



UEPB

**UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - GOVERNADOR ANTÔNIO MARIZ
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

LUIZ EDUARDO OLIVEIRA DE ARAÚJO

**ANÁLISE COMPARATIVA DO TAURI E ELECTRON: UM ESTUDO DE DOIS
FRAMEWORKS PARA APLICAÇÕES DE DESKTOP MULTIPLATAFORMA**

**PATOS - PB
2023**

LUIZ EDUARDO OLIVEIRA DE ARAÚJO

**ANÁLISE COMPARATIVA DO TAURI E ELECTRON: UM ESTUDO DE DOIS
FRAMEWORKS PARA APLICAÇÕES DE DESKTOP MULTIPLATAFORMA**

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Bacharel em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador: Profa. Giovanna Trigueiro de Almeida Araujo

**PATOS - PB
2023**

É expressamente proibido a comercialização deste documento, tanto na forma impressa como eletrônica. Sua reprodução total ou parcial é permitida exclusivamente para fins acadêmicos e científicos, desde que na reprodução figure a identificação do autor, título, instituição e ano do trabalho.

A663a Araujo, Luiz Eduardo Oliveira de.
Análise comparativa do Tauri e Electron [manuscrito] : um estudo de dois frameworks para aplicações de desktop multiplataforma / Luiz Eduardo Oliveira de Araujo. - 2023.
42 p. : il. colorido.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2023.

"Orientação : Profa. Esp. Giovanna Trigueiro de Almeida Araujo, Coordenação do Curso de Computação - CCEA. "

1. Engenharia de software. 2. Frameworks. 3. Cross-platform. I. Título

21. ed. CDD 005.1

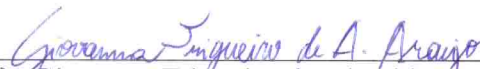
LUIZ EDUARDO OLIVEIRA DE ARAÚJO

**ANÁLISE COMPARATIVA DO TAURI E ELECTRON: UM ESTUDO DE DOIS
FRAMEWORKS PARA APLICAÇÕES DE DESKTOP MULTIPLATAFORMA.**

Trabalho de Conclusão de Curso apresentado ao
Curso de Bacharelado em Ciência da
Computação da Universidade Estadual da
Paraíba, em cumprimento à exigência para
obtenção do grau de Bacharel em Ciência da
Computação.

Aprovado em 30 de novembro de 2023

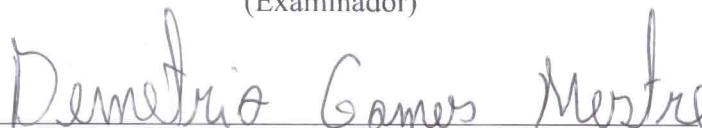
BANCA EXAMINADORA



Prof. Giovanna Trigueiro de Almeida Araújo
(Orientadora)



Prof. Pablo Ribeiro Suárez
(Examinador)



Prof. Demetrio Gomes Mestre
(Examinador)

À minha mãe Sônia (*in memoriam*), que
sempre esteve ao meu lado, guiando-me
com amor e sabedoria, DEDICO

AGRADECIMENTOS

Ao meu pai Raimundo e meu irmão Lucas, que me incentivaram nos momentos difíceis e de dúvida.

Ao professor Pablo Roberto pelo auxílio e paciência durante as fases iniciais do projeto de conclusão.

À professora Giovanna, desejo expressar meus sinceros agradecimentos por sua acolhida calorosa e valioso auxílio como orientadora neste trabalho de conclusão de curso.

Aos professores do Curso de Ciência da Computação deste campus, em especial, Pablo Suarez, Fábio Júnior, Angélica Félix, Jucelio Soares, que contribuíram com conhecimentos e inspiração ao longo dessa jornada.

Aos funcionários da UEPB, pela presteza e atendimento sempre que necessário.

Agradeço aos meus colegas de classe, em especial Caio, Patrick e Natan, que compartilharam esta jornada comigo, tornando todo o percurso mais leve. Sua amizade e apoio foram inestimáveis.

Por fim, expresso minha gratidão a todos que, de maneira direta ou indireta, contribuíram para minha formação. Muito obrigado!

RESUMO

No dinâmico cenário do desenvolvimento de *software*, a escolha cuidadosa do *framework* desempenha um papel crucial no êxito de um projeto desde o início. Esse ponto torna-se ainda mais relevante no contexto do desenvolvimento *cross-platform*, onde a seleção criteriosa de um *framework* permite a criação de aplicações que podem operar de maneira fluida em diversos sistemas operacionais. Tal abordagem elimina a necessidade de manter equipes separadas dedicadas a cada plataforma, promovendo eficiência e simplificando o processo de desenvolvimento. Tendo em vista tal perspectiva, este trabalho se propõe a desenvolver duas soluções parecidas usando diferentes *frameworks cross-platform*, e conseguir dados qualitativos e quantitativos sobre os mesmos. As aplicações foram desenvolvidas utilizando Electron e Tauri, e a coleta de dados foi realizada por meio de aplicações de *profiling*. Através dos dados coletados pelas ferramentas, pôde-se perceber que ambos *frameworks* apresentam pontos negativos e positivos, onde o Electron por ser mais antigo consegue entregar aplicações mais robustas e com mais integrações com os sistemas operacionais. Por outro lado, o Tauri ainda está se estabelecendo no mercado, porém já consegue entregar um melhor desempenho, sendo uma escolha mais ágil e eficiente para projetos de menor escala e que requerem otimização de recursos.

Palavras-Chave: Engenharia de software; Frameworks; Cross-platform.

ABSTRACT

In the dynamic landscape of software development, the careful choice of a framework plays a crucial role in the success of a project from the start. This becomes even more relevant in the context of cross-platform development, where the meticulous selection of a framework enables the creation of applications that can operate seamlessly across various operating systems. Such an approach eliminates the need to maintain separate teams dedicated to each platform, promoting efficiency and simplifying the development process. With this perspective in mind, this work aims to develop two similar solutions using different cross-platform frameworks and gather qualitative and quantitative data about them. The applications were built using Electron and Tauri, and data collection was carried out through profiling applications. Based on the data collected by these tools, it can be observed that both frameworks have their strengths and weaknesses. The more old Electron, for instance, excels in delivering robust applications with enhanced integrations with operating systems. On the other hand, Tauri is still establishing its presence in the market but already demonstrates superior performance, making it a more agile and efficient choice for smaller-scale projects that require resource optimization.

Keywords: Software Engineering; Frameworks; Cross-platform.

LISTA DE FIGURAS

Figura 1	– Desenvolvimento <i>cross-platform</i> usando JavaScript	15
Figura 2	– Tela inicial do <i>Instruments</i>	17
Figura 3	– XCode <i>Instruments</i> monitorando a aplicação Firefox	18
Figura 4	– Profiler do Visual Studio	19
Figura 5	– Funcionamento dos processos de renderização do Electron	22
Figura 6	– Comunicação entre Processos no Electron usando IPC	22
Figura 7	– Representação simplificada de uma aplicação Tauri	23
Figura 8	– Interface principal da aplicação DexTop	26
Figura 9	– Visualização detalhada de um Pokémon	27
Figura 10	– Exemplo de uma chamada IPC utilizando Electron	28
Figura 11	– Exemplo do Electron gerenciando chamadas IPC	29
Figura 12	– Código Rust que permite o Rust gerenciar chamadas IPC	29
Figura 13	– Chamada IPC utilizando a biblioteca do Tauri no JavaScript	30
Figura 14	– Utilização de CPU em porcentagem durante a execução dos aplicativos	32
Figura 15	– Utilização de memória em MegaBytes durante a execução dos aplicativos	33
Figura 16	– Tamanho final dos aplicativos em MegaBytes	34
Figura 17	– Duração da compilação dos aplicativos em segundos	36

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AWS	Amazon Web Services
CERN	European Organization for Nuclear Research
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Mark-up Language
IDE	Integrated Development Environment
IPC	InterProcess Communication
MB	Megabyte
OS	Operating System

SUMÁRIO

1	INTRODUÇÃO.....	10
1.1	Problemática.....	10
1.2	Relevância.....	11
1.3	Objetivos.....	11
1.3.1	Objetivo Geral.....	11
1.3.2	Objetivos Específicos.....	11
1.4	Metodologia.....	12
1.5	Estrutura do trabalho.....	13
2	REFERENCIAL TEÓRICO.....	13
2.1	Desenvolvimento <i>cross-platform</i>	13
2.2	Ferramentas de <i>profiling</i>	15
2.2.1	XCode Instruments.....	17
2.2.2	Visual Studio.....	18
2.3	Tecnologias selecionadas.....	19
2.3.1	HTML.....	19
2.3.2	React.....	19
2.3.3	Chromium.....	20
2.3.4	Electron.....	20
2.3.5	Tauri.....	22
2.3.5.1	WebView.....	23
2.3.6	JavaScript.....	23
2.3.7	Rust.....	24
3	ASPECTOS DE IMPLEMENTAÇÃO.....	24
3.1	DexTop.....	24
3.2	Comunicação entre processos.....	26
3.3	Dados coletados.....	29
3.3.1	Procedimentos de coleta.....	29
3.3.2	Avaliação de Desempenho: Utilização de CPU e Memória.....	30
3.3.3	Avaliação de Desempenho: Tamanho dos aplicativos e duração da compilação.....	32
3.3.4	Análise Qualitativa: Tauri e Electron em Perspectiva.....	35
4	CONCLUSÃO.....	35
4.1	Principais contribuições.....	36
4.2	Limitações da pesquisa.....	36
4.3	Trabalhos futuros.....	37
	REFERÊNCIAS.....	38

1 INTRODUÇÃO

A evolução das tecnologias de desenvolvimento tem desempenhado um papel fundamental possibilitando a criação de boas interações para os usuários e de alto desempenho em uma variedade de plataformas. Com base nisso, a escolha do *framework* adequado torna-se crucial para os desenvolvedores que desejam criar aplicações de *cross-platform* eficientes e com vários recursos nativos. Este estudo apresenta uma análise comparativa abrangente entre dois populares *frameworks*: Tauri e Electron.

Ambos *frameworks* oferecem ao desenvolvedor uma base sólida e recursos poderosos para criação de *UI's* chamativas e funcionais. O trabalho atual visa examinar e comparar os dois *frameworks* em termos de desempenho, ecossistema de desenvolvimento, entre outros pontos que serão citados no decorrer do trabalho.

Um dos benefícios significativos de optar por usar um *framework cross-platform* como Tauri e Electron, é a redução do ciclo e do custo de desenvolvimento. Como afirmado por Rehman (2018), os clientes demandam que *softwares* estejam disponíveis em vários ecossistemas diferentes, o que faz com que a criação de aplicações nativas seja demorada e custosa. Ainda segundo Rehman, ao optar por um *framework cross-platform* os desenvolvedores podem escrever apenas uma única base de código que pode ser compilada para vários sistemas operacionais diferentes.

1.1 Problemática

Embora *frameworks cross-platform* ofereçam muitas vantagens, é importante reconhecer que também podem surgir desafios e limitações ao optar por essa abordagem.

Como analisado por Heitkötter et al. (2013), aplicações *cross-platform* são muitas vezes mais lentas que a aplicações nativas, por exemplo, a abstração necessária para suportar várias plataformas pode resultar em um desempenho ligeiramente inferior em comparação com aplicativos nativos. Além disso, a dependência de camadas adicionais de software introduzidas pelos *frameworks* pode resultar em um maior consumo de recursos do sistema.

É fundamental, portanto, avaliar cuidadosamente os requisitos do projeto e considerar o equilíbrio entre a vantagem de alcançar várias plataformas e as necessidades específicas de desempenho e recursos do aplicativo.

1.2 Relevância

Como estabelecido por Alkhars et al. (2016), “Os desenvolvedores devem prestar bastante atenção na hora de escolher um *framework cross-platform* no começo do ciclo de desenvolvimento”¹.

Tendo em vista os pontos estabelecidos, esse trabalho não possui o intuito de dizer qual *framework* é melhor, mas sim ajudar os desenvolvedores a escolherem a melhor ferramenta para suas aplicações, mostrando as diferenças, pontos fortes e pontos fracos de cada *framework*.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver duas aplicações utilizando diferentes frameworks e realizar uma análise comparativa abrangente entre eles, mostrando os pontos fortes e fracos de cada um.

1.3.2 Objetivos Específicos

- Desenvolver para ambos *frameworks* uma aplicação;
- Coletar dados relevantes das aplicações criadas;
- Avaliar e comparar dados coletados;

¹ “The programmer should pay enough attention in choosing proper cross-platform technology at the beginning; because it is nearly impossible to replace”

1.4 Metodologia

A metodologia utilizada neste trabalho é o Método da Diferença, que é um tipo de método comparativo que foi proposto por Mill (1882), que tem como objetivo analisar as diferenças entre objetos semelhantes.²

Como mencionado anteriormente, os objetos a serem comparados são aplicativos desktop, que foram desenvolvidos especificamente neste trabalho. Foram criadas duas aplicações utilizando cada um dos *frameworks*. Cada aplicação terá uma interface gráfica funcional e foi capaz de realizar um teste de estresse.

No design será usado o Figma³, uma ferramenta de prototipagem de interfaces. Com intuito de facilitar o desenvolvimento do *front-end* de ambas aplicações, será usado o React⁴. Afim de possibilitar que o *front-end* criado esteja disponível para *desktops* serão usados os *frameworks* ElectronJS⁵ e Tauri⁶.

Após ambas aplicações estarem prontas, a realização da coleta de dados das mesmas será realizada utilizando um conjunto de ferramentas específicas de cada um dos sistemas operacionais alvo. No ambiente Windows, a coleta será realizada por meio da ferramenta *Visual Studio*. Já no ambiente macOS, os dados serão coletados utilizando a ferramenta *Instruments*.

Para avaliar o desempenho, a construção e outras características de cada *framework*, serão coletados e analisados diversos dados. Isso inclui a verificação da extensão e qualidade da documentação, focando também na comunidade de cada um dos *frameworks*, visando avaliar a facilidade de compreensão, implementação e o quão ativa é a comunidade em volta de cada *framework*. Será verificado o tamanho dos instaladores das aplicações desenvolvidas com cada *framework*, permitindo a comparação da eficiência na distribuição dos aplicativos. Tendo em vista os dados que serão avaliados, a abordagem deste trabalho é qualitativa e quantitativa.

Também vai ser monitorado o consumo de CPU gerado pela aplicação, proporcionando insights sobre a eficiência do código e otimizações necessárias.

² “(...) The two instances which are to be compared with one another must be exactly similar, in all circumstances except the one which we are attempting to investigate (...)”

³ <https://www.figma.com>

⁴ <https://react.dev>

⁵ <https://www.electronjs.org>

⁶ <https://tauri.app>

Cada aplicação também passará por uma avaliação do consumo de memória, auxiliando na identificação de possíveis vazamentos ou áreas de melhoria.

Com todos esses dados será feita uma comparação usando o método da diferença para apontar quais são os pontos fortes e fracos de cada *framework*.

Será medido o tempo necessário para a aplicação iniciar, fornecendo informações sobre a eficiência na inicialização e possíveis pontos de otimização. Por fim, será feita uma avaliação sobre a compatibilidade e disponibilidade das API's dos sistemas operacionais nos *frameworks*, garantindo a integração adequada com o ambiente do usuário final.

1.5 Estrutura do trabalho

O trabalho encontra-se composto pelo capítulo 2, alusivo ao referencial teórico, onde são explorados os conceitos referentes a metodologia e as ferramentas utilizadas no desenvolvimento e coleta de dados.

O desenvolvimento está descrito no terceiro capítulo, onde são desenvolvidos os aplicativos, e com as aplicações desenvolvidas, o capítulo passa a demonstrar os métodos de coleta e dados que foram capturados. Esses dados serão usados no capítulo final, onde são feitas algumas considerações finais, limitações do trabalho, contribuições e por fim possíveis trabalhos futuros.

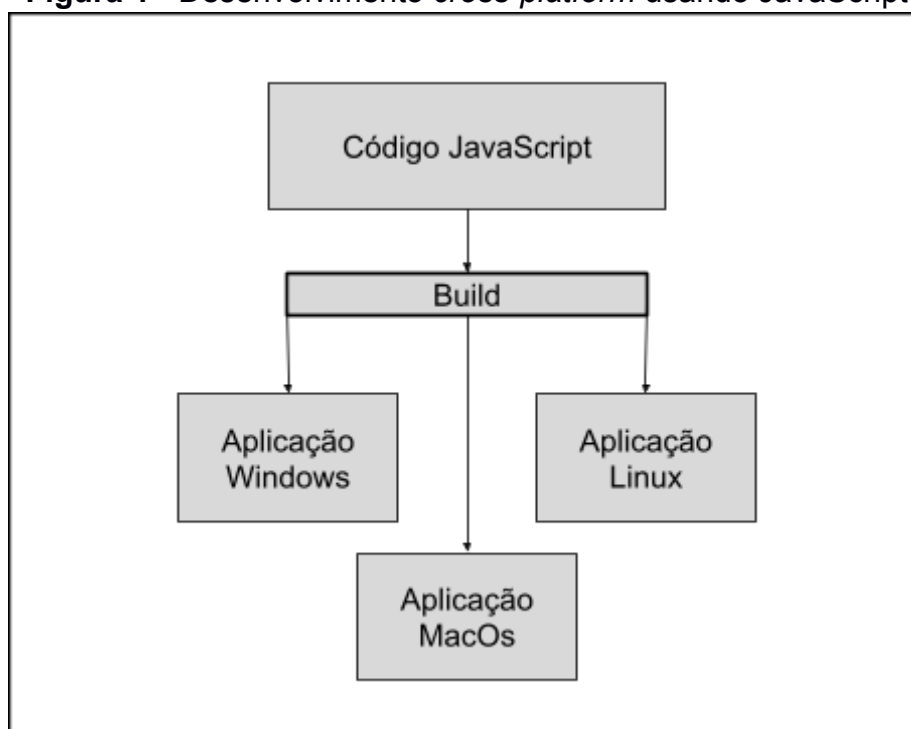
2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os conceitos e definições que esse trabalho tem como base. As seções estão dispostas da seguinte forma: Seção 2.1 apresentará um pouco sobre o desenvolvimento *cross-platform*. A seção 2.2 falará um pouco sobre ferramentas de *profiling* bem como apresentar as ferramentas usadas. Por fim, a seção 2.3 apresenta um pouco sobre cada tecnologia envolvida no desenvolvimento das aplicações.

2.1 Desenvolvimento *cross-platform*

Desenvolvimento *cross-platform* se refere a criação de aplicações que são compatíveis com vários sistemas operacionais diferentes, como Windows, Linux e Mac no desktop, ou no mobile, Android e iOS. O conceito de *cross-platform* não é novo, e existem várias opções de ferramentas que conseguem desenvolver aplicações para diferentes sistemas operacionais, alguns exemplos de ferramentas que já existem a muito tempo são o JavaFX, RAD Studio (Delphi) e QT. Outras ferramentas mais novas e populares são o Flutter, React Native, Electron e Tauri.

A Figura 1 ilustra o conceito do desenvolvimento *cross-platform*, destacando a capacidade de criar aplicações para diferentes sistemas operacionais a partir de um único código-fonte, na imagem, o código JavaScript é apresentado como a base unificada.

Figura 1 - Desenvolvimento *cross-platform* usando JavaScript

Fonte: Elaborado pelo autor (2023)

Muitas empresas de tecnologia costumam optar por soluções *cross-platform*, seja para desktop ou mobile. Para Shevtsiv et al. (2020), equipes de desenvolvimento podem ser diminuídas, trazendo benefícios tanto para empresa quanto para o cliente.

Para a empresa, os custos operacionais são reduzidos, existe uma facilitação e simplificação na comunicação e com menos vozes envolvidas nos projetos, a tomada de decisões pode ser feita de forma mais ágil e eficiente. Já para o cliente acaba sendo mais confortável comunicar-se com apenas uma pessoa, em vez de precisar explicar a dois desenvolvedores como uma tela deve funcionar. Também é possível estabelecer uma relação de confiança entre o desenvolvedor e o cliente.

Abordagens *cross-platform* também oferecem aos desenvolvedores a vantagem da reutilização de conhecimentos. Por exemplo, é possível utilizar de conhecimentos de desenvolvimento web como JavaScript, HTML e CSS para desenvolver interfaces de aplicativos Desktop e Mobile. Essa sinergia permite que os desenvolvedores explorem expertises prévias no desenvolvimento de interfaces modernas.

2.2 Ferramentas de *profiling*

Para a AWS, o *profiling* de aplicações desempenha um papel fundamental na avaliação do desempenho, eficiência e usabilidade do *software*, além de identificar as partes do sistema com maior custo computacional: “Algumas soluções de *profiling* possibilitam a visualização em tempo real de diversos aspectos de um *software*, permitindo um entendimento detalhado de sua operação”⁷.

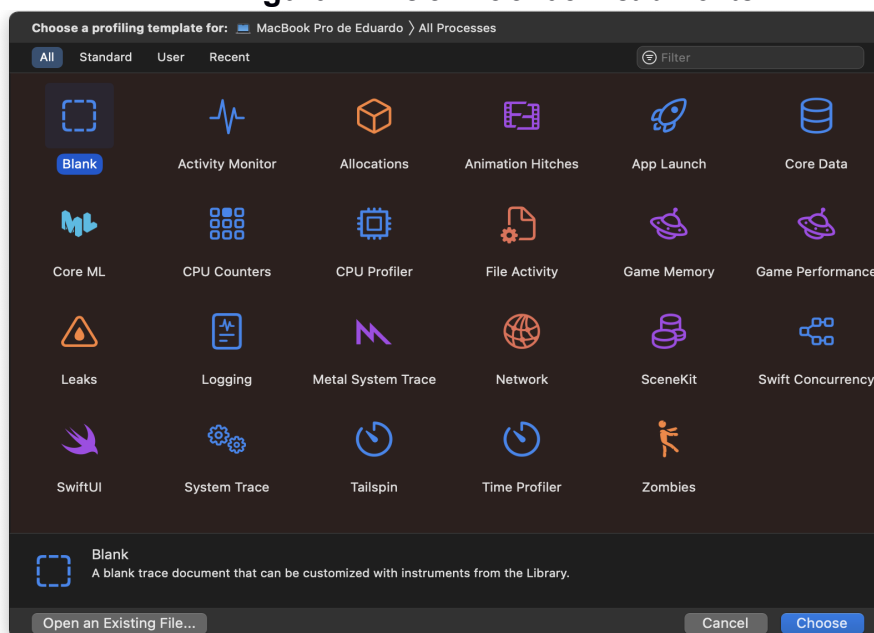
Ao analisar minuciosamente o funcionamento do programa dentro dessas soluções, é possível identificar gargalos, ineficiências e áreas que podem ser melhoradas. O objetivo do *profiling* é garantir que as aplicações operem de forma fluida, econômica e responsiva, oferecendo uma experiência agradável e eficaz aos usuários. (GOOGLE, 2023)

É importante mencionar que nem todas as soluções têm a capacidade de abranger todos os tipos de aplicativos. Tomemos, por exemplo, as aplicações desenvolvidas em C#: elas podem ser examinadas minuciosamente com o auxílio do *Instruments* da Apple e do Visual Studio, proporcionando uma análise completa. No entanto, quando tratamos de aplicações criadas em outras linguagens e *frameworks*, como o Tauri, essa abrangência nem sempre é alcançada no processo de *profiling*, no Visual Studio por exemplo, não é possível mensurar o uso de memória de uma aplicação Tauri.

2.2.1 XCode *Instruments*

O XCode da Apple disponibiliza uma ferramenta chamada *Instruments*, que possibilita a análise de vários aspectos de uma aplicação iOS e macOS, estas aplicações podem estar sendo executadas no Simulador ou em um dispositivo real. As funcionalidades do *Instruments* coletam dados do aplicativo em execução e apresentam esse dados em uma exibição gráfica com uma linha de tempo, quando necessário. Os desenvolvedores têm a possibilidade de escolher entre uma gama de modelos predefinidos ou até mesmo elaborar seus próprios modelos de *profiling*. (Apple, 2019)

⁷ “Application profiling solutions enable the discovery of resources, baseline application performance, and visualization of component interaction through flow maps built on real-time data”

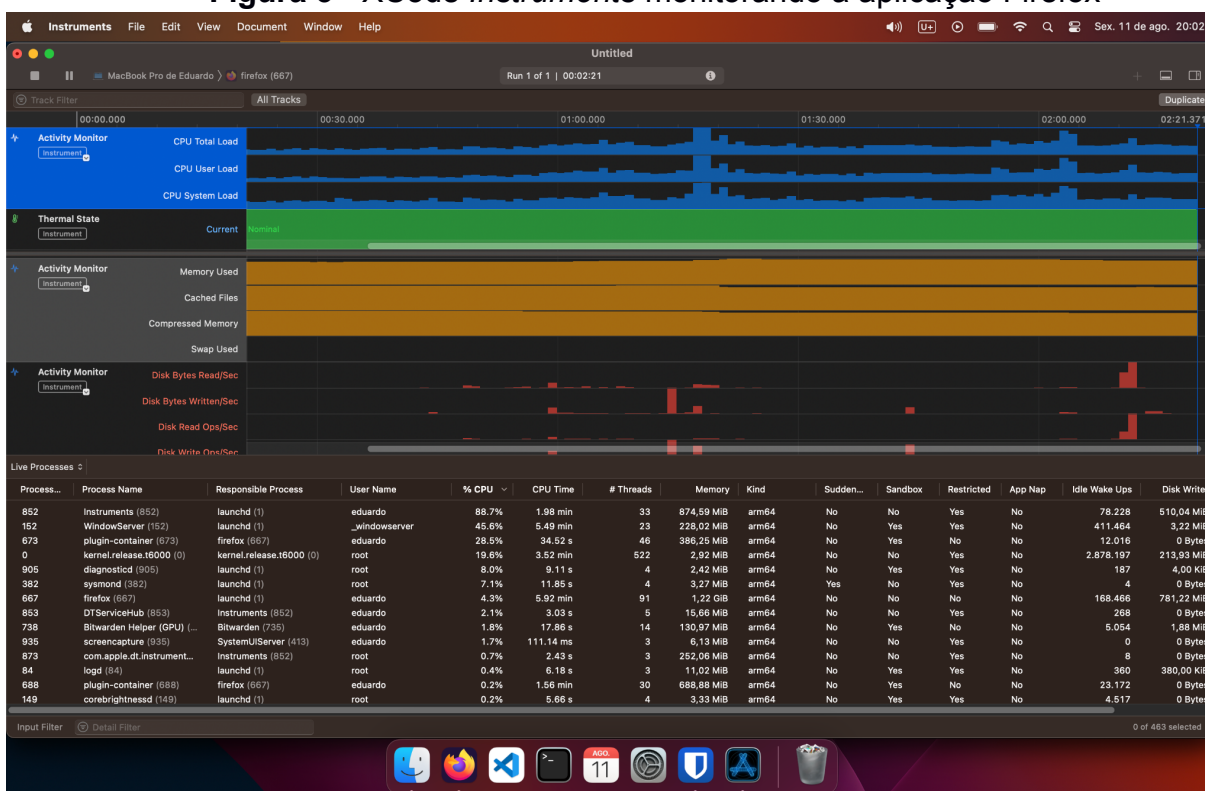
Figura 2 - Tela inicial do Instruments

Fonte: Capturado pelo autor, 2023

A Imagem 2 apresenta a tela inicial da Aplicação *Instruments*, onde podemos observar a variedade de opções disponíveis para fazer análises de outras aplicações.

A Apple (2019) afirma que é sempre importante lembrar de áreas como performance, otimização de uso de bateria e responsividade de uma aplicação, pois não importa o quanto uma UI é bonita, quando usuários encontram uma má performance ou travamentos, a experiência geral de uso é diminuída.

Figura 3 - XCode Instruments monitorando a aplicação Firefox



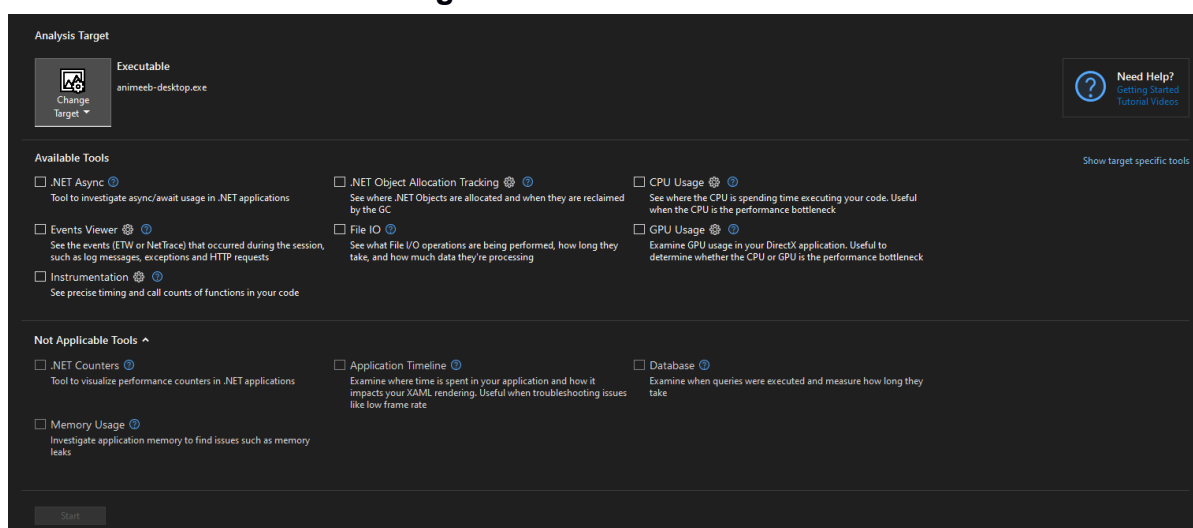
Fonte: Capturado pelo autor, (2023)

A Figura 2 apresenta o aplicativo Instruments em funcionamento enquanto ele faz a coleta de dados da aplicação Firefox que está executando em segundo plano. Podemos observar que o *Instruments* demonstra valores de uso de CPU, memória, disco e dados sobre a bateria do dispositivo. Observa-se ainda que o *Instruments* também coleta dados de outros processos que estão sendo executados no dispositivo, esses processos podem ser observados na parte de baixo da imagem.

2.2.2 Visual Studio

Visual Studio é um ambiente de desenvolvimento integrado (IDE) que é normalmente usado por desenvolvedores .NET e C#, e segundo pesquisas do StackOverflow ocorridas em 2021 com mais de 80.000 participantes, o Visual Studio e o Visual Studio Code são as ferramentas mais utilizadas pelos desenvolvedores.

Figura 4 - Profiler do Visual Studio



Fonte: Capturado pelo autor, 2023

Por ser uma IDE, o Visual Studio oferece um suporte a um depurador de códigos, junto com uma ferramenta integrada de diagnósticos, bem como uma ferramenta de *profiling* de performance. A ferramenta de diagnóstico está disponível apenas durante sessões de depuração, onde é possível criar *breakpoints* e visualizar o valor de variáveis.

Por outro lado, a ferramenta de *profiling* está disponível para ser usada com aplicativos já compilados, e prontos para serem distribuídos. Nesse contexto, Osipov (2019) afirma que os resultados dessa ferramenta serão extremamente aproximados à experiência que o usuário final vivenciará.

É importante mencionar que o profiler do Visual Studio pode apresentar algumas limitações em relação à disponibilidade de suas funções. Na Figura 2, pode-se identificar que “*Memory Usage*” (uso de memória) não está disponível para ser coletada na sessão de *profiling* da aplicação desenvolvida neste trabalho. Normalmente essas funções estão disponíveis em aplicativos criados usando .NET ou C#.

2.3 Tecnologias selecionadas

2.3.1 HTML

HTML, sigla para *HyperText Markup Language* (Linguagem de Marcação de Hipertexto), foi introduzida em conjunto com o HTTP, o Protocolo de Transferência de Hipertexto, em 1989 por Tim Berners-Lee, dentro das instalações da organização CERN, localizada na Europa.

É evidente que a linguagem concebida por Tim adquiriu uma popularidade incontestável e conquistou um amplo reconhecimento na comunidade. Conforme ressaltado por Raggett et al. (1998, p. 1), "HTML é a linguagem de publicação para a World Wide Web. Nos bastidores, cada documento que permeia a Web (...) é forjado por meio do HTML". Torna-se evidente a contínua e abrangente utilização do HTML, conforme constatado pela pesquisa conduzida pelo StackOverflow em 2021. Nesse levantamento, 53% dos participantes revelaram fazer uso do HTML em suas atividades.

2.3.2 React

Para Roldán (2018), no desenvolvimento web moderno é necessário fazer a manipulação da DOM frequentemente, e fazer muito isso afeta a performance de aplicações.⁸ Ele afirma ainda que por o React usar uma DOM virtual, todas as alterações acontecem em memória, o que é mais performático que alterar a DOM diretamente.⁹

O React é um *framework* que possibilita a criação de componentes dinâmicos e de fácil reutilização e de alta personalização. Nesse contexto, Roldán (2018) afirma que "a coisa mais poderosa do React é o fato de ele poder ser usado no cliente, servidor, aplicações de celular e até mesmo em aplicações de realidade virtual"¹⁰.

⁸ "In the modern web, we need to manipulate the DOM constantly; the problem is that doing this a lot may affect the performance of our application seriously."

⁹ "(...) React uses a Virtual DOM, which means that all updates occur in memory (this is faster than manipulating the real DOM directly) (...)"

¹⁰ The most powerful thing about React is that can be used in the client, server, mobile applications, and even VR applications

2.3.3 Chromium

Chromium é um projeto de código aberto que tem como objetivo criar uma maneira segura, rápida e estável para todas as pessoas poderem experienciar a web. O projeto Chromium é o responsável por aplicações como o Google Chrome, e o sistema Google Chrome OS.¹¹ (CHROMIUM, 2023)

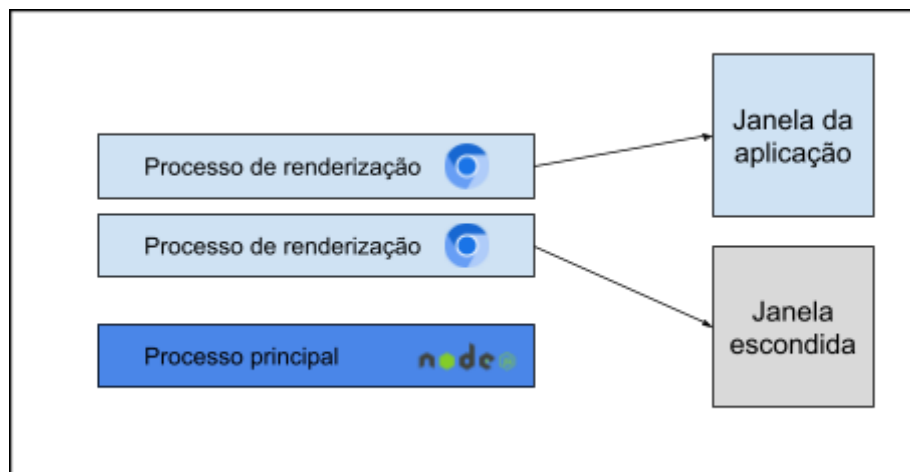
2.3.4 Electron

O Electron é um *framework* para desenvolvimento de aplicações de *desktop* utilizando JavaScript, HTML e CSS. Ao incorporar o Chromium e o Node.js em sua estrutura binária, o Electron possibilita a manutenção de um único código JavaScript e a criação de aplicativos multiplataforma que funcionam no Windows, macOS e Linux — sem a necessidade de experiência em desenvolvimento nativo. (ELECTRON, 2023).

Dentro de um aplicativo Electron, cada janela opera como um processo independente, onde um dos tipos de processo pode ser identificado como processo de renderização (*render*), onde uma aplicação pode conter múltiplos desses processos de *render*¹² (PERKAZ, 2021). É importante notar que nem todos os processos de *render* precisam exibir uma interface visual, alguns podem permanecer ocultos ao usuário, desempenhando funções em segundo plano. Para gerenciar todos esses processos o Electron dispõe de um processo principal (*main*), que é usado para o ciclo de vida da aplicação, acesso à APIs nativas do sistema operacional como notificações e bandeja de ícones. A imagem 5 demonstra os processos do Electron.

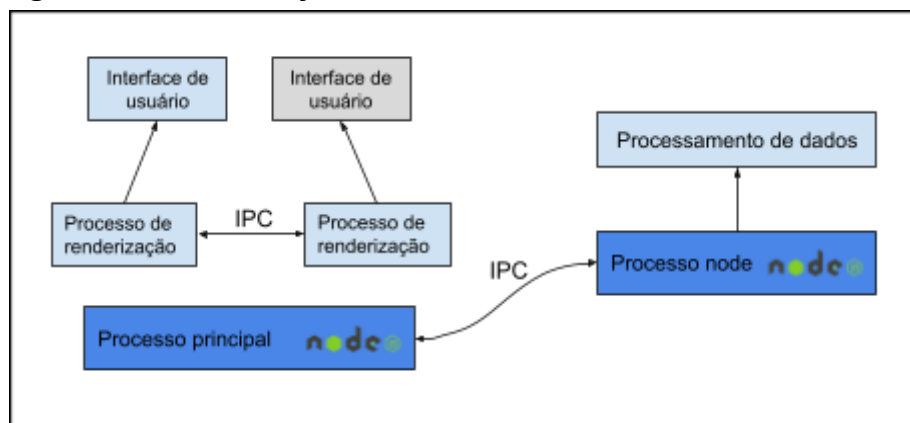
¹¹ “Chromium is an open-source browser project that aims to build a safer, faster, and more stable way for all users to experience the web. (...) the open-source projects behind the Google Chrome browser and Google Chrome OS (...)”

¹² “Each application’s window is a *render* process, which isolates the code execution at window level (...) each app is composed of one main process and a variable number of *render* processes”

Figura 5 - Funcionamento dos processos de renderização do Electron

Fonte: Elaborado pelo autor, 2023

Em aplicações de maior complexidade, pode ser necessário que processos de renderização precisem se comunicar entre si, para isso o Electron utiliza o protocolo IPC. Para Seabra et al. (2023) o IPC é uma parte vital para extrair o máximo de performance de um sistema, pois por meio dele é possível fazer trocas de mensagens com dados de diferentes partes de uma aplicação ou sistema.¹³

Figura 6 - Comunicação entre Processos no Electron usando IPC

Fonte: Elaborado pelo autor, 2023

Perkaz afirma que é ideal que o processo que renderiza a interface do usuário faça apenas isso e nada mais, e que processamentos que demorem e sejam custosos sejam feitos por outros processos. Perkaz diz ainda que nunca se deve bloquear o processo principal com processamentos longos e pesados. Caso o

¹³ "(...) is a vital part of extracting the best performance from operating systems (...) is done through distributed processes that execute message exchanges."

processo principal seja bloqueado, a aplicação Electron tende a ficar congelada até que o processo seja liberado.¹⁴

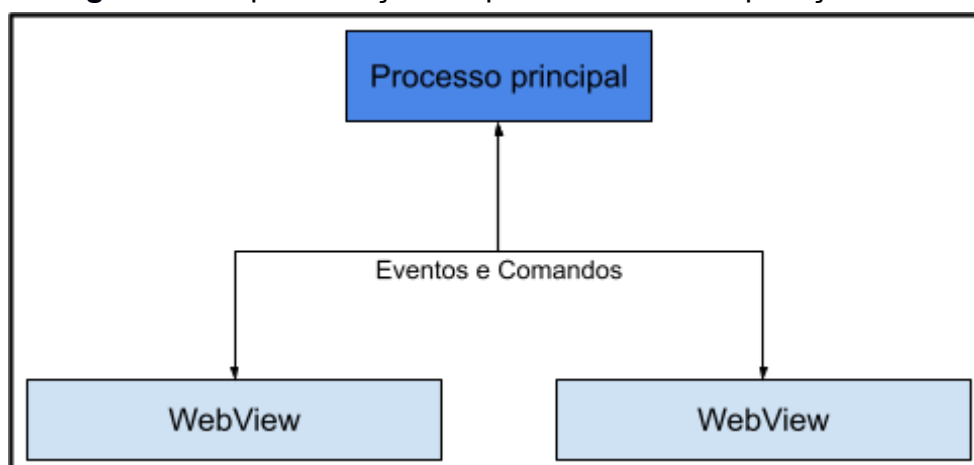
Para processamentos pesados pode-se usar um processo separado do principal, ele pode ser um processo de renderização ou apenas outro processo genérico do node. Na imagem 7 podemos observar os processos de renderização e o processo principal, que consegue criar outros processos para fazer processamentos mais demorados.

2.3.5 Tauri

Tauri é um conjunto de ferramentas que ajudam os desenvolvedores a criarem aplicações *desktop* usando praticamente qualquer *framework* frontend existente (TAURI, 2023). O Tauri é focado primeiramente em segurança, adotando uma abordagem que parte do pressuposto de que o dispositivo do usuário pode estar comprometido, para isso o Tauri implementa uma estratégia de defesa profunda, que tem como objetivo proporcionar precauções e diminuir a possibilidade de diminuir potenciais invasões (TAURI, 2023).

Parecido com o Electron, cada janela de uma aplicação Tauri é um processo de WebView, que se comunicam com o processo principal por meio de eventos e comandos:

Figura 7 - Representação simplificada de uma aplicação Tauri



Fonte: Elaborado pelo autor, 2023

¹⁴ (...) you should never run the operations in the main process. Doing so may block the main process, causing your application to freeze or crash (...)"

O processo principal do Tauri não renderiza as interfaces da aplicação, quem fica com essa responsabilidade são os processos de *WebView*, processos esses que são disponibilizados pelos sistemas operacionais (TAURI, 2023), essa lógica é representada na imagem 7. A comunicação entre processo principal e *WebView* utiliza também o protocolo IPC, seguindo ideias parecidas do Electron, onde bloquear o processo principal com longos processamentos de dados não é recomendado.

2.3.5.1 *WebView*

WebView é uma ferramenta nativa em vários sistemas operacionais diferentes capaz de utilizar tecnologias Web (JavaScript, CSS, HTML) em aplicações nativas do sistema operacional (MICROSOFT, 2023). Para a Microsoft o *WebViews* são úteis para integrar sistemas Web já construídos em uma aplicação nativa.

Diferentes sistemas usam diferentes soluções de *WebView*. A Microsoft usa o “Edge *WebView2*”, uma solução inspirada no Microsoft Edge, já o MacOS utiliza o “*WKWebView*” solução criada pela Apple baseada do Webkit, por fim, maior parte dos sistemas baseados em Linux utilizam o “*WebKitGTK*”.

Para a Microsoft, a adoção de *WebViews* oferece uma série de vantagens significativas. Elas incluem a capacidade de aproveitar o conhecimento existente em tecnologias web para criar aplicações nativas, a garantia de compatibilidade com diversos sistemas operacionais e a reutilização de códigos de aplicações já existentes, entre outros benefícios.

2.3.6 JavaScript

O JavaScript está em uma posição de dominância no cenário de desenvolvimento, de acordo com pesquisas feitas em 2023 pelo StackOverflow, a linguagem tem sido a mais utilizada por desenvolvedores durante onze anos consecutivos. Esse longo período de liderança reflete não apenas a popularidade do JavaScript, mas também a sua importância e relevância contínuas na comunidade de desenvolvimento de *software*.

Para Ryder et al. (2015), a popularidade do JavaScript se dá ao fato de ele ser uma linguagem que suporta tanto o paradigma de programação orientada a

objetos quanto o paradigma funcional. Essa versatilidade permite que os desenvolvedores utilizem a linguagem em uma ampla variedade de contextos, tornando-a uma escolha preferencial em muitos cenários de desenvolvimento.

2.3.7 Rust

Segundo Kshirsagar et al. (2020) o Rust surge como uma resposta direta aos desafios enfrentados em *softwares* desenvolvidos em C e C++. Embora essas linguagens ofereçam um desempenho superior, elas muitas vezes sofrem com problemas de segurança de memória e robustez. Ao fornecer um sistema de gerenciamento de memória mais seguro e robusto, a linguagem minimiza a probabilidade de vazamentos de memória e acessos inválidos, oferecendo, assim, uma alternativa viável para aplicações de alto desempenho.

O Rust é influenciado por várias linguagens de programação, por isso ela é considerada uma linguagem multiparadigma, combinando elementos funcionais e imperativos em sua sintaxe. Embora não seja classificada como uma linguagem orientada a objetos, Rust incorpora três dos princípios fundamentais desse paradigma: encapsulamento, abstração e polimorfismo (RUST, 2023).

Essa característica híbrida torna o Rust uma escolha versátil para uma ampla gama de aplicações, desde sistemas de alto desempenho até programas mais voltados para a expressividade e simplicidade do código.

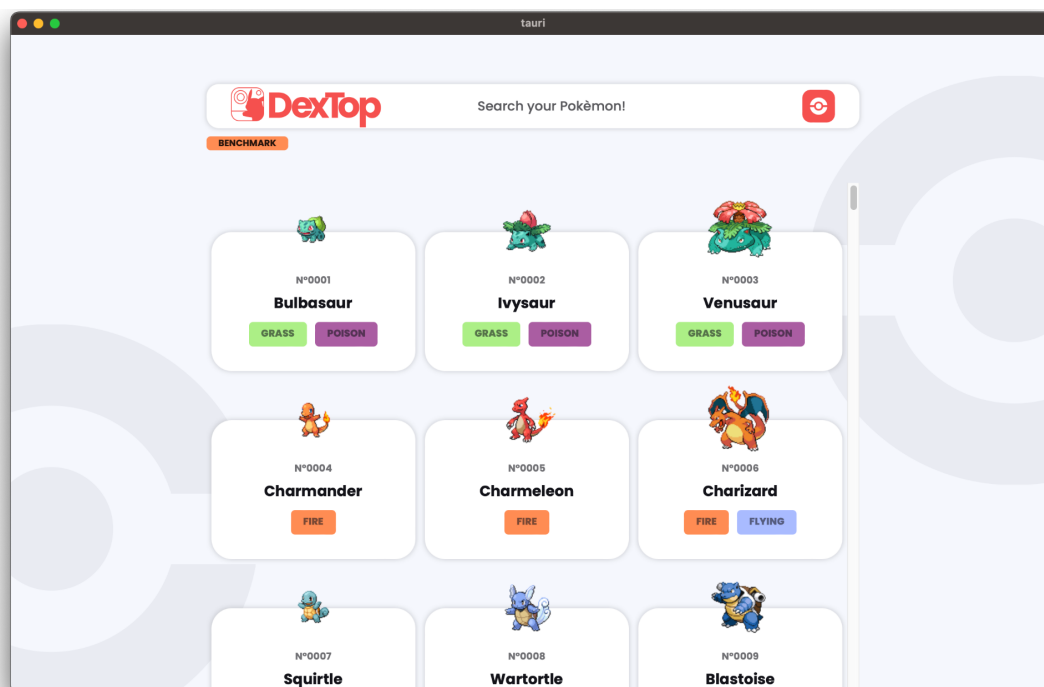
3 ASPECTOS DE IMPLEMENTAÇÃO

3.1 DexTop

A aplicação desenvolvida para o trabalho foi denominada “DexTop”, trata-se de uma Pokédex interativa que permite aos usuários explorar os Pokémon da primeira geração.

Na interface da aplicação, os Pokémon são apresentados em uma lista vertical, organizada de acordo com seus respectivos números de identificação. Cada entrada na lista exibe informações cruciais, incluindo o nome do Pokémon, seus tipos correspondentes e uma representação visual na forma de imagem. Esses elementos fornecem uma visão detalhada e organizada, facilitando a identificação e consulta de cada criatura, como apresentado na imagem 8.

Figura 8 - Interface principal da aplicação DexTop

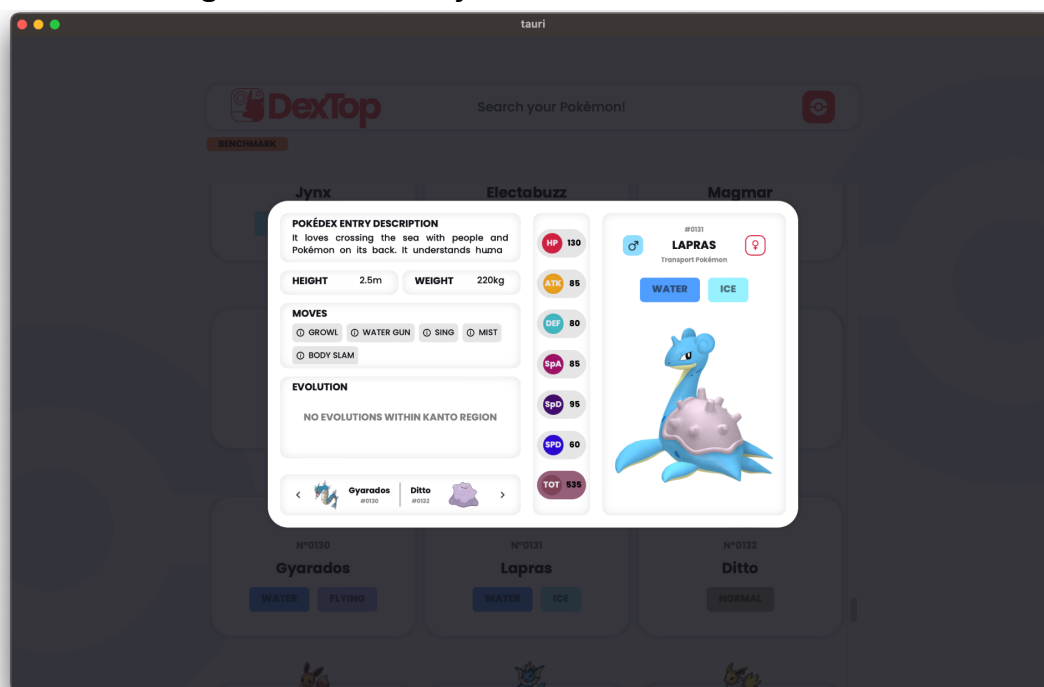


Fonte: Capturado pelo autor, 2023

Adicionalmente, o “DexTop” oferece uma funcionalidade de “benchmark”, que ao ser executada, a aplicação passa a fazer um processamento de dados simples. Essa funcionalidade é essencial para avaliar o desempenho da aplicação em diferentes cenários, garantindo eficiência em variadas situações.

Além da organização eficiente na interface da lista vertical, o DexTop também oferece uma experiência interativa ao usuário. Ao selecionar alguma das criaturas da listagem, um modal detalhado é apresentado, exibindo uma ampla gama de informações essenciais. Isso inclui dados como altura, peso, movimentos, estatísticas (como ataque, defesa, etc.), uma imagem ilustrativa e o tipo do Pokémon. Ademais, o modal também proporciona acesso direto às evoluções do Pokémon em questão, permitindo que os usuários explorem a progressão da criatura ao longo do tempo. Além disso, para facilitar a navegação, uma opção de paginação está disponível, permitindo que os usuários alternem entre Pokémon anteriores e posteriores de forma conveniente. A figura 9 apresenta esse modal detalhado com todos os detalhes previamente mencionados.

Figura 9 - Visualização detalhada de um Pokémon



Fonte: Capturado pelo autor, 2023

3.2 Comunicação entre processos

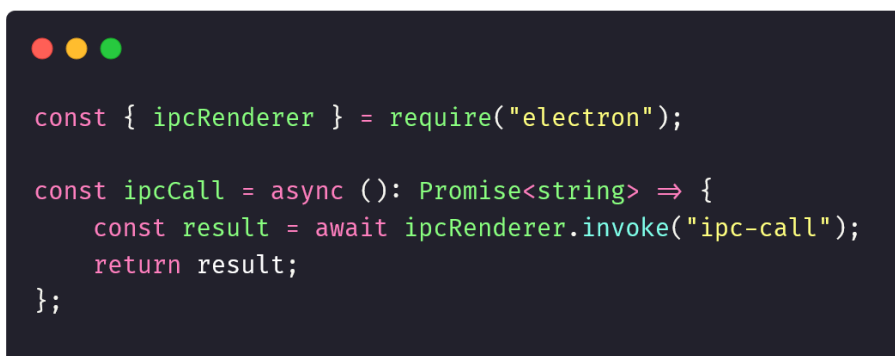
Dentro do DexTop, a comunicação entre processos é uma engrenagem crucial para o processo de coleta de dados relativos aos Pokémon. Esse mecanismo é orquestrado de forma precisa e coordenada, começando com a instância de renderização que dispara um evento em direção ao processo principal.

O processo principal trabalha como o cérebro central da aplicação, e ao receber a chamada feita pelo processo de renderização, ele assume a responsabilidade de efetuar uma consulta à PokéAPI.

A PokéAPI é uma fonte abrangente de dados sobre Pokémon, oferecendo todas as informações necessárias em um único local. Os dados são disponibilizados de forma altamente detalhada e são acessíveis por uma *API RESTful* (PokéAPI, 2023).

O *framework* Electron apresenta a classe denominada "*ipcRenderer*" que facilita a comunicação entre os processos. Dentro dessa classe, o método "*invoke*" requer, como seu primeiro argumento, o nome da "função" a ser executada em outro processo, no caso do DexTop, é o processo principal.

Figura 10 - Exemplo de uma chamada IPC utilizando Electron

A imagem mostra um editor de código com um fundo escuro e uma barra de título com três botões (vermelho, amarelo, verde) no canto superior esquerdo. O código é escrito em uma fonte monoespaçada com coloração de sintaxe: palavras-chave em verde, strings em amarelo, comentários em cinza claro e operadores em branco. O código demonstra a importação da classe ipcRenderer do Electron e a criação de uma função assíncrona que chama o método invoke da ipcRenderer com o nome "ipc-call".

```
const { ipcRenderer } = require("electron");

const ipcCall = async (): Promise<string> => {
  const result = await ipcRenderer.invoke("ipc-call");
  return result;
};
```

Fonte: Elaborado pelo autor, 2023

A imagem anterior ilustra um exemplo prático dessa classe e do método, demonstrando sua chamada à função "benchmark" no processo principal, que precisa estar preparado para receber essa chamada. Para isso, o Electron disponibiliza outra classe chamada "*ipcMain*". Essa classe é responsável pelo processo principal das aplicações, e disponibiliza o método "*handle*" para gerenciar as solicitações provenientes dos processos de renderização. Através do uso do método "*handle*", é possível definir funções que serão invocadas quando uma chamada IPC é recebida. Essa capacidade de comunicação entre os processos, facilitada pelas classes "*ipcMain*" e "*ipcRenderer*", é crucial para o desenvolvimento de aplicações interativas e eficientes no Electron.

A ilustração abaixo exemplifica o emprego do método `"handle"`, o qual invoca uma função de exemplo chamada `"someFunction"` e retorna seu resultado para o processo de renderização.

Figura 11 - Exemplo do Electron gerenciando chamadas IPC

```
const { ipcMain } = require("electron");

ipcMain.handle("ipc-call", async (event, ... args) => {
  const result = someFunction( ... args);
  return result;
});
```

Fonte: Elaborado pelo autor, 2023

A comunicação entre processos (IPC) no Tauri segue um modelo semelhante ao do Electron, o que facilita a transição para desenvolvedores familiarizados com a linguagem JavaScript e com o *framework*. Na Imagem 12, é apresentado um exemplo de código Rust para o gerenciamento da comunicação entre processos no Tauri:

Figura 12 - Código Rust que permite o Rust gerenciar chamadas IPC

```
#[tauri::command]
fn ipc_call(arg: &str) -> String {
  let result = some_function(arg);
  return result;
}

fn main() {
  tauri::Builder::default()
    .invoke_handler(tauri::generate_handler![ipc_call])
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

Fonte: Elaborado pelo autor, 2023

Neste código, estamos definindo uma função chamada `"ipc_call"` que utiliza o atributo `"#[tauri::command]"` para indicar que ela será acessível via IPC. Essa função

recebe um argumento do tipo *string* (*arg*) e retorna outra *string* como resultado. Em seguida, no bloco “*main()*”, configuramos o ambiente Tauri para lidar com a invocação dessa função. Isso permite a comunicação eficiente entre os diferentes processos da aplicação, contribuindo para a sua funcionalidade e interatividade.

Semelhante ao Electron, o Tauri também utiliza JavaScript para invocar o processo principal.

Figura 13 - Chamada IPC utilizando a biblioteca do Tauri no JavaScript

```
const { invoke } = require("@tauri-apps/api/tauri");

const ipcCall = async (): Promise<string> => {
  const result = await invoke("ipc-call");
  return result;
};
```

Fonte: Elaborado pelo autor, 2023

No código apresentado na Figura 13, estamos utilizando a biblioteca “@tauri-apps/api/tauri” para facilitar a comunicação entre os processos. Neste trecho, estamos definindo uma função chamada “*ipcCall*” que realiza a chamada ao processo principal através da função “*invoke*”. A função “*invoke*” recebe como argumento o nome da função que desejamos chamar no processo principal, que neste caso é “*ipc-call*”. Essa função retorna uma Promise, indicando que a operação é assíncrona e que irá retornar uma string.

3.3 Dados coletados

Nas seções subsequentes, serão apresentados em detalhes o procedimento adotado para a coleta de dados, bem como os próprios dados obtidos durante a execução das aplicações juntamente com uma análise dos mesmos.

3.3.1 Procedimentos de coleta

Para a coleta detalhada dos dados de desempenho das aplicações, foram empregadas ferramentas de *profiling* específicas para cada ambiente operacional.

No ambiente Windows, a ferramenta escolhida foi o Visual Studio Code (VSCode), no ambiente macOS, a coleta de dados foi conduzida utilizando o Apple Instruments.

Com o intuito de garantir uma execução padronizada e controlada das aplicações, optou-se por utilizar o ambiente macOS, aproveitando as funcionalidades oferecidas pelo Apple Automator. Essa abordagem proporcionou um ambiente de testes altamente consistente, permitindo uma comparação justa e precisa entre as performances das aplicações desenvolvidas com os *frameworks* Electron e Tauri.

A execução da aplicação pelo Automator foi projetada para replicar a interação humana com a aplicação durante um período de um minuto. Ao longo desse minuto, uma série de operações foi realizada de forma automatizada, simulando as ações que um usuário real poderia executar. Enquanto o Automator executava as ações planejadas, a ferramenta de *profiling* estava coletando os dados necessários.

Antes de cada execução o Automator foi configurado para ficar 4 segundos em idle, sendo assim as interações começam a ser executadas na marca de 5 segundos. Após o período de idle o Automator começa a realizar uma operação de rolagem na aplicação, demonstrando a habilidade de navegação fluida, essa operação dura em torno de 15 segundos.

Em seguida, em torno dos 20 segundos de execução, o Automator começa uma série de aberturas e fechamentos de um modal da aplicação. Dando sequência à execução, o Automator passa a realizar uma busca utilizando a barra de pesquisa, essa busca foi feita entre os segundos 28 e 32.

Posteriormente, no intervalo de tempo entre os segundos 33 e 48, um modal foi aberto, e dentro dele, a paginação foi utilizada para percorrer os dados dos Pokémons. A partir do segundo 50, a aplicação foi mantida em um estado de inatividade, simbolizando um cenário de espera. Por fim, a execução foi concluída com a finalização da aplicação.

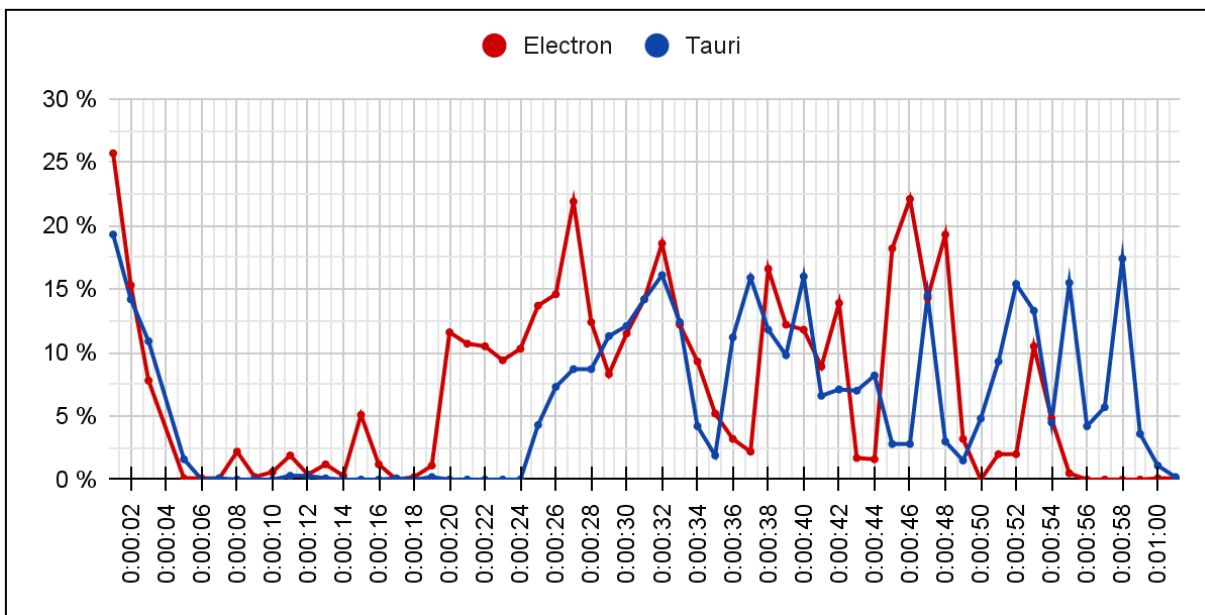
Nas seções a seguir serão apresentados todos os dados coletados.

3.3.2 Avaliação de Desempenho: Utilização de CPU e Memória

Ao observar a execução do teste automatizado, podemos ver alguns pontos de destaque que nos fornecem insights valiosos sobre o desempenho do Electron e

do Tauri. O gráfico da figura 14 ajuda a visualizar a porcentagem de CPU consumida por cada aplicação durante as execuções.

Figura 14 - Utilização de CPU em porcentagem durante a execução dos aplicativos



Fonte: Elaborado pelo autor, 2023

Analisando a inicialização do aplicativo, observamos que no segundo 0, o Electron exibiu um pico de demanda de CPU, atingindo aproximadamente 25.7%. Isso indica que o processo de inicialização do Electron pode ser mais intensivo em termos de recursos do sistema em comparação com o Tauri, que por outro lado demonstrou um uso de CPU mais moderado, mantendo-se em torno de 19.3%. Essa diferença sugere que o Tauri pode apresentar uma inicialização mais suave e eficiente em relação ao consumo de CPU.

Durante o intervalo de tempo entre os segundos 5 e 20, onde ambas as aplicações executam principalmente operações de scroll podemos observar que ambas as soluções, Electron e Tauri, apresentaram um uma utilização de CPU relativamente baixo, indicando que ambas são capazes de lidar de maneira eficaz com operações de scroll, mantendo o consumo de CPU em níveis aceitáveis.

No segundo 27, notamos que o Electron apresentou uma leve oscilação no consumo de CPU ao executar operações de abertura e fechamento de um modal, atingindo cerca de 21.9%. Por outro lado, o Tauri demonstrou uma eficiência superior, mantendo seu uso de CPU em 8.7% durante a mesma operação. Isso

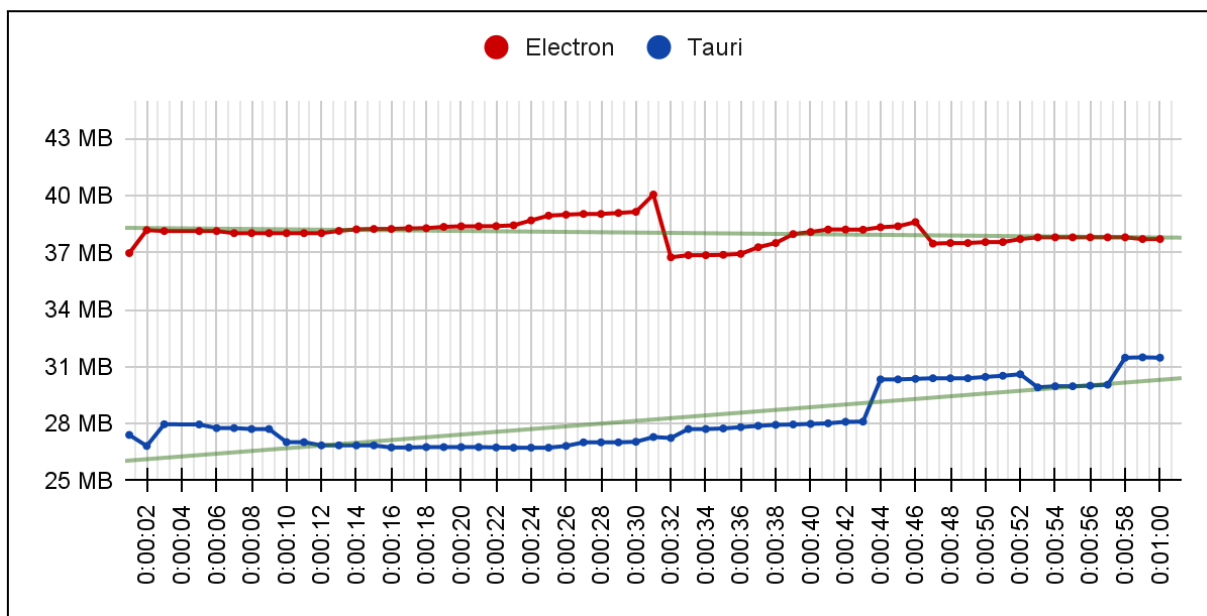
sugere que o Tauri lida de forma mais eficaz com as interações de modal em comparação com o Electron.

Em seguida, entre os segundos 28 e 32, ambos os *frameworks* enfrentaram uma carga de trabalho moderada ao realizar operações de busca usando a barra de pesquisa. O Electron registrou uma carga de CPU variando de 8.3% a 18.1%, enquanto o Tauri exibiu um desempenho muito parecido, com uma utilização de CPU variando de 8.7% a 16%. Ambos os *frameworks* demonstraram eficiência semelhante na realização de buscas.

Ao navegar usando a paginação dentro do modal, entre os segundos 33 e 48, observamos diferenças significativas entre o Electron e o Tauri. O Electron registrou um uso máximo de CPU 22.1%, indicando um impacto moderado no desempenho. Em contrapartida, o Tauri demonstrou um desempenho não tão diferente, com um consumo de CPU variando de 1.6% a 15.9%.

Por fim, após o segundo 50, durante um período de repouso a aplicação em Tauri apresentou vários picos de utilização da CPU, enquanto o Electron se manteve em repouso com utilização em torno de zero.

Figura 15 - Utilização de memória em MegaBytes durante a execução dos aplicativos



Fonte: Elaborado pelo autor, 2023

A avaliação do consumo de memória entre as aplicações Electron e Tauri revela tendências distintas ao longo do tempo. No gráfico da imagem 15, podemos

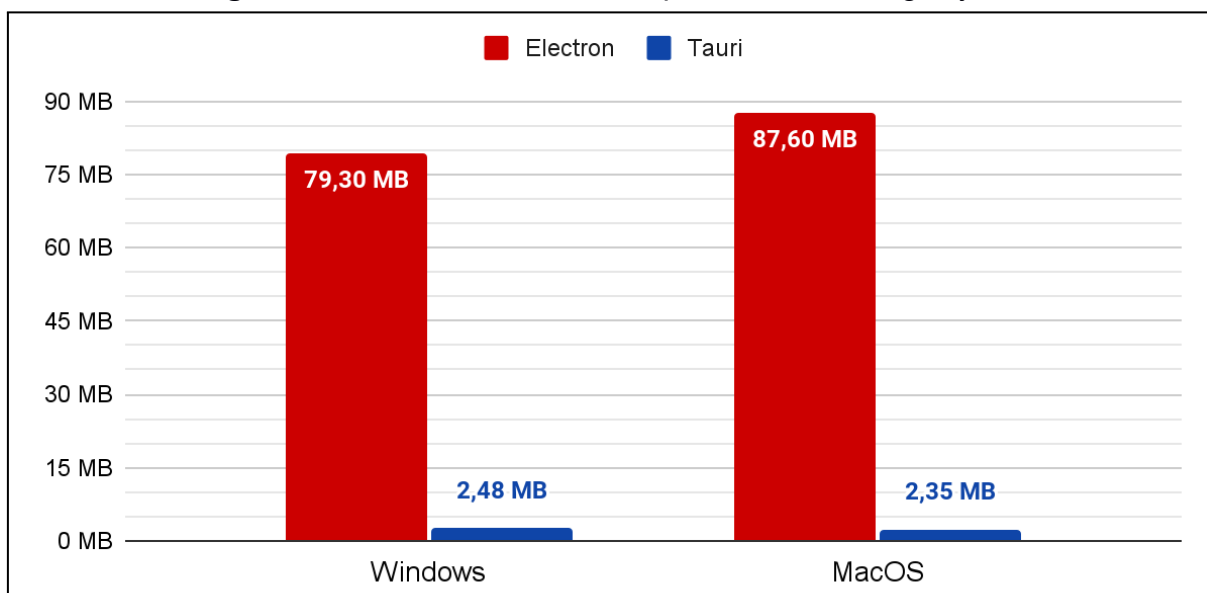
observar que a utilização de memória pelo Tauri apresenta uma tendência crescente de forma gradual, enquanto a aplicação Electron demonstra uma leve mas consistente tendência de queda. Esta diferença nas trajetórias de consumo de memória destaca as distintas abordagens de gerenciamento de recursos adotadas por cada *framework*.

O Tauri, ao longo da execução, demonstra uma maior necessidade de alocação de memória, indicando uma demanda crescente para processamento e armazenamento de dados. Por outro lado, o Electron parece otimizar a utilização de memória, mantendo-a relativamente estável e decrescente, sugerindo uma eficiência no gerenciamento de recursos.

3.3.3 Avaliação de Desempenho: Tamanho dos aplicativos e duração da compilação

Como definido por Baysal e Guerrouj (2016), parâmetros como o tamanho final dos aplicativos são cruciais para o sucesso de um aplicativo, onde os usuários tendem a instalarem aplicativos que ocupam menos espaço de armazenamento.

Figura 16 - Tamanho final dos aplicativos em MegaBytes



Fonte: Elaborado pelo autor, 2023

Os dados apresentados na imagem 16 revelam uma diferença substancial nos tamanhos dos aplicativos compilados entre o Electron e o Tauri, em diferentes sistemas operacionais. No ambiente Windows, o aplicativo compilado com o Electron possui um tamanho significativamente maior, registrando 79.30 MB, em

contraste com a aplicação Tauri, que se destaca pela sua eficiência, com um tamanho de apenas 2.48 MB.

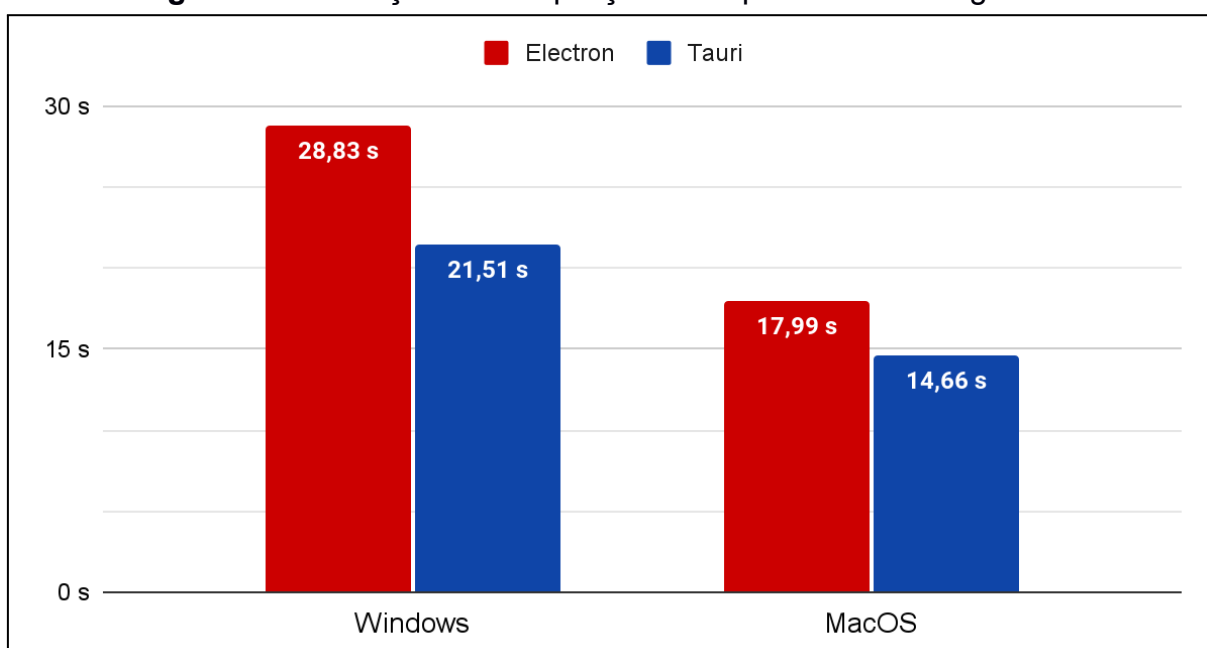
De forma semelhante, no ambiente MacOS, a disparidade persiste. O aplicativo criado com o Electron ostenta um tamanho de 87.60 MB, enquanto a aplicação Tauri continua a demonstrar sua eficácia na economia de recursos, com um tamanho notavelmente menor de 2.35 MB.

Esses dados indicam uma vantagem clara do Tauri em termos de economia de espaço, o que pode ser crucial em cenários onde a otimização do uso de disco é um fator determinante.

Além das distinções marcantes nos tamanhos dos aplicativos compilados, é igualmente essencial analisar o tempo investido em todo o processo, pois como definido por Brooks (2008), o sucesso de um projeto pode estar atrelado ao tempo de compilação das aplicações¹⁵. O Gráfico 4 destaca os tempos de compilação dos aplicativos, levando em consideração um ambiente previamente configurado tanto no MacOS quanto no Windows. Esses dados foram meticulosamente coletados com o auxílio da ferramenta de linha de comando "time".

Para garantir a precisão dos resultados, o processo de compilação foi inicialmente executado uma vez, com o tempo dessa execução sendo desconsiderado. Em seguida, o processo foi repetido por mais cinco vezes e os tempos de compilação foram registrados. A média desses tempos foi então calculada para fornecer uma visão mais abrangente e representativa do desempenho de cada *framework* em ambos os ambientes operacionais.

¹⁵ "(...) collective ownership of the build and build process made a big difference in the overall project success"

Figura 17 - Duração da compilação dos aplicativos em segundos

Fonte: Elaborado pelo autor, 2023

Ao analisar os tempos de compilação dos aplicativos desenvolvidos com Electron e Tauri apresentados na imagem 17, torna-se evidente a agilidade e eficiência que cada *framework* oferece em diferentes ambientes. No ambiente Windows, o Electron apresenta um tempo de compilação de 28.83 segundos, enquanto o Tauri demonstra-se ser alguns segundos mais rápido, completando o processo em 21.51 segundos.

No sistema operacional MacOS, ambos *frameworks* conseguem compilar as aplicações de forma mais rápida que no Windows, onde o Electron registra um tempo de compilação de 17.99 segundos, enquanto o Tauri continua a exibir sua eficiência notável, concluindo o processo em apenas 14.66 segundos.

3.3.4 Análise Qualitativa: Tauri e Electron em Perspectiva

Nos últimos 12 meses, uma análise dos dados do Google Trends¹⁶ revela uma diferença notável no interesse pelo Electron e pelo Tauri. O Electron foi pesquisado em média 69 vezes por dia, enquanto o Tauri registrou apenas 3 pesquisas diárias. Ao examinarmos as páginas do GitHub^{17,18} de cada *framework*, também podemos notar uma diferença no número de estrelas atribuídas pelos usuários. O Tauri conta

¹⁶ <https://trends.google.com/trends/explore?q=tauri,electron&hl=pt>

¹⁷ <https://github.com/tauri-apps/tauri>

¹⁸ <https://github.com/electron/electron>

com 69.8 mil estrelas, enquanto o Electron possui uma maior base de usuários com 109 mil estrelas. Outro ponto de análise se refere às discussões geradas nas páginas de issues do GitHub^{19,20}. O Tauri apresenta 1272 discussões criadas, enquanto o Electron registra um número mais elevado, com 5077 discussões.

É importante ressaltar que um dos aspectos em que o Tauri ainda está evoluindo em relação ao Electron é a implementação de APIs para conexão com o sistema operacional. Um exemplo é o "*Native Context Menu*", ou seja, o menu de contexto nativo do sistema²¹. Outro exemplo de funcionalidade ainda não disponível no Tauri é "*Deeplinking*"²², que é a possibilidade de abrir aplicações através de links personalizados. Estas funcionalidades estão disponíveis no Electron, proporcionando aos desenvolvedores uma integração mais completa com o sistema operacional. O Tauri, por sua vez, ainda está trabalhando para incorporar essa capacidade em sua plataforma.

¹⁹ <https://github.com/search?q=tauri&type=discussions>

²⁰ <https://github.com/search?q=electron&type=discussions>

²¹ <https://github.com/tauri-apps/tauri/issues/4338>

²² <https://github.com/tauri-apps/tauri/issues/323>

4 CONCLUSÃO

Com base na análise detalhada dos dados apresentados neste estudo comparativo entre o Electron e o Tauri, fica evidente que cada *framework* possui suas próprias vantagens e cenários ideais de aplicação.

Na perspectiva dos objetivos delineados neste trabalho, é notável destacar o sucesso na implementação de ambos os *frameworks*. A coleta e análise meticulosa dos dados proporcionaram uma visão abrangente das capacidades e peculiaridades distintas de cada plataforma. Com isso, este estudo se revela como um guia valioso para os desenvolvedores, oferecendo insights cruciais na seleção do *framework* mais apropriado para seus projetos, levando em conta as exigências específicas de cada aplicação.

4.1 Principais contribuições

Este estudo proporciona uma valiosa contribuição à decisão do *framework* mais apropriado para um determinado projeto, capacitando o desenvolvedor a fazer uma escolha informada, alinhada com as necessidades específicas de sua aplicação.

Ao analisar os dados apresentados como um todo, podemos observar que o Electron emerge como uma opção sólida para o desenvolvimento de aplicativos mais complexos, especialmente aqueles que demandam o uso de recursos nativos do sistema operacional. Sua ampla base de usuários, estabilidade e funcionalidades avançadas o tornam uma escolha confiável para projetos que necessitam de um alto nível de integração com o ambiente operacional.

Por outro lado, o Tauri se destaca como uma solução mais viável para problemas de menor escala e aplicações que buscam um desempenho ágil e eficiente. Sua abordagem leve e flexível o torna uma escolha atrativa para desenvolvedores que buscam uma alternativa mais simplificada e ágil.

Portanto, tanto o Electron quanto o Tauri desempenham papéis significativos no panorama do desenvolvimento de aplicativos, proporcionando soluções valiosas para diferentes cenários e necessidades.

4.2 Limitações da pesquisa

Ao conduzir esta pesquisa, algumas limitações relevantes foram identificadas. Primeiramente, observou-se que o sistema Windows apresenta restrições no que diz respeito às ferramentas de *profiling* disponíveis. Em particular, o Visual Studio, uma ferramenta comumente utilizada para desenvolvimento na plataforma Windows, não oferece uma ampla gama de dados para análise. Por exemplo, a ferramenta não dispõe de uma linha do tempo com os valores de CPU e memória.

Outra limitação relevante é a escassez de trabalhos científicos dedicados ao desenvolvimento *cross-platform* para aplicações *desktop*. Esta lacuna na literatura científica pode impactar a disponibilidade de referências e melhores práticas, tornando mais desafiador o processo de escrita e análises. Além disso, a ausência de estudos científicos específicos sobre o *profiling* de outros *frameworks* também se configura como uma limitação. A falta de outros estudos com outros *frameworks* faz com que seja difícil por exemplo decidir quais dados são mais importantes para o trabalho.

Outra consideração importante diz respeito à utilização do Instruments no macOS. Apesar de ser uma ferramenta valiosa para análise de desempenho, sua utilização pode ser complexa e demandar um aprendizado significativo por parte dos desenvolvedores. Adicionalmente, a documentação disponível sobre o Instruments não é tão abrangente quanto desejável, o que pode dificultar a exploração completa de suas funcionalidades.

4.3 Trabalhos futuros

Com base nas limitações identificadas nesta pesquisa, diversas oportunidades de aprofundamento e expansão surgem para trabalhos futuros. Uma área de investigação promissora seria a análise de outras ferramentas de *profiling* disponíveis, especialmente aquelas que podem oferecer uma visão mais abrangente e detalhada do desempenho de aplicações *cross-platform*.

Além disso, a avaliação de outros *frameworks*, como o Wails, que se assemelha ao Tauri, mas utiliza a linguagem Go em vez de Rust, pode ser um campo interessante para futuras investigações.

Outra direção de pesquisa potencial seria explorar o processo de desenvolvimento *cross-platform* em si. Investigar o ciclo de vida do desenvolvimento, práticas recomendadas e desafios comuns enfrentados pelos desenvolvedores ao

criar aplicações que visam diferentes plataformas operacionais pode fornecer um entendimento mais profundo das complexidades envolvidas nesse tipo de desenvolvimento.

REFERÊNCIAS

ALKHARS, Abeer; MAHMOUD, Wasan. **Cross-Platform Desktop Development (JavaFX vs. Electron)**. 2017. Tese (Bacharelado) – Ciências da Computação – Linnaeus University, Suécia, 2017.

AMAZON. **Application Profiling | AWS Marketplace**. Disponível em: <https://aws.amazon.com/marketplace/solutions/migration/application-profiling>. Acesso em: 30 out. 2023.

APPLE. **Getting Started with Instruments - WWDC19 - Videos**. Disponível em: <https://developer.apple.com/videos/play/wwdc2019/411/?time=45>. Acesso em: 30 out. 2023.

BROOKS, Graham. Team Pace – Keeping Build Times Down. **Agile 2008 Conference**, [s. l.], 15 ago. 2008.

FACEBOOK. GitHub. **facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces**. Disponível em: <https://github.com/facebook/react>. Acesso em: 1 jan. 2023.

GOOGLE. **Chromium: Home**. Disponível em: <https://www.chromium.org/>. Acesso em: 30 out. 2023.

GOOGLE. **Profile your app performance**. Disponível em: <https://developer.android.com/studio/profile>. Acesso em: 30 out. 2023.

GOOGLE. **tauri, electron - Pesquisar - Google Trends**. Disponível em: <https://trends.google.com/trends/explore?q=tauri,electron&hl=pt>. Acesso em: 30 out. 2023.

GUERROUJ, Latifa; BAYSAL, Olga. Investigating the android apps' success: An empirical study. **International Conference on Program Comprehension**, [s. l.], ed. 24, 7 jul. 2016.

HEITKÖTTER, Henning; HANSCHKE, Sebastian; MAJCHRZAK, Tim A. Evaluating Cross-Platform Development Approaches for Mobile Applications. **Lecture Notes in Business Information Processing**, [s. l.], p. 120–138, jun. 2013. Disponível em: <https://www3.nd.edu/~cpoellab/teaching/cse40814/crossplatform.pdf>. Acesso em: 30 out. 2023.

MILL, John Stuart. **A System of Logic: Ratiocinative and Inductive: Vol. II**. [s.l.] Library of Alexandria, [s.d.].

KSHIRSAGAR, Jayanta; DEWAN, Akshay; HAYATNAGARKAR, Harshal Ganpatrao. EpiRust: Towards A Framework For Large-scale Agent-based Epidemiological Simulations Using Rust Language. **The 61st International Conference of Scandinavian Simulation Society**, [s. l.], 1 set. 2020. DOI 10.3384/ecp20176475.

Disponível em: https://ep.liu.se/ecp/176/067/SIMS2020_article_ecp20176475.pdf. Acesso em: 30 out. 2023.

MICROSOFT. **Introduction to Microsoft Edge WebView2 - Microsoft Edge Development | Microsoft Learn**. Disponível em: <https://learn.microsoft.com/en-us/microsoft-edge/webview2/>. Acesso em: 30 out. 2023.

OSIPOV, Aleksandr. **Selecting C# Profiling Tools**. 2019. Tese (Bacharelado) – Tecnologia da informação e comunicação – Turku University Of Applied Sciences, Finlândia, 2019.

PERKAZ, Alain. **Advanced Electron.js architecture**. Disponível em: <https://blog.logrocket.com/advanced-electron-js-architecture/>. Acesso em: 30 out. 2023.

POKEAPI. **About - PokéAPI**. Disponível em: <https://pokeapi.co/about/>. Acesso em: 30 out. 2023.

RAGGETT, Dave et al. **Raggett on HTML 4**. Addison-Wesley Longman Publishing Co., Inc., 1998.

REHMAN, Faraz *et al.* Android-Platform Based Determination of Fastest Cross-Platform Framework. **International Journal of Computer Science and Mobile Computing**, [S. l.], p. 1-12, 1 set. 2017. Disponível em: <https://ijcsmc.com/docs/papers/September2018/V7I9201801.pdf>. Acesso em: 17 out. 2023.

ROLDÁN, Carlos Santana. **React Cookbook**. [S. l.: s. n.], 2018. *E-book*.

RYDER, Barbara G.; WEI, Shiyi. Adaptive Context-sensitive Analysis for JavaScript. **European Conference on Object-Oriented Programming**, [s. l.], 29 jun. 2015. Disponível em: <https://drops.dagstuhl.de/storage/00lipics/lipics-vol037-ecoop2015/LIPIcs.ECOOP.2015.712/LIPIcs.ECOOP.2015.712.pdf>. Acesso em: 30 out. 2023.

SEABRA, L. et al. **INTERPROCESS COMMUNICATION IN SOFTWARE INFRASTRUCTURE**. Editora Científica Digital eBooks, p. 670–682, 1 jan. 2023.

SHEVTSIV, Nikita A.; STRIUK, Andrii M. Cross platform development vs native development. **CS&SE@SW**, [s. l.], 27 nov. 2020. Disponível em: [http://ds.knu.edu.ua/jspui/bitstream/123456789/3550/1/Shevtsiv N. A. Cross platform development.pdf](http://ds.knu.edu.ua/jspui/bitstream/123456789/3550/1/Shevtsiv%20N.%20A.%20Cross%20platform%20development.pdf). Acesso em: 30 out. 2023.

STACKOVERFLOW. **Stack Overflow Developer Survey 2021**. Disponível em: <https://insights.stackoverflow.com/survey/2021>. Acesso em: 30 out. 2023.

STACKOVERFLOW. **Stack Overflow Developer Survey 2023**. Disponível em: <https://survey.stackoverflow.co/2023/>. Acesso em: 30 out. 2023.