INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO SERTÃO PERNAMBUCANO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JEFFERSON BARROS DOS SANTOS

COMPARANDO O DESEMPENHO DE ABORDAGENS PARA CRIAÇÃO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS: REACT NATIVE VERSUS DESENVOLVIMENTO NATIVO

SALGUEIRO

2019

JEFFERSON BARROS DOS SANTOS

COMPARANDO O DESEMPENHO DE ABORDAGENS PARA CRIAÇÃO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS: REACT NATIVE VERSUS DESENVOLVIMENTO NATIVO

Trabalho de Conclusão de Curso apresentado a Coordenação do curso de Tecnologia em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia do Sertão Pernambucano, campus Salgueiro, como requisito parcial à obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador(a): Prof. José Júnior de Oliveira Silva.

SALGUEIRO

2019

Dados Internacionais de Catalogação na Publicação (CIP)

S237 Santos, Jefferson Barros dos.

Comparando o desempenho de abordagens para criação de aplicações para dispositivos móveis : react native versus desenvolvimento nativo / Jefferson Barros dos Santos. - Salgueiro, 2019.
31 f. : il.

Trabalho de Conclusão de Curso (Sistemas para Internet) -Instituto Federal de Educação, Ciência e Tecnologia do Sertão Pernambucano, Campus Salgueiro, 2019.
Orientação: Prof. Dr. José Júnior de Oliveira Silva.

1. Programação para Aplicativos Móveis. 2. React Native. 3. Framework. 4. JavaScript. 5. Java. I. Título.

CDD 005.258

Gerado automaticamente pelo sistema Geficat, mediante dados fornecidos pelo(a) autor(a)

JEFFERSON BARROS DOS SANTOS

COMPARANDO O DESEMPENHO DE ABORDAGENS PARA CRIAÇÃO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS: REACT NATIVE VERSUS DESENVOLVIMENTO NATIVO

Trabalho de Conclusão de Curso apresentado a Coordenação do curso de Tecnologia em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia do Sertão Pernambucano, campus Salgueiro, como requisito parcial à obtenção do título de Tecnólogo em Sistemas para Internet.

Aprovado em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. José Júnior de Oliveira Silva Orientador(a)
IFAL – Campus Batalha

_____
Prof. Francenila Rodrigues Júnior Souza
IF Sertão PE – Campus Salgueiro

_____
Prof. Orlando Silva de Oliveira
IF Sertão PE – Campus Salgueiro

SALGUEIRO

2019

RESUMO

A popularidade de aplicações desenvolvidas com o uso de algum *framework* multiplataforma intensificou-se nos últimos anos. Seja pelo baixo custo no desenvolvimento ou por utilizar tecnologias consolidadas no mercado e bastante usadas para construir outro tipo de aplicação por exemplo. O presente artigo tem o intuito de realizar um comparativo entre o desenvolvimento nativo *Android* e o multiplataforma utilizando o *framework React Native* utilizando parâmetros como uso de CPU e memória RAM, para identificar qual método de desenvolvimento é mais eficaz no quesito performance e tamanho da aplicação final.

Palavras-chave: *React Native*. *Framework*. *JavaScript*. *Java*.

ABSTRACT

The popularity of applications developed using some multiplatform framework has intensified in recent years. Either for the low cost of development or for using technologies consolidated in the market and widely used to build another type of application for example. This article aims to compare Android native development with multiplatform using the React Native framework using parameters such as CPU and RAM usage, to identify which development method is most effective in terms of performance and size of the final application.

Keywords*: React Native. Framework. JavaScript. Java.*

SUMÁRIO

1. Introdução

Atualmente o mundo está repleto de inovação e tecnologia, com vários dispositivos tecnológicos de ponta como as *Smart* TVs, relógios e pulseiras inteligentes, computadores quânticos e *smartphones* modernos. Em especial, um *smartphone* possui várias funcionalidades importantes para o dia a dia da grande maioria das pessoas, comparados aos modelos atuais, os primeiros aparelhos apresentavam limitações tecnológicas.

O primeiro aparelho a ser considerado um *smartphone*, chamado de IBM *Simon*, foi lançado em 1993 pela IBM (*International Business Machines Corporation*) em parceria com a *BellSouth*. O modelo lançado apresentava para a época, uma grande variedade de recursos de outros dispositivos como PDA (*Personal Digital Assistant*), transmissão de documentos via fax, jogos, calculadora, bloco de notas, dentre outras funcionalidades [INTERNATIONAL BUSINESS TIME 2012]. Uma das principais características desse modelo é a substituição do teclado físico por um *display* sensível ao toque, que nos dias de hoje, virou uma das principais características para qualquer dispositivo inteligente lançado no mercado.

Lançado em 2000, o *Ericsson* R380 ficou conhecido como o primeiro *smartphone* do mercado. Possuía uma grande tela sensível ao toque com uma capa *flip* incluindo um teclado físico comum, uma caneta para uso com a tela sensível ao toque e a versão 5 do *Symbian* OS, o sistema operacional desenvolvido pela *Symbian Corporation* com foco em PDAs [INTERNATIONAL BUSINESS TIME 2012].

Em 2007 ocorreu uma revolução com primeiro *smartphone* lançado pela *Apple*, o *iPhone*. O aparelho possuía uma tela sensível ao toque e capacidade de gerenciar múltiplos toques, um sensor de proximidade capaz de desligar a tela ao aproximá-lo do rosto em uma chamada telefônica, espaço de armazenamento com opções de 4GB (*Gigabyte*) e 8GB de memória, sistema operacional próprio, o *iOS* e integração com serviços de outras empresas como o *Google Maps* da *Google* [INTERNATIONAL BUSINESS TIME 2012]. O modelo revolucionou o mercado de *smartphones* e possui características inovadoras presentes em modelos lançados até o momento em que esse artigo foi escrito.

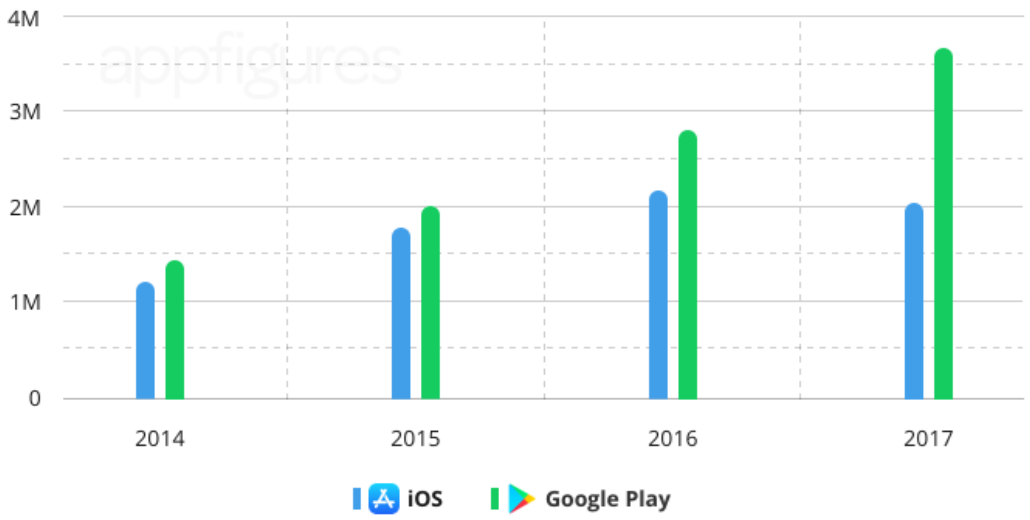
Ainda em 2007, o *Android* foi apresentado junto com a *Open Handset Alliance*, uma parceria entre *Google* e várias fabricantes, como *Motorola*, *Samsung* e LG, para desenvolver uma plataforma única no *mobile* [TECMUNDO 2017]. Logo após o lançamento do primeiro aparelho móvel da *Apple*, a *Google*, em setembro de 2008, lançou a versão 1.0 do *Android*, juntamente com a sua loja de aplicativos antes conhecida como *Android Market* e renomeada futuramente para *Google Play* em 2012. E no mesmo mês, foi lançado o primeiro *smartphone* que utilizava o sistema operacional da empresa, o HTC *Dream*.

O *Android* democratizou o uso do *smartphone*, estando presente desde os aparelhos mais simples até os mais sofisticados, tornando-se assim, em 2017, o sistema operacional líder em acesso à *internet* [TECMUNDO 2017]. Apesar de não ser forte na

produção de *smartphones*, a *Google* domina grande parte do mercado com seu sistema operacional sendo disponibilizado para várias empresas como a *Motorola*, *Samsung* e *ASUS*.

Por possuir uma gama enorme de usuários em sua plataforma, o *Android* não domina somente o mercado de sistema operacional *mobile*, mas também o de aplicativos móveis. Uma pesquisa realizada pela plataforma *AppFigures* em 2018, mostra que a loja de aplicativos da *Google*, terminou o ano de 2017 com um aumento de 30% no número de aplicativos comparado ao ano anterior, totalizando 3,6 milhões de aplicativos disponíveis e como pode-se ver também, (Gráfico 1), a *Google Play* é a loja de aplicativos que apresenta um crescimento satisfatório ao longo dos anos comparado a *Apple Store* [APPFIGURES 2018].

Gráfico 1. Crescimento da *Google Play* e *Apple Store* ao longo dos anos [APPFIGURES 2018]



1.1. Desenvolvimento de Aplicações Móveis

Nos smartphones, é possível instalar aplicações não só criadas pela empresa que mantém o sistema operacional, mas também possuem a capacidade de executar aplicativos criados por terceiros e realizar diversas outras tarefas. Para o desenvolvimento de novas aplicações atualmente, o mercado está concentrado em dois sistemas operacionais: *Android* e *iOS*.

Como pode-se ver na tabela abaixo (Tabela 1), a previsão feita entre o ano de 2017 até o ano de 2019 pela IDC (*International Data Corporation*), o mercado é liderado fortemente pelo *Android* ocupando uma parcela média de 85,73%, seguido pelo *iOS* com uma média de 14,2% do mercado [IDC 2019]. Na mesma tabela, vê-se que, outros sistemas operacionais que ocupam a parcela chamada de *Others* (outros em português), acabaram sumindo do mercado, o *Windows Phone* por exemplo, que foi bastante popular entre os anos de 2013 e 2015, acabou sendo extinto do mercado pela falta de variedade em aplicativos na sua loja.

De acordo com [TECMUNDO 2018], A loja do Windows Phone era muito inferior em comparação com App Store e Google Play e atualizações em vários aplicativos demoravam muito mais para chegar por lá. Nos últimos anos, o Windows Phone, aos poucos, acabou perdendo seu espaço no mercado e fenômenos como o caso do Pokémon GO, acabou não sendo lançado para essa plataforma por possuir uma gama menor de usuários ativos.

Tabela 1. Participação no mercado mundial de sistemas operacionais de *smartphones* [IDC 2019]

Year	2017	2018	2019
Android	85.1%	85.1%	87.0%
iOS	14.7%	14.9%	13.0%
Others	0.2%	0.0%	0.0%
TOTAL	**100.0%**	**100.0%**	**100.0%**

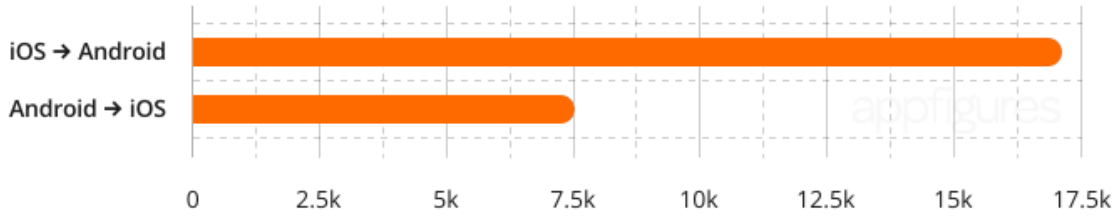
1.2. Justificativa

Por um lado, o desenvolvimento nativo é ótimo pelo fato do rápido e total acesso às funções para manipular os componentes físicos de um aparelho, como a câmera ou o microfone, por exemplo. Por outro, ao final do desenvolvimento, tem-se apenas uma aplicação construída que funcionará em apenas uma plataforma para a qual foi desenvolvida. Com isso, caso uma empresa seja contratada para desenvolver uma aplicação nativa para as duas plataformas em ascensão no mercado, por exemplo, a mesma terá custos maiores pelo fato de precisar contratar um profissional capacitado nas duas plataformas para criar uma versão para cada nicho, aumentando, assim, o custo de desenvolvimento e manutenção [MONTAN 2017]. Outra opção para solucionar esse problema é a contratação de dois profissionais com especialidades distintas para desenvolver a mesma aplicação que, no final das contas, pode custar mais que um profissional apenas.

Como os *smartphones* atuais podem utilizar sistemas operacionais diferentes, é importante desenvolver versões de uma aplicação compatíveis com todas as plataformas em uso no mercado. Para resolver o problema citado acima, pode-se utilizar o conceito de desenvolvimento multiplataforma recorrendo aos *frameworks* para o desenvolvimento *mobile*, o que reduz drasticamente o tempo e o custo no desenvolvimento.

Outro ponto a ser considerado é a portabilidade de aplicativo entre as duas plataformas em ascensão no mercado, o *Android* e *iOS*. De acordo com a pesquisa realizada pela plataforma *AppFigures* em 2018, (Gráfico 2), mais aplicativos inicialmente desenvolvidos para o *iOS* estão chegando no *Android* do que o contrário. Sem contar os aplicativos exclusivos para cada uma das plataformas, o desenvolvimento multiplataforma agiliza bastante esse processo.

Gráfico 2. Aplicativos portados entre as duas plataformas em ascensão no ano de 2017 [APPFIGURES 2018]



Com isso, o intuito principal deste trabalho é realizar uma comparação de desempenho entre duas aplicações idênticas desenvolvidas. Uma utilizando o método nativo para o desenvolvimento e outra utilizando um dos principais *frameworks* disponíveis no mercado, o *React Native* que é mantido e disponibilizado pelo *Facebook*. Baseado nisso, pode-se avaliar, no âmbito do cenário desenvolvido, se o método selecionado para o desenvolvimento multiplataforma, disponibilizado pelo *Facebook*, consegue um desempenho melhor no quesito performance e tamanho final da aplicação comparado com o método tradicional de desenvolvimento. Para o comparativo, foram utilizados os parâmetros de uso de CPU (*Central Processing Unit*), uso da Memória RAM (*Random Access Memory*), uso de *internet*, tempo de carregamento da aplicação e o tamanho do aplicativo instalado no *smartphone*.

Com base no que foi apresentado, neste trabalho usou-se como referência para a realização do comparativo descrito, o sistema operacional *Android*. Essa escolha se deu pelo fato de que é o sistema operacional mais utilizado de acordo com a pesquisa feita pela IDC e o custo da aquisição de hardware para o desenvolvimento nessa plataforma é menor comparado ao *iOS*, que necessita de equipamentos proprietários como os próprios computadores da *Apple* para o desenvolvimento de aplicações nativas.

2. Fundamentação Teórica

2.1. Desenvolvimento Nativo

Uma aplicação considerada nativa é desenvolvida na linguagem padrão do dispositivo. No caso desse estudo, a linguagem utilizada para desenvolver uma aplicação nativa para

a plataforma *Android* é o *Java*. A principal característica de uma aplicação nativa é o acesso total a recursos de *hardware* e, ao final do desenvolvimento, funcionará apenas no sistema operacional para qual a aplicação foi desenvolvida [MONTAN 2017].

Para o desenvolvimento dessas aplicações na plataforma *Android*, é possível utilizar as linguagens de programação *Java* ou o *Kotlin* distintamente ou, o código baseado em *Java* com o *Kotlin* ou o *Kotlin* com código baseado em *Java* [ANDROID 2019]. A *Google* disponibiliza, sem nenhum custo, o SDK (*Software Development Kit*) do *Android* que é responsável pela distribuição de todas as ferramentas necessárias para o desenvolvimento e é executada sobre a plataforma *Java*.

Também é importante, dependendo da proporção do projeto, o uso de alguma API (*Application Programming Interface*), que é disponibilizada pela empresa permitindo que as aplicações tenham comunicação com outras aplicações da própria *Google* ou outro padrão desenvolvido e disponibilizado por outra companhia ou grupo, para acessar seus produtos específicos e serviços. Para unificar tudo isso, a *Google* disponibiliza também a IDE (*Integrated Development Environment*) *Android Studio*, que é o ambiente de desenvolvimento oficial e recomendado pela companhia que facilita ao desenvolvedor trabalhar com todas essas ferramentas de uma maneira mais produtiva e eficiente.

2.2. *Frameworks* no Desenvolvimento *Mobile*

Ao decorrer do tempo, várias soluções foram desenvolvidas com o intuito de melhorar a vida do programador no quesito curva de aprendizado, ou seja, não é mais preciso aprender duas ou mais linguagens de programação para criar um mesmo aplicativo que funcione de forma nativa em duas plataformas distintas. Ao decorrer desta sessão, são apresentados alguns métodos de desenvolvimento e como eles funcionam.

2.2.1. Aplicações Baseadas na *Web*

Aplicações baseadas na *Web* são *sites* responsivos que conseguem se adaptar ao tamanho da tela de um *smartphone* ao ser acessado pelo navegador do dispositivo. Esses *sites* não são construídos utilizando, especificamente, a linguagem nativa do dispositivo, como *Kotlin*, por exemplo. É necessária uma linguagem pensada na *Web* para construir uma aplicação desse tipo.

São semelhantes ao próprio aplicativo (Figura 1), podendo ou não, apresentar a ausência de algumas funcionalidades, já que esse tipo de aplicação não tem tanto acesso ao *hardware* do dispositivo comparado a um aplicativo *mobile*. Esse tipo de aplicação é bastante utilizado por usuários que ainda possuem em seus *smartphones*, uma versão defasada do sistema operacional que não aceite a instalação de uma aplicação mais recente.

Figura 1. Página no *Facebook* do Instituto Federal do Sertão Pernambucano, campus Salgueiro funcionando na versão Nativa e na baseada na *Web*

2.2.2. Aplicações Híbridas

Existem *frameworks* que utilizam um componente existente desde a versão 5.0 do *Android* chamada *WebView*. De acordo com a própria *Google*, *WebView* é um componente do sistema com tecnologia do *Google Chrome*, que permite que aplicativos *Android* exibam conteúdo da *Web* [GOOGLE PLAY 2019].

Para o desenvolvimento de páginas *Web* pode-se utilizar no *front-end*[1], HTML (*HyperText Markup Language*), responsável pela estruturação da página, a parte de cores, estilização das fontes e muito mais ficam a cargo do CSS[2] (*Cascading Style Sheets*), e, por fim, a parte funcional que dá vida a qualquer página, fica a cargo do *JavaScript*[3], que é uma linguagem de programação.

Já para o *back-end*[4] pode-se utilizar o *React* que basicamente consegue levar o *JavaScript* antes rodando no lado do cliente (navegador), para funcionar no lado do servidor. Além dele, também é possível utilizar o PHP (*PHP: Hypertext Preprocessor*), que é uma linguagem muito popular atualmente que foi criada com o intuito de trabalhar

---

[1] *Front-end* é a interface de interação entre o usuário e o sistema.

[2] CSS é uma linguagem que permite estilizar páginas *Web*.

[3] JavaScript é uma linguagem de programação que faz parte do conjunto principal no desenvolvimento *Web*.

[4] *Back-end* é responsável por todas as regras de uma aplicação, desde requisições a um banco de dados, até a comunicação com outra aplicação.

diretamente no *back-end* para dar as funcionalidades finais de qualquer plataforma *Web*, seja na comunicação com o banco de dados ou com alguma plataforma externa.

Esses *frameworks* baseados na *Web*, basicamente são *sites* responsivos que rodam na forma de aplicativo. Esse tipo de aplicação acaba perdendo performance por conta da dependência de um navegador para execução e não possuem acesso direto aos componentes físicos do dispositivo, dependendo de um *plugin* [5]específico que somente o desenvolvedor do *framework* ou a comunidade que contribui podem disponibilizar.

2.2.3. Aplicações Interpretadas

Esse tipo de aplicação utiliza um interpretador empacotado junto do aplicativo que, durante a execução da aplicação, transforma o código escrito em sua linguagem padrão para código nativo *Java* no *Android* e/ou podendo também ser transformado em código nativo para *Objective-C* ou *Swift* [6]no caso do *iOS* [BASTOS 2017]. Se tornando, assim, um *framework* multiplataforma de código nativo.

Um exemplo de *framework* que trabalha desta forma é o *React Native*, desenvolvido pelos engenheiros do *Facebook*, o qual permite a criação de aplicações baseadas na linguagem de programação *JavaScript* em conjunto com o *framework Node.js*[7]. O *React Native* possui a capacidade de criar componentes nativos nas duas plataformas (Figura 2), e em determinados momentos, pode ser necessário desenvolver alguns trechos de códigos nativos para a construção dos mesmos.

---

[5] *Plugin* é o nome dado a uma extensão de um programa que tem como objetivo principal, adicionar novas funcionalidades ao mesmo.

[6] *Objective-C* e *Swift* são linguagens de programação nativas para o desenvolvimento de aplicações para o sistema operacional iOS.

[7] *Node.js* é um ambiente desenvolvido para levar o *JavaScript* antes no *front-end,* para o *back-end*.

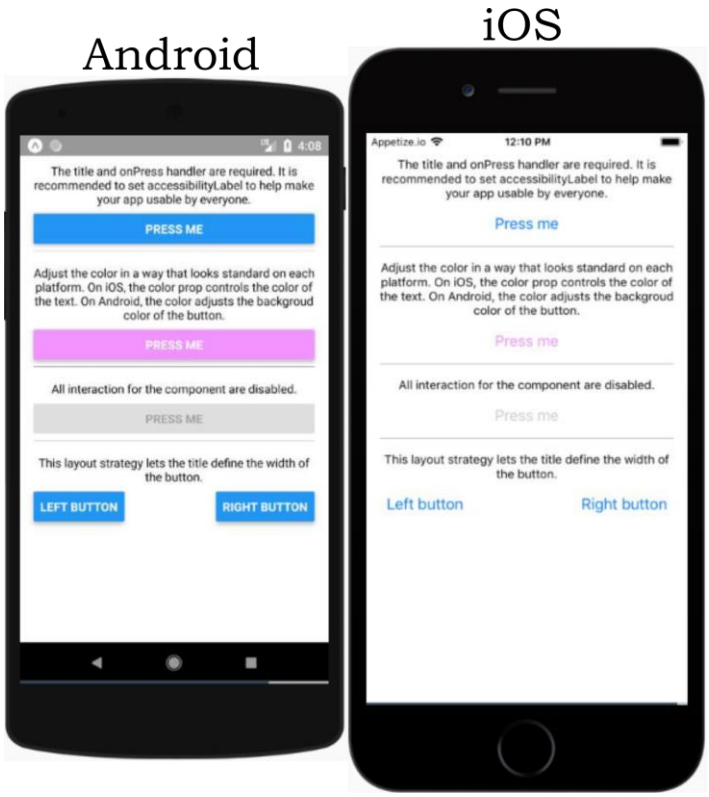


Figura 2. Exemplo do componente *Button* programado utilizando o *framework React Native* e rodando nativamente nas duas plataformas, *Android* e *iOS* no emulador *Expo*

O núcleo do *React Native* é o *React*, que é um *framework JavaScript* que foi originalmente desenvolvido para a criação de interfaces de usuário complexas e permite uma abordagem semelhante ao desenvolvimento *Web*. O *React* exige a utilização do JSX que permite escrever estruturas baseadas no HTML, no formato de *tags*, no mesmo arquivo onde é escrito o código *JavaScript*. O JSX consiste em uma extensão versátil que utiliza XML[8] (*Extensible Markup Language*), tanto para interface nativa em dispositivos móveis, quanto para a *Web* [CRUZ 2017].

2.3. *Firebase*

O *Firebase* é um conjunto de serviços *back-end* lançado pela *Google* em 2004. Possui a filosofia de que o cliente não precisa se preocupar com o *back-end*, tudo fica a cargo das ferramentas disponibilizadas pelo *Firebase* e sua estrutura. Possui ferramentas como o *Cloud Firestore* que permite armazenar e sincronizar dados entre usuários e dispositivos por meio de um banco de dados *NoSQL*[9], *Authentication* que disponibiliza vários meios de autenticação, como através do *GitHub*[10], *Google*, *Facebook* e *Twitter*. E o *Cloud*

[8] XML é uma linguagem de marcação que permite o fácil compartilhamento de informações por intermédio da *internet*.

[9] *NoSQL* é um modelo de banco de dados não relacional.

[10] *GitHub* é um repositório de código aberto que permite o controle de versionamento.

Storage que permite o armazenamento e compartilhamento de conteúdo gerado pelo usuário como imagens, áudio e vídeo [FIREBASE 2019].

O *Firebase* possui planos pagos que custam, em média, U$ 25/mês e um gratuito, chamado de Plano *Spark*, que possui vantagens como no banco de dados *Realtime Database*, disponibilizando 100 conexões simultâneas e 1GB para armazenamento de dados. No *Storage*, é disponibilizado 5GB para armazenamento de arquivos e *download* dos mesmos com o limite de tráfego de 1GB por dia [FIREBASE 2019]. Caso o desenvolvedor queira aumentar a capacidade de armazenamento, por exemplo, é cobrada uma taxa por cada *Gigabyte*[11] a mais.

2.3.1. *Realtime Database*

O *Realtime Database* é um dos diversos serviços disponibilizados pelo *Firebase*. Consiste em um banco de dados baseado em *NoSQL* hospedado na nuvem. Os dados são armazenados em formato JSON (*JavaScript Object Notation*), e os mesmos podem ser sincronizados em tempo real através de funções disponibilizadas pelo *Firebase* com os aplicativos clientes conectados ao serviço [SILVA 2018].

2.4. Ciclo de Vida de uma Aplicação

2.4.1. Ciclo de Vida no Desenvolvimento Nativo *Android*

No desenvolvimento nativo *Android*, é chamado de *activity* todas as telas desenvolvidas para a aplicação que permite a interação com o usuário. A mesma é uma classe *Java* que permite aos desenvolvedores, manipular os estados da *activity* de acordo com a aplicação em desenvolvimento. Ou seja, é permitido manipular uma aplicação desde o momento que a mesma é iniciada, até o momento da sua finalização.

De acordo com o guia para desenvolvedores *Android* [ANDROID 2019], pode-se manipular as telas da aplicação com os seguintes métodos (Figura 3):

[11] *Gigabyte* é uma unidade de medida de informação.

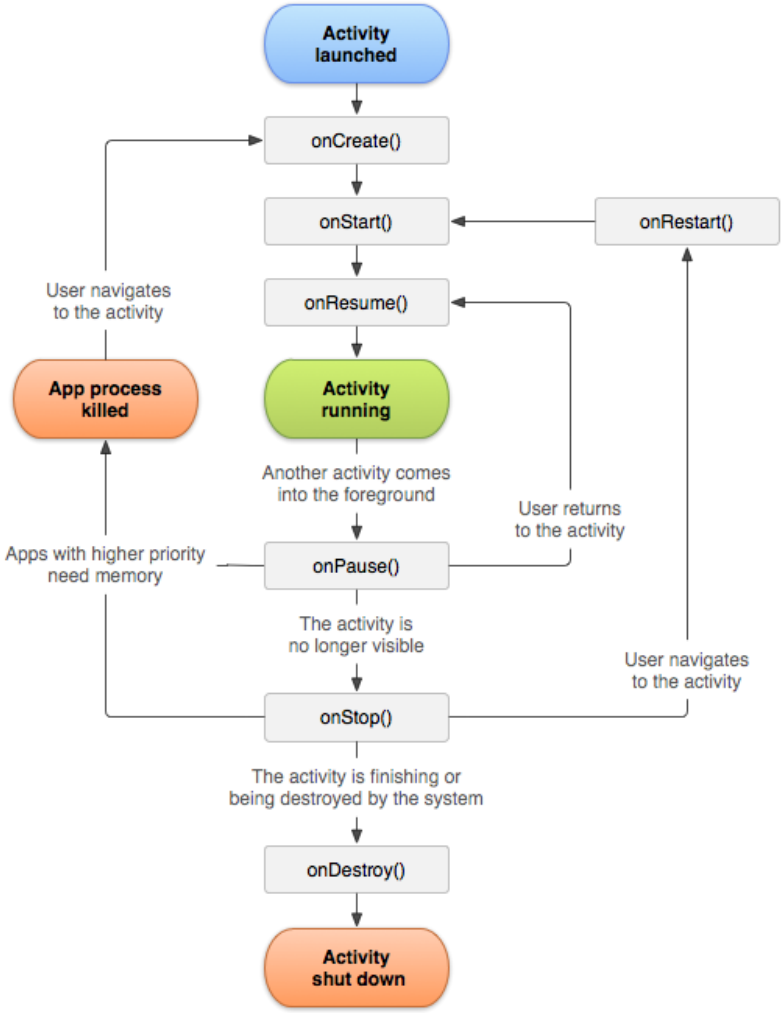


Figura 3. Ciclo de vida de uma *activity* no desenvolvimento nativo *Android*

- *onCreate*: Chamado quando a *activity* é criada.
- *onStart*: Chamado logo antes da *activity* se tornar visível ao usuário.
- *onResume*: Chamado quando a *activity* está visível para o usuário.
- *onPause*: Chamado quando o sistema está prestes a chamar outra *activity*.
- *onStop*: Chamado quando a *activity* não está mais visível ao usuário.
- *onRestart*: Chamado antes da *activity* ser reiniciada.
- *onDestroy*: Chamado antes da *activity* ser destruída.

2.4.2. Ciclo de Vida no Desenvolvimento com o *React Native*

No desenvolvimento multiplataforma escolhido para este comparativo, o ciclo de vida funciona de uma forma um pouco diferente do desenvolvimento *Android* nativo. As telas da aplicação são chamadas de componentes e os mesmos também possuem um ciclo de vida que pode sofrer alterações antes, durante e depois da execução. Sendo

assim, pode-se chamar métodos interceptando sua renderização durante todo o ciclo [FERNANDES 2017].

No *React* pode-se manipular os estados do componente visando três fases: montagem, atualização e desmontagem [FERNANDES 2017]. Baseado nisso, os principais métodos disponíveis são (Figura 4):

Figura 4. Ciclo de vida de um componente no *React* [CODEVOILA 2016]

- *componentWillMount*: É chamado apenas uma vez antes da renderização inicial.
- *render*: Retornando apenas conteúdo JSX, esse método é chamado na construção do componente e toda vez que o mesmo é atualizado.
- *componentDidMount*: É chamado após a renderização do componente.
- *componentWillReceiveProps*: É chamado quando o componente estiver recebendo novas propriedades.
- *componentWillUpdate*: É chamado após ocorrer alguma alteração nos estados do componente.
- *componentDidUpdate*: É chamado após ocorrer a alteração do componente.
- *componentWillUnmount*: É chamado imediatamente antes de um componente ser desmontado.

3. Metodologia

O *React Native* foi o *framework* escolhido para a realização da comparação juntamente com o a ferramenta escolhida para o desenvolvimento nativo no *Android*. A sua escolha se deu por conta que é um *framework* bastante conhecido para o desenvolvimento de aplicações multiplataforma, possui uma comunidade ativa e extensa, é *Open Source*, possui um interpretador incorporado ao aplicativo que permite a conversão do código escrito utilizando o *React Native* para código nativo tanto na plataforma *Android*, quanto no *iOS*. Fazendo com que assim, o mesmo não utilize outros tipos de ferramentas limitadas que serão descritas nesse artigo para a construção de suas aplicações.

3.1. Aplicação Desenvolvida

A aplicação desenvolvida baseia-se em um jogo chamado *Jokenpo*. O intuito do jogo é escolher entre pedra, papel ou tesoura com o auxílio de botões e a aplicação fará o restante do trabalho. Será gerado um número aleatório utilizando em ambas as aplicações, a classe *Random* e esse número será comparado em um conjunto de instruções para determinar a escolha da máquina. Caso o usuário vença a máquina, o dispositivo irá vibrar durante meio segundo simbolizando a vitória.

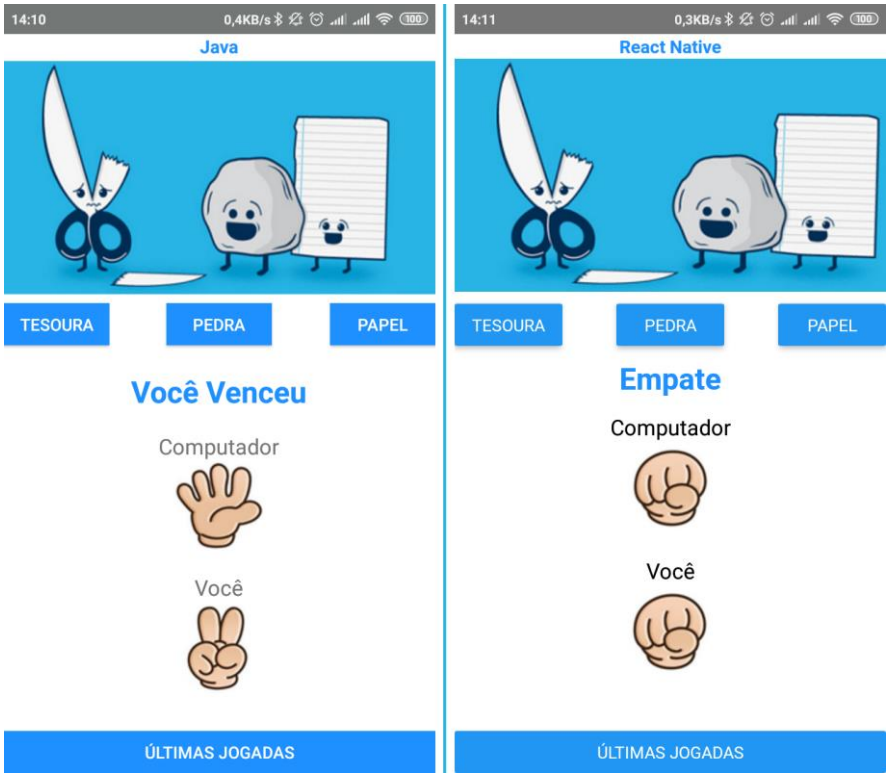


Figura 5. Tela inicial da aplicação *Jokenpo* [Pelo autor]

Após mostrar o resultado na tela (Figura 5), o mesmo será enviado para armazenamento na nuvem utilizando o serviço *Realtime Database* do *Firebase* com o intuito de obter as informações geradas durante a execução dos mesmos. Ao clicar no botão Últimas Jogadas (Figura 6), será listado para o usuário todos os dados salvos de jogadas feitas pelo mesmo em ambas as aplicações.

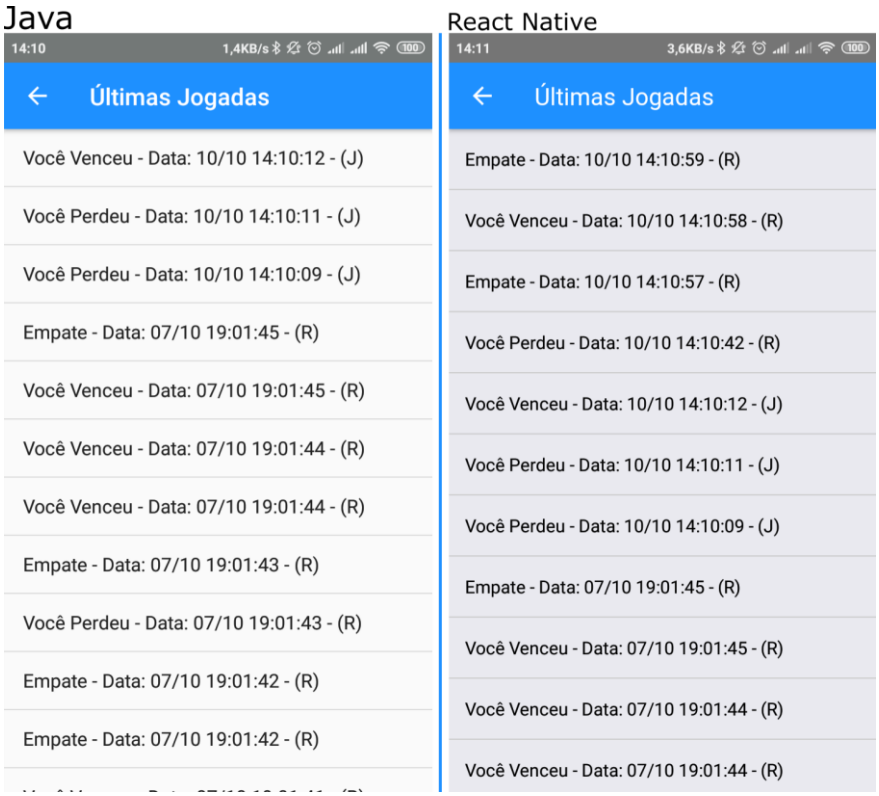


Figura 6. Tela Últimas Jogadas que apresenta ao jogador, todo o histórico em ambas as aplicações [Pelo autor]

3.2. Ambiente de Desenvolvimento

Para desenvolver a aplicação nativa utilizando o *Java*, as ferramentas utilizadas e configurações realizadas nos arquivos principais do *Android* seguem descritas na tabela abaixo (Tabela 2).

Tabela 2. Configurações e ferramentas utilizadas no desenvolvimento da aplicação nativa [Pelo autor]

Descrição	Versão
Java JDK	1.8.0_211
Android Studio	3.5
Gradle	5.4.1
compileSdkVersion	29
buildToolsVersion	29.0.2
minSdkVersion	16
targetSdkVersion	29
Firebase	16.1.0

Já para o desenvolvimento da aplicação multiplataforma utilizando o *framework React Native*, é descrito na tabela abaixo, (Tabela 3), todas as ferramentas utilizadas para a finalização da aplicação multiplataforma.

Tabela 3. Ferramentas utilizadas para o desenvolvimento da aplicação multiplataforma [Pelo autor]

Descrição	Versão
Java JDK	1.8.0_211
Node.js	10.16.3
React	16.8.6
React Native	0.60.4
Firebase	6.3.3

3.3. Ambiente de Testes

Para realizar os testes de performance, foi utilizado um aparelho físico. Deu-se essa escolha pelo fato de que no mesmo, as aplicações podem utilizar todos os recursos necessários para uma melhor performance comparada ao emulador disponibilizado pela própria *Google* em sua IDE oficial que, durante a própria execução dos programas do computador, pode afetar no desempenho da aplicação rodando no emulador. Assim, os dados referentes ao uso de memória RAM, uso de *internet*, uso de CPU e tempo de carregamento, são mais precisos trabalhando com o dispositivo físico. Vale ressaltar que as aplicações foram testadas com o *smartphone* configurado para o uso normal do cotidiano, ou seja, Wi-Fi (*Wireless Fidelity*), *Bluetooth* e GPS (*Global Positioning System*) ligados. Abaixo, Tabela 4, pode-se conferir a configuração de *hardware* e *software* do *smartphone* utilizado.

Tabela 4. Configurações de *hardware* e *software* referente ao dispositivo físico utilizado nos testes [Pelo autor]

Descrição	Especificação
Marca e Modelo	*Xiaomi Redmi 5A*
Versão do *Android*	*8.1.0 Oreo*
Modificação do *Android*	*MIUI 10.3.2*
Processador	*Qualcomm Snapdragon 425 Quad-Core 1.4GHz*
Arquitetura do Processador	*ARMv8-A*
Conjunto de Instruções	*arm64-v8a*
Processador Gráfico	*Qualcomm Adreno 308*
Memória RAM	*2GB*
Bateria	*3000 mAh*

3.3.1. Plano de *Internet*

É preciso conexão com a *internet* no *smartphone* para o envio dos dados para o serviço *Realtime Database* do *Firebase*. O plano de *internet* utilizado durante os testes foi de 35Mbps (megabit por segundo) de *download* mantendo uma média de 22,85Mbps, e 10Mbps de *upload* mantendo uma média de 4,68Mbps durante os testes manuais

realizados. Foi utilizado o aplicativo *Speedtest* para obtenção dos dados referente a velocidade da *internet* momento antes de cada teste.

3.4. Testes Realizados

3.4.1. Uso de Memória RAM e CPU

Para obter os dados referentes ao uso de memória RAM e CPU pelos aplicativos em execução no *smartphone*, foi utilizada uma ferramenta de linha de comando versátil que permite a comunicação com o dispositivo, utilizando um cabo USB (*Universal Serial Bus*), chamada ADB (*Android Debug Bridge*) [ANDROID 2019]. Com o comando *adb shell top*[12], é permitido monitorar a cada três segundos, informações como porcentagem de memória RAM, porcenagem do uso de CPU, tamanho virtual do processo e prioridade das aplicações em execução naquele momento diretamente pelo terminal do computador.

3.4.2. Uso de *Internet*

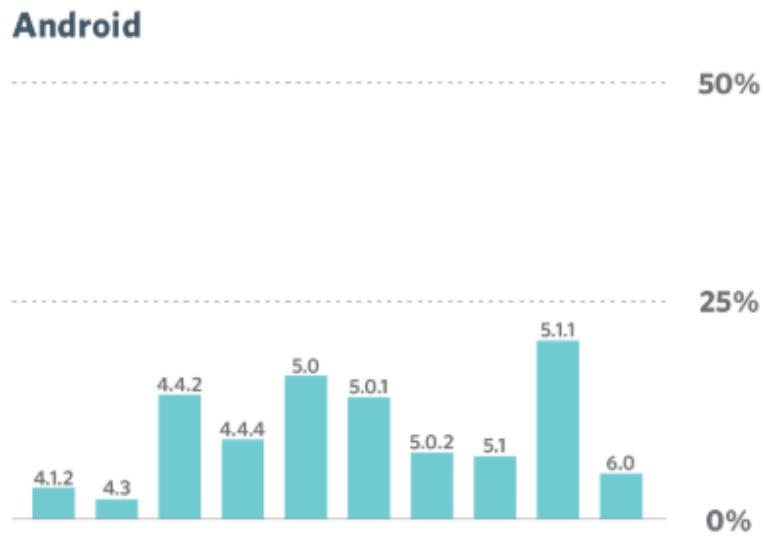
No menu de configurações do dispositivo utilizado, há a opção chamada Uso de Dados que permite a visualização e gerenciamento do uso de dados pelas aplicações em execução, tanto dos dados móveis, quanto da rede Wi-Fi. Seguindo na parte de visualização, ao selecionar uma aplicação, é mostrado um gráfico referente ao uso de dados pela mesma em um determinado tempo.

3.4.3. Tamanho do APK

Em uma pesquisa feita pela *Duo Labs* (2016), no Gráfico 3, é mostrado que grande parte dos usuários *Android* ainda possuem instalado em seus *smartphones*, a versão 4.0 ou inferior do mesmo, que nos dias atuais já está obsoleta [DUO 2016]. O problema disso não é só na questão de segurança, como mostrado na pesquisa, mas também usuários que ainda possuem esses dispositivos nos dias atuais, sofrem na questão espaço de armazenamento. Sem contar com as aplicações que retiram o suporte para sistemas operacionais mais antigos e *hardwares* menos modestos, com o grande número de aplicações disponíveis na loja nos dias atuais, o desejo por mais aplicações instaladas é um problema para usuários com esse tipo de aparelho. Com no máximo 8GB de memória de armazenamento, incluindo a acomodação do próprio sistema operacional, o espaço disponível nesses dispositivos acaba se tornando menor e o espaço livre acaba sendo muito pouco para tantos aplicativos instalados.

[12] Comando que permite monitorar os processos em aberto no *smartphone*.

Gráfico 3. Fragmentação do sistema operacional *Android* [DUO 2016]



Para a obtenção dos dados referente ao tamanho do APK gerado pelas ferramentas de desenvolvimento, foi utilizada a ferramenta de propriedades do *Windows* que permite gerenciar arquivos que estão salvos no disco de armazenamento do computador. Na aba geral, é possível visualizar o tamanho do arquivo APK gerado tanto em *megabytes*, quanto em *bytes*.

3.4.4. Tempo de Carregamento

Para obter dados referentes ao tempo de carregamento das aplicações desenvolvidas em milissegundos, no desenvolvimento nativo *Android*, foi implementado um algoritmo nos métodos *onCreate* e *onResume* que calcula o tempo desde o momento da abertura da aplicação, até o momento em que a mesma está pronta para interação com o usuário.

Já para o *React Native*, foi implementado um algoritmo semelhante nos métodos *componentWillMount* e *componentDidMount* que realizam o mesmo procedimento que foi implementado no desenvolvimento nativo e ao final, é mostrado o resultado na tela das duas aplicações.

4. Resultados e Discussões

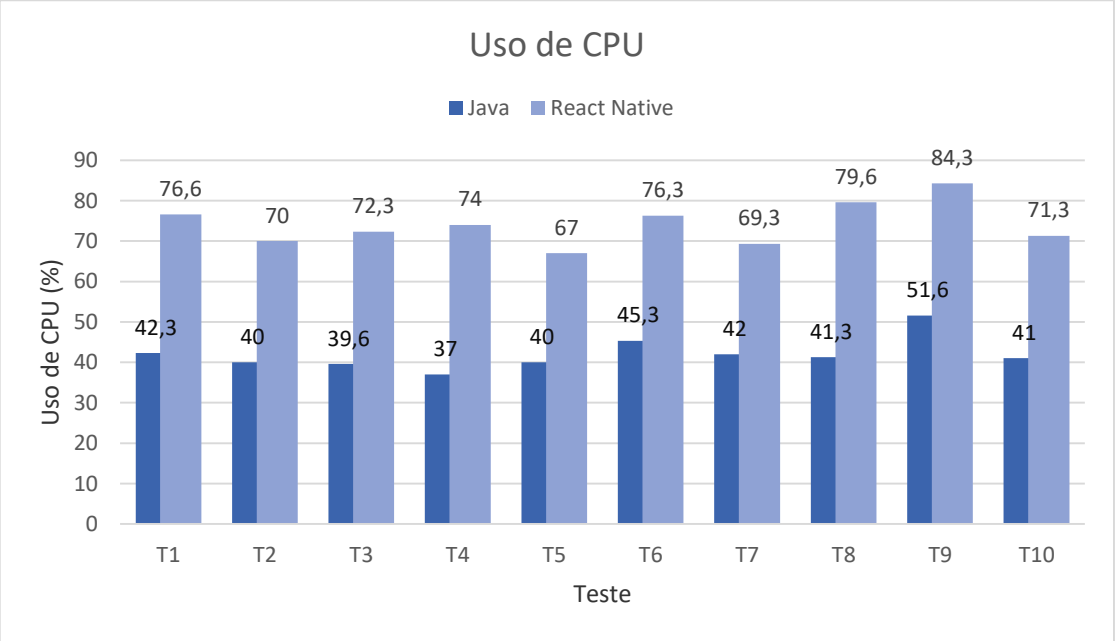
Para a obtenção de todos os dados utilizados para o comparativo, foram feitas cem jogadas em dez testes realizados em diferentes horários por cada aplicação, ou seja, as aplicações foram abertas dez vezes e a memória RAM do *smartphone* foi limpa sempre antes das aplicações desenvolvidas serem abertas e os testes nas duas aplicações foram realizados na ordem da aplicação nativa *Java* primeiro, logo após a desenvolvida com o *framework React Native*.

Após a realização dos testes, foram obtidos os seguintes dados apresentados nos subtópicos a seguir.

4.1. Uso de CPU e Memória RAM

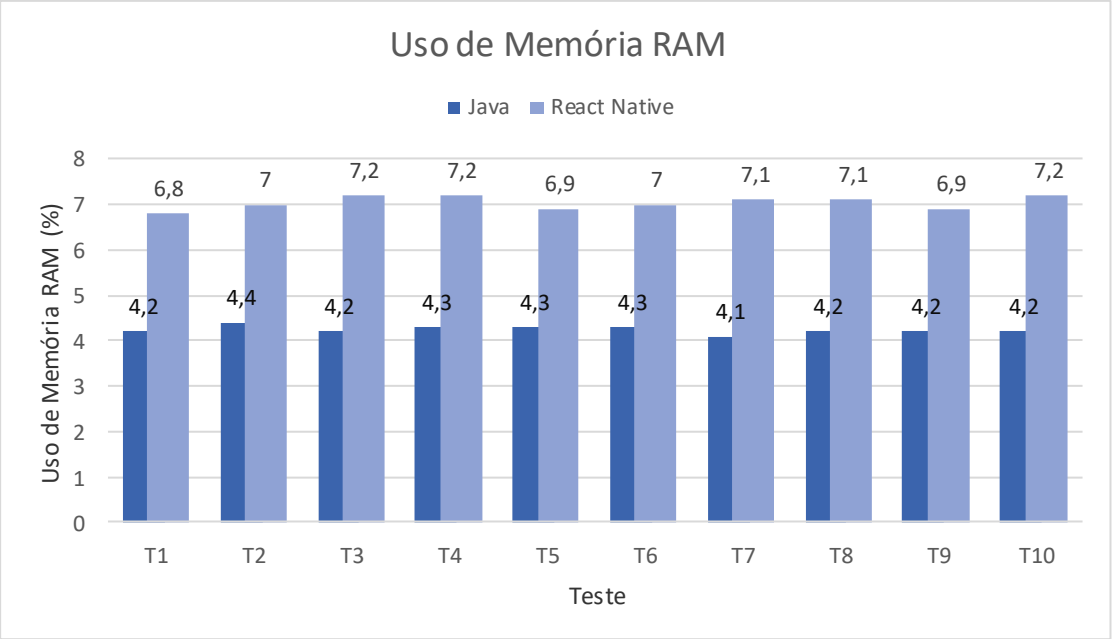
Como pode-se ver nos gráficos abaixo (Gráfico 4 e 5), o *React Native* teve um uso de CPU e memória RAM bastante elevado comparado com o desenvolvimento nativo *Android* utilizando o *Java*. Esse consumo elevado pode-se atribuir ao fato de que o *React Native* utiliza o *Node.js* para interpretar o *JavaScript* utilizado para a escrita dos seus códigos e passar esse código por uma *bridge* (ponte), que posteriormente irá acessar os módulos nativos da plataforma para qual a aplicação foi desenvolvida [MIRANDA 2019].

Gráfico 4. Uso de CPU pelas aplicações desenvolvidas [Pelo autor]



A aplicação nativa obteve uma média de 42,01% de uso, enquanto a aplicação multiplataforma obteve uma média de 74,07% no uso de CPU. Baseado na média de consumo, a aplicação nativa consumiu 1,76 vezes menos processamento durante a execução dos testes.

Gráfico 5. Uso de memória RAM pelas aplicações desenvolvidas [Pelo autor]

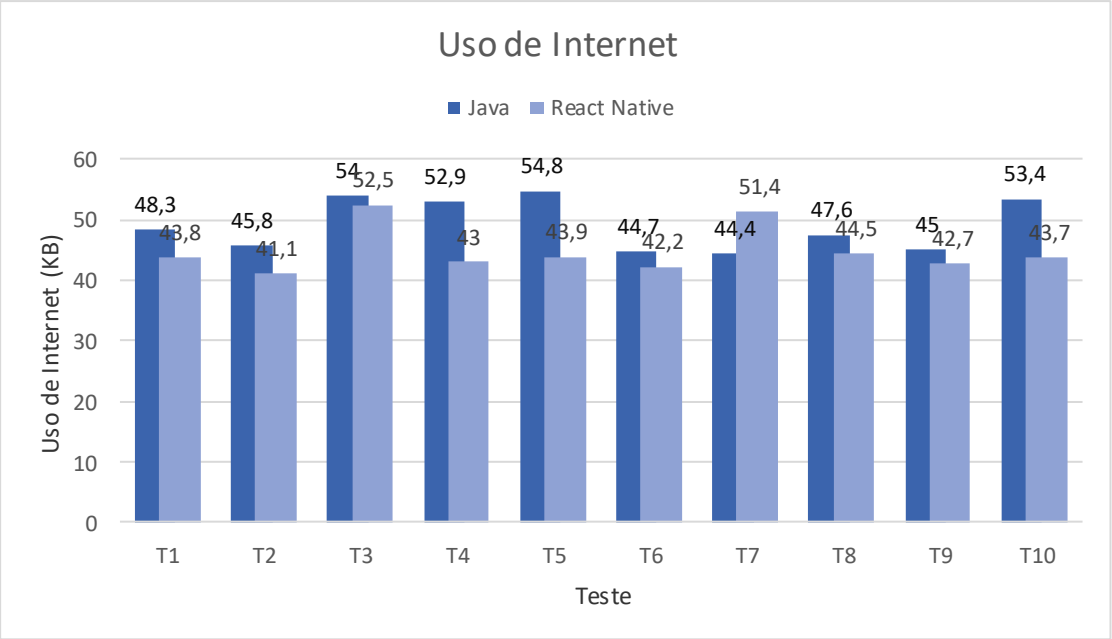


No quesito uso de memória RAM, enquanto a aplicação nativa consumiu uma média de 4,24% de uso durante os testes, a aplicação utilizando o *framework React Native* consumiu 7,04% de uso. Ou seja, de acordo com a média apresentada, a aplicação nativa obteve um consumo 1,66 vezes menor comparado com a aplicação multiplataforma.

4.2. Uso de *Internet*

Para o uso de *internet* no envio das cem jogadas durante os testes (Gráfico 6), a aplicação nativa obteve uma média no consumo de 49,09KB (*Kilobyte*), enquanto a aplicação multiplataforma obteve uma média de 44,88KB. Baseado na média obtida, no quesito consumo de *internet* a aplicação multiplataforma se saiu bem consumindo 1.09 vezes menos comparado com a aplicação nativa.

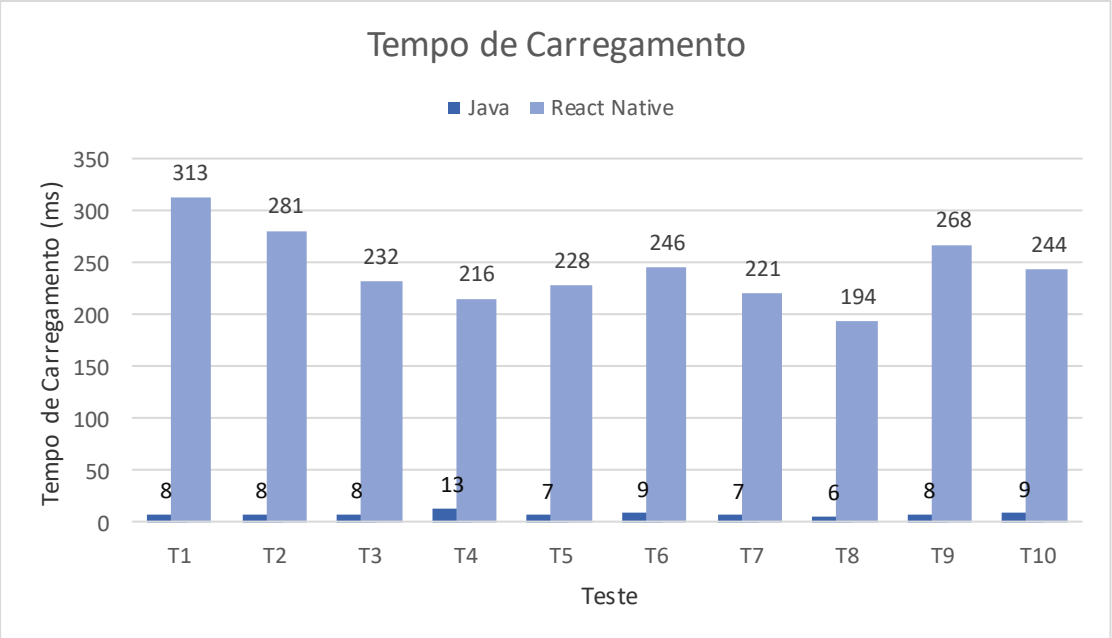
Gráfico 6. Uso de *internet* pelas aplicações desenvolvidas [Pelo autor]



4.3. Tempo de Carregamento

No quesito tempo de carregamento das aplicações (Gráfico 7), é calculado o tempo a partir da execução da função *onCreate* até a finalização da *onResume* na aplicação nativa. Enquanto na aplicação multiplataforma, foi calculado o tempo levado para a execução da função *componentWillMount* até a finalização da função *componentDidMount*. Na aplicação multiplataforma foi observado um tempo de carregamento muito elevado que no desenvolvimento nativo. Pelo fato de realizar uma conversão de componentes criados pelo *React* para o nativo no *Android*, o tempo para esse processo é elevado. Na aplicação multiplataforma, foi obtida uma média de 244,3ms (milissegundos) de carregamento enquanto na aplicação nativa foi obtida uma média de 8,3ms.

Gráfico 7. Tempo de carregamento em milissegundos pelas aplicações desenvolvidas [Pelo autor]



Com isso, baseado nas médias obtidas, a aplicação nativa foi 29,4 vezes mais rápida no tempo de carregamento comparado com a aplicação multiplataforma.

4.4. Tamanho do APK

No quesito tamanho do APK gerado pelas duas ferramentas de desenvolvimento, ao gerar o mesmo, o APK da aplicação nativa ficou com um tamanho total de 2,77MB, enquanto a aplicação multiplataforma obteve um tamanho total de 20,2MB. Ou seja, a aplicação nativa ficou 7,29 vezes menor que a aplicação multiplataforma.

Cabe salientar que nos testes realizados, foram utilizados os APKs padrões gerados (Tabela 5).

Tabela 5. APK gerado após o término do desenvolvimento [Pelo autor]

Ferramenta	Tamanho
Java	2,77MB
React Native	20,2MB

De acordo com [FERNANDES 2019], o *React Native* implementou em uma de suas últimas atualizações, o suporte a várias arquiteturas de processadores disponíveis no mercado. Com isso, ao gerar um APK utilizando o *framework React Native*, todas as arquiteturas estão agrupadas em um mesmo arquivo e consequentemente, o mesmo acaba ficando maior. Para separar as arquiteturas, basta modificar o arquivo *android/app/build.gradle* procurando o trecho de código *def*

enableSeparateBuildPerCPUArchitecture = false, trocar de *false* para *true* o valor da variável e gerar novamente o APK da aplicação (Tabela 6).

Tabela 6. APK gerado após a separação das arquiteturas no *React Native* [Pelo autor]

Arquitetura	Tamanho
Arm64-v8a	7,12MB
Armeabi-v7a	6,83MB
X86	6,85MB
X86_64	7,70MB

5. Considerações Finais

No presente trabalho, foi apresentado um comparativo entre o desenvolvimento nativo na plataforma *Android* e o desenvolvimento multiplataforma com o *framework React Native*. Com base nos dados obtidos, no âmbito do ambiente proposto e baseado nos testes realizados, pode-se observar que o desenvolvimento nativo ainda é uma boa opção no quesito performance, comparado com o desenvolvimento multiplataforma escolhido para o comparativo.

O desenvolvimento nativo obteve dados favoráveis em quatro dos cinco testes realizados, perdendo apenas no quesito uso de dados. Apesar da boa colocação, dependendo da grandiosidade da aplicação a ser desenvolvida, o desenvolvimento multiplataforma escolhido pode ser uma boa opção por conta de pontos como alta produtividade, baixo custo de manutenção e curva de aprendizado.

Como já foi mencionado nos resultados referentes ao tamanho do APK gerado, o *React Native* realiza a junção de todas as arquiteturas já descritas em um único arquivo APK. Ao finalizar os testes para o comparativo, foi testado a parte, o APK com a arquitetura *arm64-v8a*. Que nesse caso, é específica para o dispositivo físico utilizado nos testes. Após realizar uma bateria de testes, não foi perceptível a diferença nos dados obtidos entre o APK com a arquitetura *arm64-v8a* (específica para o dispositivo utilizado), com os dados obtidos na comparação principal entre o *React Native* e o *Java* nativo. Mas, mesmo assim, o APK gerado pelo desenvolvimento nativo é 2,57 vezes menor comparado com o APK correspondente à arquitetura *arm64-v8a* específica para o dispositivo utilizado.

Contudo, por um lado, o desenvolvimento nativo é vantajoso pelo fato do rápido e otimizado acesso aos recursos físicos do dispositivo, uma melhor segurança e como mostrado nos testes realizados em um ambiente específico, mostrou-se uma performance exepcional. Por outro, o desenvolvimento multiplataforma tem pontos positivos que pode deixar mais difícil a escolha da melhor solução para o desenvolvimento de uma solução. Seja pelo fato da escrita de um único código que funcione em duas plataformas distintas ou o baixo custo no desenvolvimento. No caso do *React Native*, essa ferramenta ainda está se desenvolvendo e futuramente, poderá apresentar resultados melhores comparado com os vistos aqui. Sabendo disso, cabe ao desenvolvedor decidir, se vale a pena trocar o desempenho de uma aplicação nativa, por

um baixo custo no desenvolvimento e manutenção, uma ótima curva de aprendizado e ter uma aplicação multiplataforma utilizando o *framework React Native* que foi utilizado no comparativo apresentado.

6. Trabalhos Futuros

Como possíveis trabalhos futuros, uma das opções almejadas é o aprimoramento da aplicação desenvolvida aplicando-a novas funcionalidades. Também é viável realizar mais testes com base nos parâmetros definidos neste trabalho em outros dispositivos físicos e/ou também utilizando emuladores como o próprio emulador que vem junto com o SDK do *Android*. Por fim, existe um amplo conjunto de opções para o desenvolvimento multiplataforma. Sabendo disso, abre espaço também o desenvolvimento da mesma aplicação em outros *frameworks* disponíveis no mercado para servir como base para comparação entre as diversas ferramentas multiplataforma ou até mesmo entre as duas principais ferramentas, *Java* ou *Objective-C*, para o desenvolvimento de aplicações nativas.

Referências

ANDROID. Android Debug Bridge. 2019. Disponível em: <https://developer.android.com/studio/command-line/adb.html?hl=pt-br>. Acesso em: 01 nov. 2019.

ANDROID. Atividades. 2019. Disponível em: <https://developer.android.com/guide/components/activities>. Acesso em: 31 out. 2019.

ANDROID. Desenvolver apps para Android com o Kotlin. 2019. Disponível em: <https://developer.android.com/kotlin?hl=pt>. Acesso em: 13 ago. 2019.

BASTOS, G; [et. al]. REACT-NATIVE, AVALIAÇÃO DE PERFORMANCE COMPARANDO COM CÓDIGO JAVA NATIVO COM LISTVIEW. JETMA, 2017. Disponível em: <http://publicacoes.facthus.edu.br/index.php/geral/article/view/19>. Acesso em: 23 set. 2019.

CODEVOILA. React.js. Tutorial: React Component Lifecycle. 2016. Disponível em: < https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle>. Acesso em: Acesso em: 31 out. 2019.

CRUZ, V; PETRUCELLI, E. Tecnologias Web para o Desenvolvimento Mobile Nativo. SIMTEC, 2017. Disponível em: < http://simtec.fatectq.edu.br/index.php/simtec/article/view/269>. Acesso em: 23 set. 2019.

EXPO. Disponível em: <https://expo.io/>. Acesso em: 23 set. 2019.

FERNANDES, Diego. React do zero: ciclo de vida, stateless components e arquitetura flux. 2017. Disponível em: <https://blog.rocketseat.com.br/react-do-zero-ciclo-de-vida-stateless-components-e-arquitetura-flux/>. Acesso em: 31 out. 2019.

FERNANDES, Diego. Reduzindo o tamanho do APK para Android no React Native. 2019. Disponível em: <https://blog.rocketseat.com.br/reduzindo-o-tamanho-das-builds-para-android-no-react-native/>. Acesso em: 06 nov. 2019.

FIREBASE. Comece gratuitamente e depois pague conforme o uso. 2019. Disponível em: < https://firebase.google.com/pricing?hl=pt-br >. Acesso em: 23 set. 2019.

FIREBASE. O Firebase ajuda as equipes de aplicativos para dispositivos móveis e da Web a alcançar o sucesso. 2019. Disponível em: < https://firebase.google.com/products?hl=pt-br >. Acesso em: 23 set. 2019.

GOOGLE PLAY. WebView do sistema Android. 2019. Disponível em: <https://play.google.com/store/apps/details?id=com.google.android.webview&hl=pt_BR>. Acesso em: 15 ago. 2019.

HANLEY, Mike. Duo Analytics: Android Device Security. 2016. Disponível em: <https://duo.com/blog/duo-analytics-android-device-security-article>. Acesso: 13 out. 2019.

IDC. Smartphone Market Share. 2019. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 13 ago. 2019.

KLEINA, Nilton. A história do Android, o robô que domina o mercado mobile. 2017. Disponível em: <https://www.tecmundo.com.br/ciencia/120933-historia-android-robo-domina-o-mercado-mobile-video.htm>. Acesso em: 10 out. 2019.

KLEINA, Nilton. A história do Windows Phone, do início à queda. 2018. Disponível em: <https://www.tecmundo.com.br/mercado/130500-historia-windows-phone-do-inicio-queda-video.htm>. Acesso em: 02 ago. 2019.

MICHAELI, Ariel. iOS Developers Ship 29% Fewer Apps In 2017, The First Ever Decline – And More Trends To Watch. 2018. Disponível em: <https://blog.appfigures.com/ios-developers-ship-less-apps-for-first-time/>. Acesso em: 10 out. 2019.

MIRANDA, G; MARQUES, D. Análise comparativa entre ferramentas de desenvolvimento de aplicativos móveis multiplataforma utilizando características da ISO/IEC 25010 por meio da implementação de um cenário pré-definido. IFSP, 2019. Disponível em: <http://hto.ifsp.edu.br/portal/images/thumbnails/images/IFSP/Cursos/Coord_ADS/Arquivos/TCCs/2019/TCC_GustavoRibeiroMiranda.pdf>. Acesso em: 06 nov. 2019.

MONTAN, J; SANTOS, M. AVALIAÇÃO DE PLATAFORMAS HÍBRIDAS PARA DESENVOLVIMENTO DE APLICAÇÕES PARA O ANDROID. Juiz de Fora: Multiverso, 2017. Disponível em: <http://periodicos.jf.ifsudestemg.edu.br/multiverso/article/view/178>. Acesso em: 02 ago. 2019.

MUKHERJEE, Sangeeta. Smartphone Evolution: From IBM Simon To Samsung Galaxy S3. 2012. Disponível em: <https://www.ibtimes.com/smartphone-evolution-ibm-simon-samsung-galaxy-s3-697340>. Acesso em: 10 out. 2019.

SILVA, Werliton. Aplicações Móveis Nativas com React Native e Firebase: Um Estudo de Caso. UFMA, 2018. Disponível em: < http://rosario.ufma.br:8080/jspui/handle/123456789/3498 >. Acesso em: 23 set. 2019.