UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUCAS MEDEIROS MARTINS

MONITORAÇÃO DA SANIDADE DE WEB SCRAPERS COM OPENTELEMETRY

CAMPINA GRANDE - PB

2024

ANDRÉ LUCAS MEDEIROS MARTINS

MONITORAÇÃO DA SANIDADE DE WEB SCRAPERS COM OPENTELEMETRY

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Eliane Cristina de Araújo

CAMPINA GRANDE - PB

2024

ANDRÉ LUCAS MEDEIROS MARTINS

MONITORAÇÃO DA SANIDADE DE WEB SCRAPERS COM OPENTELEMETRY

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Eliane Cristina de Araújo
Orientador – UASC/CEEI/UFCG

Dalton Dario Serey Guerrero
Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15 de MAIO de 2024.

CAMPINA GRANDE - PB

RESUMO

Web Scrapers são ferramentas para coletar dados de *sites web* sendo uma estratégia amplamente usada para fornecer diversos tipos de serviços. No entanto, quando se tem um ou mais serviços de extração de dados de *sites*, como saber a "saúde" desse funcionamento? Não existe um compromisso de compatibilidade entre *site* e *scraper*, pois, em geral, são desenvolvidos por equipes diferentes. Dessa forma, o *scraper* pode inadvertidamente deixar de funcionar devido a mudanças realizadas no *site*.

Neste trabalho foi desenvolvida uma estratégia usando *OpenTemeletry*, para emissão de métricas e rastreamentos de funcionamento para solucionar esta falta de visão da sanidade do *scraper*. Espera-se que ao final deste trabalho seja possível aplicar a solução na prática e, assim, ter um único local com as informações de funcionamento dos *Web Scrapers*.

MONITORING THE HEALTH OF WEB SCRAPERS WITH OPENTELEMETRY

ABSTRACT

Web Scrapers are tools for collecting data from websites, widely used to provide various types of services. However, when you have one or more data extraction services from websites, how do you know the "health" of their operation? There is no compatibility commitment between the site and the scraper because they are generally developed by different teams. Thus, the scraper may inadvertently fail to function due to changes made to the site. In this work, a strategy using OpenTelemetry was developed to emit metrics and operational traces to address this lack of insight into the scraper's health. It is expected that by the end of this work, it will be possible to apply the solution in practice and thus have a single location with information on the operation of Web Scrapers.

Monitoração da sanidade de Web Scrapers com OpenTelemetry

André Lucas Medeiros Martins
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 — 58.109-970 — Campina Grande — PB — Brasil
andre.martins@ccc.ufcg.edu.br

RESUMO

Web Scrapers são ferramentas para coletar dados de sites web sendo uma estratégia amplamente usada para fornecer diversos tipos de serviços. No entanto, quando se tem um ou mais serviços de extração de dados de sites, como saber a "saúde" desse funcionamento? Não existe um compromisso de compatibilidade entre site e scraper. Pois em geral, são desenvolvidos por equipes diferentes. Dessa forma, o scraper pode inadvertidamente deixar de funcionar devido a mudanças realizadas no site. Neste trabalho apresentamos e construímos uma estratégia usando OpenTemeletry, para emissão de métricas e rastreamentos de funcionamento para solucionar esta falta de visão da sanidade do scraper.

Palavras chave

Web Scraper, monitoramento, sanidade, Open Telemetry.

1. INTRODUÇÃO

"Se a programação é mágica, o web scraping é a prática dessa magia" (MITCHELL, 2019). Hoje essa frase faz total sentido e diversas empresas praticam essa abordagem, pois a *web* se move a partir de dados. Aqueles que têm mais informações e quanto mais atualizadas forem, se põe à frente no mercado. É comum, por exemplo, ver lojas que, buscando vender mais, mostram ao cliente o preço de seus concorrentes e comparam com o seu.

Os *web scrapers* (que se traduz e será usado em diante como raspadores), são motores para diversos tipos de produtos de *software*. Nesta técnica, as informações são extraídas a partir de um conjunto de passos automatizados que consultam um servidor *web*. Lá acessam o que lhes é desejável e, então, esse dado pode ser usado como for necessário (métricas, armazenamento, consulta, etc). No entanto, há fatores que podem preocupar quem consome o resultado das raspagens. Por exemplo, como sabemos se o conteúdo está válido e atualizado? E se tivermos um grande conjunto de *scrapers*? A preocupação do mal funcionamento cresce proporcionalmente com o número de raspadores, pois diariamente sites atualizam o conteúdo, mudam seu formato, deixam de existir, ficam fora de funcionamento e outros diversos motivos que podem impedir a coleta do dado. Sendo então a natureza do código de raspagem tão baseada no site e por não existir um compromisso de compatibilidade *site*, o algoritmo está suscetível a erros.

Assim, surge a necessidade de se ter um método de checar o funcionamento de um conjunto de *scrapers*.

2. FUNDAMENTAÇÃO

Nessa seção serão apresentados alguns conceitos e tecnologias que serão citados ao longo do trabalho.

2.1 Observabilidade

O conceito principal deste trabalho é a observabilidade. Ela nos permite compreender um sistema a partir do exterior e para isto usamos a instrumentação [2].

2.2 Instrumentação

É a estratégia de emitir dados de funcionamento enquanto o processo é executado [2]. Esses dados são chamados de telemetria. Durante o desenvolvimento do trabalho foi necessário instrumentar uma tecnologia e, com isso, conseguimos obter o monitoramento.

2.3 Telemetria

Refere-se aos dados emitidos por um sistema, sobre o seu comportamento. No *OpenTelemetry* esses dados estão na forma de métricas, *traces* (rastreamentos) e *logs* (registros de funcionamento) [2].

2.4 OpenTelemetry

É um projeto de código aberto projetado para criar e gerenciar dados de telemetria [2]. Além disso, o projeto possui diversos componentes que agregam a esta funcionalidade principal, um deles é o Collector – um proxy que recebe, processa e exporta dados de telemetria.

3. OBJETIVOS

3.1 Objetivos gerais

Construir um meio de observar o funcionamento e sanidade de um conjunto de *Web Scrapers*, de forma que a coleta das informações para a construção desse relatório não cause impactos no bom funcionamento dos *scrapers*.

3.2 Objetivos específicos

Para alcançar o objetivo principal do trabalho, foi elaborado um conjunto de atividades:

- *Buscar avanços*

Buscar avanços sobre o assunto, usando as tecnologias citadas;

- *Coletar Web Scrapers*

Coletar 5 *Web Scrapers* para usá-los como amostra;

- *Instrumentação*

Instrumentar os raspadores com o *OpenTelemetry* para coleta de dados de funcionamento, para evitar a necessidade de alterar o código do raspador;

- *Gerar relatório com dados coletados*

Tendo Web Scrapers sido instrumentados e executados, coletar os dados e gerar relatório de sua sanidade;

4. ARQUITETURA E TECNOLOGIAS

O sistema desenvolvido segue a estratégia de micro serviços. Em resumo, temos 3 serviços: Dashboard de Monitoramento, Serviço

de *Scrapers* e Coletor *OpenTelemetry*. Abaixo, cada parte é explicada e suas tecnologias detalhadas.

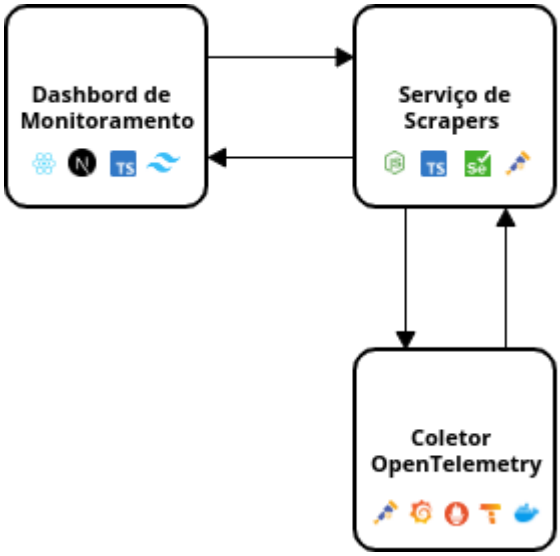

Figura 1: Descrição da arquitetura e tecnologias presentes

4.1 *Dashboard* de Monitoramento

Este serviço é responsável por interagir com o usuário, possibilitar a entrada de dados e gerar relatórios dos raspadores e suas execuções. Neste conseguimos ver métricas e gráficos que são o objetivo final da aplicação. Nesta seção, serão apresentadas as tecnologias utilizadas para a construção deste serviço.

- *React*

Uma biblioteca Javascript para construção de componentes da interface.

- *Next.js*

Um *framework* do React que facilita a configuração de todo o projeto e torna mais rápido os processos de criar o roteamento de páginas, construí-las e publicá-las.

- *Tailwind*

Uma api para facilitar a estilização dos componentes, o *Tailwind* torna o uso de estilos CSS muito mais rápido e intuitivo.

4.2 Serviço de *Scrapers*

O principal artefato do sistema são os *scraper*, então existe um serviço responsável somente por manter, executar e observar o seu funcionamento. Este, age em um papel de servidor, e irá prover as informações e as ações necessárias Dashboard de Monitoramento

Reactpara que o Dashboard funcione. Abaixo, estão descritas as tecnologias usadas.

- *Node.js*

Com a conceito de MVC (Model, View e Controller), usamos o Node.js para criar um serviço Back-end que vai servir de intermédio entre os três serviços. Com ele, construímos *endpoints* que recebe e envia informações para o *Dashboard*

- *Selenium*

Um raspador *web* é uma automação de um browser (também chamado de navegador). Então, para que as sequências de passos de um *scraper* que manipulam um navegador possam ser executadas, o Selenium provê as ferramentas.

- *OpenTelemetry API*

Para que seja possível observar o funcionamento do *web scraper*, criamos então um envelopamento do Selenium e usamos o *Otel* (abreviação de *OpenTelemetry*, que será usada deste ponto em diante) para criar as observações e enviar para o serviço de Coletor. Esta API torna fácil a criação de métricas e rastreamentos, elementos que são usados como base para o relatório do *Dashboard*.

4.3 Coletor *OpenTelemetry*

- *OpenTelemetry Collector*

Um componente do projeto OTel que recebe, processa e exporta os dados de telemetria (métricas, *traces*) e que, neste sistema, são gerados pelo Serviço de *Scrapers*.

- *Prometheus e Grafana Tempo*

A tecnologia anterior não possui em suas funções o armazenamento, dessa forma, utilizamos fontes de dados para que desempenhem este papel e posteriormente possam ser consultados para a geração de relatórios. Essas fontes funcionam juntamente com o Grafana

- *Grafana*

Por sua vez, o Grafana, é uma ferramenta que busca facilitar a comunicação entre o Serviço de Scrapers e o Coletor *OpenTelemetry*. seria possível também construir um relatório dos dados recebidos dentro desta tecnologia, no entanto, não seria tão customizável quando um próprio serviço de Dashboard.

- *Docker e Docker Compose*

Por fim, para facilitar toda essa construção do coletor e sua comunicação, usamos a tecnologia de conteinerização do Docker e estratégia do Docker Compose para executar tudo isto com um único comando

5. SOLUÇÃO

A solução é baseada em uma arquitetura cliente-servidor. O Serviço de *Scrapers* é responsável por ter a lógica da aplicação e a no Dashboard, incluímos uma interface para que o usuário possa interagir e customizar a aplicação.

O diagrama de casos de uso UML, da figura 2, apresenta as possibilidades de interação do usuário com o sistema ao exercitar as funcionalidades disponíveis.

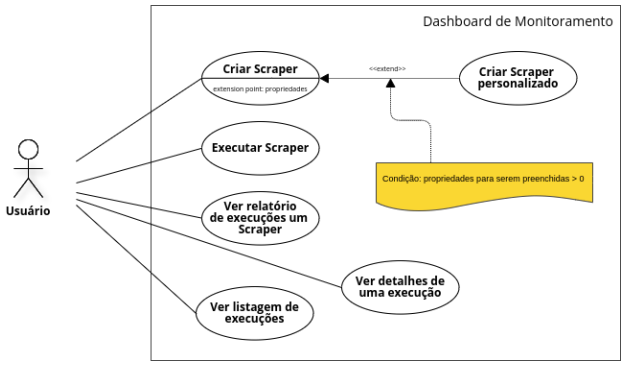

Figura 2: Diagrama de Caso de Uso

A seguir, as funcionalidades serão detalhadas considerando os casos de uso apresentados.

5.1 Criar *Scraper* e *Criar Scraper personalizado*

O Serviço de *Scrapers* é o local onde os raspadores são implementados usando o Selenium. Para a sua criação deve ser usada a *interface* chamada *Scraper*, conforme Figura 3.

```
scraper-service > src > models > TS Scraper.ts > ...
1  import { WebDriver } from "selenium-webdriver";
2
3  import InstrumentedWebDriver from "../wrappers/selenium";
4
5  export interface ScraperProps {
6    [key: string]: string;
7  }
8
9  export default interface Scraper {
10   name: string;
11   description: string;
12
13   run(driver: WebDriver | InstrumentedWebDriver,props: ScraperProps): Promise<void>;
14
15   necessaryProps: () => ScraperProps;
16 }
```

Figura 3: Interface de um Scraper

Na Figura 3 é possível ver o método "run". Nele estão as instruções de funcionamento do raspador. Além disso, o método recebe como propriedades um *driver* que é a api de interação com o *browser*. Este *driver* recebe uma instância pura, vinda diretamente do Selenium ou uma versão que foi instrumentada com o *OpenTelemetry* para fins de monitoramento. Além do *driver*, o método irá receber uma propriedade "props" que será detalhada mais à frente.

No serviço de Dashboard, vamos lidar diretamente com esses *Scraper* que implementam a *interface*.

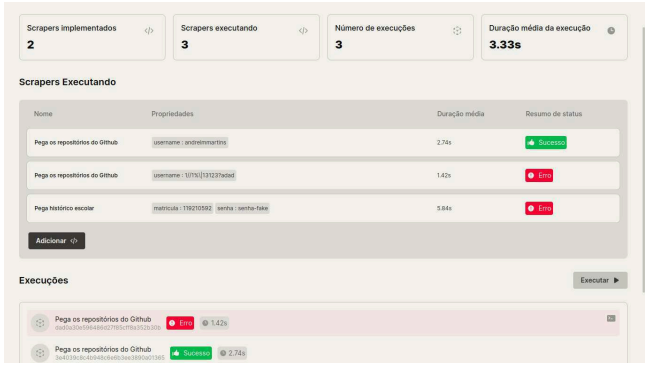

Figura 4: Listagem e criação de *Scrapers*

Podemos observar no Dashboard (Figura 4) uma listagem e um botão "Adicionar", no entanto o funcionamento desta seção não é a mesma citada anteriormente sobre implementar a interface. Aqui são criados e gerenciados os *Scrapers* que foram selecionados para serem observados usando *OpenTelemetry*. A execução deles alimenta o dashboard. Ao clicar no botão "Adicionar", é possível criar uma instância de execução e então são providas as informações da propriedade "props" presente no método "run" da interface.

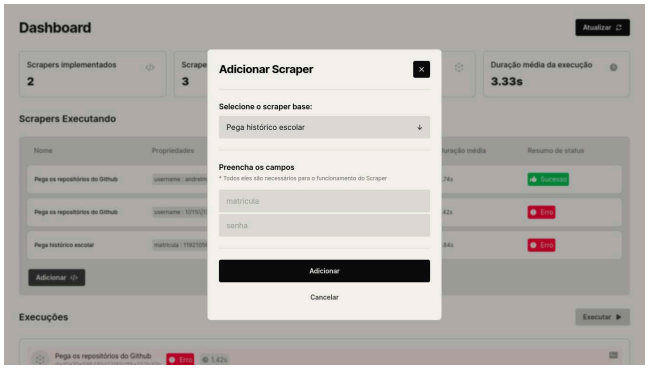

Figura 5: Modal de adição de *Scraper*

Ao clicar no botão, é exibido uma caixa de diálogo modal. Na figura 5, a base para a execução pode ser selecionada, neste caso o raspador escolhido foi "Pega histórico escolar". Então, com a comunicação com o Serviço de *Scrapers* é possível ver quais são as propriedades que existem e as que precisam ser preenchidas para a execução. Com isso, o formulário de preenchimento é gerado. Clicando no botão de "Adicionar", a instância de execução estará disponível para ser executada a qualquer momento.

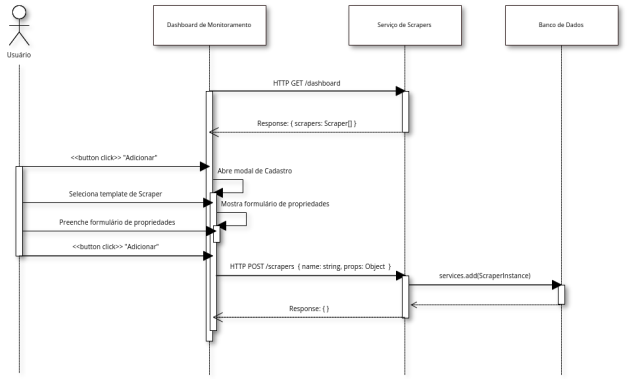

Figura 6: Diagrama de sequência da funcionalidade de criação de *scraper*

5.2 Executar Scraper e Ver relatório de execução de um Scraper

Na figura 4, é possível ver um botão "Executar". Ao clicar nesse botão, cada uma das instâncias será executada. Logo após isso, todo o relatório é atualizado com as novas informações.

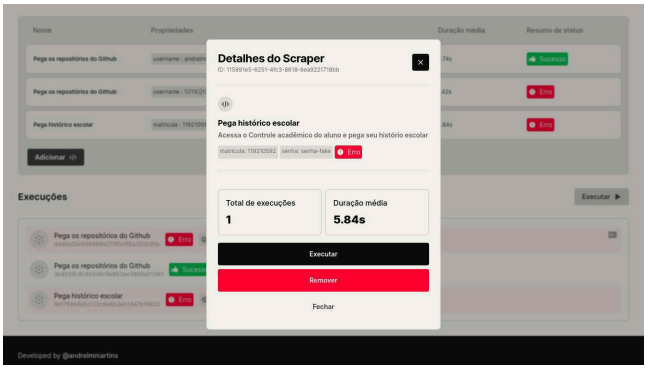

Figura 7: Detalhes de um *scraper*

Ademais, uma outra funcionalidade presente é a de detalhar um *scraper*. Aqui, são reunidas todas as execuções de um raspador e com essas informações é montado um breve relatório. Esse relatório mostra informações como: o número de vezes que foi executado, a duração média de cada execução e um resumo dos *status* de cada execução.

5.3 Ver detalhes de uma execução

Para se ter um relatório mais completo do funcionamento de um *scraper* é preciso ver mais detalhes de uma execução. Então, é possível clicar em uma execução e uma caixa de diálogo irá abrir com o detalhamento.

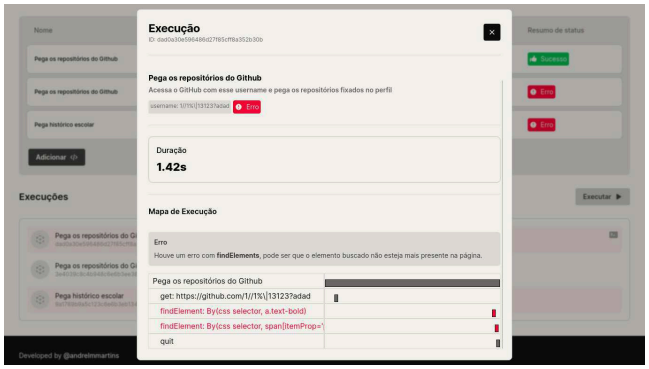

Figura 8: Detalhes de uma execução

Na figura 8, estão presentes as informações do raspador que foi executado e, abaixo, um relatório com duração, status, um mapa de atividades do *scraper* em formato de diagrama de Gantt, e se houver algum erro durante a execução será mostrado ao usuário uma possível explicação do erro. Dessa forma, o sistema notifica e auxilia o usuário a manter o raspador funcionando.

5.4 Ver listagem de execuções

O Dashboard também apresenta uma listagem de execuções. Veja a Figura 9. Essas são as principais informações do relatório gerado e nelas está o diferencial da aplicação.

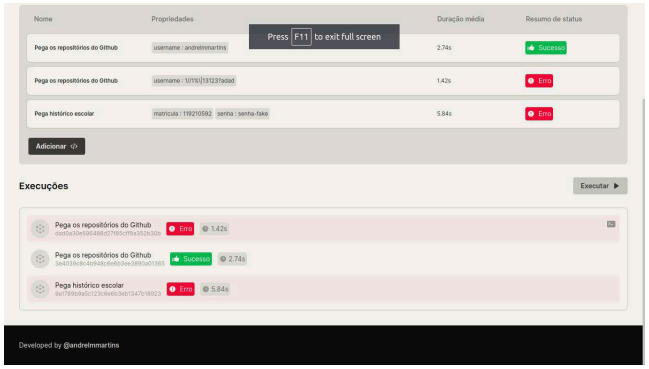

Figura 9: Listagem e criação de *Scrapers*

Uma execução é um objeto presente em todo o sistema. Começando por sua criação no Serviço de *Scrapers*, quando o raspador recebe um *driver* instrumentado com *OpenTelemetry*. A cada passo que acontece no *browser,* traces (rastreamentos) e/ou métricas vão sendo criadas, ao fim da execução, são enviadas para o coletor *OpenTelemetry*. Dessa forma, quando abrimos o Dashboard, solicitamos ao servidor as informações de listagem, que são buscadas no Coletor, interpretadas pelo Serviço de Scraper e enviadas para a interface do usuário.

Uma execução tem como informações básicas: a sua duração, o *scraper* a qual se refere e um *status* – seja de erro, sucesso ou indeterminado, quando não se sabe a causa da falha.

6. PERFORMANCE

É preciso considerar o custo adicionado ao *scraper* por ter sido instrumentado. Será que os tempos de duração coletados são mesmo reais? Ou o *OpenTelemetry* estaria onerando a execução do *scraper*?

Um ponto positivo do *OTel* é ser simples em seu uso. Ao instrumentar o *driver* do Selenium, envelopamos as funções com pequenos trechos de código. Passos esses, que não geram um peso considerável à execução. Além disso, a comunicação com o Coletor para enviar a telemetria acontece posteriormente à execução do *scraper*. Dessa forma, não existe impacto considerável no funcionamento do scraper se compararmos a sua versão sem instrumentação.

7. CONCLUSÃO E TRABALHOS FUTUROS

Com esse trabalho, propomos uma solução, que usa *OpenTelemetry* para visualizar o real estado de funcionamento de *Web Scrapers.*

A missão do *OTel* é tornar os sistemas observáveis de forma padronizada e eficaz. Sendo um projeto de código aberto, essa tecnologia está em constante desenvolvimento e, com isso, diversos aplicativos e serviços já possuem nativamente o monitoramento com *Otel*, como: *Kubernetes*, *Next.js*, entre outros.

Trabalhos futuros podem ser conduzidos para buscar ainda mais melhorias para este. Com avanços diários do *OpenTelemetry*, podemos encontrar na comunidade uma instrumentação completa e nativa do Selenium. O que viabiliza diversas *features* para a aplicação atual. Além disso, a comparação com mais sistemas de monitoramento, traria um relatório ainda mais completo para o sistema, com gráficos e métricas mais específicas.

Finalmente, este trabalho contribui com uma arquitetura para o monitoramento de *Web Scrapers,* oferecendo uma solução de fácil adaptação, sem trazer dificuldades aos programadores ou impactos no desempenho do funcionamento do scraper.

8. REFERÊNCIAS

[1] MITCHELL, Ryan. Web Scraping com Python: Coletando Mais Dados da web Moderna. 2. ed. São Paulo: Novatec Editora, 2019.

[2] The OpenTelemetry Authors. OpenTelemetry. 2023. Disponível em: <https://opentelemetry.io/>. Acesso em: 8 nov. 2023.

[3] Vercel, Inc. Next.js. 2024. Disponível em: <https://nextjs.org/>. Acesso em: 22 abr. 2024.

[4] Tailwind CSS. 2024. Disponível em: <https://tailwindcss.com/>. Acesso em: 28 abr. 2024.

[5] Meta Open Source. React. 2024. Disponível em: <https://react.dev/>. Acesso em: 28 abr. 2024.

[6] OpenJS Foundation. Node.js. 2024. Disponível em: <https://nodejs.org/>. Acesso em: 28 abr. 2024.

[7] Grafana Labs. Grafana. 2024. Disponível em: <https://grafana.com/>. Acesso em: 28 abr. 2024.

[8] Grafana Labs. Grafana Tempo. 2024. Disponível em: <https://grafana.com/oss/tempo/>. Acesso em: 28 abr. 2024.

[9] Prometheus Authors. Prometheus. 2024. Disponível em: <https://prometheus.io/>. Acesso em: 28 abr. 2024.

[10] Docker. 2024. Disponível em: <https://www.docker.com/>. Acesso em: 28 abr. 2024.