JORNAL DE ENGENHARIA, TECNOLOGIA E MEIO AMBIENTE
Nº 2. Vol. 2 2017/2

REACT-NATIVE, AVALIAÇÃO DE PERFORMANCE COMPARANDO COM CÓDIGO JAVA NATIVO COM LISTVIEW

Guilherme Borges Bastos[1]; Bruno Alves Romero[2]; Eduardo Fernandes Saad[3]

[1,2,3] Faculdade de Talentos Humanos - FACTHUS, Uberaba (MG), Brasil
[3]Centro Brasileiro de Pesquisas Físicas - CBPF, Rio de Janeiro (RJ), Brasil

guilhermeborgesbastos@gmail.com, bruno.romero@facthus.edu.br, eduardo.saad@facthus.edu.br

RESUMO: A demanda por aplicações mais complexas impulsiona o aprimoramento das linguagens para dispositivos móveis, lançando novas opções para o desenvolvimento de aplicativos. Este estudo traz uma análise comparativa de desempenho entre uma ferramenta recente no mercado mundial React Native, comparando com a linguagem de programação JAVA, analisando os recursos utilizados na operação de um componente (ListView) nos dois ambientes de programação, verificando os mais importantes aspectos técnicos: gerenciamento e consumo de memória, tempo de resposta de requisição ao servidor e o tempo de resposta da inicialização da aplicação. As aplicações foram configuradas para realizar uma chamada a API da Amazon, recuperando a lista dos álbuns mais populares. Essa lista apresenta um resultado médio consistente de aproximadamente 1000 entradas. Os resultados mostram que o ambiente JAVA apresentou um valor médio de 1258 ms para inicialização, enquanto o React Native levou 1369 ms, uma diferença a favor do JAVA de 8,82%. Pode-se observar uma superioridade pequena, mas ainda assim, relevante das aplicações em JAVA Nativo em contraponto ao React Native. Portanto o JAVA Nativo mostrou-se nos testes mais eficiente, principalmente quando se necessita que desempenho seja fator indispensável, como em uma aplicação de alta performance. Foi observado que o framework React Native traz um meio único de desenvolvimento onde pode-se aplica-lo tanto ao ambiente iOS quanto Android ao contrário do JAVA, diminuindo o custo de treinamento e aprendizado de linguagens e ferramentas tão distintas.

PALAVRAS CHAVE: Comparativo, JAVA, Mobile, Performance, React-Native.

REACT-NATIVE, NATIVE APPLICATION COMPARISION USING LISTVIEW COMPONENT

ABSTRACT: The demand for more complex applications drives the enhancement of the languages for mobile devices, launching new options for the development of applications. This study brings a comparative performance analysis between a recent tool in the world market React Native, comparing with the programming language JAVA, analyzing the resources used in the operation of a component (ListView) in the two programming environments, verifying the most important techniques aspects: memory management and consumption, server request response time, and application startup response time. Applications have been configured to call the Amazon API, retrieving the list of the most popular albums. This list shows a consistent average result of approximately 1000 entries. The results show that the JAVA environment presented an average value of 1258 ms for initialization, while React Native took 1369 ms, a difference in favor of JAVA of 8.82%. One can observe a small, but still relevant, superiority of native JAVA applications as opposed to React Native. Therefore JAVA Native proved to be more efficient in the tests, especially when performance is an indispensable factor, as in a high-performance application. It has been noted that the React Native framework brings an unique development environment where you can apply it to both the iOS and Android environments unlike JAVA, reducing the cost of training and learning. languages and so different tools.

KEYWORDS: Comparative, JAVA, Mobile, Performance, React-Native.

INTRODUÇÃO

Bosomworth (2015) demonstra que em meados de 2014, os dispositivos móveis superaram os desktops no acesso à *internet*, ocasionando uma revolução na *World Wide Web*, obrigando empresas a adaptar seus sites e ferramentas aos dispositivos móveis emergentes no mercado. Em sua pesquisa, Bosomworth (2015) demonstra que o público passa muitas horas a mais utilizando aplicativos do que navegando pela internet, uma tendência que deve se manter nos próximos anos.

O crescimento da demanda e o aparecimento de aplicações mais complexas impulsiona o aprimoramento das linguagens *mobile*. Assim, todos os anos são lançados novas opções para o desenvolvimento de aplicativos, isso gera dúvidas sobre qual seria a melhor linguagem ou framework para determinadas aplicações, uma vez que, se entende que cada ferramenta tem seus pontos fortes e fracos.

Este estudo analisa uma ferramenta atual no mercado mundial React Native, comparando o desempenho de um componente (ListView) com o mesmo aplicativo/componente desenvolvido em linguagem de programação JAVA, verificando os mais importantes aspectos técnicos: gerenciamento de memória, tempo de resposta de requisição ao servidor e o tempo de resposta da inicialização da aplicação. Os dados serão tabulados e os resultados apresentados em gráficos e tabelas. Ao final será verificado qual a melhor plataforma de desenvolvimento para este tipo de aplicação.

Cabe salientar que todos os testes e desenvolvimentos, foram realizados utilizando a arquitetura Windows 10 Home, versão 1607.

MATERIAIS

Para o desenvolvimento deste trabalho, foram criadas duas aplicações que realizavam o mesmo trabalho, utilizando um componente comum entre o ambiente JAVA e o framework React-Native, chamado ListView. Este componente apresenta uma lista contendo um resultado por linha, sendo utilizado na maioria dos aplicativos móveis.

As aplicações estão configuradas para realizar uma chamada a API da Amazon, recuperando a lista dos álbuns mais populares. Essa lista apresenta um resultado médio consistente de aproximadamente 1000 entradas. Essa lista é inicialmente armazenada em um objeto comum ArrayList e, posteriormente, enviado para o componente adequado a cada ambiente para ser inicializado.

O tempo de inicialização da aplicação, isto é, o tempo desde a chamada, até que o ambiente esteja funcional e com todos os 1000 registros estejam carregados na lista, foi medido.

Também foi averiguado o consumo de memória RAM conforme a lista era apresentada. Para isso, foi criado um método para rolagem dos elementos, em uma velocidade controlada e constante de 1 elemento por segundo. Isso permite avaliar o gerenciamento de memória do componente contendo vários elementos. Assim, a cada 5 segundos, um método lia o consumo de memória do aplicativo.

A LINGUAGEM JAVA

Java é uma linguagem de programação e plataforma computacional lançada pela primeira vez pela *Sun Microsystems* em 1995. Existem muitas aplicações e sites que não funcionarão, a menos que você tenha o Java instalado, e mais desses são criados todos os dias. O Java é rápido, seguro e confiável. De *laptops* a *datacenters*, consoles de games a supercomputadores científicos, telefones celulares à Internet, o Java é utilizado todos os lugares. (JAVA, 2017).

A versão utilizada neste trabalho foi a 1.8.161.

NODE.JS

Node.js é uma plataforma construída sobre o motor JavaScript do Google Chrome para construir aplicações em rede rápidas e escaláveis. Node.js usa um modelo de I/O direcionada a evento não bloqueante, que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

Em outras palavras, hoje o Node.js é uma das ferramentas mais simples para criar um servidor web com banco de dados, podendo faze-lo até mesmo um minicomputador como o Raspberry PI, pois é leve, aceita uma grande quantidade de conexões, altamente escalável, relativamente fácil de programar, tem uma comunidade em crescimento, além de vários frameworks de código aberto disponibilizados gratuitamente para uso e estudo. (PUCCINELLI, 2015).

ANDROID STUDIO

O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android e é baseado no IntelliJ IDEA, adquirido pela Google em negociações com a empresa JetBrains. Além do editor de código e das ferramentas de desenvolvedor avançados do IntelliJ, o Android Studio oferece um ambiente integrado para criação de aplicativos Android (ANDROID, 2017).

A versão utilizada neste trabalho foi a 2.3.

REACT NATIVE

Segundo Cabral (2016), o React Native é framework para o desenvolvimento. Desenvolvido pelos engenheiros do Facebook e que consiste em uma série de ferramentas que viabilizam a criação de aplicações móveis nativas para a plataforma iOS e Android, utilizando o que há de mais moderno no desenvolvimento Front-end.

O React é o competidor direto do Angular em relação à popularidade, tutoriais e comunidade. É um framework veloz, sendo utilizado pelo Facebook para seu aplicativo. O React Native permite desenvolver aplicativos móveis com HTML, CSS e JavaScript. O código é compilado para código nativo Java, Object-C, Swift entregando uma experiência nativa ao usuário final (CABRAL, 2016).

O React Native não é uma plataforma de desenvolvimento de aplicativos híbridos, diferentemente do Ionic. A filosofia do React Native é "*learn once, write anywhere*" (aprenda uma vez, escreva em qualquer lugar), isso significa que será necessário escrever um código para IOS e outro para Android, obrigando a manutenção de códigos em plataformas diferentes, o que implica em maior custo em desenvolvimento e manutenção. O que é compensado na melhor organização de código que o React oferece é a vantagem dos aplicativos serem nativos, diminuindo gargalos de performance e discrepâncias entre

componentes, o que os aplicativos que utilizam webview, apresentam. A versão utilizada neste trabalho foi a 0.4.

METODOLOGIA

A aplicação criada em JAVA envolve mais componentes para o seu funcionamento, destacando-se:

- Mains_activity.xml: template a ser carregado no projeto principal, elemento de layout;
- Main.java: classe principal da aplicação, contendo o método main;
- Adapter.java: classe responsável pode gerenciar as entradas no componente ListView;
- Api.java: classe responsável por efetuar as requisições para a API da Amazon.

A aplicação em React-Native, utilizando apenas o componente ListView, se baseia em módulos oferecidos pelo framework, simplificando e diminuindo o tempo de codificação. Os arquivos constantes limitam-se a:

- List.js: o componente ListView em sua forma padrão;
- RowContent.js: um modelo de layout a ser aplicado em cada linha da lista.

COLETA DOS DADOS

Para a coleta dos dados posteriormente utilizados para a tabulação e à exibição dos gráficos, foi desenvolvida uma função auxiliar chamada readCPUUsage(), que calcula o uso do CPU. A função é exibida a seguir.

```
private float readCPUUsage() {
   try {
      RandomAccessFile reader = new
RandomAccessFile("/proc/stat", "r");
      String load = reader.readLine();

      String[] toks = load.split(" +");

      long idle1 = Long.parseLong(toks[4]);
      long cpu1 =
         Long.parseLong(toks[2]) +
         Long.parseLong(toks[3]) +
         Long.parseLong(toks[5]) +
         Long.parseLong(toks[6]) +
         Long.parseLong(toks[7]) +
         Long.parseLong(toks[8]);

      try {
         Thread.sleep(360);
      } catch (Exception e) {}

      reader.seek(0);
      load = reader.readLine();
      reader.close();

      toks = load.split(" +");

      long idle2 = Long.parseLong(toks[4]);
      long cpu2 = Long.parseLong(toks[2]) +
               Long.parseLong(toks[3]) +
               Long.parseLong(toks[5]) +
               Long.parseLong(toks[6]) +
               Long.parseLong(toks[7]) +
               Long.parseLong(toks[8]);

      return (float)(cpu2 - cpu1) / ((cpu2 + idle2) - (cpu1
+ idle1));

   } catch (IOException ex) {
      ex.printStackTrace();
   }

   return 0;
}
```

Também foi desenvolvido um método para a leitura do uso de memória RAM, neste método foi feito uma sobrescrita (Override) na classe principal MemoryInfo. A função readRAMUsage() pode ser vista a seguir:

```
private double readUsageRAM() {

   try {

      MemoryInfo mi = new MemoryInfo();

      ActivityManager activityManager =
(ActivityManager)
getSystemService(ACTIVITY_SERVICE);
      activityManager.getMemoryInfo(mi);
      double availableMegs = mi.availMem /
0x100000L;

      //Percentage can be calculated for API 16+
      double percentAvail = mi.availMem /
(double)mi.totalMem;

   } catch (IOException ex) {
      ex.printStackTrace();
   }

   return percentAvail;
}
```

No que tange a abertura da aplicação, foram realizadas 25 leituras do tempo de inicialização da mesma. Para tanto, um método denominado time() foi desenvolvido para a averiguação de tempo de abertura da aplicação. Este é executada no início da classe main de ambas as aplicações para capturar o monento de início. E, da mesma forma, no método onCreate() da página que renderiza a lista, visando capturar o término da criação da

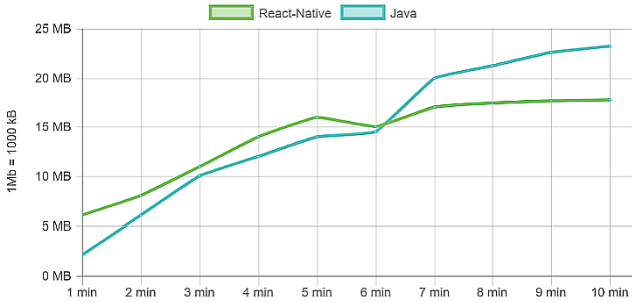
lista. O intervalo entre a primeira chamada e a seguinte é o tempo para a abertura da aplicação.

```
private int timer (Boolean trigger) {
    // variável membro da classe de controle
    if(trigger) {
        start = System.currentTimeMillis();
    } else {
        end = System.currentTimeMillis();
    }

    if( NumberUtils.isNumber(start) &&
NumberUtils.isNumber(start)) {
        return duration = end - start;
    }

    Return false;
}
```

RESULTADOS

A Figura 1 apresenta desempenho de ambas as aplicações, JAVA Nativo e React Native, no aspecto do uso de memória. As leituras do consumo foram realizadas a cada 5 segundos paralelamente a rolagem automática da lista em razão de uma linha por segundo. É obserservável que o React Native exige mais memória para a inicialização. Este fato pode decorrer do número de componentes e bibliotecas oferecidas pelo React Package por padrão. Até os 6:10 de uso da aplicação, o JAVA mostra-se mais eficiente no gerenciamento de memória, após os 6:10 o React Native apresenta um gerenciamento mais eficiente, é provável que a otimização esteja atrelada ao fato de que o React Native possui um gerenciador de linhas nativo em seu código fonte, tornando a performance da aplicação satisfatória para centenas de registros simultaneamente carregados em um ListView. Ressalta-se a possibilidade do uso de RecyclerView ao invés de ListView no desenvolvimento JAVA, porém este ainda não tem seu uso disseminado nas aplicações, por ser encontrado a partir da versão 7 do ambiente Android.

Figura 1: Gráfico de desempenho no gerenciamento de memória RAM entre Java e React Native.



Fonte: Os autores, 2007

O Quadro 1 detalha, por minuto, o consumo (em megabytes) de memória.

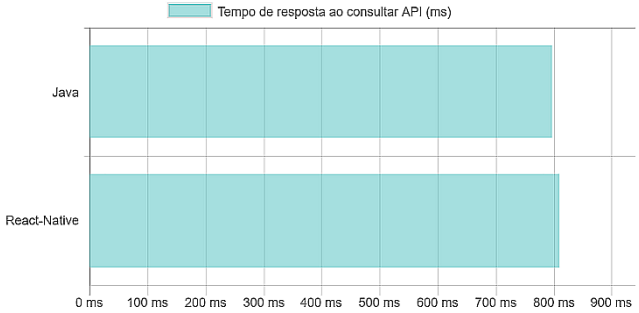
Quadro 1: Uso de memória nos 10 primeiros minutos da aplicação.

	JAVA	React Native
1 min	2,3	6,1
2 min	6,3	8,2
3 min	10,3	11,2
4 min	12,4	14,2
5 min	14,1	16,7
6 min	14,5	15,3
7 min	20,7	17,1
8 min	21,4	17,5
9 min	22,6	17,7
10 min	23	17,9

Fonte: Os autores, 2007

Na Figura 2, apresenta-se o desempenho de ambos os aplicativos em relação ao tempo de resposta na requisição HTTP para uma API externa. Observa-se que o React Native possui um desempenho levemente inferior, sendo 13 milissegundos mais lento que a requisição realizada em JAVA. A comparação foi realizada atavés de 25 chamadas em cada ambiente.

Figura 2: Gráfico de tempo médio de resposta na requisição HTTP entre Java e React Native.



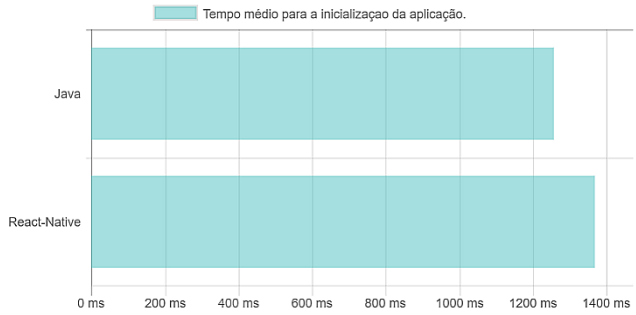
Fonte: Os autores, 2007

A Figura 2 mostra que, em média, a aplicação JAVA nativa precisou de 797 ms para obter e armazenar os dados no ArrayList, enquanto o React-Native precisou de 810 ms, uma diferença de 1,63%.

A Figura. 3 apresenta o tempo de inicialização da aplicação, isto é, desde o momento que a aplicação é chamada até a mesma estar pronta para utilização (com os dados carregados na ListView). Nota-se que o React Native possui um desempenho levemente inferior, sendo

111 ms mais lento que o processo feito em JAVA. Foram realizados 25 inicializações em cada ambiente para a obtenção dos valores apresentados.

Figura 3: Tempo médio de inicialização da aplicação (JAVA x React Native).



Fonte: Os autores, 2007

O ambiente JAVA apresentou um valor médio de 1258 ms para inicialização, enquanto o React Native levou 1369 ms, uma diferença a favor do JAVA de 8,82%.

CONCLUSÃO

Observa-se uma superioridade pequena, mas ainda assim, relevante das aplicações em JAVA Nativo em contraponto ao React Native. Cabe citar que o framework já foi apresentado com a frase: "[...] React Native is all about bringing the speed and agility of web app development to the hybrid space - with native results. " (WODEHOUSE, 2016).

Importa ressaltar que, a análise deste projeto toma apenas um componente (ainda que importante e de utilização massiva nas aplicações) para avaliação e, ainda assim, em uma situação bastante específica. Não sendo possível, de forma alguma que, as conclusões observadas neste artigo, possam ser estendidas a toda uma ferramenta, ambiente, framework, desenvolvedor ou qualquer outro elemento envolvido. Ou seja, o resultado deste trabalho deve ser avaliado somente a luz da situação averiguada.

O JAVA Nativo mostrou-se nos testes mais eficiente, principalmente quando se necessita que desempenho seja fator indispensável, como em uma aplicação de alta performance.

Também foi observado que em aplicações se exija maior flexibilidade, o JAVA apresenta uma maior variedade de componentes e uma arquitetura mais aberta.

O React Native se destacou nos testes de gerenciamento de memória RAM, que neste estudo foi aplicado na renderização e disponibilização de uma lista em um componente ListView.

Apesar do framework React Native não se mostrar tão eficiente quanto o JAVA, cabe destacar que este traz um meio único de desenvolvimento, isto é, uma vez que o desenvolvedor aprende o funcionamento do framework, pode-se aplica-lo tanto ao ambiente iOS quanto Android. O que seria impossível com o JAVA, uma vez que este seria funcional apenas para o ambiente Android.

Assim, mesmo que o React Native não ofereça o mesmo desempenho, robustez e flexibilidade que o ambiente JAVA, este pode ser aplicado em muitas aplicações que precisam funcionar nos maiores expoentes do mercado móvel atual (iOS e Android). Diminuindo o custo de treinamento e aprendizado de linguagens e ferramentas tão distintas.

REFERÊNCIAS

ANDROID. **O IDE oficial do Android**. 2017. Disponível em: <https://developer.android.com/studio/index.html>. Acesso em: 25 abr. 2017.

BOSOMWORTH, Danyl. Mobile marketing statistics 2015. **Leeds: Smart Insights (Marketing Intelligence) Ltd**.

CABRAL, Carlos. **Construa aplicações móveis nativas com JavaScript.** 2016. Disponível em: <https://tableless.com.br/react-native-construa-aplicacoes-moveis-nativas-com-javascript/>. Acesso em: 27 abr. 2017.

PUCCINELLI, Maurilio. **O que é Node.js**. 2015. Disponível em: <http://agencia.yesbr.com.br/aplicativos/o-que-e-node-js-para-que-serve-e-como-crio-um-servidor-web-simples-com-uma-pagina-simples-tutorial-rapido-e-que-funciona/>. Acesso em: 07 maio 2017

WODEHOUSE, Carey. **7 Reasons Why Facebook's React Native Is the Future of Hybrid App Development**, 2010. Disponível em: <https://www.upwork.com/hiring/mobile/react-native-hybrid-app-development/>. Acesso em: 28 junho. 2017