UNIVERSIDADE FEDERAL DO PAMPA

Felipe Bedinotto Fava

Ferramenta para Visualização do Consumo de Recursos de Hardware para Aplicações Flutter

Alegrete
2023

Felipe Bedinotto Fava

Ferramenta para Visualização do Consumo de Recursos de Hardware para Aplicações Flutter

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Claudio Schepke

Alegrete
2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

F272f Fava, Felipe Bedinotto
Ferramenta para Visualização do Consumo de Recursos de Hardware para Aplicações Flutter / Felipe Bedinotto Fava.
45 p.

Tese(Doutorado)-- Universidade Federal do Pampa, CIÊNCIA DA COMPUTAÇÃO, 2023.
"Orientação: Claudio Schepke".

1. Flutter. 2. Android. 3. Plugin. 4. Monitor de Rede. 5. Memória. I. Título.

FELIPE BEDINOTTO FAVA

FERRAMENTA PARA VISUALIZAÇÃO DO CONSUMO DE RECURSOS DE HARDWARE PARA APLICAÇÕES FLUTTER

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 06 de dezembro de 2023.

Banca examinadora:

_____

Prof. Dr. Claudio Schepke

Orientador

UNIPAMPA

_____

Profa. Dra. Aline Vieira de Mello

UNIPAMPA

Prof. Me. Jean Felipe Patikowski Cheiran
UNIPAMPA

Vilson Blanco Dauinheimer
Analista de Sistemas

Assinado eletronicamente por **ALINE VIEIRA DE MELLO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 06/12/2023, às 20:40, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **JEAN FELIPE PATIKOWSKI CHEIRAN**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 06/12/2023, às 20:41, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Vilson Blanco Dauinheimer**, **Usuário Externo**, em 06/12/2023, às 20:41, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **CLAUDIO SCHEPKE**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 06/12/2023, às 20:42, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1312942** e o código CRC **C141C7F5**.

Este trabalho é dedicado em parte a minha família que sempre me apoiou na minha jornada de vida, a minha mãe Liliane, meu pai Mauricio e minha irmã Renata, sem vocês eu não seria quem sou hoje e não teria chegado onde cheguei. Mas além de minha família também dedico este trabalho aos amigos que a universidade me trouxe, que ao longo de todos esses anos foram minha familia nesta cidade. Faço também uma dedicatória especial para minha namorada Ellen, que surgiu em minha vida como uma amiga através da universidade e hoje em dia é minha grande companheira de vida, me apoiando, inspirando e ajudando a lutar cada dia para buscar alcançar os meus e os nossos sonhos.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a meu orientador, Professor Dr. Claudio Schepke, por aceitar ser meu guia nesta etapa final da minha graduação, toda sua ajuda foi muito importante para a elaboração deste trabalho e para minha formação como cientista da computação. Agradeço aos integrantes da banca avaliadora por aceitarem meu convite, Professora Dra. Aline Vieira de Mello, Professor Me. Jean Felipe Patikowski Cheiran e ao Analista de Sistemas Vilson Blanco Dauinheimer, sinto-me verdadeiramente grato a todos vocês por fazerem parte da avaliação deste trabalho. Agradeço também aos demais professores que tive na UNIPAMPA durante meu período de graduação, posso garantir que aprendi algo com cada um deles e esses ensinamentos agora fazem parte do profissional que estou me tornando. Agradeço a minha família por todo apoio todos esses anos de formação, vocês são minha base. Por fim, agradeço a minha namorada Ellen Rodrigues Castro que me incentivou, auxiliou, apoiou e aguentou todo tempo que elaborei este trabalho.

RESUMO

O presente Trabalho de Conclusão de Curso (TCC) apresenta o desenvolvimento de um *plugin* que visa auxiliar desenvolvedores Flutter a identificar problemas de desempenho em suas aplicações enquanto as desenvolvem, de uma forma simples e direta, por meio da visualização de três aspectos chaves: taxa de quadros que estão sendo renderizados por segundo, quantidade de *Kilobytes* sendo enviados e recebidos pela rede e quantidade de *Random Access Memory* (RAM) em uso e disponível. Com base nisso, o documento descreve o que é Flutter e como ele funciona, enfatizando as diferenças entre pacotes e *plugins* em Flutter. Também são abordados os ambientes de desenvolvimento para plataformas móveis, *desktop* e *web*, descrevendo algumas das linguagens de programação que geralmente são descritas para a criação de aplicações em cada uma das plataformas. Como metodologia da criação do *plugin* descreve-se cada etapa percorrida durante o desenvolvimento da primeira versão do mesmo, apontando os obstáculos encontrados e as decisões tomadas para buscar solucionar os problemas, bem como os sucessos e as falhas do desenvolvimento. Os resultados apresentados dão-se em formato de imagens de duas aplicações reais e uma de teste, utilizando o *plugin* para demonstrar como o mesmo funciona e como ele se integra com a aplicação que o está utilizando. Por fim são descritas as conclusões finais deste trabalho bem como os possíveis trabalhos futuros.

Palavras-chave: Flutter. Android. Plugin. Monitor de Rede. FPS. Memória.

ABSTRACT

This Bachelor's Thesis describes the development of a plugin that aims to help Flutter developers identify performance problems in their applications while developing them. It operates simply and directly through three key visualization aspects: frame rate rendered per second, number of Kilobytes being sent and received through the network, and amount of RAM in use and available. Based on this, the document describes what Flutter is and how it works, emphasizing the differences between packages and plugins in Flutter. Development environments for mobile, desktop, and web platforms are also covered, describing the programming languages for creating applications on each platform. The methodology for creating the plugin includes the description of each step taken during the development of the first version of the plugin, pointing out the obstacles encountered and the decisions made to solve the problems, as well as the successes and failures of the development. The results are presented in figures of two real applications and a test one, using the plugin to demonstrate how it works and how it integrates with the application that is using it. Finally, the document describes the conclusions of this work and possible future work.

Key-words: Flutter. Android. Plugin. Network Monitor. FPS. Memory.

LISTA DE FIGURAS

Figura 1 – Exemplo execução de uma aplicação Flutter através do SDK em linha de comando 21
Figura 2 – Página do DevTools disponibilizada pelo link da Figura 1 22
Figura 3 – Teste do pacote **performance** 24
Figura 4 – Foco em uma execução do pacote **performance** 24
Figura 5 – Testes do pacote **lemberfpsmonitor** 26
Figura 6 – Testes do pacote **statsfl** 27
Figura 7 – Testes do pacote **show_fps** 27
Figura 8 – Contador de quadros por segundo 30
Figura 9 – Informações sobre memória RAM 32
Figura 10 – Gráfico de perfil de rede 34
Figura 11 – Página sobre Cristo Redentor com plugin ativo 38
Figura 12 – Página sobre A Grande Pirâmide com plugin ativo 39
Figura 13 – Comparação entre página carregada e em carregamento do aplicativo OpenLeaf 40
Figura 14 – Comparação do comportamento da aplicação ao ser feita uma requisição para a Internet 41

LISTA DE SIGLAS

ACM *Association for Computing Machinery*

API *Application Programming Interface*

CPU *Central Processing Unit*

FPS *Frames Per Second*

HTTP *Hypertext Transfer Protocol*

IA *Inteligência Artificial*

IaaS *Infrastructure as a Service*

IEEE *Institute of Electrical and Electronics Engineers*

IoT *Internet of Things*

PaaS *Platform as a Service*

RAM *Random Access Memory*

SaaS *Software as a Service*

TCC Trabalho de Conclusão de Curso

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

UID *Unique Identifier*

SUMÁRIO

1 INTRODUÇÃO 19
1.1 Objetivos 20
1.2 Justificativa 21
1.3 Organização deste trabalho 21

2 TRABALHOS RELACIONADOS 23

3 METODOLOGIA 29
3.1 Contador de FPS 30
3.2 Uso de Memória 31
3.3 Tráfego de Rede 32
3.4 Parâmetros de customização 34

4 RESULTADOS 37
4.1 Wonderous 37
4.2 OpenLeaf 37
4.3 Aplicativo de exemplo 38

5 CONSIDERAÇÕES FINAIS 43

REFERÊNCIAS 45

1 INTRODUÇÃO

O número de dispositivos móveis ativos é bem maior do que o número de *laptops*, *desktops* e *tablets* somados no mundo atual, chegando a até 4.8 vezes como apontam dados do ano de 2022 (TAYLOR, 2023) e com projeções que perpetuam esta margem. Somado a isto, o número de aplicações móveis tem crescido significativamente nos últimos anos, devido é claro ao aumento da popularidade de dispositivos móveis e à crescente demanda por aplicações que possam ser utilizadas em qualquer lugar e a qualquer hora.

Com o crescimento no mercado de aparelhos móveis houve também um aumento de vagas no mercado para especialistas em desenvolvimento para estas plataformas, acarretando o aumento do número de desenvolvedores de aplicativos, encadeando assim um grande número de aplicações criadas. Segundo dados na Google Play (CECI, 2023) e na Apple App Store (CAMINADE, 2022), existem milhões de aplicativos disponíveis para download, abrangendo uma ampla variedade de categorias, como jogos, redes sociais, notícias, entre outros. Além disso, a criação, desenvolvimento e o amadurecimento de outras tecnologias, como a *Internet of Things* (IoT) e a *Inteligência Artificial* (IA), permitem a criação de novos tipos de aplicativos que oferecem funcionalidades e recursos inovadores.

O *hardware* de aparelhos móveis geralmente busca a melhor eficiência energética possível. Por isso, ao analisarmos suas características, esses possuem processadores mais modestos e com restrições de memória RAM quando comparados com *desktops* ou *laptops*. Além disso, as aplicações geralmente executam diretamente no aparelho e se comunicam através da Internet com servidores somente quando estritamente necessário. Somando isto ao fato de que a energia é um recurso caro, torna-se um pouco mais claro um dos grandes motivos pelos quais são criadas aplicações para serem instaladas nos dispositivos móveis, ao invés de simplesmente serem utilizados sistemas *web*, através do navegador de um aparelho móvel, como bem é apontado por Horn et al. (2023).

Um ponto chave que sempre é almejado em uma aplicação *web* é o baixo uso de recursos, dado que normalmente o servidor no qual um sistema está hospedado não executa exclusivamente aquele sistema. Com o advento da computação em nuvem e dos grandes provedores desta solução, cada vez mais tem-se tornado comum a utilização de *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS), além é claro de outras soluções de serviços. Para aplicações móveis ou *desktop*, o baixo consumo de recursos é um objetivo claro e importante. Além disso, vários cuidados devem ser tomados durante o desenvolvimento em sistemas rodando diretamente na máquina do usuário, para garantir que este não evidencie comportamentos indesejados como travamentos, atraso na responsividade, fechamentos inesperados, entre outros. Por exemplo, é preciso determinar quais são as características mínimas que o aparelho do usuário deve apresentar para conseguir executar a aplicação. Tudo isso para tornar a Experiência do Usuário a mais consistente possível, preservando a confiabilidade do sistema.

Atualmente para que um sistema seja minimamente aceito por seus usuários, este

deve estar funcionando de maneira eficiente e satisfazendo as necessidades dos usuários. É muito difícil um usuário, ao abrir uma aplicação, ver que a bateria de seu aparelho está sendo drenada mais rápido do que o normal, seguir utilizando a aplicação. Além do consumo, o desempenho do sistema é algo importante e facilmente afugenta novos usuários. Uma aplicação com baixo desempenho pode causar problemas como lentidão, travamentos e erros, o que pode levar a insatisfação do usuário, desinstalação da aplicação e até mesmo a problemas mais graves como perda monetária, por exemplo.

O desenvolvimento de aplicativos móveis é uma área em constante evolução e existem várias opções disponíveis para os desenvolvedores. Cada opção possui suas próprias vantagens e desvantagens, e a escolha do melhor caminho a seguir dependerá das necessidades específicas de cada projeto. Uma das opções mais populares é o desenvolvimento nativo, que envolve o uso de linguagens e ferramentas específicas para cada plataforma (como Swift e Objective-C para iOS e Java e Kotlin para Android). Isso geralmente resulta em aplicativos com desempenho superior. Mas isso pode ser mais trabalhoso e caro devido à necessidade de criar e manter dois códigos-fontes diferentes. Além disso, existem ferramentas de desenvolvimento como Flutter, React Native, Ionic e Xamarin, que possibilitam o desenvolvimento de aplicativos multiplataforma, com o potencial de alcançarem excelente desempenho quando comparados a aplicações nativas e facilitarem o processo de desenvolvimento ao diminuírem a necessidade de projetos distintos para cada plataforma.

1.1 Objetivos

Considerando tudo o que foi apresentado até agora, o presente trabalho apresenta um plugin Flutter para ser utilizado durante o desenvolvimento de aplicações Flutter. Este visa auxiliar o desenvolvedor a identificar, enquanto programa o sistema, possíveis mudanças que afetem o desempenho da aplicação, aumentando a velocidade de detecção de *bugs*.

O presente trabalho tem como objetivo principal entregar um plugin para auxiliar desenvolvedores Flutter a identificarem problemas de desempenho em suas aplicações enquanto as desenvolvem. Dentre os objetivos específicos almejados durante o desenvolvimento do plugin estão:

- Ser simples para o desenvolvedor usar;
- Fazer a interface a menos invasiva possível;
- Apresentar indicativo da geração de quadros por segundo;
- Mostrar indicativo do uso de memória;
- Visualizar indicativo do tráfego de rede.

1.2 Justificativa

Flutter possui um conjunto de ferramentas disponíveis para depuração de código e análise de desempenho, chamado DevTools apresentado oficialmente pela documentação Google (2023b). Trata-se de uma ferramenta poderosa que auxilia desenvolvedores a entender como sua aplicação está desempenhando. Porém, ainda assim, este é um processo complexo e, de certo modo, custoso, dado que requer um acompanhamento externo à aplicação. O desenvolvedor necessita acessar a interface do DevTools através de um link disponibilizado pelo SDK Flutter quando a aplicação é colocada em execução através do SDK, como pode-se observar através na Figura 1.

Figura 1 – Exemplo execução de uma aplicação Flutter através do SDK em linha de comando

Fonte: Autor

Além do fato de ser uma aplicação externa ao sistema, sendo desenvolvido em Flutter, outro ponto importante que foi considerado para a elaboração deste trabalho é que exatamente por ser uma ferramenta tão robusta e poderosa, sua interface é considerada por muitos como sendo de difícil entendimento, possuindo uma curva de aprendizagem que assusta muitos desenvolvedores, causando assim receio da utilização da ferramenta. Na Figura 2 pode ser visto a interface do DevTools quando acessado pelo navegador Google Chrome no sistema operacional macOS.

1.3 Organização deste trabalho

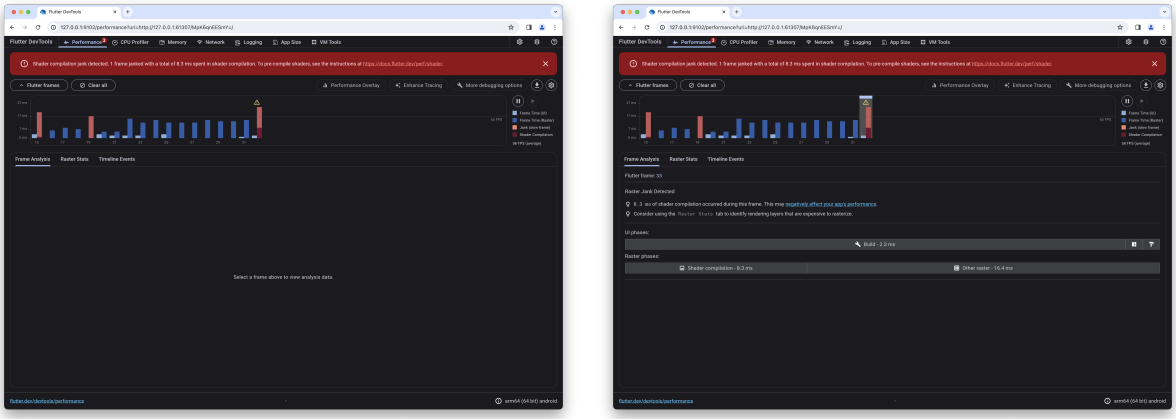
Este trabalho está organizado em 56 capítulos.

No Capítulo 2 encontra-se a apresentação de pacotes relacionados que foram encontrados no repositório oficial do Flutter e Dart, estes pacotes são considerados trabalhos relacionados devido a natureza deste trabalho.

Dentro do Capítulo 3 é abordado como o plugin foi desenvolvido, descrevendo quais são as funcionalidades e limitações do mesmo.

No Capítulo 4 são apresentados os resultados deste trabalho, mostrando, através de capturas de tela, a versão atual da interface do plugin. Por fim, no Capítulo 5 é descrita a conclusão deste trabalho e possíveis trabalhos futuros.

Figura 2 – Página do DevTools disponibilizada pelo link da Figura 1



Fonte: Autor

2 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos relacionados encontrados na literatura acadêmica e na Internet. Para a pesquisa dos trabalhos foram utilizadas as bases *Institute of Electrical and Electronics Engineers* (IEEE), através do portal *IEEE Xplore* e *Association for Computing Machinery* (ACM), através do portal *ACM Digital Library*, com as chaves de busca: **Flutter**, **Flutter Performance**, **Flutter Memory**, **Flutter Evaluation**, **Mobile Performance**, **Android**, **Android Performance**, **Android Memory**, **iOS**, **iOS Performance**, **iOS Memory**. Notou-se uma dificuldade na busca por artigos relacionados que abordem o domínio deste trabalho, sendo este o desenvolvimento de uma ferramenta para monitoramento de recursos durante o desenvolvimento de aplicações.

Diante deste fato, foi necessário buscar referências na Internet em geral. Com isso, foram utilizadas as documentações oficiais do Flutter (GOOGLE, 2023c), Android (GOOGLE, 2023a), e iOS (APPLE, 2023), as quais abordam vários tópicos sobre desempenho, monitoramento de rede e de memória. Além das documentações oficiais, foram pesquisados artigos, postagens em *blogs* e repositórios na Internet que fossem ao encontro dos seguintes critérios:

- Ser sobre Flutter, Android ou iOS;
- Abordar monitoramento de *Central Processing Unit* (CPU), Memória RAM ou tráfego de rede.

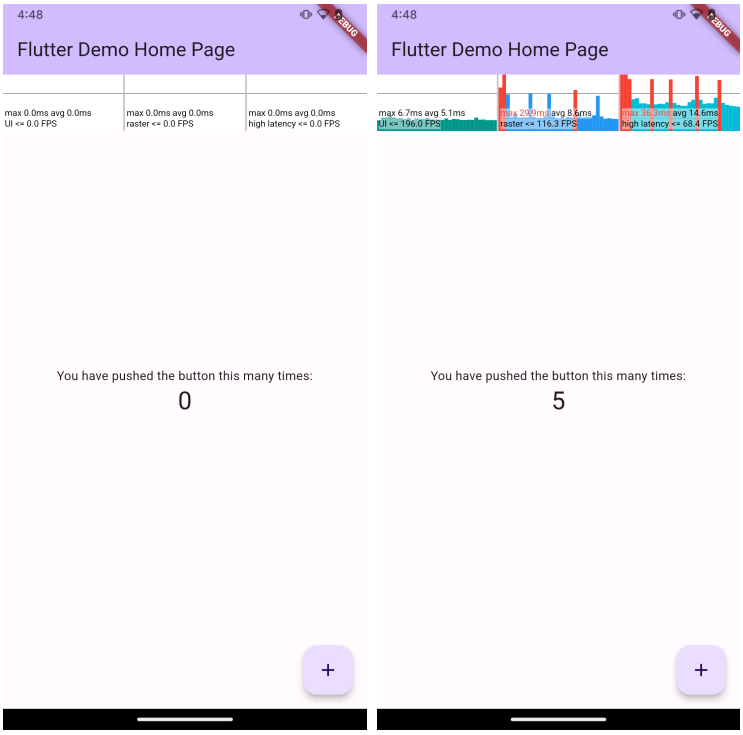
Todos os pacotes encontrados no repositório oficial de pacotes e *plugins* de Flutter e Dart, foram testados usando a aplicação base criada pelo Flutter quando o comando `flutter create {nome_para_projeto_flutter}` é executado. Os pacotes que mais se encaixaram com este trabalho, seguindo os critérios antes descritos são enumerados a seguir.

1. **performance**[1] - este pacote foi uma das inspirações para a criação deste trabalho. Ele cria sobre o aplicativo base três gráficos que apresentam, da esquerda para a direita, a geração de quadros pela *isolate* de UI, de *raster* e por fim o tempo total entre o início do *vsync* e a finalização da rasterização, respectivamente como pode ser visto nas Figura 3 e Figura 4.

   Um ponto importante deste pacote é que o mesmo só atualiza os gráficos quando realmente a tela é atualizada, como se pode ver na imagem da esquerda da Figura 3, em que sem interações com a aplicação, o gráfico permanece sem dados. Enquanto não houve uma necessidade da tela ser atualizada, nada é mostrado nos gráficos. Porém quando houve interação (foram apertadas cinco vezes o botão com um +

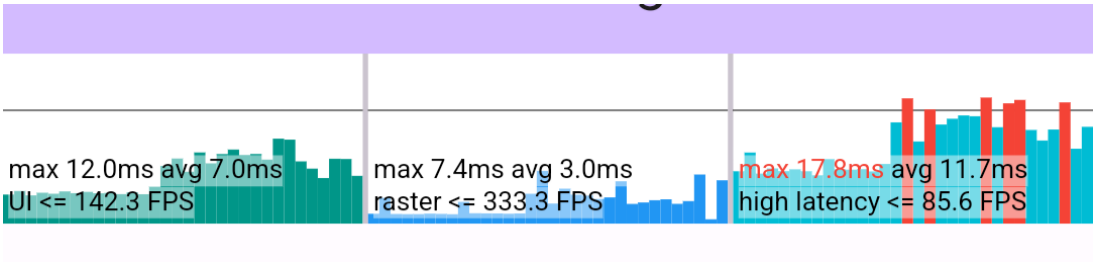
[1] https://github.com/creativecreatorormaybenot/performance

Figura 3 – Teste do pacote **performance**



Fonte: Autor

Figura 4 – Foco em uma execução do pacote **performance**



Fonte: Autor

localizado no canto inferior esquerdo), os gráficos foram atualizados, como pode-se ver na imagem da direita da Figura 3.

Para possuir tal comportamento, este pacote utiliza de recursos próprios do Flutter para a aquisição dos dados usados na geração dos gráficos. Deparando-se com esta situação, o pensamento seguinte pensamento surgiu: "Se é possível coletar essas informações, direto no Flutter, provavelmente deve ser possível coletar outros dados também". A partir deste ponto, surgiu então a ideia de buscar modos de capturar outros tipos de dados para fazer o monitoramento do aparelho enquanto a aplicação

é desenvolvida. Com isso, o foco deste trabalho foi determinado.

2. **performance_fps**[2] - outro pacote que se propõe a mostrar a taxa de *Frames Per Second* (FPS) da aplicação. Porém, diferentemente do pacote anterior, este não apresenta nenhum elemento de interface, somente fica disponível ao desenvolvedor chamar uma função que retorna dois valores, sendo estes: FPS e *dropCount*. Além disso, este *plugin* utiliza mecanismos específicos das plataformas destino para a obtenção dos dados, mecanismos estes que já se encontram descontinuados.

3. **lemberfpsmonitor**[3] - de simples instalação e utilização dentro do código, este pacote apresenta ao desenvolvedor um gráfico e três informações, sendo estas a taxa de quadros por segundo, no canto superior direito de sua interface, bem como ao lado desta, entre parenteses, a menor e a maior taxas de quadros registradas dentro da janela de amostras.

   Vale ser destacado que ao seguir-se o exemplo presente no repositório do pacote os resultados são os presentes nas imagens presentes na Figura 5. Um ponto válido de observação é o travamento a 60 FPSs. Esse comportamento pode ser mudado passando um novo valor para o parâmetro *maxFPS* ao *widget* do pacote. Por padrão ele possui valor de 60.

   Também é interessante notar na Figura 5 que mesmo sem interações com a aplicação, o contador já está sendo atualizado. Esse comportamento se dá devido a origem dos dados e ao modo como a reatividade foi arquitetada no pacote. Diferentemente do pacote **performance**, este aqui não usa métodos sofisticados para a aquisição dos dados. Ao invés de modificar o gráfico reagindo somente quando um novo quadro for gerado, esta ferramenta realiza chamadas periódicas para atualizar o gráfico, fazendo assim com que os gráficos sejam redesenhados sem nem mesmo haver interação para com a interface do sistema.

4. **statsfl**[4] - este pacote tem a mesma interface do anterior, devido ao fato de que o pacote **lemberfpsmonitor** é inspirado neste, bem como sua própria documentação oficial informa.

   Porém este pacote não apresenta o gráfico presente no outro, mostrando assim somente os valores e uma pequena linha representando quais quadros alcançaram o alvo de 60 FPS, como pode-se ver na Figura 6, que contém um teste completo e outro com foco na ferramenta.

5. **show_fps**[5] - Pacote simples, fornece um visualizador de FPS. Porém foram encontrados problemas na execução do exemplo de código presente na página oficial

[2] https://github.com/allenymt/flutter_fps
[3] https://github.com/lember-ecu/fpsmonitoring
[4] https://github.com/gskinnerTeam/flutter-stats-fl
[5] https://github.com/mantreshkhurana/show_fps

Figura 5 – Testes do pacote **lemberfpsmonitor**

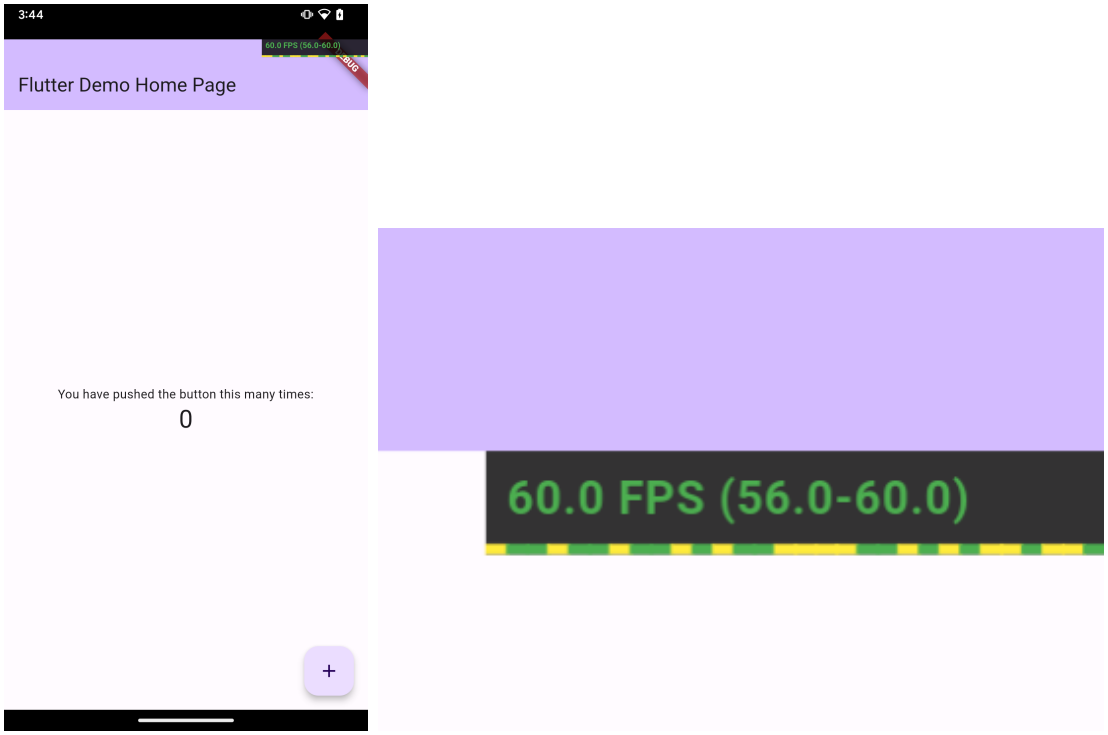
Fonte: Autor

do pacote. Ao implementá-lo, seguindo as informações fornecidas, o contador de quadros não era mostrado na aplicação. Somente um texto "FPS:" no canto superior direito da página foi apresentado. Com a mudança do parâmetro *showChart* de *false* para *true*, o contador passou a funcionar junto com o gráfico, como mostra a Figura 7 em que são apresentados dois testes feitos com a ferramenta, contendo uma representação completa e outra com foco na interface do pacote. Um ponto importante constatado neste pacote é que a atualização do gráfico é muito rápida, sendo difícil de acompanhar os eventos.

6. **memory_info**[6] - Sem uma interface de apresentação de dados pronta para o desenvolvedor, este pacote disponibiliza alguns métodos que retornam informações de memória RAM e armazenamento interno do aparelho. Sendo algumas destas informações: total de memória RAM do dispositivo, quantidade de memória RAM livre e tamanho total de armazenamento interno bem como quando espaço há livre deste.

[6] https://github.com/MrOlolo/memory_info/tree/master/memory_info

Figura 6 – Testes do pacote **statsfl**



Fonte: Autor

Figura 7 – Testes do pacote **show_fps**

Fonte: Autor

3 METODOLOGIA

Este capítulo descreve o processo de criação de uma ferramenta de desempenho de aplicativos.

Os objetivos da ferramenta foram definidos com base nos fatores considerados na primeira etapa do trabalho de conclusão de curso, a saber:

- Uso de CPU: A quantidade de poder de processamento que o aplicativo requer e o impacto no desempenho do dispositivo;
- Uso de memória: A quantidade de memória que a aplicação usa enquanto está em execução.
- Uso de bateria: O impacto que o aplicativo tem na vida útil da bateria do dispositivo.
- Uso de rede: A quantidade de dados que o sistema transfere pela rede e o impacto no desempenho da rede.
- Tempo de inicialização: O tempo que leva para o aplicativo ser iniciado e ficar pronto para uso.
- Taxa de quadros: O número de quadros por segundo que a aplicação é capaz de exibir, o que pode impactar a suavidade de animações e rolagem.
- Latência da experiência: O tempo que leva para o sistema responder a entrada do usuário.
- Taxa de *crashes*: O número de *crashes* por usuário ou sessão.

Todos esses pontos são importantes e devem ser analisados durante o desenvolvimento e vida de uma aplicação. No entanto, na maioria das vezes, para identificar um ou mais problemas, vários desses fatores devem ser observados e analisados para encontrar maneiras de corrigi-los.

É importante lembrar que os fatores mais importantes dependerão do aplicativo e do seu uso previsto. Também é válido avaliar o desempenho da aplicação em diferentes dispositivos e em diferentes condições para obter uma compreensão abrangente de seu desempenho.

Dentre todos esses fatores, três foram escolhidos para a primeira implementação da ferramenta: uso de memória, uso de rede e taxa de quadros. Utilizando esses três fatores em conjunto, já é possível para o desenvolvedor entender como é o comportamento normal de sua aplicação enquanto a desenvolve. A cada nova alteração, pode ser visualizada em tempo real quais foram os impactos analisando esses três fatores.

A ferramenta desenvolvida permite que os desenvolvedores avaliem o desempenho de seus aplicativos de forma rápida e fácil. Ela fornece informações valiosas que podem ser usadas para melhorar o desempenho dos aplicativos e proporcionar uma melhor experiência aos usuários.

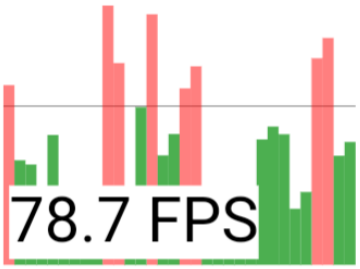
3.1 Contador de FPS

O desenvolvimento da ferramenta iniciou-se pela etapa com mais referências e exemplos de implementação. Foram encontrados cinco pacotes que disponibilizam a funcionalidade desejada de exibição de quadros por segundo, como apresentado anteriormente no Capítulo 2. No entanto, após a análise do funcionamento e das implementações, somente o pacote **performance** foi escolhido. Ele foi analisado em profundidade para compreender como os dados são obtidos e gerenciados até serem apresentados na tela.

A utilização de `SchedulerBinding` é o grande mecanismo por trás do pacote **performance**. Ele é descrito em Flutter (2023), sendo este uma ferramenta especial chamada de `mixin`, disponível na biblioteca `scheduler` presente no Flutter fazendo desnecessária a instalação de módulos externos. Com `SchedulerBinding` é adquirido uma lista de `FrameTiming`, classe disponível na ***dart:ui***, uma biblioteca que disponibiliza os serviços de baixo nível para o *framework* do Flutter usar na inicialização das aplicações.

Na classe `FrameTiming` são armazenadas as métricas relacionadas a tempo de um quadro, como por exemplo: `vsyncStart, buildStart, buildFinish, rasterStart, rasterFinish`. Somando isso com a reatividade e os *callbacks* do `SchedulerBinding`, temos um modo muito simples e efetivo de coletar os dados necessários para apresentar no contador de quadros. A Figura 8 mostra como ficou a interface do contador de quadros por segundo.

Figura 8 – Contador de quadros por segundo



Fonte: Autor

3.2 Uso de Memória

O monitoramento de memória foi a tarefa mais complexa do desenvolvimento da ferramenta. Inicialmente, a ideia era listar e mostrar em tempo real os objetos em memória criados dentro da aplicação que mais estavam consumindo memória RAM.

Procurou-se entender como a ferramenta DevTools adquire informações sobre memória. Para isso, foi acessado o repositório oficial da ferramenta[1] e pesquisado pelo módulo de memória. A leitura do código foi iniciada para compreender seu funcionamento, mas foi decidido abandonar esta abordagem devido à grande complexidade e quantidade de código necessária para sua implementação. Seria necessário implementar centenas de linhas de código que são desenvolvidas no DevTools, apenas para utilizar algumas poucas funcionalidades.

Após pesquisas na internet, foi encontrada a classe `MemoryAllocations`, disponível na biblioteca **foundation** do Flutter. Essa classe permite escutar eventos do ciclo de vida dos objetos, ou seja, quando um objeto é criado ou coletado pelo coletor de lixo, o código que chama a função é avisado.

A identificação dos eventos não era um problema. O desafio era listar em tempo real os objetos ativos na memória, ordenados por tamanho ocupado, do maior para o menor. O problema é que os eventos notificados são de todos os objetos gerenciados durante a execução da aplicação, alcançando a casa de dezenas de milhares de eventos por segundo em testes realizados. Isso gera um desafio imenso de criar e manter uma estrutura de dados atualizada com somente os objetos ativos na memória, ou seja, que ainda não tenham sido coletados pelo coletor de lixo. Além disso, a estrutura deveria priorizar a leitura dos objetos que mais ocupam espaço na RAM.

Durante meses, foi buscada uma solução viável para este problema, mas infelizmente não foi alcançada. O maior problema foi o gasto computacional, não sendo alcançado uma otimização de código para tornar o processamento dos dados dos eventos de memória e a gestão da estrutura de dados eficiente.

Outra abordagem que não foi tentada foi aplicar os algoritmos criados em Dart na linguagem C. Através da biblioteca **dart:ffi**, é possível utilizar código C dentro de aplicações desenvolvidas com Dart. Não é possível afirmar que isso tornaria viável a disponibilidade desta funcionalidade dentro do *plugin*, mas é uma alternativa que por falta de tempo, não foi testada.

Por fim, para esta opção não ser removida da primeira versão do *plugin*, foi decidido informar para o desenvolvedor dados básicos sobre a memória RAM do dispositivo. O *plugin* mostra três informações, que podem ser observadas na Figura 9, sendo estas: Memória total do dispositivo em *gigabytes*; Quantidade total de memória sendo utilizada no dispositivo; Quantidade de memória livre no dispositivo.

[1] https://github.com/flutter/devtools

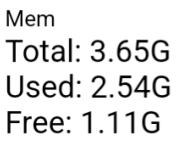
Diferentemente da taxa de quadros, que foi desenvolvida somente com código Dart, para conseguir ter acesso às informações do *hardware* onde a aplicação está sendo executada, fez-se necessário ir a nível de plataforma devido a algumas limitações do Flutter. Então, ao invés deste trabalho ser um pacote, que somente possui códigos Dart, ele é um *plugin* pois possui códigos específicos da plataforma Android.

Usando Kotlin como linguagem de programação para o desenvolvimento específico para Android, utilizou-se a classe `MemoryInfo` para retornar através do `MethodChannel` as informações para o código Dart e com isso apresentar na tela os dados da memória.

A rotina que busca as informações da plataforma não utiliza de reatividades internas do Flutter para ser executada. Ela é invocada a cada segundo dentro de uma função em código Dart. Por causa deste comportamento, quando o monitor de quadros é executado sozinho em uma aplicação recém-criada, por exemplo, ele tem comportamento igual ao pacote **performance**, em que somente quando há alterações na interface há atualizações no gráfico. Porém, quando o módulo de monitoramento de memória ou tráfego de rede são executados junto com o monitor de quadros, a cada segundo há uma atualização da interface e o monitor atualiza seu gráfico.

Esta solução ainda não é a que mais agrada o autor, porém, para a versão inicial do *plugin*, é uma métrica que pode ajudar o desenvolvedor a identificar um consumo anormal de memória RAM.

Figura 9 – Informações sobre memória RAM



Fonte: Autor

3.3 Tráfego de Rede

O objetivo deste monitor de tráfego é sinalizar ao desenvolvedor a quantidade de dados que entram e saem do dispositivo pela Internet. Atualmente, é comum que aplicações móveis sejam apenas *front-ends* para *Application Programming Interface*s (APIs), portanto, é comum enviar e receber informações pela Internet. Com um monitor de tráfego de rede, o desenvolvedor pode identificar a quantidade de dados que são enviados durante uma solicitação e recebidos em uma resposta. Isto pode ajudar a identificar anomalias no comportamento da aplicação, como por exemplo, quando uma quantidade X de dados é esperada ser recebida ao ser feita uma requisição a uma API, porém uma quanti-

dade Y está sendo retornada. Isso indica alguma anormalidade que pode ser identificada com o monitor de tráfego.

No início, foram pesquisados modos de adquirir dados de rede usando recursos disponíveis no Flutter. No entanto, foi descoberto que Flutter não possui acesso direto à camada de rede dos dispositivos. Essa camada está disponível apenas por meio de APIs do sistema. Como acessar a camada de rede por meio do Flutter e contar os dados que entram ou saem do dispositivo não poderias ser uma opção, foi então pesquisada outra abordagem para resolver este problema.

O segundo conceito estudado para implementação foi interceptar as solicitações feitas pela aplicação que importa o *plugin*. No entanto, essa ideia provou-se não ser adequada. Normalmente, os desenvolvedores Flutter usam pacotes dedicados para fazer solicitações *Hypertext Transfer Protocol* (HTTP) à Internet. Portanto, seria necessário analisar como cada um desses pacotes funciona e procurar brechas que permitissem a captura dos pacotes.

Além de ser uma solução ineficiente em termos de tempo e esforço, também é provável que existam mecanismos de segurança que impeçam essa abordagem.

Não sendo possível realizar a aquisição de dados por meio do Flutter ou Dart, foram pesquisadas soluções voltadas para plataformas específicas. Neste caso, também apenas para Android nesta primeira versão.

Com acesso às APIs do Android, foram investigadas formas de acessar os valores desejados. Primeiramente, foi encontrada a classe `TrafficStats`, que é a mais citada na Internet para captura dos valores.

O objetivo era capturar as informações enviadas e recebidas apenas pela aplicação. Para isso, foram encontrados os métodos `getUidTxBytes` e `getUidRxBytes`, para pegar a quantidade de *bytes* enviadas e recebidas, respectivamente.

Para surpresa, os dados não eram consistentes. Estava claro que algo não estava certo. Foi verificado novamente se a captura do *Unique Identifier* (UID) estava sendo feita de maneira correta. Sim, estava correta.

Dada a situação, partiu-se para a implementação de uma classe que é indicada para acessar históricos e informações detalhadas sobre a rede do dispositivo. Esta classe chama-se `NetworkStatsManager`, sendo esta até mesmo indicada para uso na página oficial do `TrafficStats`.

Dentro desta classe, foi encontrado um método muito interessante chamado `queryDetailsForU` que retorna detalhes estatísticos sobre o uso de rede de um UID. Com isso, é possível acessar as informações sobre a aplicação. Contudo, este método tem um preço: tempo. Na própria documentação é descrito que este método demora muito e não deve ser invocado pela thread principal da aplicação.

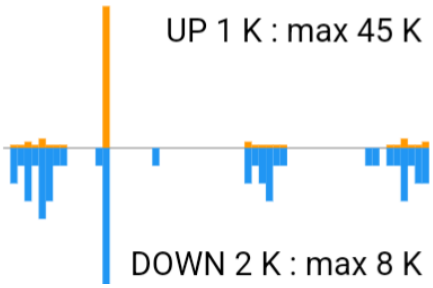
Mesmo sabendo-se disso, esta alternativa foi implementada e o método funcionava através de uma thread dedicada. No entanto, dada a natureza e a proposta do plugin, a

utilização deste método não poderia ser mantida, ele é muito custoso computacionalmente.

Após retornar à estaca zero, a atenção voltou-se novamente para a classe `TrafficStats` e seus métodos. Foi novamente tentado usar os métodos já citados desta, mas novamente houve a mesma inconsistência. Em seguida, buscou-se então usar um novo método desta mesma classe.

Ao invés de tentar pegar somente dados da aplicação, utilizaram-se os métodos `getTotalTxBytes` e `getTotalRxBytes` para pegar o número de bytes enviados e recebidos por todo o sistema, respectivamente e o resultado pode ser visualizado na Figura 10. Como esta classe coleta os dados na camada de rede, ela não diferencia entre pacotes *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP).

Figura 10 – Gráfico de perfil de rede



Fonte: Autor

3.4 Parâmetros de customização

O plugin possui diversos parâmetros que podem ser usados para a customização do mesmo, facilitando assim ao desenvolvedor manipular os módulos na tela.

1. **bool disable**: Indica se o plugin será ou não desabilitado, impedindo o mesmo de ser inicializado. Seu valor padão é *falso*.

2. **bool activateFPS**: Indica se o módulo de geração de quadros será ou não ativado. Seu valor padrão é *verdadeiro*.

3. **double scaleFPS**: Indica a escala que o módulo de geração de quadros será renderizado. Seu valor padrão é *1*.

4. **Alignment alignmentFPS**: Indica qual o alinhamento do módulo de geração de quadros dentro da tela do dispositivo. Seu valor padrão é *Alignment.topRight*.

5. **int sampleSizeFPS**: Indica o número de amostras *frame time* que serão extrapoladas para o módulo de geração de quadros. Seu valor padrão é *32*.

6. **Duration targetFrameTimeFPS**: Indica o tempo máximo que um quadro pode levar para ser construido, todos quadros que levarem mais tempo do que o valor deste parâmetro, serão desenhados em vermelho, caso contrário, serão desenhados na cor do parâmetro *chartBarsColor*. Este parâmetro indica também ao plugin onde desenhar a linha preta horizontal presente no módulo de geração de quadros. Seu valor padrão é *16.7 milisegundos*, o que corresponde a 60 quadros por segundo.

7. **Color backgroundFPS**: Indica a cor de fundo do módulo de geração de quadros. Seu valor padrão é *Color(0x00ffffff)*.

8. **Color textColorFPS**: Indica a cor do texto do módulo de geração de quadros. Seu valor padrão é *Color(0xff000000)*.

9. **Color textBackgroundColorFPS**: Indica a cor de fundo do texto do módulo de geração de quadros. Seu valor padrão é *Color(0xffffffff)*.

10. **Color chartBarsColor**: Indica a cor dos retângulos representando os quadros superiores ao *targetFrameTimeFPS* do módulo de geração de quadros. Seu valor padrão é *Color(0xff4caf50)*.

11. **bool activateMemory**: Indica se o módulo de memória será ou não ativado. Seu valor padrão é *verdadeiro*.

12. **Alignment alignmentMemory**: Indica qual o alinhamento do módulo de memória dentro da tela do dispositivo. Seu valor padrão é *Alignment.topRight*.

13. **double scaleMemory**: Indica a escala que o módulo de memória será renderizado. Seu valor padrão é *1*.

14. **Color backgroundMemory**: Indica a cor de fundo do módulo de memória. Seu valor padrão é *Color(0x00ffffff)*.

15. **Color textColorMemory**: Indica a cor do texto do módulo de memória. Seu valor padrão é *Color(0xff000000)*.

16. **Color textBackgroundColorMemory**: Indica a cor de fundo do texto do módulo de memória. Seu valor padrão é *Color(0xffffffff)*.

17. **bool activateNetwork**: Indica se o módulo de rede será ou não ativado. Seu valor padrão é *verdadeiro*.

18. **Alignment alignmentNetwork**: Indica qual o alinhamento do módulo de rede dentro da tela do dispositivo. Seu valor padrão é *Alignment.topLeft*.

19. **double scaleNetwork**: Indica a escala que o módulo de rede será renderizado. Seu valor padrão é *1*.

20. **Color backgroundNetwork**: Indica a cor de fundo do módulo de rede. Seu valor padrão é *Color(0x00ffffff)*.

21. **Color textColorNetwork**: Indica a cor do texto do módulo de rede. Seu valor padrão é *Color(0xff000000)*.

22. **Color textBackgroundNetwork**: Indica a cor de fundo do texto do módulo de rede. Seu valor padrão é *Color(0xffffffff)*.

23. **int timeBox**: Indica quantos segundos apareceram no gráfico de rede. Seu valor padrão é *60*

4 RESULTADOS

Este capítulo apresenta a interface do plugin desenvolvido, sendo utilizado através de dois aplicativos reais e um de teste. As aplicações são *Wonderous* e *OpenLeaf*, estando as duas disponíveis para download na loja de aplicativos *App Store*, além de que *Wonderous* também está disponível na *Play Store*. Além da disponibilidade nas lojas de aplicativos os dois possuem repositórios públicos no GitHub. O aplicativo de teste apresentado também, foi criado durante o desenvolvimento deste *plugin*.

Os resultados apresentados neste capítulo, foram adquiridos ao executar as aplicações em um *smartphone* físico, sendo este um Motorola Z3 Play, com processador Qualcomm SDM636 Snapdragon 636 e 4GB de memória RAM. Com isto pode-se verificar o impacto das aplicações um *hardware* já datado, podendo ser considerado como um aparelho com caracteristicas de dispositivos de entrada atualmente.

Desta-se que mesmo sendo disponível a utilização do plugin em modo *debug*, para ter resultados mais condizentes com o real desempenho que a aplicação terá, é recomendado realizar análises de comportamento da aplicação no modo *profile* ou até mesmo *release*, para somente assim poder entender qual é a real capacidade de desempenho da mesma. Em modo *debug* há um *overhead* das funções utilizadas para depurar a aplicação que influenciam e prejudicam os valores encontrados durante a execução da aplicação.

4.1 Wonderous

Este aplicativo traz diversas informações sobre as mais famosas estruturas do mundo, como por exemplo: A Grande Muralha da China, o Cristo Redentor, as Pirâmides de Gizé, entre outros. É um aplicativo desenvolvido utilizando diversas funcionalidades de *layout* do Flutter, com o intuito de demonstrar a grande capacidade de estilização e desenvolvimento de animações.

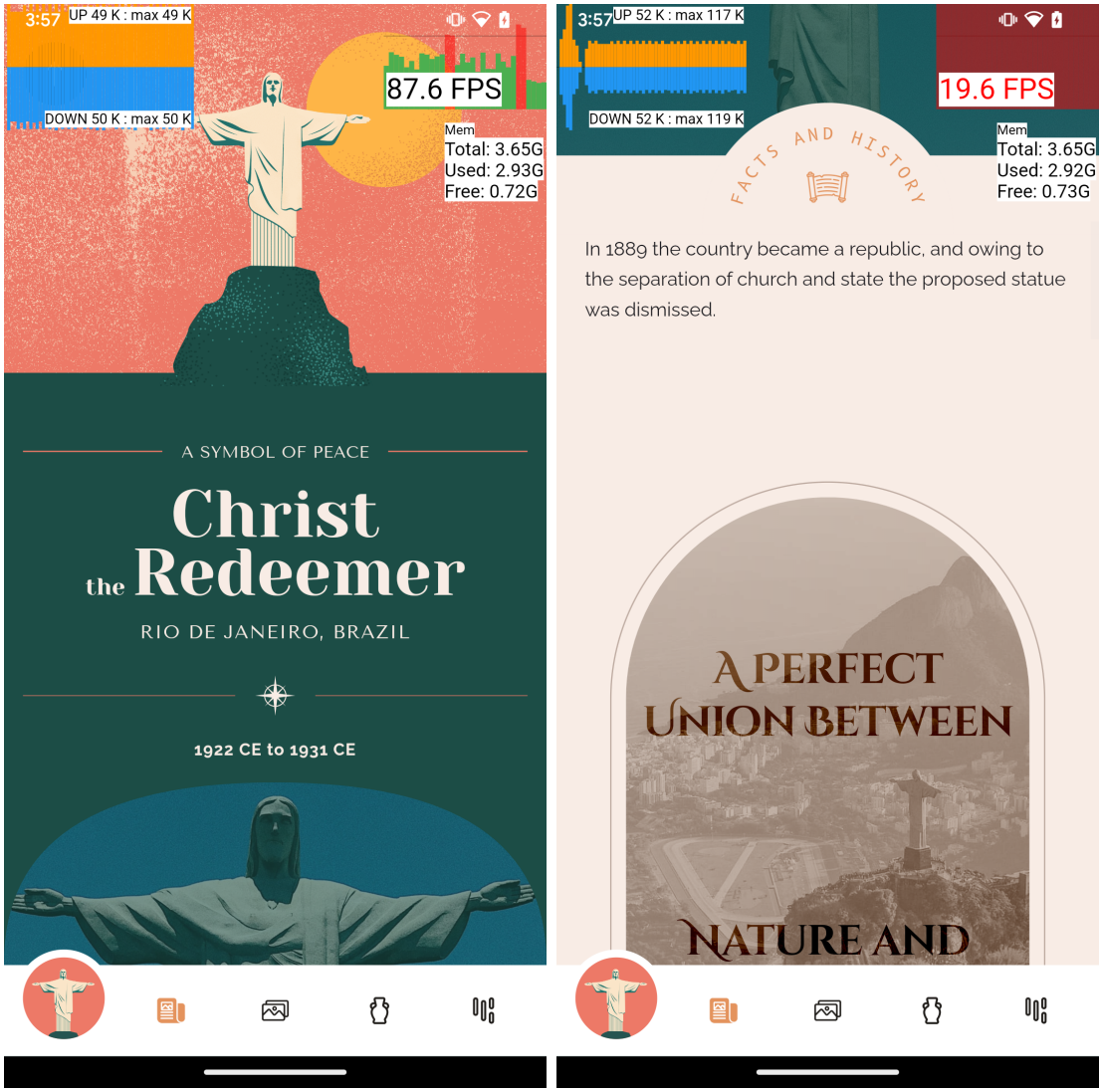
Na Figura 11 é apresentado a página da aplicação Wonderous que traz informações sobre o Cristo Redentor, com as configurações padrões do plugin, onde foi constatado um trecho em que há uma queda brusca de desempenho. Justamente na seção da página onde há uma grande concentração de animações e efeitos visuais, a taxa de quadros gerados por segundo cai para 1/4 da presente no resto desta página.

Foi constatado o mesmo comportamento em todas outras páginas da aplicação que possuem o componente de *layout* repleto de animações. Na Figura 12 pode-se ver o comportamento, além de demonstrar também a utilização de configurações customizadas de alinhamento dos componentes do plugin.

4.2 OpenLeaf

Este aplicativo funciona como uma biblioteca virtual, tendo disponíveis diversos livros de domínio público, além disso também possui a funcionalidade de leitor de arquivos

Figura 11 – Página sobre Cristo Redentor com plugin ativo



Fonte: Autor

PDF.

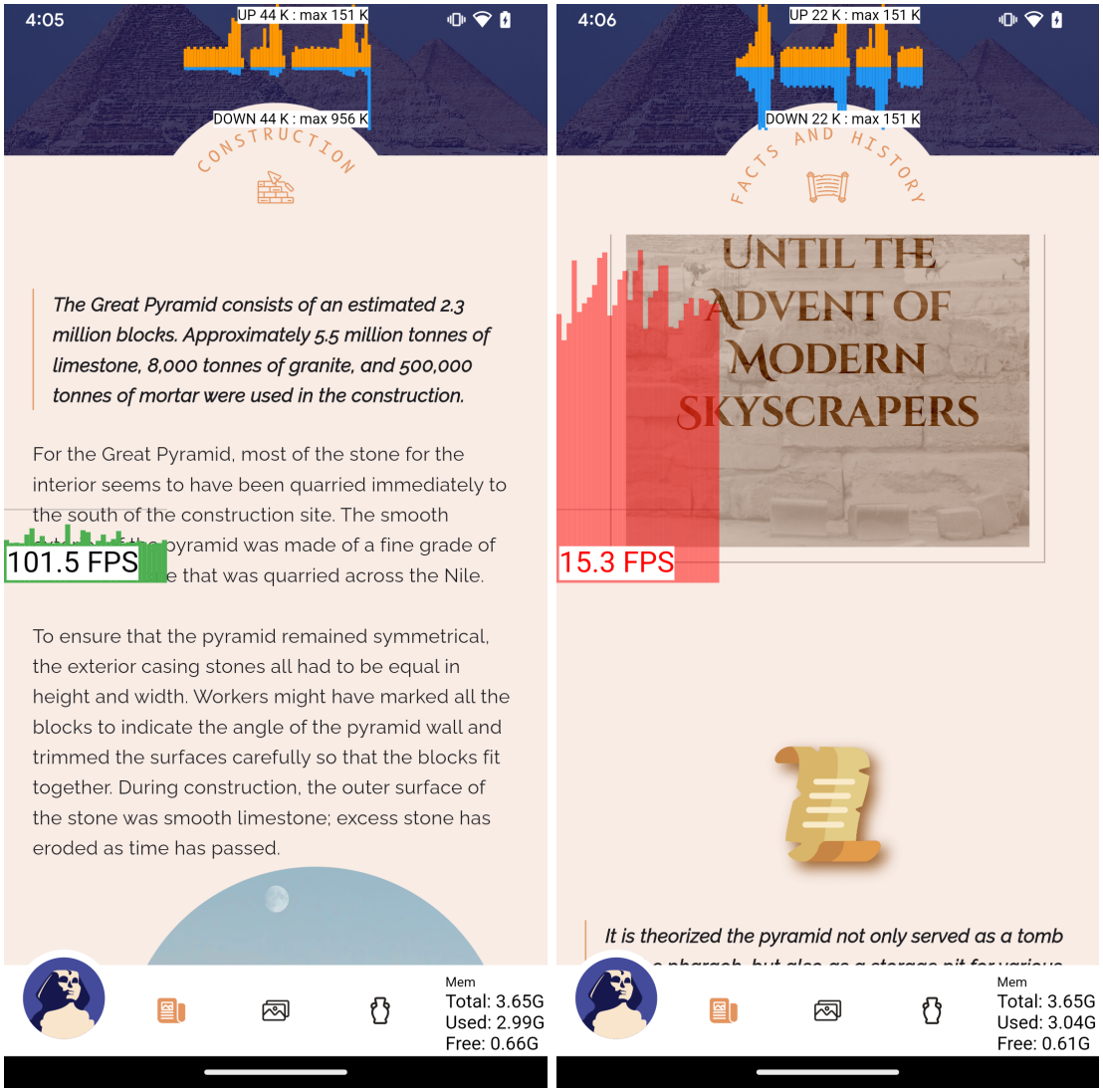
A Figura 13 apresenta uma comparação entre o desempenho normal da aplicação quando sua interface está carregada, entregando uma geração de quadros excelente no dispositivo de teste e o momento em que uma nova página é carregada.

4.3 Aplicativo de exemplo

Este aplicativo foi criado para testar o plugin durante o desenvolvimento do mesmo, executando ações simples como: requisições HTTP para uma API online quando solicitado, incremento de variável através de interação com botão e geração de uma lista com 1000000 de itens, apresentada em outra página do aplicativo.

A Figura 14 apresenta uma comparação entre o comportamento da aplicação antes

Figura 12 – Página sobre A Grande Pirâmide com plugin ativo

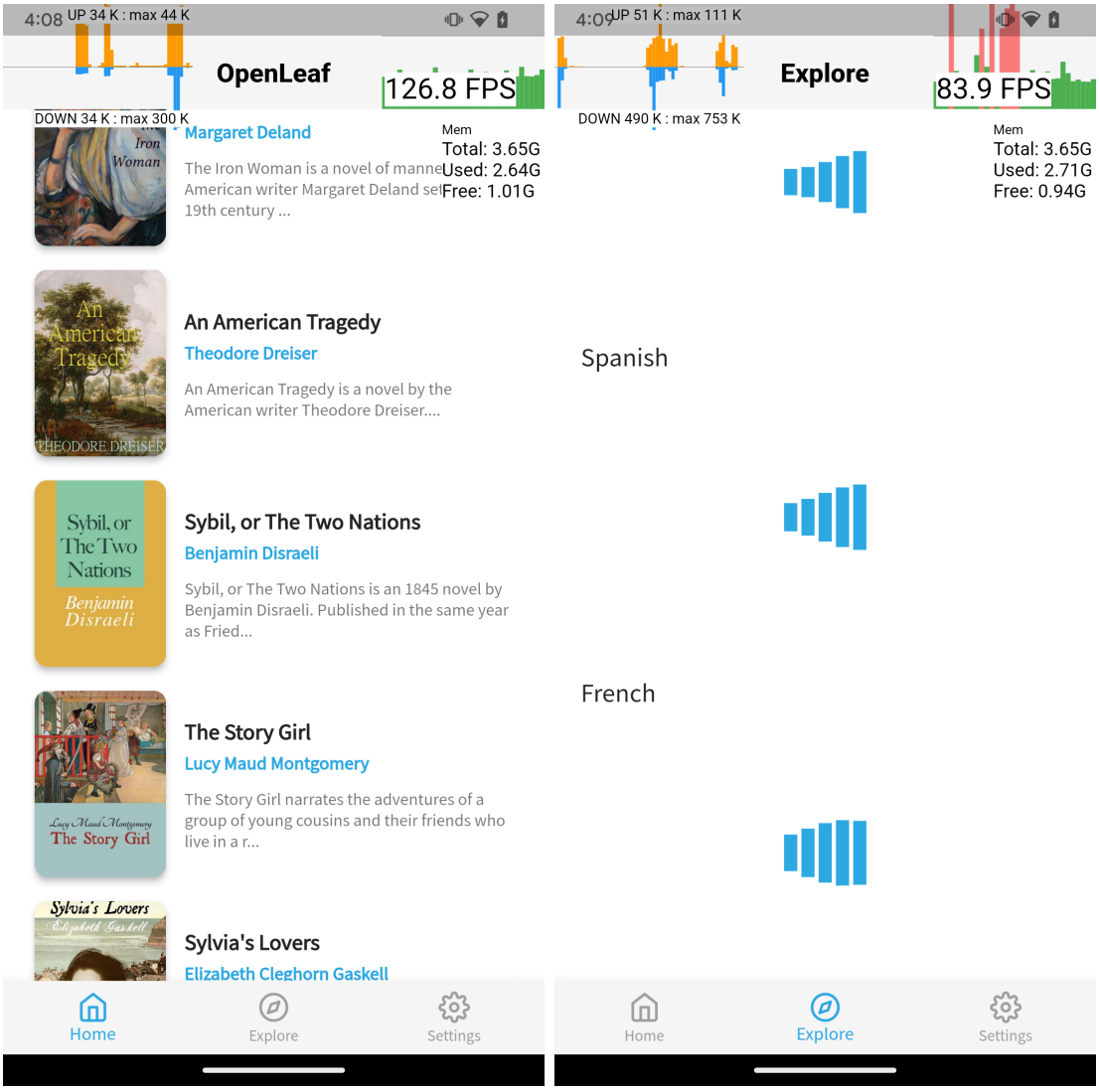


Fonte: Autor

e após uma requisição HTTP ser feita.

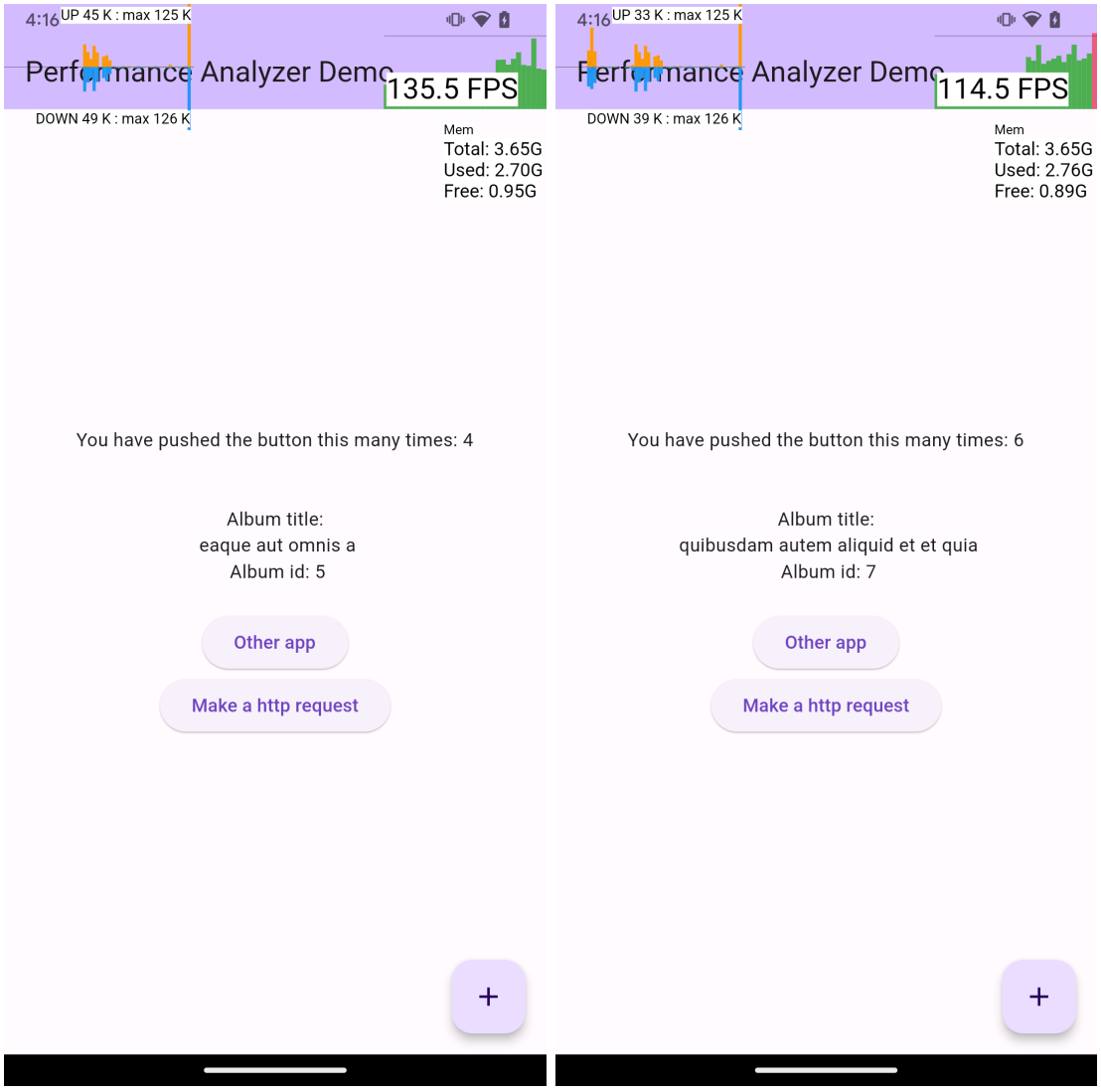
Destaca-se que a primeira versão do *plugin* já foi publicada no repositório oficial de pacotes para aplicações Dart e Flutter, bem como possui um repositório aberto, sendo possível acompanhar todo o desenvolvimento do mesmo.

Figura 13 – Comparação entre página carregada e em carregamento do aplicativo OpenLeaf



Fonte: Autor

Figura 14 – Comparação do comportamento da aplicação ao ser feita uma requisição para a Internet



Fonte: Autor

5 CONSIDERAÇÕES FINAIS

Neste trabalho de conclusão de curso foram apresentadas as motivações para a criação de um plugin para a mensuração de performance de aplicações desenvolvidas em Flutter. A ferramente provê a visualização do consumo de recursos de rendenização de quadros, quantidade de memória disponível e consumida e envio e recebimento de bytes via rede. O trabalho discorre sobre as etapas percorridas durante o desenvolvimento do software.

As contribuições deste trabalho são: (a) a criação de uma ferramenta alternativa para auxiliar desenvolvedores Flutter a identificarem problemas de desempenho em suas aplicações; (b) uma extensão da bibliografia sobre Flutter, dado que ainda é um tópico ainda não muito presente em trabalhos acadêmicos.

O plugin foi disponibilizado publicamente e gratuitamente no repositório oficial de pacotes e plugins do Dart e Flutter[1]. O código-fonte do plugin também está hospedado no repositório de versionamento da plataforma Github[2], também de forma pública e gratuita. Com a disponibilização do trabalho para a comunidade de Flutter, espera-se que o plugin seja difundido entre outros desenvolvedores.

Como trabalhos futuros tem-se a análise da viabilidade da utilização das funcionalidades introduzidas na versão 3.16 de Flutter, que tornou recentemente disponível, para autores de pacotes, a extensão de funcionalidades do DevTools. Pretende-se também avaliar o impacto computacional que o uso do plugin traz para o desempenho das aplicações, isto é a intrusão. É possível ainda a criação de projetos com o mesmo intuito do que o que foi aqui exposto em outras ferramentas utilizadas para o desenvolvimento de aplicações, visando auxiliar os desenvolvedores que utilizam estas ferramentas.

[1] <https://pub.dev/packages/perf_view>
[2] <https://github.com/bedinotto/perf_view>

REFERÊNCIAS

APPLE. **iOS**. 2023. Disponível em: <https://developer.apple.com/documentation/technologies>. Citado na página 23.

CAMINADE, M. v. W. J. **The Success of Third-Party Apps on the App Store**. 2022. Disponível em: <https://www.apple.com/newsroom/pdfs/the-success-of-third-party-apps-on-the-app-store.pdf>. Citado na página 19.

CECI, L. **How many apps are there in the Play Store?** 2023. Disponível em: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Citado na página 19.

FLUTTER. **SchedulerBinding**. 2023. Disponível em: <https://github.com/flutter/flutter/blob/master/packages/flutter/lib/src/scheduler/binding.dart>. Citado na página 30.

GOOGLE. **Android**. 2023. Disponível em: <https://developer.android.com>. Citado na página 23.

GOOGLE. **Flutter**. 2023. Disponível em: <https://docs.flutter.dev/tools/devtools/overview>. Citado na página 21.

GOOGLE. **Flutter**. 2023. Disponível em: <https://docs.flutter.dev>. Citado na página 23.

HORN, R. et al. Native vs web apps: Comparing the energy consumption and performance of android apps and their web counterparts. In: **2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)**. [S.l.: s.n.], 2023. p. 44–54. Citado na página 19.

TAYLOR, P. **Global figure of connected devices 2014-2028**. 2023. Disponível em: <https://www.statista.com/statistics/512650/worldwide-connected-devices-amount/>. Citado na página 19.