais (ZANETTE, 2022) de se contornar problemas de implementação de Scrapy para a elaboração de testes unitários: por ser uma biblioteca mais antiga que o mecanismo de `async/await` de Python 3.5+, a Scrapy se baseou na Twisted, uma biblioteca cuja implementação se utiliza de estado global e mecanismos de controle que impedem a execução de uma ou mais *spiders* mais de uma vez em um mesmo processo, dificultando a implementação de testes unitários (já que cada teste executaria uma *spider* novamente). Com a implementação da extração via API JSON utilizando o mecanismo de `async/await`, porém, é possível reescrever a extração de processos via HTML utilizando uma biblioteca substituta para o uso de expressões XPath em cima desse mecanismo.

Além das otimizações possíveis, outros trabalhos que se desenrolam a partir deste incluem o suporte a outros TJs que não o TJ-RJ, o que demanda a investigação de quais possuem um processo vigente de informatização, e o tratamento de processos que possuam múltiplas instâncias.

Por fim, do ponto de vista das escolhas de terceiros, observa-se que as decisões de um portal de dados e transparência, como é o caso dos TJs, impactam além do usuário-alvo original. Primeiramente, este trabalho só foi possível pois há, ainda que de maneira espalhada (múltiplos subdomínios com funcionamentos diferentes, por exemplo), algum grau de informatização. Ele, cumprindo sua função, se torna mais uma ferramenta para que outros setores de pesquisa e/ou jornalísticos possam analisar e explicar à sociedade sobre fenômenos que ela vive, e portanto informatizar é também dar conhecimento. Da mesma forma, porém, ele surge por consequência de uma "deficiência" nos mecanismos de busca dos TJs do ponto de vista de facilitar que sejam feitas análises estatísticas, visto que um mero parâmetro "Assunto" nas consultas processuais ou uma forma de listagem de processos facilitaria já o trabalho de quem faça tais análises ou de implementação de uma ferramenta semelhante à TJScraper . Observando as decisões arquiteturais, algumas afetam de maneira potencialmente não esperada, a exemplo da existência de uma API JSON que, apesar de criada provavelmente para melhor implementar um sistema de SPA (*Single-Page-Application*) no subdomínio ww3 do TJ-RJ, foi útil para se reduzir o esforço necessário para extração dos campos dos dados dos processos. A falta de documentação pública dessa API, porém, frequentemente se mostrou uma barreira durante a implementação, uma vez que novos formatos de resposta do servidor do TJ-RJ eram descobertos para processos específicos, ou mesmo com relação a não se saber se há formas de simpli-

ficação da implementação da ferramenta devido ao não conhecimento de parâmetros ou rotas que podem existir na API mas que não são conhecidos.

APÊNDICE A – ARQUIVO DE PÁGINAS

Tabela de páginas arquivadas para reprodutibilidade.

Descrição da Página	Arquivo
TJ-RJ: Página de processo válido no subdomínio ww4.	https://web.archive.org/web/20220714040302/http://www4.tjrj.jus.br/consultaProcessoWebV2/consultaProc.do?numProcesso=2021.004.015548-9
Tj-RJ: Página de processo inexistente no subdomínio ww4.	https://web.archive.org/web/20220714043304/http://www4.tjrj.jus.br/consultaProcessoWebV2/consultaProc.do?numProcesso=2020.004.015548-9
Tj-RJ: Página de processo inválido no subdomínio ww4.	https://web.archive.org/web/20220714043454/http://www4.tjrj.jus.br/consultaProcessoWebV2/consultaProc.do?numProcesso=0

REFERÊNCIAS

AMAERJ. Tribunal do Rio recebeu 1,8 milhão de novos processos no ano passado, 7 jan. 2020. Disponível em: https://amaerj.org.br/noticias/tribunal-do-rio-recebeu-18-milhao-de-novos-processos-no-ano-passado/. Acesso em: 21 jul. 2022.

AMANDA LEMOS DANIEL E. DE CASTRO, Natália Portinari. Morar em Favela do Rio é Agravante em Condenação por Tráfico de Drogas. **Folha de São Paulo**, 27 abr. 2018. Disponível em: https://www1.folha.uol.com.br/cotidiano/2018/04/morar-em-favela-do-rio-e-agravante-em-condenacao-por-trafico-de-drogas.shtml. Acesso em: 24 jul. 2022.

BRAINY. **Intima.ai**. Disponível em: https://documenter.getpostman.com/view/11707205/T17GgoJW?version=latest#intro. Acesso em: 24 jul. 2022.

CODILO. **Codilo API — API de captura de dados públicos**. Disponível em: https://www.codilo.com.br/. Acesso em: 24 jul. 2022.

CONJUR. TJ do Rio recebeu mais de 1,6 milhão de novos processos em 2018, 3 jan. 2019. Disponível em: https://www.conjur.com.br/2019-jan-03/tj-rio-recebeu-16-milhao-novos-processos-2018. Acesso em: 21 jul. 2022.

CRAWLY. **Plexi API**. Disponível em: https://crawly.atlassian.net/wiki/spaces/PLEXIAPI/pages/1147994113/TJRJ+-+Consulta+processual. Acesso em: 24 jul. 2022.

FSF, Free Software Foundation —. What is Free Software?, 30 jul. 2019. Disponível em: https://www.gnu.org/philosophy/free-sw.htm. Acesso em: 24 jul. 2022.

GLEZ-PEÑA, Daniel; LOURENÇO, Anália; LÓPEZ-FERNÁNDEZ, Hugo; REBOIRO-JATO, Miguel; FDEZ-RIVEROLA, Florentino. Web scraping technologies in an API world. **Briefings in Bioinformatics**, v. 15, n. 5, p. 788–797, abr. 2013. ISSN 1467-5463. DOI: 10.1093/bib/bbt026. eprint: https://academic.oup.com/bib/article-pdf/15/5/788/17488715/bbt026.pdf. Disponível em: https://doi.org/10.1093/bib/bbt026.

STANDARDIZATION, International Organization for. **Information technology — Security techniques — Check character systems**. 2003. [*S.l.*], 2003. Disponível em: https://www.iso.org/standard/31531.html. Acesso em: 8 jul. 2022.

JUSTIÇA, Conselho Nacional de. **Numeração Única - Portal CNJ**. Disponível em: https://www.cnj.jus.br/programas-e-acoes/numeracao-unica/. Acesso em: 8 jul. 2022.

JUSTIÇA, Conselho Nacional de. **Resolução Nº 65 de 16/12/2008**. [*S.l.*], 2008. Disponível em: https://atos.cnj.jus.br/atos/detalhar/atos-normativos?documento=119. Acesso em: 8 jul. 2022.

JUSTIÇA DO ESTADO DO RIO DE JANEIRO, Tribunal de. **Código das Serventias**. 2009. Disponível em: https://www.tjrj.jus.br/consultas/cod_serventias/cons_cod_serventias. Acesso em: 8 jul. 2022.

JUSTIÇA DO ESTADO DO RIO DE JANEIRO, Tribunal de. **Poder Judiciário do Estado do Rio de Janeiro**. Disponível em: http://www.tjrj.jus.br/. Acesso em: 18 jul. 2022.

KROLL, John. Como humanizar os dados. **Rede de Jornalistas Internacionais**, 26 jan. 2022. Disponível em: https://ijnet.org/pt-br/story/como-humanizar-os-dados. Acesso em: 24 jul. 2022.

KROLL, John. Examinando mais a fundo com as perguntas básicas do jornalismo. **Rede de Jornalistas Internacionais**, 30 out. 2018. Disponível em: https://ijnet.org/pt-br/story/examinando-mais-fundo-com-perguntas-basicas-do-jornalismo. Acesso em: 24 jul. 2022.

NIKOLAY KIM AND ANDREW SVETLOV. **AIOHTTP — Asynchronous HTTP Client/Server for asyncio and Python**. Disponível em: https://docs.aiohttp.org/en/stable/. Acesso em: 18 jul. 2022.

PALLETS. **Flask Documentation**. Disponível em: https://flask.palletsprojects.com/en/2.1.x/. Acesso em: 18 jul. 2022.

POETRY. **Poetry - Python package management and packaging made easy**. Disponível em: https://python-poetry.org. Acesso em: 18 jul. 2022.

PRESTON-WERNER, Tom. **TOML: Tom's Obvious Minimal Language**. Disponível em: https://toml.io/en/. Acesso em: 18 jul. 2022.

RAMÍREZ, Sebastián. **Typer**. Disponível em: https://typer.tiangolo.com/. Acesso em: 18 jul. 2022.

SOUZA, Wiliane Maria; FILHO, Wladimir; SANTOS, Wylliams. Ferramenta de Web-Scraping: Impactos da COVID-19 na Indústria de Software. *In*: ANAIS Estendidos do XVII Simpósio Brasileiro de Sistemas de Informação. On-line: SBC, 2021. P. 49–52. DOI: 10.5753/sbsi.2021.15354. Disponível em: https://sol.sbc.org.br/index.php/sbsi_estendido/article/view/15354.

SQLITE. **SQLite3**. 2022. Disponível em: https://www.sqlite.org/about.html. Acesso em: 18 jul. 2022.

TAYLOR IENCZAK ZANETTE, João Paulo. **TJScraper: v0.1.1**. [*S.l.*: *s.n.*], jul. 2022. DOI: 10.5281/zenodo.6899201. Disponível em: https://github.com/jptiz/tj-scraper.

W3C. **XPath Path Language 3.1**. Disponível em: https://www.w3.org/TR/2017/REC-xpath-31-20170321/. Acesso em: 18 jul. 2022.

WOUTER BOLSTERLEE. **JSONLines**. Disponível em: https://jsonlines.readthedocs.io/. Acesso em: 18 jul. 2022.

YURY SELIVANOV. **PEP-492**. [*S.l.*], 2015. Disponível em: https://peps.python.org/pep-0492/. Acesso em: 18 jul. 2022.

ZANETTE, João Paulo Taylor Ienczak. **TJScraper**. 2022. Acesso em: 19 jul. 2022.

ZHAO, Bo. Web scraping. **Encyclopedia of big data**, Springer Living ed. Cham, p. 1–3, 2017.

ZYTE. **Scrapy**. Disponível em: `https://scrapy.org/`. Acesso em: 18 jul. 2022.