UNIVERSIDADE FEDERAL DO PAMPA

Fernando Fortunato de Lima

Avaliação de Frameworks para o Desenvolvimento de Aplicações Híbridas

Alegrete
2019

Fernando Fortunato de Lima

Avaliação de Frameworks para o Desenvolvimento de Aplicações Híbridas

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

Orientador: Prof. Doutora Aline Vieira de Mello

Alegrete
2019

Fernando Fortunato de Lima

Avaliação de Frameworks para o Desenvolvimento de Aplicações Híbridas

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

Trabalho de Conclusão de Curso defendido e aprovado em 24 de junho de 2019
Banca examinadora:

Aline Vieira de Mello

Prof. Doutora Aline Vieira de Mello
Orientador
UNIPAMPA

Fábio Paulo Basso

Prof. Doutor Fábio Paulo Basso
UNIPAMPA

Prof. Doutor João Pablo Silva da Silva
UNIPAMPA

Dedico este trabalho ao meu falecido avô Benedito Luiz, que se estivesse entre nós, faria qualquer coisa para ver a apresentação deste trabalho, serei sempre grato pela dedicação do senhor em vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me dado força, sabedoria e paciência para que eu pudesse chegar onde estou.

Aos meus pais, Claudete e Júlio César, pela paciência que apresentaram quando eu caia ou me desviava do meu caminho; compreensão pelas dificuldades que enfrentei ao me deparar com alguns desafios; apoio nos momentos que precisei de conselhos; sacrifícios que fizeram para que eu pudesse chegar até aqui e o amor incondicional que demonstraram no decorrer dos anos.

À minha namorada e companheira Mayara, por sempre me incentivar a ser um homem melhor, capacitado e paciente, e pela companhia durante esta estrada cheia de pedras e espinhos chamada vida.

Ao meu amigo e irmão Raul e sua esposa Talita, que me auxiliaram de forma esplendida neste trabalho, apesar da distância que eu estava de casa, nunca deixaram nossa amizade morrer.

À minha orientadora Aline por me acolher, auxiliar e guiar para que pudesse desenvolver e apresentar este trabalho.

E aos meus amigos Willian, Pedro, Zenio, Claudio, Ricardo, Cortez, Leonardo, Guilherme e Elias, que fizeram parte da minha vida nestes anos de graduação, onde acredito que de amigos tornaram-se meus irmãos.

"Não tenha pressa, não viva rápido demais
Dificuldades virão e passarão
Encontre uma mulher, e encontrará o amor
E não esqueça, filho, existe alguém lá em cima."
(Lynyrd Skynyrd)

RESUMO

O uso de aplicativos móveis auxilia no cotidiano da vida das pessoas. Estes aplicativos podem ajudar em diversos contextos, como na vida social, na comunicação, no entretenimento, entre outros. Por este motivo, o aumento na demanda de aplicativos publicados nas lojas virtuais cresce a cada dia, resultando no crescimento do mercado de desenvolvimento de aplicações. Para realizar a publicação de uma aplicação nas lojas virtuais, deve ser implementado duas ou mais aplicações, dependendo no número de Sistemas Operacionais (SO) que se deseja atingir. O custo para a criação de cada aplicação é alto, contando com o custo dos recursos, tempo, manutenção e evolução, portanto, manter dois ou mais aplicativos é uma tarefa cara. Para amenizar este problema, foram criados *frameworks* para desenvolvimento de aplicações móveis, ou seja, é utilizado uma plataforma base para implementar uma única aplicação, da qual será gerado aplicativos para um ou mais SO. Apesar de ser uma ótima solução para os profissionais da área, cada *framework* possui um conjunto de vantagens e desvantagens que dependem das características da aplicação que será desenvolvida, dificultando a decisão de qual utilizar. Em vista desse problema, o objetivo deste trabalho é realizar a comparação entre *frameworks* e apresentar dados que auxiliem os profissionais na melhor escolha para seu contexto. Para alcançar esse objetivo, foi realizado um mapeamento sistemático visando levantar as principais características comparadas pelos trabalhos relacionados a este. Mini-aplicações para cada funcionalidade nativa dos dispositivos foram implementadas usando os *frameworks* Flutter, Ionic, NativeScript-Vue.js e React Native. Uma bateria de testes foi executada nos SO Android e iOS para obter as médias de tempo de execução, consumo de memória e uso do processador. Adicionalmente, um questionário foi elaborado e distribuído para a comunidade de usuários destes *frameworks*, com o objetivo de obter a opinião dos profissionais e dados que não podem ser evidenciados nos testes. Os resultados apontam que o React Native é o *framework* mais conhecido pelos respondentes, sendo o que apresentou menor tempo de desenvolvimento e mais indicado para aplicações complexas. Em contrapartida, o Ionic provou ser o menos aconselhado para aplicações complexas ou com acesso as funcionalidades nativas do dispositivo. O Flutter apresentou os menores valores médios dos testes de consumo de memória e uso do processador, porém, foi apontado pelos participantes do questionário o fato de utilizar DART como linguagem de programação ser um ponto negativo. O NativeScript-Vue.js é o menos conhecido e utilizado, apresentando os melhores tempos médios para o SO iOS, menos para a mini-aplicação Bluetooth.

Palavras-chave: Desenvolvimento. Comparação. Aplicações Híbridas. Aplicações Móveis. Framework.

ABSTRACT

The widespread use of mobile applications facilitates people's daily tasks. These applications can help people in a variety of ways, such as social life, communications, entertainment and several others. As a consequence, the demand for this applications is in constant rising creating a huge market for the development of such applications. Most apps target one or more operational system, considering that, developers must build the same applications more than once one for each OS targeted. The cost of development of said applications is high, it must take into account resources, planning, time, maintenance and evolution of each application developed. Targeting two or more OSs will most likely double the price of an application. To mitigate the aforementioned cost problem, mobile application development frameworks were built to provide a base platform to build one single application that can target one or more OS. In spite of being a great solution for developers, it is a challenge to choose the right framework for the task at hand. This work aims to compare frameworks, with the objective to present data for developers to facilitate choosing the right framework for the problem they have. For that, this work presents a systematic mapping aiming to find the main features of a framework looking into related works. Several small applications were also implemented to test some of the native functionalities of each device, and a battery of tests to obtain data about excecution time, memory consumption and use of processing power. Furthermore, a questionaire was created and sent to a community of mobile developers looking to get some opinions from professionals and data that cannot be collected through testing. The results indicate that the most recognized Framework by the respondents was React Native; this framework also presented the fastest development time and is also the most indicated for complex applications. Ionic presented itself as the least recommended framework for complex applications or to use native tools. Flutter had the least median values in the memory consumption and processor usage tests, however the respondents pointed that using a DART as a programming language is a major drawback. Native Script-Vue.js is the least known and used. Nevertheless presented the best mean times for iOS, excluding the Bluetooth mini application.

Key-words: Development. Comparison. Hybrid Applications. Mobile Applications. Framework.

LISTA DE FIGURAS

Figura 1 – Sistemas operacionais vendidos entre 2009 a 2018. 29
Figura 2 – Funcionamento das aplicações nativas. 34
Figura 3 – Funcionamento das aplicações web. 35
Figura 4 – Funcionamento das aplicações híbridas. 35
Figura 5 – Etapas do processo de seleção. 42
Figura 6 – Protótipo gráfico da funcionalidade Bluetooth. 58
Figura 7 – Protótipo gráfico da funcionalidade Câmera. 59
Figura 8 – Protótipo gráfico da funcionalidade GPS. 59
Figura 9 – Protótipo gráfico da funcionalidade Armazenamento Interno. 60
Figura 10 – Diagrama de pacotes das mini-aplicações. 61
Figura 11 – Diagrama de atividade do script de teste Bluetooth. 62
Figura 12 – Diagrama de atividade do script de teste Câmera. 63
Figura 13 – Diagrama de atividade do script de teste GPS. 64
Figura 14 – Diagrama de atividade do script de teste Armazenamento Interno. . . . 64
Figura 15 – Quantidade de respondentes que conhecem os *Frameworks*. 76
Figura 16 – Relatório do desempenho e uso de memória Bluetooth/Android. 94
Figura 17 – Relatório do desempenho e uso de memória Camera/Android. 96
Figura 18 – Relatório do desempenho e uso de memória GPS/Android. 98
Figura 19 – Relatório do desempenho e uso de memória Armazenamento/Android. 100
Figura 20 – Relatório do desempenho e uso de memória Bluetooth/iOS. 102
Figura 21 – Relatório do desempenho e uso de memória Câmera/iOS. 104
Figura 22 – Relatório do desempenho e uso de memória GPS/iOS. 106
Figura 23 – Relatório do desempenho e uso de memória Armazenamento Interno/iOS.108

LISTA DE TABELAS

Tabela 1 – Características positivas e negativas do iOS e Android. 30
Tabela 2 – *Strings*. 41
Tabela 3 – Resultados das bibliotecas. 43
Tabela 4 – Classificação dos trabalhos. 44
Tabela 5 – *Frameworks* apresentados pelos estudos. 44
Tabela 6 – Número de estudos que avaliam cada *Framework*. 45
Tabela 7 – Grupos dos tipos de aplicações. 45
Tabela 8 – Características comparadas em cada estudo. 46
Tabela 9 – Metodologias de comparação em cada estudo. 50
Tabela 10 – Estágios para o desenvolvimento de mini-aplicações. 53
Tabela 11 – Descrição dos dispositivos. 55
Tabela 12 – Estágios para produção do questionário. 55
Tabela 13 – Questões sobre os *Frameworks*. 66
Tabela 14 – Tempo de execução da mini-aplicação Bluetooth. 69
Tabela 15 – Tempo de execução da mini-aplicação Câmera. 70
Tabela 16 – Tempo de execução da mini-aplicação GPS. 70
Tabela 17 – Tempo de execução para a mini-aplicação Memória. 70
Tabela 18 – Soma do tempo médio de execução para todas as mini-aplicações. . . . 71
Tabela 19 – Consumo de memória da mini-aplicação Bluetooth. 71
Tabela 20 – Consumo de memória da mini-aplicação Câmera. 71
Tabela 21 – Consumo de memória da mini-aplicação GPS. 72
Tabela 22 – Consumo de memória da mini-aplicação Memória. 72
Tabela 23 – Soma do consumo médio de memória para todas as mini-aplicações. . . 72
Tabela 24 – Uso do processador para a mini-aplicação Bluetooth. 73
Tabela 25 – Desempenho da mini-aplicação Câmera. 73
Tabela 26 – Uso do processador para a mini-aplicação GPS. 74
Tabela 27 – Uso do processador para a mini-aplicação Memória. 74
Tabela 28 – Soma do uso médio do processador para todas as mini-aplicações. . . . 74
Tabela 29 – Nível de dificuldade para aprender o *Framework*. 76
Tabela 30 – Nível de satisfação da documentação dos *Framework*. 77
Tabela 31 – Velocidade de construção/compilação do *Framework*. 77
Tabela 32 – Complexidade do *framework* para acessar as funcionalidades. 77
Tabela 33 – Vantagens dos *frameworks*. 78
Tabela 34 – Desvantagens dos *frameworks*. 79
Tabela 35 – Dificuldades dos *frameworks*. 80
Tabela 36 – Pontos positivos encontrados no uso dos *Frameworks*. 81

LISTA DE ABREVIATURAS

Inc. *Incorporation*

LISTA DE SIGLAS

`AOT` A Frente do Tempo (do inglês *Ahead-Of-Time*)

API Interface de Programação de Aplicações (do inglês *Application Program Interface*)

CE Critérios de Exclusão

CI Critérios de Inclusão

`CLI` Interface de Linha de Comando (do inglês *Command-Line Interface*)

CQ Critérios de Qualidade

`CSS` Folha de Estilo em Cascatas (do inglês *Cascading Style Sheets*)

`D&D` Arrastar e Soltar (do inglês *Drag and Drop*)

GPS Sistema de Posicionamento Global (do inglês *Global Positioning System*)

GSM Sistema Global para Comunicações Móveis (do inglês *Global System for Mobile communications*)

`HU` Histórias de Usuários

`IDE` Ambiente de Desenvolvimento Integrado (do inglês *Integrated Development Environment*)

`IHC` Interação Humano-Computador

`MVC Model-View-Controller`

`SO` Sistemas Operacionais

PICO População/Problema, Intervenção, Comparação, Resultado (do inglês *Population/Problem, Intervention, Comparison, Outcome*)

QP Questões de Pesquisa

`QQ` Questões do Questionário

`SDK` Kit de Desenvolvimento de Software (do inglês *Software Development Kit*)

W3C Consórcio *World Wide Web* (do inglês *World Wide Web Consortium*)

SUMÁRIO

1 INTRODUÇÃO 27
1.1 Objetivo 27
1.2 Organização do Trabalho 28

2 FUNDAMENTAÇÃO TEÓRICA 29
2.1 Plataformas 29
2.1.1 Android 30
2.1.2 iOS 31
2.2 Aplicações Móveis 31
2.2.1 Características 31
2.2.2 Categorias 33
2.3 *Frameworks* 36
2.3.1 Flutter 37
2.3.2 Ionic 37
2.3.3 NativeScript-Vue.js 38
2.3.4 React Native 38

3 REVISÃO DE LITERATURA 39
3.1 Mapeamento Sistemático 39
3.1.1 Planejamento 39
3.1.1.1 Escopo e Objetivo 39
3.1.1.2 Estrutura das Questões 39
3.1.1.3 Questões de Pesquisa 40
3.1.1.4 Processo de Busca 40
3.1.1.5 Critérios de Inclusão e Exclusão 40
3.1.1.6 Critérios de Qualidade 41
3.1.1.7 Processo de Seleção 42
3.1.2 Condução 43
3.2 Discussão Sobre os Trabalhos Relacionados 43
3.2.1 *Frameworks* Para Desenvolvimento de Aplicações Móveis . . 44
3.2.2 Características Comparadas Entre as *Frameworks* 45
3.2.3 Vantagens e Desvantagens Encontradas em Cada *Framework* 46
3.2.3.1 PhoneGap 47
3.2.3.2 Appcelerator Titanium Studio 2.0 47
3.2.3.3 Sencha Touch 2 47
3.2.3.4 Ionic 47
3.2.3.5 JQuery e JQuery Mobile 48
3.2.3.6 MoSync 48

3.2.3.7 DragonRAD . . . 48
3.2.3.8 NativeScript . . . 49
3.2.3.9 React Native . . . 49
3.2.3.10 Rhodes . . . 49
3.2.4 Metodologias de Comparação Apresentadas nos Trabalhos . . . 49
3.3 Considerações Finais . . . 50

4 METODOLOGIA . . . 53
4.1 Mini-Aplicações . . . 53
4.1.1 Funcionalidades . . . 53
4.1.2 Métricas . . . 54
4.1.3 Ambiente de Execução . . . 54
4.1.4 Infraestrutura . . . 54
4.2 Questionário . . . 54
4.3 Procedimento . . . 54

5 DESENVOLVIMENTO . . . 57
5.1 Desenvolvimento de Mini-Aplicações . . . 57
5.1.1 Histórias de Usuário . . . 57
5.1.2 Protótipo . . . 58
5.1.3 Arquitetura . . . 60
5.1.4 Implementação . . . 61
5.1.5 Script de Teste . . . 62
5.2 Desenvolvimento do Questionário . . . 65
5.2.1 Objetivos . . . 65
5.2.2 Público-Alvo . . . 65
5.2.3 Amostragem . . . 65
5.2.4 Elaborar Questões . . . 65
5.2.5 Distribuição . . . 66
5.2.6 Análise . . . 67

6 RESULTADOS . . . 69
6.1 Mini-Aplicações . . . 69
6.1.1 Tempo de Execução . . . 69
6.1.2 Consumo de Memória . . . 70
6.1.3 Uso do Processador . . . 72
6.1.4 Resumo . . . 74
6.2 Questionário . . . 75
6.2.1 Resumo . . . 81

7 CONSIDERAÇÕES FINAIS . . . 83

REFERÊNCIAS 85

APÊNDICES 91

APÊNDICE A – RESULTADOS BLUETOOTH ANDROID 93

APÊNDICE B – RESULTADOS CÂMERA ANDROID . . . 95

APÊNDICE C – RESULTADOS GPS ANDROID 97

APÊNDICE D – RESULTADOS ARMAZENAMENTO INTERNO ANDROID 99

APÊNDICE E – RESULTADOS BLUETOOTH IOS 101

APÊNDICE F – RESULTADOS CÂMERA IOS 103

APÊNDICE G – RESULTADOS GPS IOS 105

APÊNDICE H – RESULTADOS ARMAZENAMENTO INTERNO IOS 107

APÊNDICE I – QUESTIONÁRIO 109

1 INTRODUÇÃO

Aplicações móveis são utilizadas para facilitar e auxiliar o cotidiano das pessoas em diversos contextos, como por exemplo na realização de compras online em segundos, compartilhar novidades da vida pessoal nas redes sociais, relaxar com jogos e aplicativos de entretenimento, conversar com amigos ou familiares em outras cidades, estados ou países com aplicativos de comunicação, entre outros.

O crescimento do uso de aplicativos móveis implica no crescimento do mercado de desenvolvimento de aplicações móveis (NELSON, 2016; STATISTA, 2019). Contudo, a publicação de um aplicativo para ambos `SO` `Android` e `iOS` demanda tempo e recurso, como o custo do desenvolvimento de dois aplicativos, tempo de produção para ambos, manutenção e evolução das aplicações (STATE, 2018).

Com o objetivo de amenizar os problemas com custos do desenvolvimento, foram criados os *frameworks* de desenvolvimento para aplicações híbridas (GUPTA, 2018). Esses *frameworks* são plataformas de desenvolvimento que permitem gerar aplicações para dois ou mais `SO`, realizando a implementação de apenas um aplicativo. O resultado da adoção dessa solução reduz significativamente os custos, o tempo de produção, a manutenção e a evolução dos aplicativos.

Atualmente, existem diversos *frameworks* que se diferenciam pela linguagem de programação utilizada, arquitetura implementada, modos de gerar arquivos executáveis e estratégia de reutilização de artefatos e ativos (XANTHOPOULOS; XINOGALOS, 2013). A decisão de qual *framework* utilizar para a criação do aplicativo móvel depende de alguns fatores como a expertise dos desenvolvedores com a linguagem de programação, a curva de aprendizagem caso não haja conhecimento da linguagem, qual o tipo de aplicação será criada, quais funcionalidades serão implementadas, se será necessário utilizar de funcionalidades nativas do dispositivo e para quais `SO` serão publicados.

De acordo com UnfoldLabs (2019), apesar da quantidade de *frameworks* disponíveis, os desenvolvedores devem possuir conhecimentos básicos de cada um, para que assim, tenham base para tomar a decisão de qual utilizar. Entretanto, para conhecer esses *frameworks*, os desenvolvedores necessitariam realizar pesquisas e desenvolver aplicativos teste e, portanto, dispensar tempo para gerar esses conhecimentos, o que muitas vezes é inviável no período de desenvolvimento de um produto.

1.1 Objetivo

Este trabalho tem como objetivo geral realizar a comparação entre *frameworks* para desenvolvimento de aplicações móveis, visando auxiliar desenvolvedores a decidir qual a melhor opção. Os *frameworks* descritos e comparados neste trabalho foram selecionadas a partir dos trabalhos relacionados encontrados nas bibliotecas digitais, de buscas por inovações nesta área de desenvolvimento de aplicações e indicações em meio a indústria de

software. A partir dos trabalhos relacionados, foram selecionados Ionic[1] e React Native[2]. O NativeScript-Vue.js[3] foi definido por indicações de desenvolvedores que atuam nesta área no mercado de trabalho. O Flutter[4] foi encontrado como inovação de *framework*, considerando como funciona e a linguagem de programação que utiliza.

Para melhor organização e garantia de que este trabalho atinja seu objetivo geral, foram estipulados os seguintes objetivos específicos:

- Identificar as características que são frequentemente utilizadas para comparar os *frameworks* de desenvolvimento para aplicações híbridas;
- Reconhecer as metodologias comumente utilizadas para realizar a comparação entre os *frameworks*;
- Obter a opinião dos desenvolvedores sobre sua experiência no uso dos *frameworks*;
- Apresentar e discutir os resultados obtidos das comparações.

1.2 Organização do Trabalho

Este trabalho está organizado seguindo a descrição dos capítulos a seguir:

- O Capítulo 2 descreve os conceitos sobre sistemas operacionais, aplicações móveis e *frameworks*;
- O Capítulo 3 apresenta o mapeamento sistemático executado visando obter trabalhos relacionados a este, seus objetivos, critérios utilizados nas comparações e quais características foram comparadas.
- O Capítulo 4 explica a metodologia seguida para o desenvolvimento deste trabalho;
- O Capítulo 5 detalha como foi realizado cada etapa deste trabalho, sendo elas: a implementação das mini-aplicações, o desenvolvimento do questionário e a execução dos testes;
- O Capítulo 6 apresenta a comparação dos resultados obtidos pela execução dos testes e aplicação do questionário;
- O Capítulo 7 apresenta as considerações finais e descreve os trabalhos futuros.

[1] <https://ionicframework.com>
[2] <https://facebook.github.io/react-native>
[3] <https://www.nativescript.org>
[4] <https://flutter.io/>

2 FUNDAMENTAÇÃO TEÓRICA

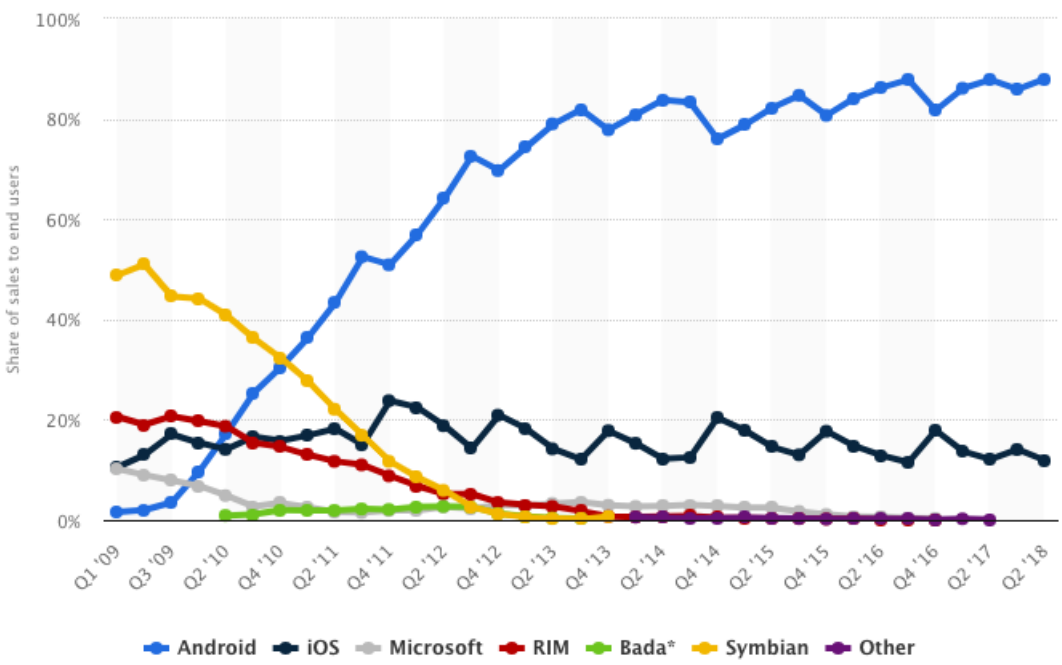
Neste Capítulo são revisados os conceitos necessários para a compreensão deste trabalho, apresentando conceitos relacionados as aplicações móveis que podem ser divididos em aplicações nativas, web e híbridas, sistemas operacionais e *frameworks* para desenvolvimento de aplicações híbridas.

2.1 Plataformas

Plataformas, ou `SO`, realzam a ponte de interação entre o usuário e o dispositivo. Sua tarefa é gerenciar os recursos, ou seja, é utilizado para o *software* e *hardware* de aparelhos celulares, tablets, televisões entre outros dispositivos (GOODWILL, 2018).

As estatísticas, apresentadas por Statista (2018), mostram que os `SO` mais vendidos no mundo, são `Android` (GOOGLE, 2018a) e `iOS` (APPLE, 2018a). A Figura 1 apresenta um gráficos dos `SO` mais vendidos entre 2009 e 2018. Pode-se observar a queda nas vendas do Symbian, pois era o pioneiro em `SO` para dispositivos móveis e atualmente foi descontinuado. O `iOS` sempre manteve-se no mesmo patamar de vendas, enquanto o `Android` aumenta continuamente suas vendas.

Figura 1 – Sistemas operacionais vendidos entre 2009 a 2018.



Fonte – (STATISTA, 2018)

A Tabela 1 apresenta pontos positivos e negativos dos `SO` `Android` e `iOS`, que foram levantados por Bergher (2012). Pode-se notar que, com a estratégia de possuir o código aberto, o `Android` possui uma grande quantidade e diversidade de jogos e aplicações. Por

outro lado, o iOS possui um número menor de aplicações, porém, a qualidade de desempenho e gráficos são melhores. A maioria dos aplicativos para iOS são para necessidades do tipo de compras, notícias e gerenciamentos de tarefas.

Tabela 1 – Características positivas e negativas do iOS e Android.

Características	**Sistemas**	
Sistema Operacional	iOS	Android
Empresa	Apple	Google
Loja de Aplicativos	Possui as melhores aplicações móveis do mercado	Por ter código aberto, possui uma lista extensa de aplicações móveis
Navegações na Internet e Comunicação	Veloz, funcional e intuitivo. Peca pela incompatibilidade com sites em Flash	Ótimo para quem usa os serviços Google. Funciona com flash
Uso para Jogos	Possui os melhores jogos também exibe ótimos gráficos	Tem a maior diversidade e variedade de games

Fonte – Adaptado de Bergher (2012)

2.1.1 Android

O Android é um SO baseado em Núcleo Linux (LINUX, 2018). Foi criado pela empresa Android *Incorporation* (Inc.) (SCHUMAN, 2018), posteriormente sendo comprado pela Google (CALLAHAM, 2018). É desenvolvido com enfoque na interface do usuário, disponibilizando atualmente SO Android para *Smartphones*, *Smartwatches*, *Tablets*, *Smart TVs* e carros.

A produção de aplicativos para Android é realizada através do Ambiente de Desenvolvimento Integrado (do inglês *Integrated Development Environment*) (IDE) Android Studio[1]. Esta IDE utiliza Java (ORACLE, 2018) como linguagem de programação nativa, atualmente, também dando suporte a Kotlin (ANDROID, 2018).

Para publicar e disponibilizar os aplicativos para Android, é necessário criar uma conta de desenvolvedor na loja de aplicativos, filmes, músicas e livros da Google, a Google Play[2]. Para criar esta conta, o usuário deve aceitar o Contrato de Desenvolvedor [3] e realizar o pagamentos da taxa de registro no valor de U$25 para obter licença vitalícia de desenvolvedor.

[1] <https://developer.android.com/studio>
[2] <https://play.google.com/store>
[3] <https://www.androidauthority.com/google-android-acquisition-884194>

2.1.2 iOS

O iOS é um SO desenvolvido em Darwin (PUREDARWIN, 2018), baseado em XNU (APPLE, 2018b). Foi criado e distribuído em Junho de 2007 (CAPUTO, 2017) pela empresa Apple Inc.. Diferente do Android, a Apple disponibiliza o SO iOS somente para iPhones e iPads. Cada tipo de dispositivo possui um SO correspondente ao seu dispositivo: as Smart TVs utilizam o tvOS, os computadores o macOS e os Smartwatches o watchOS (APPLE, 2018c).

Os aplicativos para iOS devem ser desenvolvidos na IDE XCode[4], que dá suporte as linguagens de programação Swift (REBOUÇAS et al., 2016), C (KYFONIDIS; MOUMOUTZIS; CHRISTODOULAKIS, 2017), C++ (RUIQING; XIAOHUI, 2010), e Objective-C (RAWLINGS, 1989). Uma desvantagem desta IDE é sua exclusividade com o SO macOS.

A distribuição é realizada através da App Store[5], a loja de aplicativos da Apple. Para a criação de uma conta de desenvolvedor, é necessário o pagamento da taxa anual de U$99[6] e aceitar os Termos e Condições[7].

2.2 Aplicações Móveis

De acordo com o dicionário de Cambridge[8], a definição de Aplicação Móvel é: "Um *software* executado em telefones móveis" (CAMBRIDGE, 2018). Em outras palavras, aplicações móveis são *softwares* semelhantes aos usados em computadores, porém, as funcionalidades possuem escopo limitadas, objetivos específicos e portáteis, para possibilitar seu uso a qualquer momento. Apesar desta definição tratar somente de um tipo de dispositivo, atualmente quaisquer aplicações que estejam sendo executadas em dispositivos que possuam SO, tais como *Smartphones*, *Tablets*, *Smart TVs*, e *Smartwatches* são consideradas como aplicações móveis.

2.2.1 Características

Como as aplicações móveis são desenvolvidas para *hardwares*, SO e objetivos específicos, há um conjunto de características (ou requisitos) que devem ser levadas em consideração pelos desenvolvedores em sua criação e distribuição (WASSERMAN, 2010). Sendo tais características:

[4] <https://developer.apple.com/xcode/ide/>
[5] <https://www.apple.com/lae/ios/app-store>
[6] <https://developer.apple.com/support/compare-memberships>
[7] <https://developer.apple.com/terms>
[8] <https://dictionary.cambridge.org/>

- **Consumo de bateria**

  Apesar do processo de criação de *hardwares* para dispositivos possuírem seu desenvolvimento focado em desempenho e economia de bateria, algumas aplicações móveis utilizam aspectos e processos mais complexos que outras, podendo vir a consumir mais recursos dos dispositivos (BALASUBRAMANIAN; BALASUBRAMANIAN; VENKATARAMANI, 2009).

- **Sensores**

  Os dispositivos atuais são compostos por diversos sensores que garantem maiores oportunidades para funcionalidades e aplicações móveis para estas funcionalidades, tais como: telas suscetíveis ao toque, acelerômetro, giroscópio, magnetômetro, Sistema de Posicionamento Global (do inglês *Global Positioning System*) (GPS), barômetro, sensor de proximidade, e sensor de luz ambiente (NIELD, 2017). Sabendo disso, desenvolvedores devem levar em consideração tais sensores, dependendo do objetivo da aplicação (KORPIPÄÄ; MÄNTYJÄRVI, 2003).

- **Interação com outras aplicações**

  Na aquisição do dispositivo, o usuário dispõe de aplicações instaladas por padrão pela empresa que desenvolveu o aparelho, pelo `SO` ou pela empresa de telefonia contratada. Como o `SO` disponibiliza a loja para adquirir novas aplicações móveis, o desenvolvedor pode desenvolver uma aplicação que pode vir a ter interação com outra (WASSERMAN, 2010).

- **Armazenamento**

  As aplicações instaladas, geralmente, não ocupam muito espaço no armazenamento interno ou externo do dispositivo. A preocupação dos desenvolvedores se baseia nos dados gerados ou baixados pela aplicação (CHARLAND; LEROUX, 2011).

- **Consumo de dados**

  Algumas aplicações são desenvolvidas para apresentação de informações, como fotos, contatos, agendas, entre outros. Essas possuem funções que necessitam de comunicação com a internet constantemente, para transferência de dados na rede. A preocupação que se deve atentar é com o alto consumo de dados para usuários que utilizam tecnologias como dados móveis (3G, 4G, Sistema Global para Comunicações Móveis (do inglês *Global System for Mobile communications*) (GSM)) (TRIPATHI et al., 2011).

- **Interface do usuário**

  A interface do usuário é uma das características críticas que devem ser consideradas ao desenvolver uma aplicação, pois é a forma de contato direto entre o sistema e

o usuário. Existem padrões em pesquisa e desenvolvimento para questões visuais, tais como cores e tamanhos das letras, fundo da tela, a coloração de abas e menus, e algumas restrições (GOOGLE, 2018b). Não somente limitando-se a estes critérios, salienta-se que interações com sensores estão evoluindo e demandam mais cuidado e atenção no desenvolvimento de aplicação.

- **Segurança**

  A segurança é uma característica que profissionais da área da computação devem atentar-se, pois se trata do nível da vulnerabilidade da aplicação, considerando o armazenamento de senhas, autenticação de usuários e proteção contra ataques externos (MANCHANDA, 2018).

2.2.2 Categorias

O crescimento expressivo de aplicativos nas lojas digitais é resultado do aumento contínuo da demanda dos usuários. Para aumentar a produção e atender a estas demandas, *frameworks* ou tecnologias web podem ser adotadas durante o desenvolvimento, pois os custos e tempos para o aprendizado e produção das aplicações são consideravelmente menores (BRITO et al., 2018). De acordo com a estratégia de desenvolvimento, as aplicações podem ser categorizadas de três formas, sendo elas:

- **Aplicações Nativas**

  São definidas como aplicações nativas, aplicativos que são desenvolvidas através da linguagem de programação definida pelo seu `SO` e que não possuem compatibilidade com outros `SO`. Por exemplo, ao desenvolver aplicativos para `Android`, a mesma não irá executar em um dispositivo com `iOS`, e vice-versa (MORE; CHANDRAN, 2016).

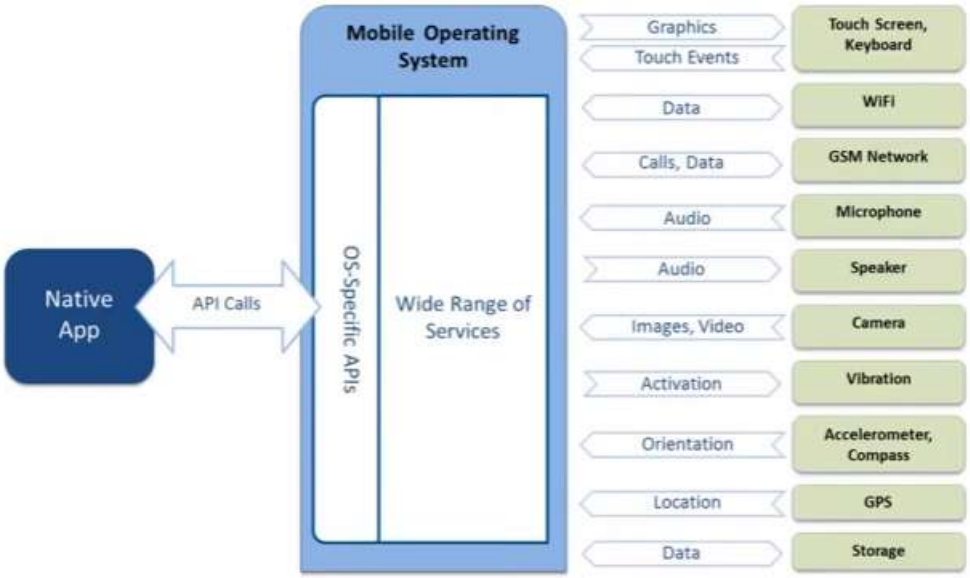
  Como pode ser observado na Figura 2, aplicações nativas não utilizam de pontes ou outras aplicações para serem executadas. As chamadas de Interface de Programação de Aplicações (do inglês *Application Program Interface*) (API) do `SO` são executadas direto da aplicação, garantindo desempenho e maior personalização.

  Quando instalado no dispositivo, o aplicativo acessa diretamente o *hardware*, sendo possível utilizar todos os sensores e funcionalidades disponibilizadas pelo `SO`, tais como teclado, WiFi, câmera, telas suscetíveis ao toque, GPS, entre outros.

- **Aplicações Web**

  São aplicações baseadas em navegadores em que não é necessário a instalação ou atualização, porém, é necessário carregar a aplicação no navegador toda vez que iniciada, sendo assim, há a obrigatoriedade do uso da Internet.

Figura 2 – Funcionamento das aplicações nativas.



Fonte – Vendors e Resources (2018)

Estas aplicações são desenvolvidas utilizando tecnologias web, tais como `JavaScript`, `HTML` e Folha de Estilo em Cascatas (do inglês *Cascading Style Sheets*) (`CSS`). Desvantagens encontradas nestas aplicações são: a limitação do acesso às funcionalidades do *hardware*, o custo e tempo de carregar o conteúdo da aplicação como as páginas e imagens.

A Figura 3 ilustra o funcionamento das aplicações web. Pode ser observado que as funcionalidades gráficas e eventos no toque de tela possuem todo o acesso disponível através de chamadas Consórcio *World Wide Web* (do inglês *World Wide Web Consortium*) (W3C) [9]. Em contra-partida, como essas aplicações são executadas via navegadores, contêineres e `Web Views`, os acesso ou chamadas para funcionalidades de transferência de dados, sistema de orientação e armazenamento são limitados. Não há acesso ou chamadas para as funcionalidades de gravação de áudio, câmera e notificações (XANTHOPOULOS; XINOGALOS, 2013).

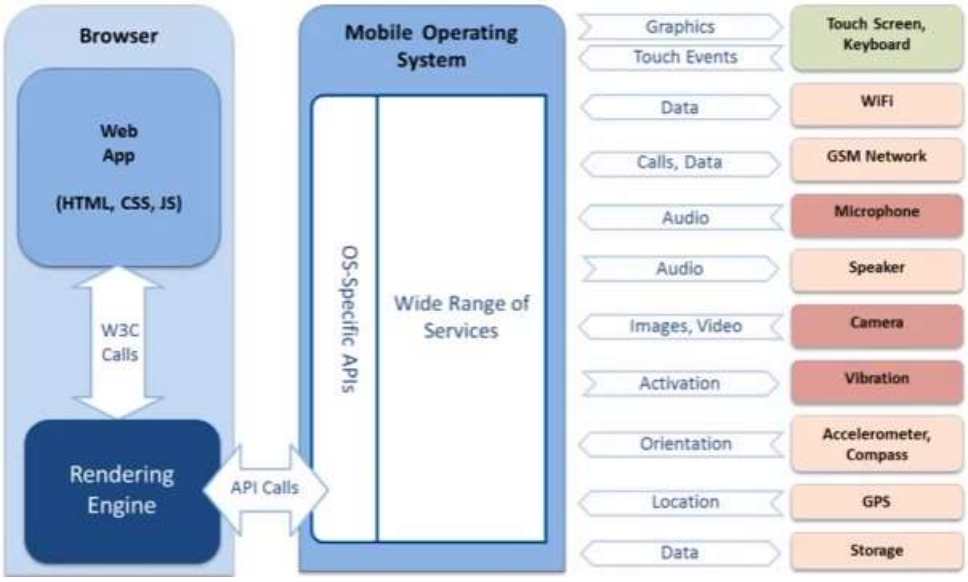
- **Aplicações Híbridas**

  As aplicações híbridas são uma mescla de aplicações nativas e web (JOBE, 2013). Essas aplicações são geralmente desenvolvidas utilizando `JavaScript`, `HTML` e `CSS`.

  A Figura 4 ilustra a divisão entre partes web e nativa dentro da aplicação desenvolvida, sendo a parte web mais voltada a interface do usuário e eventos do toque na tela, enquanto a parte nativa é acessada via chamadas `JavaScript` para APIs

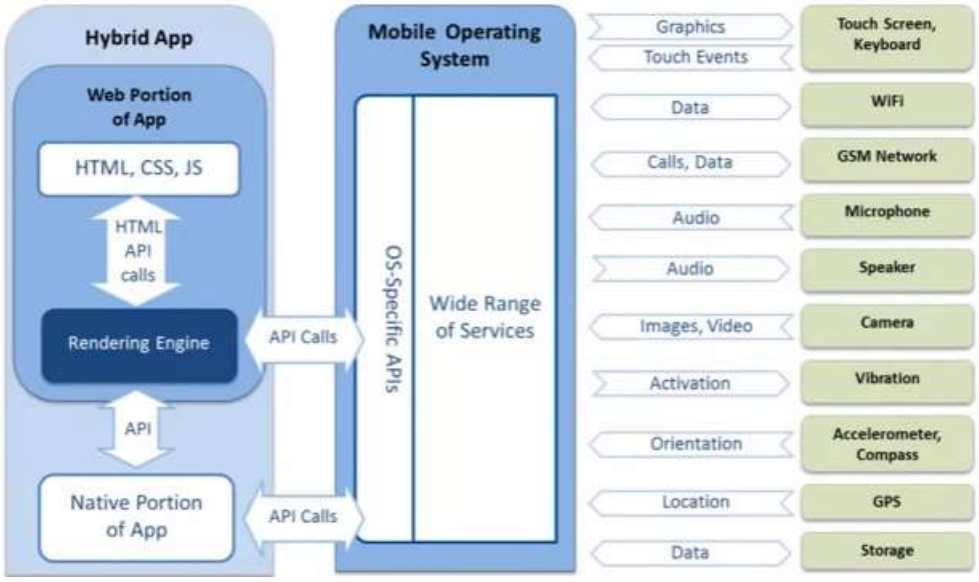
[9] https://www.w3.org/

Figura 3 – Funcionamento das aplicações web.



Fonte – Vendors e Resources (2018)

Figura 4 – Funcionamento das aplicações híbridas.



Fonte – Vendors e Resources (2018)

externas. Dessa forma, é possível utilizar as funcionalidades nativas do *hardware* do dispositivo independente do SO e personalizar a aplicação utilizando as tecnologias web (MALAVOLTA et al., 2015).

2.3 *Frameworks*

O desenvolvimento de aplicações nativas para Android e iOS é um processo com alto custo, pois é necessário produzir e manter dois produtos idênticos, porém, utilizando duas linguagens de programação diferentes (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2012).

Os *frameworks* de desenvolvimento de aplicações híbridas foram criados com o objetivo de diminuir os custos de produção e manutenção, já que é necessário produzir e manter apenas um produto para ambos SOs (FERREIRA et al., 2018). Os *frameworks* tendem a utilizar linguagens de programação para web, geralmente JavaScript. O código-fonte é encapsulado em uma aplicação nativa e utiliza de WebView ou componentes de web para apresentar a interface, acessar bibliotecas internas do SO e para comunicar-se com os sensores embutidos no aparelho (BRISTOWE, 2017).

Os *frameworks* possuem algumas características que podem ser observadas e comparadas em primeira instância, não sendo necessário desenvolvimentos e testes técnicos. Tais características são, de forma geral, superficiais e podem ser úteis para filtrar os *frameworks* no primeiro contato. Tais características são:

- **Linguagem de Programação:** Trata-se da linguagem de programação ou tecnologia web com a qual os desenvolvedores utilizam para implementações no *Framework* (PALMIERI; SINGH; CICCHETTI, 2012).

- **Curva de Aprendizagem:** A curva de aprendizagem se baseia no nível de dificuldade encontrado para aprender a tecnologia, de tal forma, é classificada em níveis baixo, médio e alto (BRITO et al., 2018) (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

- **Reutilização:** A reutilização se baseia na capacidade dos *frameworks* de criar blocos de códigos ou funções, criar repositórios para estes blocos e utilizar tais blocos de forma genérica no mesmo ou em outros projetos (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

- **Manutenção:** A manutenção é medida através da manutenibilidade proporcionada pelos *frameworks*, é baseada na capacidade de manter e evoluir projetos (HEITKöTTER; HANSCHKE; MAJCHRZAK, 2012).

- **Fundadores:** O suporte fornecido pelos fundadores é baseado na quantidade de dúvidas ou *bugs* (*issues*) resolvidas no repositório (CIMAN; GAGGI; GONZO, 2014).

- **Comunidade:** O suporte fornecido pela comunidade é baseada em questões apresentadas pelo site `Stack Overflow`[10], onde a comunidade auxilia com soluções para problemas similares encontrados por programadores iniciantes ou experientes (CIMAN; GAGGI; GONZO, 2014).

A seguir são descritos os *frameworks* para aplicações híbridas que são comparados neste trabalho: `Flutter`, `Ionic`, `NativeScript-Vue.js` e `React Native`.

2.3.1 Flutter

Flutter é o atual *Framework* de código aberto da empresa Google. Lançada em 2017, é a novidade entre as *frameworks*, já que utiliza a linguagem de programação Dart[11]. De acordo com a pesquisa realizada pelo site Stack Overflow (OVERFLOW, 2018), esta linguagem de programação não é popular atualmente, porém, seu uso tem crescido com o decorrer dos meses.

A *Framework* Flutter possui uma alta escalabilidade e com a utilização de componentes chamados `Widget` (Componente), dispõe de interfaces personalizadas e únicas. De acordo com o trabalho desenvolvido por Gažo (2018), Flutter evita os problemas de desempenhos encontrados em *frameworks* que utilizam pontes de compilação da linguagem de programação utilizada para a nativa. Isto ocorre pela linguagem Dart dar suporte a compilação A Frente do Tempo (do inglês *Ahead-Of-Time*) (`AOT`), que realiza a compilação antes da implantação e início da aplicação.

2.3.2 Ionic

Ionic é um *Framework* de código aberto criado pela empresa `Drifty Co.` em 2012. O seu objetivo é proporcionar a possibilidade dos desenvolvedores produzirem aplicações com alto desempenho e alta qualidade utilizando tecnologias webs conhecidas (`HTML`, `CSS` e `JavaScript`). O Ionic utiliza `Apache Cordova`[12] como base, depende de componentes `Webview` para apresentar sua interface. Sendo assim, simplifica e diminui o tempo de desenvolvimento das interfaces (IONIC, 2018).

Uma característica relevante desta *Framework* é sua compatibilidade com o tecnologia web `AngularJS`, que proporciona um aumento substancial nas funções que podem ser utilizadas e criadas, na utilização de componentes visuais populares do `AngularJS`, criando assim maior flexibilidade e facilidade para o desenvolvimento das aplicações (PATIL, 2017).

[10] <https://stackoverflow.com>
[11] <https://www.dartlang.org>
[12] <https://cordova.apache.org/>

2.3.3 NativeScript-Vue.js

NativeScript é um *Framework* de código aberto para criação de aplicações nativas usando JavaScript, criado em 2014 pelo ex-funcionário da Google, Evan You. Utiliza o Model-View-Controller (MVC) como padrão arquitetural pré-definido e da suporte ao Angular (SOFTWARE, 2018b), Vue.js (SOFTWARE, 2018c) e tecnologias web (SOFTWARE, 2018a). Neste trabalho, é explorado e apresentado o *Framework* utilizando a extensão do Vue.js, que é focada somente para a camada de Visão do MVC, facilitando a integração com outras bibliotecas ou projetos existentes.

NativeScript-Vue.js é um *plugin* para NativeScript, que permite a criação de aplicações nativas. Utiliza componentes de interfaces nativos dos SO, não possuindo os limites tecnológicos apresentados por *frameworks* que utilizam WebView para sua apresentação. Outra vantagem desta *Framework* é sua baixa curva de aprendizagem, sendo basicamente JavaScript, torna rápido e fácil a geração de soluções.

2.3.4 React Native

React Native é o *Framework* de código aberto para desenvolvimento de aplicações da empresa Facebook[13] disponibilizada em 2005, durante a React.js Conf 2015[14] (OCCHINO, 2015). É uma extensão do *Framework* React (FACEBOOK, 2018), do qual o intuito é facilitar a criação de interfaces gráficas.

O React Native segue os mesmos padrões de código do React, sendo assim, é possível criar funções que podem ser padronizadas e utilizadas de formas genéricas dentro do projeto em que está trabalhando (ALCANTARA, 2018).

Uma desvantagem desta *Framework* é a utilização de pontes para a compilação da linguagem React ou JavaScript para a linguagem nativa do SO, sendo isto responsável por problemas de desempenho. Entretanto, uma vantagem a ser citada é a forma como é estruturado em componentes e ao alto e ativo apoio da comunidade, que reflete no alto grau de reutilização e personalização de funções e telas.

[13] <https://opensource.fb.com/>
[14] <https://reactjs.org/blog/2015/02/18/react-conf-roundup-2015.html>

3 REVISÃO DE LITERATURA

Este Capítulo descreve o mapeamento sistemático da literatura que foi realizado com o objetivo de identificar os trabalhos que apresentam comparações entre *Frameworks* de desenvolvimento de aplicações móveis. Adicionalmente apresenta também discussões sobre os resultados encontrados.

3.1 Mapeamento Sistemático

"Um mapeamento sistemático da literatura é um modo de identificação, avaliação e interpretação de todas as pesquisas relevantes disponíveis para questões de pesquisa particulares" (BUDGEN; BRERETON, 2006, p. 1052). Tal método de pesquisa fornece estudos relevantes ao contexto da pesquisa, apresenta uma visão geral da área de conhecimento e mostra possíveis oportunidades para a pesquisa a ser desenvolvida.

3.1.1 Planejamento

Este mapeamento sistemático da literatura foi uma adaptação dos protocolos presentes nos trabalhos apresentados por Kitchenham (2004), Kitchenham et al. (2007) Kitchenham et al. (2009), porém, foi acrescentado a estratégia para melhor estruturar as questões de pesquisa, conhecida como População/Problema, Intervenção, Comparação, Resultado (do inglês *Population/Problem, Intervention, Comparison, Outcome*) (PICO) (SANTOS; PIMENTA; NOBRE, 2007) (SILVEIRA et al., 2011).

3.1.1.1 Escopo e Objetivo

Este mapeamento sistemático foi realizado com o objetivo de identificar trabalhos que apresentem comparações de *Frameworks* para desenvolvimento de aplicativos móveis. Através desses trabalhos, espera-se identificar as características semelhantes e diferentes entre as *Frameworks*, os critérios e as metodologias utilizadas para a realização das comparações.

3.1.1.2 Estrutura das Questões

As questões de pesquisa foram criadas e estruturadas utilizando a estratégia PICO:

- População (*Population*): Pesquisas sobre desenvolvimento de aplicações móveis.
- Intervenção (*Intervention*): Uso de *frameworks* para o desenvolvimento.
- Contexto (*Context*): *Frameworks* sendo usados para desenvolver aplicações móveis para diferentes plataformas.
- Saída (*Outcome*): Características comparadas e como foram comparadas.

3.1.1.3 Questões de Pesquisa

Para mapear a área de conhecimento que embasa o trabalho a ser desenvolvido, foram identificadas 5 Questões de Pesquisa (QP). Estas questões nortearam a extração de informações relevantes e precisas no contexto

QP1. Quais *Frameworks* são apresentados no trabalho?

O objetivo é listar os *Frameworks* que são apresentados e comparados nos trabalhos, para ter base de quais são os mais estudados e/ou usados.

QP2. Quais as características dos *Frameworks* são apresentadas ou comparadas no trabalho?

Obtendo uma lista de características para cada *Framework*, pode-se cruzá-las e verificar quais as características em comum entre os *Frameworks* e quais são exclusivas para cada. Também é importante salientar quais critérios e formas são utilizados para recolher dados.

QP3. Quais problemas ou dificuldades são apresentados?

Identificar a listagem de *issues* para cada *Framework* e interseccioná-las para obter as *issues* em comum e distintas entre elas.

QP4. Quais as vantagens ou benefícios são apresentados?

Tem como objetivo gerar uma lista de vantagens em comum e distintas entre elas.

QP5. Qual a metodologia utilizada para realizar a comparação?

Tem como objetivo identificar a metodologia utilizada para a comparação entre *Frameworks*, ou seja, fazer um levantamento de como foram realizadas as comparações.

3.1.1.4 Processo de Busca

Para a busca dos estudos foi levantado um conjunto de palavras-chaves que devem estar presentes nos títulos, palavras-chaves e/ou *abstract* dos estudos, essas palavras-chaves foram baseadas nos trabalhos Brito et al. (2018) e Heitkötter, Hanschke e Majchrzak (2012). Após o levantamento, definiu-se uma *string* base, apresentada na Tabela 2, que varia de acordo com o sistema de busca de cada biblioteca digital. As bibliotecas utilizadas neste estudo foram: ACM Digital Library, IEEE Xplore, Springer Link e Scopus.

3.1.1.5 Critérios de Inclusão e Exclusão

Critérios de Inclusão (CI) e Critérios de Exclusão (CE) são levantados e usados para selecionar os estudos encontrados na busca. Os CI foram definidos com o intuito

Tabela 2 – *Strings*.

Estudo	Características
Base	((comparison) AND (mobile) AND (cross-platform OR hybrid) AND (application OR app) AND (development))
ACM	((acmdlTitle:("mobile"))) OR (recordAbstract: ("mobile")) AND ((acmdlTitle:("cross-platform" OR "hybrid"))) OR (recordAbstract:("cross-platform" OR "hybrid")) AND ((acmdlTitle:("application" OR "app"))) OR (recordAbstract:("application"OR "app")) AND ((acmdlTitle:("development"))) OR (recordAbstract:("development"))
IEEE	(('comparison') AND ('mobile') AND (('cross-platform') OR ('hybrid')) AND (('application') OR ('app')) AND ('development'))
Scopus	(TITLE-ABS-KEY (comparison) AND TITLE-ABS-KEY (mobile) AND TITLE-ABS-KEY (cross-platform) OR TITLE-ABS-KEY (hybrid) AND TITLE-ABS-KEY (application) OR TITLE-ABS-KEY (app) AND TITLE-ABS-KEY (development))
Springer Link	comparison AND mobile AND cross-platform AND OR AND hybrid AND application AND OR AND app AND development

Fonte – Própria

de incluir estudos que apresentassem ou citassem *Frameworks* de desenvolvimento de aplicações móveis. Em contrapartida, os CE visam excluir estudos que não podem ser obtidos ou que não apresentam relevância para os objetivos do presente mapeamento. Os CI e CE são listados a seguir.

CI1. O objetivo do trabalho é comparar *Frameworks* de desenvolvimento multi-plataforma de aplicações móveis?

CI2. O trabalho apresenta detalhes ou descrição sobre o *Framework*?

CI3. São apresentados exemplos ou cases para cada *Framework*?

CI4. É apresentado a metodologia utilizada para realizar as comparações?

CI5. Os critérios comparados são explicados ou exemplificados?

CE1. Estudos disponíveis em formato de *posters* ou que apresentem somente o *abstract*.

CE2. Estudos incompletos ou duplicados.

3.1.1.6 Critérios de Qualidade

A classificação dos estudos se realizou a partir dos pontos atribuídos para cada um dos estudo conforme os Critérios de Qualidade (CQ) definidos. Os pontos são concedidos

de acordo com o quão satisfatório os estudos respondem os CQ, sendo assim, são pontuados com: Sim (**S**), o critério é respondido de forma satisfatória e relevante, corresponde a 1 ponto; Parcial (**P**), o critério é parcialmente respondido, corresponde a 0,5 ponto; Não (**N**), o critério não é respondido ou a resposta não é satisfatória, não é acrescentado pontos. Os 4 CQ levantados e como é definido sua pontuação, são apresentados a seguir:

CQ1. O estudo apresenta de forma detalhada as características dos *Frameworks*?

Critério de Análise:
S: Caso as características sejam apresentadas, explicadas e exemplificadas;
P: Caso as características sejam apenas citadas.
N: Caso não seja apresentada características.

CQ2. São apresentados exemplos de uso da *Framework*?

Critério de Análise:
S: Os exemplos mostrados são apresentados de forma detalhada com imagens e linhas de código;
P: São apenas citados alguns exemplos;
N: Não são apresentados exemplos.

CQ3. São apresentados os critérios utilizados para a realização da comparação?

Critério de Análise:
S: Os critérios utilizados para as comparações, são explicados e detalhados;
P: Os critérios são apenas apresentados;
N: Não são apresentados os critérios.

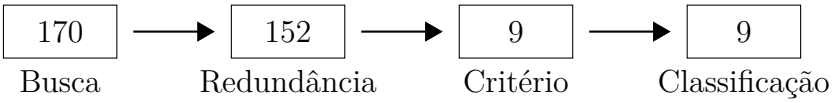
CQ4. É mostrado e detalhado a metodologia utilizada para a realização da comparação?

Critério de Análise:
S: É explicado de forma detalhada como foi realizado as comparações;
P: Apenas são apresentados os resultados da comparação;
N: Não são apresentados os resultados.

3.1.1.7 Processo de Seleção

O processo de seleção dos estudos foi dividido em 4 etapas, como pode ser observado na Figura 5.

Figura 5 – Etapas do processo de seleção.



Fonte – Própria (2019)

- `Busca`: é a primeira etapa em que se realiza as buscas nas bibliotecas digitais citadas na Subseção 3.1.1.4, o resultado total de estudos encontrados foi de 170.

- `Redundância`: nesta etapa, lê-se o título do estudo, os nomes dos autores e o ano em que foi publicado, removendo trabalhos duplicados. O número de estudos resultantes foi de 152.

- `Critérios`: na terceira etapa do processo de seleção, foram aplicados os CI e os CE citados na Sub-Seção 3.1.1.5. Do resultado total da etapa anterior, somente 9 estudos passaram pelos CQ.

- `Classificação`: na última etapa foi realizada a leitura completa dos 9 estudos, classificando-os de acordo com sua relevância. Nesta etapa, não foi removido nenhum estudo.

3.1.2 Condução

Este mapeamento sistemático foi conduzido entre Dezembro de 2018 e Janeiro de 2019. O total de estudos encontrados na busca realizada nas bibliotecas, foi de 170 estudos. A Tabela 3 apresenta os números de estudos que retornaram de cada biblioteca, ressaltando que, não foi estipulado um limite de ano para a publicação.

Tabela 3 – Resultados das bibliotecas.

Bibliotecas	**Resultados**
ACM	15
IEEE	23
Scopus	83
Springer Link	60

Fonte – Própria (2019)

O total de estudos selecionados foram 9. A Tabela 4 mostra detalhadamente como foi realizado a etapa de classificação dos estudos. O número de trabalhos não reflete a quantidade de *Frameworks* comparados.

3.2 Discussão Sobre os Trabalhos Relacionados

Esta seção apresenta a análise dos 9 estudos levantados durante a condução do mapeamento sistemático. Os trabalhos foram analisados levando em consideração os *Frameworks* apresentados e comparados, suas características, exemplos de usos e critérios utilizados para realizar as comparações.

Tabela 4 – Classificação dos trabalhos.

Estudos	**CQ1**	**CQ2**	**CQ3**	**CQ3**	**Pontos**
Dalmasso et al. (2013)	S	N	S	P	2,5
Ahti, Hyrynsalmi e Nevalainen (2016)	P	P	P	P	2
Heitkötter, Hanschke e Majchrzak (2012)	P	N	S	P	2
Brito et al. (2018)	P	N	P	S	2
Xanthopoulos e Xinogalos (2013)	N	P	S	N	1,5
Palmieri, Singh e Cicchetti (2012)	S	N	P	N	1,5
Botella, Escribano e Peñalver (2016)	P	N	P	N	1,5
Ciman, Gaggi e Gonzo (2014)	S	N	N	N	1
Mercado, Munaiah e Meneely (2016)	N	N	N	P	0,5

Fonte – Própria (2019)

3.2.1 *Frameworks* Para Desenvolvimento de Aplicações Móveis

A partir dos resultados do mapeamento sistemático descrito na Seção 3.1, foram extraídos os nomes dos *Frameworks* apresentados e comparados em cada estudo, sendo eles mostrados na Tabela 5.

Tabela 5 – *Frameworks* apresentados pelos estudos.

Estudos	***Frameworks***
Dalmasso et al. (2013)	PhoneGap, PhoneGap + Sencha Touch 2.0, PhoneGap + JQuery mobile, Sencha Touch 2.0, Appcelerator Titanium Studio 2.0
Ahti, Hyrynsalmi e Nevalainen (2016)	PhoneGap
Heitkötter, Hanschke e Majchrzak (2012)	PhoneGap, Appcelerator Titanium Studio 2.0
Brito et al. (2018)	React Native, Ionic, NativeScript
Xanthopoulos e Xinogalos (2013)	Não se aprofunda em nenhuma, pois realiza a comparação entre os grupos
Palmieri, Singh e Cicchetti (2012)	Rhodes, PhoneGap, DragonRAD, MoSync
Botella, Escribano e Peñalver (2016)	Sencha Touch 2.0, Ionic
Ciman, Gaggi e Gonzo (2014)	MoSync, Appcelerator Titanium Studio 2.0, Jquery & Jquery Mobile, PhoneGap
Mercado, Munaiah e Meneely (2016)	Não se aprofunda em nenhuma, pois realiza a comparação entre os grupos

Fonte – Própria (2019)

O estudo de Ahti, Hyrynsalmi e Nevalainen (2016) faz a comparação direta entre aplicações desenvolvidas utilizando PhoneGap e as desenvolvidas nativamente. Brito et al. (2018) apresenta três *Frameworks* que são comparadas neste trabalho, sendo elas: `React Native`, `Ionic` e `NativeScript`. Heitkötter, Hanschke e Majchrzak (2012) e Botella, Escribano e Peñalver (2016) mostram os resultados da comparação entre aplicações

criadas utilizando PhoneGap, Titanium Mobile, Sencha Touch 2.0 e Ionic, sendo elas aplicações web e híbridas, e as aplicações nativas. Os trabalhos do Dalmasso et al. (2013), Xanthopoulos e Xinogalos (2013) e Mercado, Munaiah e Meneely (2016) não apresentam a comparação entre *Frameworks*, mas sim entre os 3 grandes grupos de tipos de aplicações: `Aplicações Nativa`, `Aplicações Web` e `Aplicações Híbridas`.

Tabela 6 – Número de estudos que avaliam cada *Framework*.

Framework	**Trabalhos**
PhoneGap	5
Appcelerator Titanium Studio 2.0	3
Sencha Touch 2	3
Ionic	2
JQuery	2
JQuery Mobile	2
MoSync	2
DragonRAD	1
NativeScript	1
React Native	1
Rhodes	1

Fonte – Própria (2019)

Foi obtido também, os dados de quais são os *Framework* mais citados nas pesquisas. Nota-se na Tabela 6 que a *Framework* PhoneGap[1] é o mais citado e comparado, sendo encontrado em 5 dos 9 estudos.

A Tabela 7 mostra a divisão dos *Frameworks* apresentados nos estudos encontrados entre os dois tipos de aplicações, a web e a híbrida. Nota-se que a maioria é pertencente as aplicações híbridas.

Tabela 7 – Grupos dos tipos de aplicações.

Aplicações Web	**Aplicações Híbridas**
JQuery Mobile, Sencha Touch	PhoneGap, JQuery, MoSync, Ionic React Native, Rhodes, NativeScript DragonRAD, Appcelerator Titanium Mobile

Fonte – Própria (2019)

3.2.2 Características Comparadas Entre as *Frameworks*

Os autores dos estudos selecionados geraram grupos distintos de características dos *Frameworks* utilizados na comparação, que podem ser observadas na Tabela 8. Cada trabalho possui um diferencial em seu grupo de características, exceto o apresentado por Xanthopoulos e Xinogalos (2013), o qual realiza a comparação entre abordagens para desenvolvimento de aplicativos móveis.

[1] https://phonegap.com/

O estudo publicado por Heitkötter, Hanschke e Majchrzak (2012) não apresenta a comparação direta entre as características dos *Frameworks*. Neste trabalho as características são divididas em grupos de critérios de perspectiva de infraestrutura e critérios de perspectiva de desenvolvimento.

Pode-se notar que as características mais comparadas entre os trabalhos foram: as licenças necessárias para o desenvolvimento de aplicações, os SO ou versões que são suportadas e custo do desenvolvimento (é comparado desde o treinamento até a disponibilização da aplicação).

Algumas características como segurança, arquitetura, desempenho e a Interação Humano-Computador (IHC) são relevantes para o desenvolvimento de aplicações, porém, não foram apresentados em todos os trabalhos, apenas o estudo dos autores Mercado, Munaiah e Meneely (2016) engloba a comparação utilizando tais características.

Tabela 8 – Características comparadas em cada estudo.

Estudo	Características
Dalmasso et al. (2013)	Uso de memória RAM, uso do processador, consumo de energia
Ahti, Hyrynsalmi e Nevalainen (2016)	Tempo de início da aplicação, uso da memoria RAM e tamanho do espaço de armazenamento
Heitkötter, Hanschke e Majchrzak (2012)	Licença e custo, plataformas suportadas, acesso a funcionalidades específicas do SO, aparência nativa, desempenho, distribuição, ambiente de desenvolvimento, interface de usuário, dificuldade, manutenção, escalabilidade, tempo de desenvolvimento
Brito et al. (2018)	Aprendizagem e qualidade da documentação, Custo de Desenvolvimento, Emuladores e Depuração, Tempo de resposta e velocidade, Reconhecimento comercial, Reutilização de código e trabalho em equipe, Manutenção e Atualizações
Xanthopoulos e Xinogalos (2013)	Não apresenta características de *Frameworks*, pois realiza comparações entre as abordagens
Palmieri, Singh e Cicchetti (2012)	SOs suportadas, licença das ferramentas, linguagem de programação, disponibilidade da API, arquitetura, IDE
Botella, Escribano e Peñalver (2016)	Funcionalidade, interface de usuário, versão da SO, tempo de desenvolvimento, reuso, acesso nativo
Ciman, Gaggi e Gonzo (2014)	Licença, comunidade, tutoriais, complexidade, IDE, dispositivos, interface de usuário, conhecimento
Mercado, Munaiah e Meneely (2016)	Desempenho, confiabilidade, usabilidade, segurança

Fonte – Própria (2019)

3.2.3 Vantagens e Desvantagens Encontradas em Cada *Framework*

Foi realizado o levantamento das vantagens e desvantagens citadas em cada estudo para cada *Framework*. A partir do resultado deste levantamento, foi criado uma lista, onde

são apresentados estes critérios e os autores que as citaram.

3.2.3.1 PhoneGap

- **Vantagens:** Este *Framework* pode ser personalizado pelo desenvolvedor, já que fornece todo seu código fonte. Ele também facilita a adoção por desenvolvedores Web , pois sua implementação é em grande parte `HTML`, `JavaScript` e `CSS`, (DALMASSO et al., 2013).

- **Desvantagens:** Há falta de suporte para componentes de interface nativos, padrões de design e ferramentas de desenvolvimento (DALMASSO et al., 2013).

3.2.3.2 Appcelerator Titanium Studio 2.0

- **Vantagens:** O código nativo gerado apresenta alto desempenho quando executado, sua configuração é relativamente fácil para desenvolvedores inexperientes e é disponibilizado com ótima documentação, além de suportar desenvolvimento para *Tablets* (DALMASSO et al., 2013). Possui muitas bibliotecas que auxiliam no desenvolvimento (CIMAN; GAGGI; GONZO, 2014).

- **Desvantagens:** Possui APIs restritivas e um pequeno conjunto de modelos de dispositivos são suportados (DALMASSO et al., 2013). Há funcionalidades adicionais que podem ser compradas, tornando o ecossistema do Titanium menos *Open Source* comparado com outras plataformas (HEITKöTTER; HANSCHKE; MAJCHRZAK, 2012).

3.2.3.3 Sencha Touch 2

- **Vantagens:** Pode ser utilizado em conjunto com outras *Frameworks* para criação de interfaces (DALMASSO et al., 2013). É executado do navegador, portanto não possui dependências com `SO` ou modelos de dispositivos (HEITKöTTER; HANSCHKE; MAJCHRZAK, 2012). O tempo de desenvolvimento utilizando este *Framework* é relativamente menor que outros (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

- **Desvantagens:** As funcionalidades de *hardware* possuem acesso limitadas devido ao fato de que o aplicativo obtido com o Sencha é um aplicativo da web. O tempo de carregamento da aplicação é maior quando comparada uma desenvolvida com aplicativos híbridos (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

3.2.3.4 Ionic

- **Vantagens:** Após realizar alterações, só é necessário salvar o código e recarregar a página no navegador para ver a aplicação (BRITO et al., 2018). Comparado ao San-

cha Touch 2.0, o tempo de desenvolvimento é consideravelmente menor (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

- **Desvantagens:** O tempo de resposta ao clique é maior que outras aplicações híbridas e a latência é perceptível por qualquer utilizador (BRITO et al., 2018). Não foi possível instalar o aplicativo em versões `Android` menor que 4.1 (BOTELLA; ESCRIBANO; PEÑALVER, 2016);

3.2.3.5 JQuery e JQuery Mobile

- **Vantagens:** Depende exclusivamente do navegador, portanto não possui dependências com `SO` ou versionamentos (HEITKöTTER; HANSCHKE; MAJCHRZAK, 2012). Permite criação de aplicações web que funcionarão tanto nos dispositivos móveis quanto em computadores (CIMAN; GAGGI; GONZO, 2014).

- **Desvantagens:** Não possibilita o desenvolvimento de aplicativos móveis, mas somente aplicações web. Também não dá suporte a funcionalidades nativas do dispositivo (CIMAN; GAGGI; GONZO, 2014).

3.2.3.6 MoSync

- **Vantagens:** Oferece acesso a APIs externas através de C/C# (PALMIERI; SINGH; CICCHETTI, 2012) e a documentação disponibilizada para JavaScript é completa e explicativa (BOTELLA; ESCRIBANO; PEÑALVER, 2016).

- **Desvantagens:** Não é um *Framework* de código aberto (PALMIERI; SINGH; CICCHETTI, 2012). Não da suporte a bibliotecas de animações, porém, possibilita suas criações manualmente. As funcionalidades presentes utilizando a função de deputação e a documentação encontrada para C++ são básicas, não auxiliando em casos complexos. O tempo para criação e configuração para criação de testes é muito alto comparado a outros *Frameworks* (CIMAN; GAGGI; GONZO, 2014).

3.2.3.7 DragonRAD

- **Vantagens:** Facilita a integração e sincronização da base de dados com funcionalidades nativas. Não possui uma linguagem de programação, é desenvolvido totalmente em Arrastar e Soltar (do inglês *Drag and Drop*) (`D&D`) (PALMIERI; SINGH; CICCHETTI, 2012).

- **Desvantagens:** Não é um *Framework* de código aberto. Este *Framework* necessidade de sua própria `IDE`, por apresentar desenvolvimento em `D&D` (PALMIERI; SINGH; CICCHETTI, 2012).

3.2.3.8 NativeScript

- **Vantagens:** Oferece a funcionalidade "Hot Reloading" para acompanhamento das alterações ou implementações realizadas e seu tempo de resposta das aplicações é muito semelhante as nativas (BRITO et al., 2018).
- **Desvantagens:** Apresenta a configuração inicial do ambiente de desenvolvimento muito complexa (BRITO et al., 2018).

3.2.3.9 React Native

- **Vantagens:** Apresenta a documentação bem escrita e explicativa, muito semelhante a nativa do Android. Também oferece a funcionalidade "Hot Reloading" (BRITO et al., 2018).
- **Desvantagens:** Possui versão estável, porém, recebe constantemente melhorias (BRITO et al., 2018).

3.2.3.10 Rhodes

- **Vantagens:** Dos *Frameworks* híbridos é o que apresenta maior quantidade de `SO` suportados. Trabalha com a arquitetura `MVC` (PALMIERI; SINGH; CICCHETTI, 2012).
- **Desvantagens:** Não é um *Framework* de código aberto e não apresenta versão teste (PALMIERI; SINGH; CICCHETTI, 2012).

3.2.4 Metodologias de Comparação Apresentadas nos Trabalhos

A Tabela 9 mostra a síntese extraída de cada estudo. Pode-se observar que, o trabalho apresentado por Ahti, Hyrynsalmi e Nevalainen (2016) não é uma comparação entre *Frameworks*, e sim entre o PhoneGap e o desenvolvimento nativo.

Os estudos de Heitkötter, Hanschke e Majchrzak (2012) e Botella, Escribano e Peñalver (2016) definem critérios do *Framework* do ponto de vista do desenvolvedor e do usuário, comparando-os.

Uma das metodologias base para este trabalho, é a apresentada por Brito et al. (2018), no qual foram criados critérios para desenvolvimento de aplicações que devem ser respeitados e pontuados de acordo com o nível de satisfação encontrado.

O trabalho apresentado por Mercado, Munaiah e Meneely (2016) possui uma metodologia diferente das outras, pois não apresenta comparações baseadas em implementações, códigos, arquiteturas ou critérios de tecnologias. A comparação é realizada utilizando a classificação dos usuários a partir de métricas que avaliam as reclamações em relação a características específicas.

Tabela 9 – Metodologias de comparação em cada estudo.

Estudo	**Características**
Dalmasso et al. (2013)	É apresentada uma pesquisa detalhada que abrange vários aspectos das ferramentas como API e documentação, ambiente de desenvolvimento, implantação, estabilidade do *Framework*, vantagens e desvantagens.
Ahti, Hyrynsalmi e Nevalainen (2016)	Realiza uma comparação direta entre o *Framework* PhoneGap e o desenvolvimento nativo.
Heitkötter, Hanschke e Majchrzak (2012)	Uma lista de critérios foi criada a partir de discussões com pessoas que trabalham com desenvolvimento de software, esta lista foi posteriormente aumentada através de pesquisas na literatura.
Brito et al. (2018)	Os critérios selecionados para realizar a comparação foram escolhidos de acordo com as necessidades principais do desenvolvimento de aplicações móveis.
Xanthopoulos e Xinogalos (2013)	Uma análise comparativa para desenvolvimento multiplataforma. Para isso os autores utilizaram um conjunto de características que convergem com o conjunto de critérios proposto por Heitkötter, Hanschke e Majchrzak (2012).
Palmieri, Singh e Cicchetti (2012)	Uma lista de critérios foi montada. Os critérios escolhidos foram selecionados com intuito de serem úteis para os desenvolvedores entenderem qual ferramenta pode ser apropriada para seus propósitos.
Botella, Escribano e Peñalver (2016)	Foram definidos critérios para comparar os *Frameworks* do ponto de vista do usuário final e do ponto de vista do desenvolvedor.
Ciman, Gaggi e Gonzo (2014)	Uma lista de critérios para realizar a comparação entre os *Frameworks* é criada a partir fatores que os autores acreditam ser os mais importantes no desenvolvimento de aplicativos.
Mercado, Munaiah e Meneely (2016)	As avaliações de usuários classificadas são analisadas usando uma métrica que quantifica a proporção de avaliações nas quais os usuários reclamam de uma preocupação de qualidade específica.

Fonte – Própria (2019)

3.3 Considerações Finais

Neste capítulo foi apresentado a revisão de literatura, sendo realizado um mapeamento sistemático para selecionar e classificar os estudos mais relevantes para a pesquisa apresentada neste trabalho. Este mapeamento sistemático foi divido em planejamento e condução, sendo detalhado e discutido os resultados obtidos.

No planejamento, inicialmente foram definidas as questões de pesquisa, para que

fosse possível deixar claro qual eram os objetivos da pesquisa. Após esta definição, foi criada a *String* base, facilitando a adaptação para a busca nas bibliotecas digitais. O próximo passo foi o levantamento de critérios de inclusão, exclusão e qualidade. Este procedimento foi necessário para a realização da filtragem e classificação dos estudos.

A condução da pesquisa teve seu início com a busca nas bibliotecas digitais, o que obteve o total de 170 estudos. Estes trabalhos passaram pelas etapas de remoção dos trabalhos redundantes, critérios de inclusão, exclusão e classificação, resultando em 9 trabalhos ordenados pela pontuação alcançada.

A partir destes 9 estudos, foi possível realizar o levantamento dos *Frameworks* que são comparados e apresentados, a quais grupos cada um destes pertencem, as características que foram utilizadas para comparação, vantagens, desvantagens e qual a metodologia utilizada para executar a comparação.

Para a realização da comparação entre *Frameworks* descrita neste trabalho, foi utilizada como base uma síntese das metodologias levantadas a partir da leitura e entendimento dos estudos selecionados. Salienta-se que, a metodologia apresentada neste trabalho não apresenta exclusivamente características ou processos presentes nos estudos resultantes do mapeamento sistemático.

Com base nos trabalhos selecionados, foram definidos os critérios a serem comparados e a metodologia a ser adotada. O trabalho apresentado por Heitkötter, Hanschke e Majchrzak (2012) serviu de referência para quais tipo de características são comparadas, pois apresentou uma forma de comparar os critérios diferente dos outros, já que foi agrupado em características de desenvolvedor e de usuário. As características comparadas foram definidas a partir dos trabalhos publicados por Dalmasso et al. (2013) e Brito et al. (2018).

Este trabalho, diferente dos apresentados neste Capítulo, tem como objetivo comparar as características dos *Frameworks* para desenvolvimento de aplicativos híbridos para auxiliar desenvolvedores ou empresas a selecionar o *Framework* mais adequado para cada caso. Nesta comparação, é apresentado características técnicas relevantes para desenvolvedores, tais como percentual de uso do processador, consumo de memória e tempo de execução da aplicação.

4 METODOLOGIA

Neste Capítulo é apresentado a metodologia realizada neste trabalho, que de acordo com Prodanov e Freitas (2013), é um estudo classificado como pesquisa experimental, pois é executada de forma controlada, ou seja, com um número definido de vezes e com ambiente supervisionado, utilizando de ferramentas de precisão para a coleta de métricas para comparação.

Para comparar os *Frameworks* Flutter, Ionic, NativeScript e React Native, duas estratégias são adotadas: (i) desenvolver um conjunto de mini-aplicações com o propósito de comparar os *Frameworks* em relação a determinadas funcionalidades dos dispositivos móveis; e (ii) elaborar e aplicar um questionário para conhecer a opinião de desenvolvedores em relação a diferentes aspectos desses *Frameworks*. As seções a seguir apresentam os detalhes de cada uma das estratégias.

4.1 Mini-Aplicações

A criação destas mini-aplicações foi dividida em 5 estágios, visando a melhor organização e garantir processos bem definidos. A Tabela 10 apresenta estes estágios, juntamente com a atividade vinculada a eles.

Tabela 10 – Estágios para o desenvolvimento de mini-aplicações.

Estágio	Atividade
Funcionalidades	Definir quais funcionalidades devem ser implementadas, testadas e seus resultados colhidos e analisados.
Métricas	Definir quais serão as métricas coletadas para analise.
Ambiente de Execução	Definir o ambiente em que as mini-aplicações serão desenvolvidas, testadas e como serão recolhidos os dados.
Infraestrutura	Definir em quais dispositivos serão utilizados como base para o desenvolvimento e para execução dos testes.
Desenvolvimento e Testes	Desenvolver as mini-aplicações para cada *Framework* e executar os testes definidos no ambiente de execução e recolher os dados.

Fonte – Própria (2019)

4.1.1 Funcionalidades

A partir do artigo disponibilizado pelo site Gasparotto (2014), foi definido algumas funcionalidades nativas do *Hardware* dos dispositivos móveis, sendo elas:

- **Câmera:** acesso as ações vinculadas a câmera, por exemplo, iniciar a aplicação de câmera do dispositivo e captura de imagem;
- **GPS:** utilização do GPS para definir atual localização ou posicionamento no mapa;

- **Armazenamento Interno (Memória):** a gravação, leitura e remoção de dados no armazenamento interno do dispositivo;
- **Bluetooth:** ativação e pareamento do Bluetooth.

4.1.2 Métricas

As métricas analisadas foram coletadas a partir de testes executadas em cada mini-aplicação e para cada *Framework*. Para obter dados precisos, cada teste foi executado no mínimo 30 vezes, para que assim, seja obtido o tempo mínimo, médio e máximo de execução, o consumo mínimo, médio e máximo de memória e a porcentagem mínima, média e máxima de uso do processador necessário para a execução da mini-aplicação.

4.1.3 Ambiente de Execução

Todos os testes foram executados utilizando a ferramenta `Bitbar Cloud`[1]. Esta ferramenta é comercial, oferecendo dois tipos de planos, individual e para times. O individual custa U$45 (Quarenta e cinco dólares) mensais e o para times custa U$390 (Trezentos e noventa dólares) mensais. Ao criar uma conta, é necessário informar um e-mail corporativo, recebendo 30 dias de utilização gratuitos. Esta ferramenta não executa os testes em dispositivos virtuais, ela possui uma biblioteca de dispositivos físicos que ficam a disposição dos usuários, porém, para a utilização da maioria dos dispositivos é necessário a aquisição de um dos planos pagos.

4.1.4 Infraestrutura

A Tabela 11 mostra os dispositivos utilizados como alvo para o desenvolvimento das aplicações e para a execução dos testes. Pode-se observar que foram definidos dois dispositivos, um para cada `SO`, sendo eles `Android` e `iOS`. Ambos dispositivos são físicos, e não virtuais, foram utilizados na ferramenta `Bitbar Cloud`.

4.2 Questionário

Como definido por Kasunic (2005), um questionário é uma abordagem para coleta e análise de dados na qual os participantes respondem a questões direcionadas ou abertas. Baseado em seu estudo, a Tabela 12 apresenta os estágios para a produção deste questionário.

4.3 Procedimento

Com o objetivo de obter os dados para efetuar a comparação entre os *Frameworks*, os seguintes passos foram realizados:

[1] https://cloud.bitbar.com/

Tabela 11 – Descrição dos dispositivos.

Descrição	**LG Google Nexus 5 6.0.1**	**Apple iPhone 5S A1453 11.2.6**
SO	Android 6.0.1	iOS 11.2.6
API level	23	-
Fabricante	LG	Apple
CPU	Quad-core 2.3 GHz Krait 400	Dual-core 1.3 GHz Apple Swift cores
RAM	2048 MB	1024 MB
Armazenamento Interno	16 GB	16 GB
Tela	4.95"Full HD (1920 x 1080)	4"WDVGA (1136 x 640)
Câmera	8.0 megapixels	8.0 megapixels

Fonte – Própria (2019)

Tabela 12 – Estágios para produção do questionário.

Estágio	**Atividade**
Objetivos	Definir quais são os objetivos que devem ser atingidos a partir das respostas obtidas.
Público-alvo	Definir qual é o tipo de público para qual este questionário é direcionado.
Amostragem	Definir a quantidade da amostragem desejada.
Elaborar questões	Elaborar as questões seguindo os objetivos definidos.
Distribuição	Definir o canal que será utilizado para a distribuição do questionário.
Análise	Definir o método de análise dos dados e executá-lo nos resultados obtidos.

Fonte – Própria (2019)

1. Desenvolver mini-aplicações visando a realização de ações singulares e específicas;
2. Formular o questionário para os desenvolvedores;
3. Recolher os questionários respondidos;
4. Criar conjuntos de testes para obtenção dos dados referentes as funcionalidades nativas;
5. Executar os conjuntos de testes nas mini-aplicações, em média 30 vezes por aplicação, e recolher os resultados;
6. Sintetizar as respostas dos questionários e os resultados dos testes;
7. Apresentar e descrever os resultados da sintetização.

5 DESENVOLVIMENTO

Neste Capítulo é descrito o processo de desenvolvimento das mini-aplicações que foram utilizadas na coleta de dados e as etapas de definição e elaboração do questionário para coletar dados de desenvolvedores e estudantes com conhecimento sobre os *Frameworks*.

5.1 Desenvolvimento de Mini-Aplicações

Para a realização da coleta de dados sobre as funcionalidades vinculadas à câmera, GPS, armazenamento interno e conectividade via Bluetooth, foram desenvolvidas mini-aplicações para cada *Framework*. Tais mini-aplicações são:

- **CameraAppTeste:** A mini-aplicação voltada para os testes da câmera possibilita as ações de gravar vídeos e fotografar, porém, não tem como objetivo testar o armazenamento das imagens ou vídeos. O objetivo é validar a possibilidade de fotografar e filmar a partir do acesso a este recurso via *Framework*, o tempo para a execução de fotografar e filmar.

- **GPSAppTeste:** Os testes voltados ao GPS possibilitam identificar a posição atual do usuário, o objetivo é validar o acesso a este recurso via *Framework* e o tempo para identificação da localização;

- **StorageAppTeste:** Essa mini-aplicação foi feita para testar o armazenamento interno, ou seja, gravação, alteração e remoção de dados salvos em banco de dados interno, imagens ou vídeos.

- **BluetoothAppTeste:** A conectividade do Bluetooth também foi testada a partir de uma mini-aplicação, que tem como objetivo validar o acesso a esta funcionalidade nativa.

5.1.1 Histórias de Usuário

Para cada mini-aplicação, foram definidas Histórias de Usuários (`HU`). Esta etapa visa definir os objetivos de cada mini-aplicação e fornecer detalhes de seu funcionamento, estabelecendo um padrão entre os *Frameworks*. Sendo assim, as `HU` são:

HU1. **Bluetooth**

COMO usuário, **DESEJO** ligar/desligar o bluetooth do dispositivo, **PARA** que possa ser validado que está acessando a funcionalidade corretamente.

HU2. **Câmera**

COMO usuário, **DESEJO** acessar a funcionalidade da câmera e ter a possibilidade de fotografar, **PARA** que possa ser validado que está acessando a funcionalidade corretamente.

HU3. GPS

COMO usuário, **DESEJO** acessar a funcionalidade do GPS e obter a minha localização atual, **PARA** que possa ser validado que está acessando a funcionalidade corretamente.

HU4. Armazenamento Interno (Memória)

COMO usuário, **DESEJO** informar um texto, salvá-lo no dispositivo e depois visualizá-lo na tela, **PARA** que possa ser validado que está acessando a funcionalidade corretamente.

5.1.2 Protótipo

Para evitar discrepâncias nos testes das telas, foram definidos protótipos para cada mini-aplicação.

A Figura 6 representa o protótipo gráfico da mini-aplicação Bluetooth. Visando ser simples e objetiva, é apresentado somente um botão para ativar/desativar o Bluetooth e um texto para indicar o estado atual da funcionalidade, podendo ser desativo (Figura 6a) ou ativado (Figura 6b).

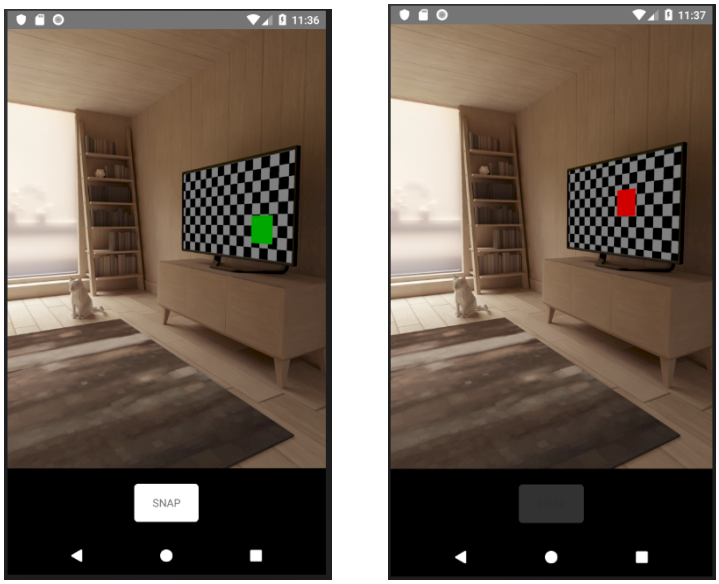
Figura 6 – Protótipo gráfico da funcionalidade Bluetooth.

(a) Início do teste (b) Resultado do teste

Fonte – Própria (2019)

A Figura 7 apresenta o protótipo gráfico da mini-aplicação Câmera. Visando ser simples e objetiva, é apresentado somente um botão para iniciar a câmera do dispositivo.

Figura 7 – Protótipo gráfico da funcionalidade Câmera.

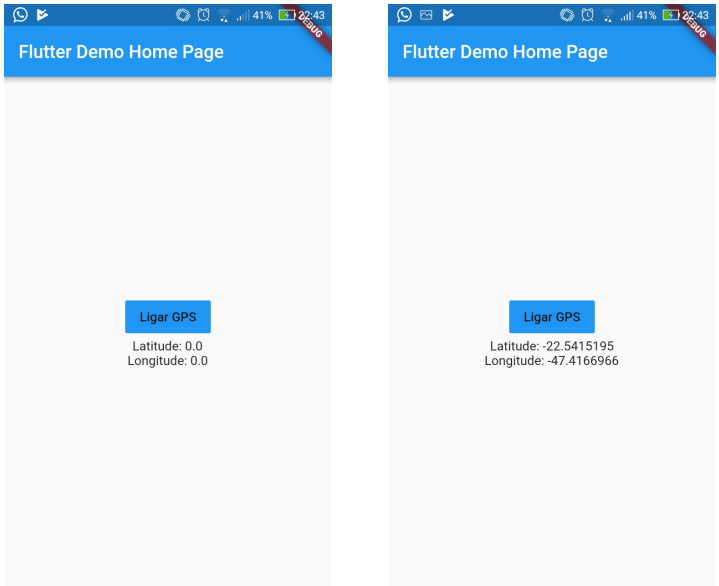


(a) Início do teste (b) Resultado do teste

Fonte – Própria (2019)

A Figura 8 apresenta o protótipo gráfico da mini-aplicação GPS. Visando ser simples e objetiva, é apresentado um botão para obter a atual localização do dispositivo e um texto para apresentar esta localização em longitude e latitude

Figura 8 – Protótipo gráfico da funcionalidade GPS.

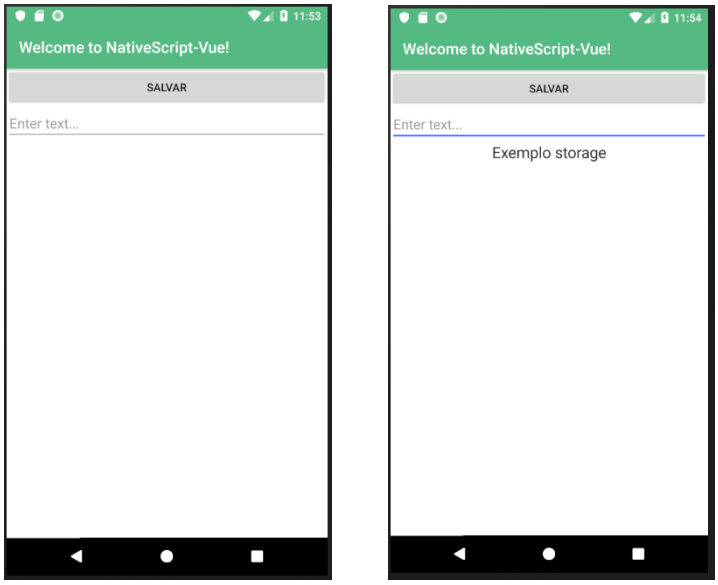


(a) Início do teste (b) Resultado do teste

Fonte – Própria (2019)

A Figura 9 apresenta o protótipo gráfico da mini-aplicação Memória (Armazenamento Interno). Visando ser simples e objetiva, é apresentado um campo para inserção do texto desejado, um botão com a ação de gravar o texto no armazenamento interno do dispositivo e um texto para apresentar o texto gravado para o usuário.

Figura 9 – Protótipo gráfico da funcionalidade Armazenamento Interno.



(a) Início do teste (b) Resultado do teste

Fonte – Própria (2019)

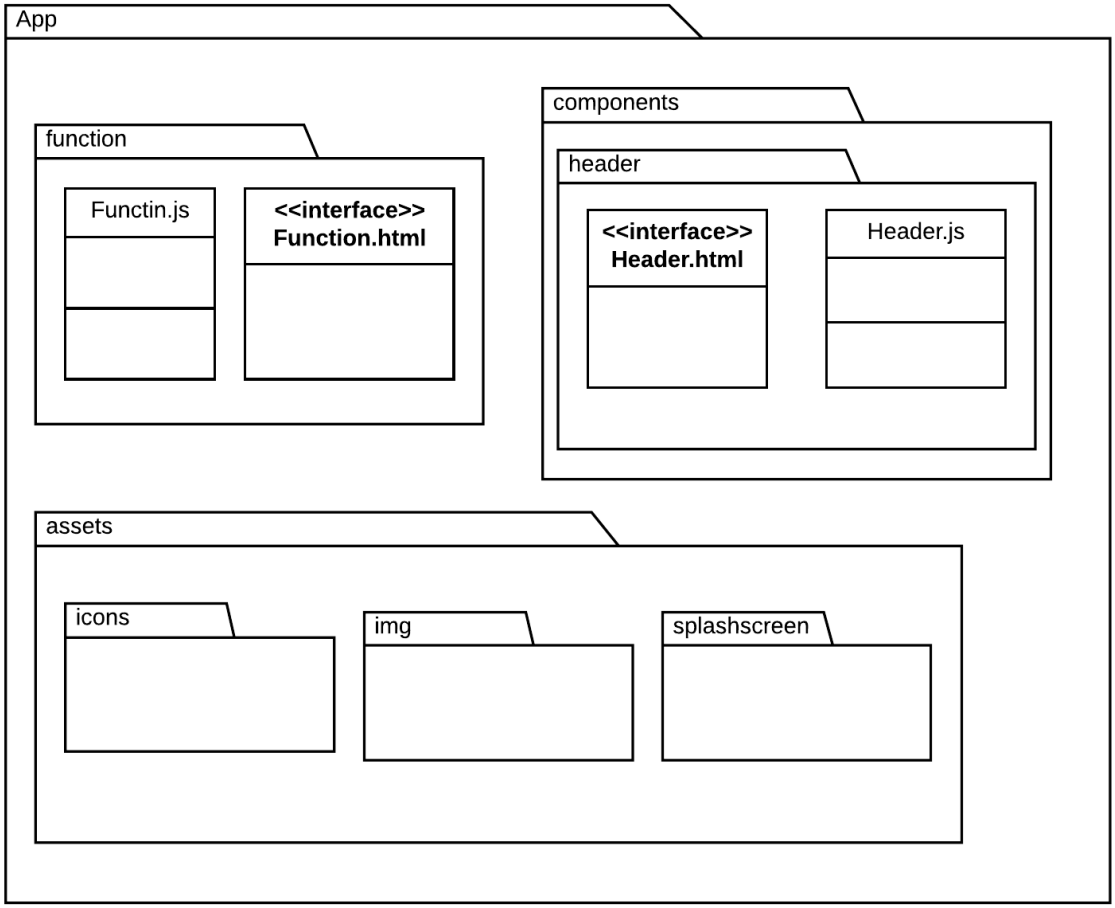
5.1.3 Arquitetura

Os *Frameworks* apresentados neste trabalho possuem a flexibilidade de utilizar padrões de arquitetura consolidados (FOWLER, 2009) ou a personalização da arquitetura. A Figura 10 mostra o diagrama de pacotes utilizado para exemplificar a arquitetura definida para todas as mini-aplicações.

O pacote `App` representa o projeto ao qual foi implementado os códigos-fontes que são compilados para os dois `SO`, sendo composto por pacotes internos contendo a tela desenvolvida, componentes reutilizáveis e ativos como imagens e ícones.

As telas e funcionalidades implementadas ficam no pacote `function`. Esse pacote deve conter apenas uma funcionalidade, portanto, cada tela/funcionalidade tem seu próprio pacote. Nas mini-aplicações implementadas neste trabalho, foi necessário a criação de um pacote. A funcionalidade, indicada na figura pela classe `Function.js`, é responsável pela parte lógica, em contra-partida, a interface `Function.html` possui a responsabilidade sobre a composição gráfica sobre a tela. Dependendo do *Framework*, pode-se ter um arquivo adicional para personalização dos componentes separado do `Function.html`, sendo ele arquivo de estilo (`.css` ou `.scss`).

Figura 10 – Diagrama de pacotes das mini-aplicações.



Fonte – Própria (2019)

O pacote `components` tem como objetivo dividir e possibilitar o acesso por qualquer tela a componentes reutilizáveis, como cabeçalhos, rodapés, botões ou outro componente personalizado para telas específicas. Esse pacote é opcional neste trabalho não foi necessário sua criação.

Os ícones, imagens e outros ativos que podem vir a ser utilizados no projeto, ficam separados no pacote `assets`. Este pacote é convertido para cada um dos `SO` no momento da compilação, sendo gravado dentro do arquivo de instalação da aplicação.

5.1.4 Implementação

A implementação das mini-aplicações foi executada utilizando o editor de texto `Visual Studio Code`[1] e o `IDE Android Studio`[2]. Para executar as mini-aplicações no `SO`

[1] https://code.visualstudio.com/
[2] https://developer.android.com/studio

Android é necessário a instalação e configuração do Java[3] e Kit de Desenvolvimento de Software (do inglês *Software Development Kit*) (SDK)[4]. Por outro lado, para a execução no SO iOS, é necessário utilizar o IDE XCode, exclusivo para dispositivos da Apple, além de ser necessário uma conta de desenvolvedor, que tem o custo anual de 99 Dólares, para realizar os testes em dispositivo físico.

É necessário a instalação do gerenciador de pacotes NPM[5] e o interpretador de JavaScript NodeJS[6] para realizar a compilação e geração de executáveis das mini-aplicações, exceto para o Flutter, que necessita de bibliotecas específicas para a linguagem Dart.

Todos os códigos gerados para a realização deste trabalho estão localizados em uma conta pública do GitHub[7] criada somente para este objetivo.

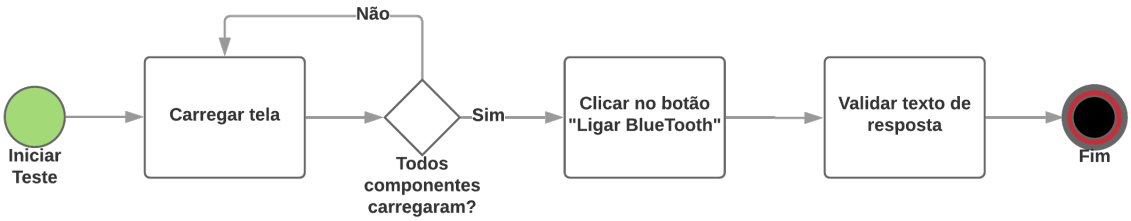
5.1.5 Script de Teste

Para padronizar os testes e obter resultados homogêneos e confiáveis, foram definidos scripts de testes. Esses scripts são compostos por ações que devem ser realizadas automaticamente pela aplicação, ou seja, um guia que automatiza ações simuladas na aplicação. Para cada mini-aplicação foi definido um script, os quais são descritos a seguir.

- **Bluetooth**

  A Figura 11 ilustra os passos que devem ser executados para a realização dos testes da mini-aplicação Bluetooth.

Figura 11 – Diagrama de atividade do script de teste Bluetooth.



Fonte – Própria (2019)

As ações executadas pelo script são:

  – **Carregar Tela:** o script aguarda a tela carregar todos seus componentes gráficos, sendo executado uma validação a cada segundo;

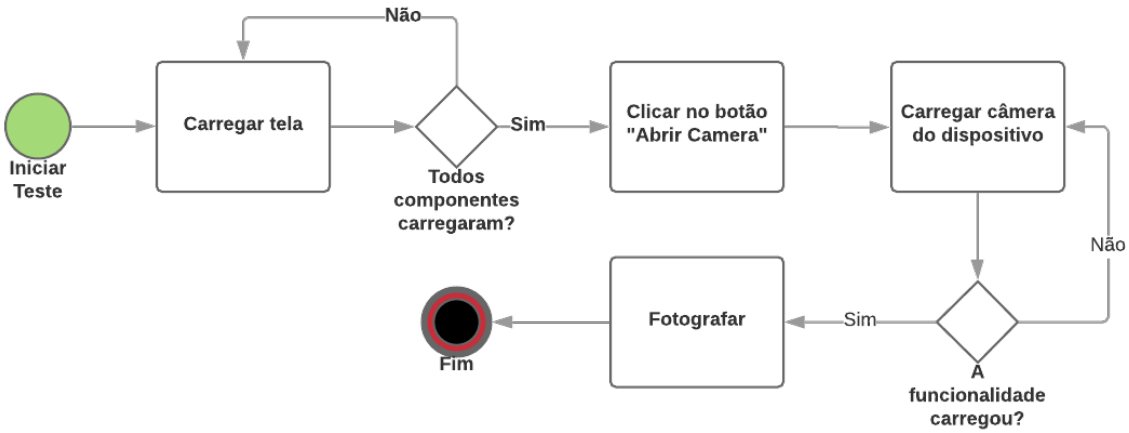
[3] https://www.oracle.com/technetwork/java/index.html
[4] https://www.oracle.com/technetwork/java/embedded/javame/javame-sdk/overview/index.html
[5] https://www.npmjs.com/
[6] https://nodejs.org/en/
[7] https://github.com/tccframeworkcomparison

- **Clicar no Botão "Ligar Bluetooth":** é selecionado o botão com o texto "Ligar Bluetooth";
- **Validar Texto de Resposta:** é realizado a validação se o texto presente na tela como resposta é igual a "Ativado".

- **Câmera**

A Figura 12 ilustra os passos que devem ser executados para a realização dos testes da mini-aplicação Câmera.

Figura 12 – Diagrama de atividade do script de teste Câmera.



Fonte – Própria (2019)

As ações executadas pelo script são:

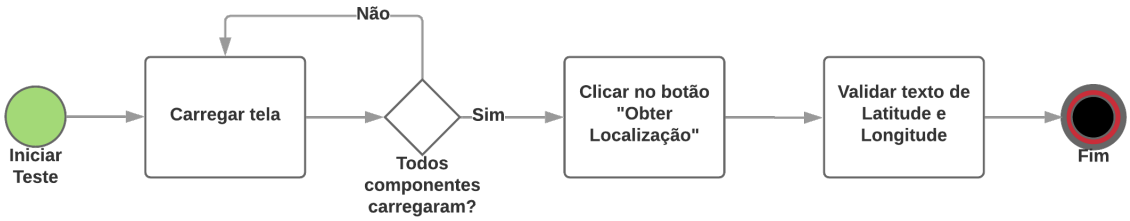
- **Carregar Tela:** o script aguarda a tela carregar todos seus componentes gráficos, sendo executado uma validação a cada segundo;
- **Clicar no Botão "Abrir Camera":** é selecionado o botão com o texto "Abrir Camera";
- **Carregar Câmera do Dispositivo:** script aguarda a funcionalidade nativa do dispositivo carregar, realizando validações a cada segundo.
- **Fotografar:** é realizado a ação de fotografar utilizando a funcionalidade nativa da câmera.

- **GPS**

A Figura 13 ilustra os passos que devem ser executados para a realização dos testes da mini-aplicação GPS.

As ações executadas pelo script são:

Figura 13 – Diagrama de atividade do script de teste GPS.



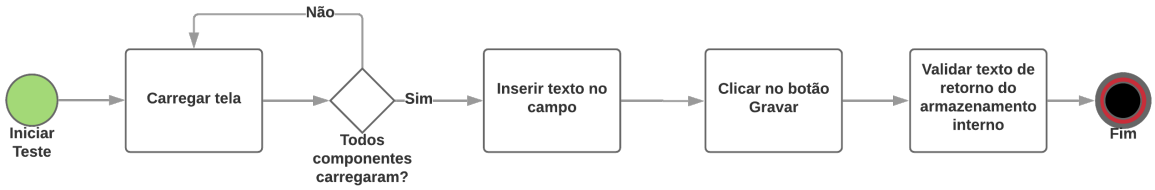
Fonte – Própria (2019)

- **Carregar Tela:** o script aguarda a tela carregar todos seus componentes gráficos, sendo executado uma validação a cada segundo;
- **Clicar no Botão "Obter Localização":** é selecionado o botão com o texto "Obter Localização";
- **Validar Texto de Latitude e Longitude:** é realizado a validação do texto presente na tela, para garantir que é igual ao esperado e informado no script.

- **Armazenamento Interno**

A Figura 14 ilustra os passos que devem ser executados para a realização dos testes da mini-aplicação Memória.

Figura 14 – Diagrama de atividade do script de teste Armazenamento Interno.



Fonte – Própria (2019)

As ações executadas pelo script são:

- **Carregar Tela:** o script aguarda a tela carregar todos seus componentes gráficos, sendo executado uma validação a cada segundo;
- **Inserir Texto no Campo:** é inserido o texto "Testando Armazenamento Interno" no campo de texto disponível na tela;
- **Clicar no Botão "Gravar":** é selecionado o botão com o texto "Gravar";
- **Validar Texto de Retorno do Armazenamento Interno:** é realizado a validação do texto presente na tela, que é o retorno do dados salvos no

armazenamento interno do dispositivo, a fim de garantir que é igual ao texto "Testando Armazenamento Interno".

5.2 Desenvolvimento do Questionário

Nesta seção é apresentado o desenvolvimento do questionário sobre os *Frameworks* de desenvolvimento de aplicações híbridas, seguindo as etapas descritas na Seção 4.2, que são: objetivos, público alvo, amostragem, elaborar questões, distribuição e análise.

5.2.1 Objetivos

O questionário tem como objetivo obter informações específicas de cada *Framework* a partir da percepção do desenvolvedor. Informações como por exemplo: a dificuldade de desenvolvimento, os pontos positivos e negativos, os problemas encontrados e o quão explicativo é a documentação.

5.2.2 Público-Alvo

O público alvo definido para este questionário são desenvolvedores ou estudantes que tenham contato teórico ou prático com qualquer um dos *Frameworks* comparados neste trabalho. Enfatiza-se que, todos os participantes devem concordar com a distribuição de suas respostas e o anonimato de seus dados pessoais.

5.2.3 Amostragem

A amostragem para este trabalho não será definida, para que não haja limitadores na quantidade de respostas obtidas.

5.2.4 Elaborar Questões

A Tabela 13 apresenta as Questões do Questionário (`QQ`) que está disponível online[8] e foram levantadas para que profissionais da área de desenvolvimento de aplicativos ou sistemas e estudantes que tenham experiência com os *Frameworks* respondam. Cada questão possui:

- **Objetivo:** Deve apresentar qual o motivo da realização da questão, e quais dados podem ser extraídos da resposta;
- **Tipo:** Identifica se as respostas da questão são fechadas, utilizando alternativas, ou descritivas, sendo uma resposta aberta para os participantes;

[8] https://docs.google.com/forms/d/1vtJfaWz-9wpZDVQIaGXh7zVL9vqCcZ7jnSN-Az1AI4Q/viewform?edit_requested=true

- **Respostas:** Apresenta as possíveis respostas, somente quando a questão for do tipo fechada.

Tabela 13 – Questões sobre os *Frameworks*.

Questões	Objetivo	Tipo	Opções de Resposta
QQ1. Qual a dificuldade encontrada para o aprendizado do *Framework*?	Identificar a curva de aprendizagem necessária para cada funcionalidade, no final, é calculado a média e definido a curva de aprendizagem para o *Framework*.	Alternativas	Muito Fácil; Fácil; Mediano; Difícil; Muito Difícil.
QQ2. Quão satisfatória é a didática da documentação do *Framework*?	O objetivo desta questão é entender o quão esclarecedora é a documentação do *Framework*.	Alternativas	Muito Satisfatória; Satisfatória; Indiferente; Insatisfatória; Muito Insatisfatória.
QQ3. Quão veloz é a construção/compilação da aplicação?	Medir qual a velocidade que o *Framework* constrói/compila uma aplicação.	Alternativas	Muito Rápido; Rápido; Normal; Lento; Muito Lento.
QQ4. Quão complexos são os códigos fontes necessários para o *Framework* acessar as funcionalidade nativas do dispositivo?	Entender a complexidade e dificuldade na implementação dos métodos para acesso nativo do dispositivo.	Alternativas	Muito Simples; Simples; Normal; Complexo; Muito Complexo.
QQ5. Quais as vantagens deste *Framework*?	Tem como objetivo listar as vantagens encontradas durante o desenvolvimento utilizando o *Framework*.	Descritiva	-
QQ6. Quais as desvantagens deste *Framework*?	Tem como objetivo listar as desvantagens encontradas durante o desenvolvimento utilizando o *Framework*.	Descritiva	-
QQ7. Problemas ou dificuldades encontrados no desenvolvimento de aplicações utilizando este *Framework*?	Visa listar problemas ou dificuldades durante o desenvolvimento utilizando o *Framework*.	Descritiva	-

Fonte – Própria (2019)

5.2.5 Distribuição

O questionário foi elaborado utilizando o `Formulário do Google`[9], sendo assim, o canal utilizado para distribuição do mesmo foi o e-mail. O questionário foi enviado para

[9] https://www.google.com/forms/about/

estudantes dos cursos de Engenharia de Software e Ciência da Computação da Universidade Federal do Pampa (UNIPAMPA) e desenvolvedores da empresa de *software* `Logic Solutions Provider`[10]. Não foram enviados para outras instituições ou empresas para que houvesse um ambiente controlado de respondentes, com a intenção de posteriormente segmentar as respostas por perfis e/ou cargos.

5.2.6 Análise

As questões foram divididas em dois tipos, sendo elas com respostas descritivas e fechadas. As questões que utilizam alternativas foram analisadas levando em consideração a quantidade de vezes que uma das alternativas foi selecionada, a partir disso, são apresentadas em gráficos, mostrando de forma quantitativa a opinião dos respondentes.

Por outro lado, as respostas descritivas foram sintetizadas e destacados os pontos mais relevantes para o trabalho, tais pontos são apresentadas em tabelas, visando representar a opnião dos respondetes.

[10] http://www.logicsp.com.br/

6 RESULTADOS

Neste Capítulo são apresentados e discutidos os resultados obtidos referentes aos testes das mini-aplicações desenvolvidas para cada *framework* e ao questionário respondido por acadêmicos ou profissionais que atuam na área.

6.1 Mini-Aplicações

Três critérios foram utilizados para avaliar as mini-aplicações: tempo de execução, consumo de memória e percentual de uso do processador. Os resultados foram obtidos através dos testes realizados. Esses resultados são mostrados em tabelas, permitindo a comparação entre os *frameworks* nos `SO Android` e `iOS`.

6.1.1 Tempo de Execução

O tempo de execução é medido entre o início da aplicação até o momento em que não há mais tarefas para serem executadas. A Tabela 14 mostra os tempos mínimos, médios e máximos para a mini-aplicação `Bluetooth` para `Android` e `iOS`. Nota-se que os *frameworks* apresentam tempos semelhantes para os dois `SO`. O `Flutter` apresentou o melhor desempenho em relação ao tempo de execução para essa mini-aplicação. Embora o `Ionic` tenha obtido o menor tempo mínimo para `Android`, ele apresentou os maiores tempos de execução para essa mini-aplicação. O `NativeScript-Vue.js` apresentou o menor tempo médio para `Android` e o `React Native` obteve os maiores tempos máximos para ambos `SO`.

Tabela 14 – Tempo de execução da mini-aplicação Bluetooth.

Framework	**Mínimo (min)**		**Médio (min)**		**Máximo (min)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	0:41	**0:45**	0:47	**0:53**	**1:03**	**1:02**
Ionic	**0:40**	1:04	1:10	1:14	1:18	1:19
NativeScript-Vue.js	0:43	0:54	**0:46**	1:11	1:16	1:13
React Native	0:55	0:52	0:56	1:06	1:17	1:20

Fonte – Própria (2019)

Os tempos de execução da mini-aplicação `Câmera` são apresentados na Tabela 15. É possível notar que o `Ionic` apresentou os menores tempos de execução para `Android`, enquanto o `React Native` apresentou os menores tempos mínimo e médio para `iOS`. Entretanto, ao analisarmos os resultados para os dois `SO`, o `Ionic` possui vantagens porque `React Native` possui os maiores tempos médio e máximo para o `Android`.

Os resultados para a mini-aplicação `GPS` são apresentados na Tabela 16. O *framework* `NativeScript-Vue.js` apresentou o melhor desempenho em relação ao tempo de execução para ambos `SO`, enquanto o *framework* `Flutter` obteve o pior desempenho.

Tabela 15 – Tempo de execução da mini-aplicação Câmera.

Framework	Mínimo (min)		Médio (min)		Máximo (min)	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	2:02	1:56	2:04	1:59	2:12	**2:01**
Ionic	**1:41**	1:50	**1:51**	1:53	**2:00**	2:08
NativeScript-Vue.js	1:54	1:47	2:03	1:57	2:10	2:09
React Native	1:45	**1:37**	2:15	**1:49**	2:17	2:13

Fonte – Própria (2019)

Tabela 16 – Tempo de execução da mini-aplicação GPS.

Framework	Mínimo (min)		Médio (min)		Máximo (min)	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	0:22	0:20	0:25	0:26	0:28	0:28
Ionic	0:25	0:22	0:26	0:23	0:27	0:26
NativeScript-Vue.js	**0:17**	0:20	**0:20**	**0:22**	**0:23**	**0:23**
React Native	0:18	**0:17**	0:23	0:24	0:26	0:26

Fonte – Própria (2019)

A Tabela 17 apresenta os tempos de execução para a mini-aplicação `Memória`. Como pode ser observado, o `NativeScript-Vue.js` novamente obteve o melhor desempenho quando considerado os dois `SO`. O `React Native` apresentou o menor tempo médio para `iOS`, porém seu tempo máximo para esse `SO` é 67% maior do que o tempo máximo apresentado pelo `NativeScript-Vue.js`.

Tabela 17 – Tempo de execução para a mini-aplicação Memória.

Framework	Mínimo (min)		Médio (min)		Máximo (min)	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	**1:10**	1:15	1:19	1:44	2:07	2:17
Ionic	1:16	1:17	1:59	1:55	2:02	2:11
NativeScript-Vue.js	1:12	**1:10**	**1:15**	1:31	**1:35**	**1:39**
React Native	1:18	1:19	1:35	**1:24**	1:54	2:07

Fonte – Própria (2019)

Ao somarmos os tempos médios obtidos na execução de todas as mini-aplicações para ambos os `SO` (mostrado na Tabela 18), o `NativeScript-vue.js` é o `framework` com o menor tempo médio (09:25), seguido do `Flutter` (09:37), do `React Native` (09:52) e, por fim, do `Ionic` (10:51).

6.1.2 Consumo de Memória

O consumo de memória é a quantidade, em Megabyte (MB), do espaço que é alocado na memória para a execução da aplicação, sendo medida do momento em que a aplicação é iniciada até o momento de sua finalização. Para o consumo de memória são apresentados os valores mínimo, médio e máximo obtidos a partir das 30 execuções de

Tabela 18 – Soma do tempo médio de execução para todas as mini-aplicações.

Framework	**Android**	**iOS**	**Total**
Flutter	04:35	05:02	09:37
Ionic	05:26	05:25	10:51
NativeScript-Vue.js	**04:25**	05:01	**09:25**
React Native	05:09	**04:43**	09:52

Fonte – Própria (2019)

cada mini-aplicação. A Tabela 19 apresenta os valores de consumo de memória para a mini-aplicação `Bluetooth` executada nos `SO Android` e `iOS`. Nota-se que a mini-aplicação do `Ionic` apresenta os maiores valores mínimo, médio e máximo. Em contra-partida, o `Flutter` apresenta os menores valores de médio e máximo consumo entre os *frameworks*.

Tabela 19 – Consumo de memória da mini-aplicação Bluetooth.

Framework	**Mínimo (MB)**		**Médio (MB)**		**Máximo (MB)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	64,9	64,7	**71,2**	**71,3**	**75,9**	**75,9**
Ionic	94,5	66,7	137,8	134,7	146,4	148,6
NativeScript-Vue.js	49,6	47,8	87,1	111	103	117
React Native	**34,7**	**35,7**	81,2	81	87,5	87,4

Fonte – Própria (2019)

A Tabela 20 apresenta os valores de consumo de memória para a mini-aplicação `Câmera`. O `Flutter` foi o *framework* que menos consumiu memória para ambos `SO`, apresentando os menores valores médio e máximo. Já o `Ionic` apresentou os maiores valores mínimo, médio e máximo (com exceção do valor máximo para o `SO iOS`).

Tabela 20 – Consumo de memória da mini-aplicação Câmera.

Framework	**Mínimo (MB)**		**Médio (MB)**		**Máximo (MB)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	64,5	**75,8**	**71,3**	**71,4**	**61,4**	76
Ionic	108,2	70,8	114,8	112,3	115,9	116
NativeScript-Vue.js	**59**	**59,3**	99,1	111,5	104,2	118,1
React Native	69,8	70,7	82,3	80,5	88	84

Fonte – Própria (2019)

A Tabela 21 apresenta os consumos mínimo, médio e máximo de memória para a mini-aplicação `GPS`. Pode-se observar que, o `Flutter` e o `React Native` são os *frameworks* que em média consomem menos memória. Já o `Ionic` é o *framework* que apresenta em média um consumo maior de memória do dispositivo.

Os resultados de consumo mínimo, médio e máximo de memória para a aplicação `Memória` são apresentadas na Tabela 22. Observa-se que o `Ionic` apresenta a maior distância entre os valores mínimo e máximo de consumo de memória, o que indica que há

Tabela 21 – Consumo de memória da mini-aplicação GPS.

Framework	**Mínimo (MB)**		**Médio (MB)**		**Máximo (MB)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	**35,2**	58,8	**71**	71,1	75,7	76
Ionic	68	56.1	112,7	110,8	121,6	115,8
NativeScript-Vue.js	57,3	**48,1**	98,3	110,1	104,8	119,1
React Native	67,1	67,7	71,1	**71**	**74,6**	**74,4**

Fonte – Própria (2019)

muito oscilação, isto pode ocorrer por ser executado através de navegadores ou `WebView`. Enquanto o `Flutter` encontra-se com o menor consumo entre todos, em ambos `SO`.

Tabela 22 – Consumo de memória da mini-aplicação Memória.

Framework	**Mínimo (MB)**		**Médio (MB)**		**Máximo (MB)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	62,9	64,6	**70,1**	**71,2**	**75,6**	**75,8**
Ionic	89,6	**55,6**	120,5	117,3	128	124,9
NativeScript-Vue.js	**57,7**	58,3	99,7	112,4	105,1	120
React Native	62,8	67,9	73,1	73,4	77,5	77,5

Fonte – Própria (2019)

Ao somarmos os consumos médios de memória obtidos na execução de todas as mini-aplicações para ambos os `SO` (mostrado na Tabela 23), o `Flutter` é o `framework` com o menor consumo médio de memória (668,6 MB), seguido do `React Native` (613,6 MB), do `NativeScript-vue.js` (829,2 MB) e, por fim, do `Ionic` (960,9 MB). A razão para o alto consumo do `Ionic` é o fato dele necessitar de `WebView` para executar as aplicações. Já o `NativaScript-Vue.js` apresentou esses valores pelo fato de sua biblioteca possuir baixa otimização, como pode ser observado nas *issues* abertas no `GitHub`[1].

Tabela 23 – Soma do consumo médio de memória para todas as mini-aplicações.

Framework	**Android**	**iOS**	**Total**
Flutter	**283,6** MB	**285** MB	**568,6** MB
Ionic	485,8 MB	475,1 MB	960,9 MB
NativeScript-Vue.js	384,2 MB	445 MB	829,2 MB
React Native	307,7 MB	305,9 MB	613,6 MB

Fonte – Própria (2019)

6.1.3 Uso do Processador

O uso do processador é medido em porcentagem (%), sendo o processador responsável pela execução das tarefas das aplicações, o objetivo é apresentar a porcentagem

[1] https://github.com/eddyverbruggen/nativescript-bluetooth/issues

mínima, média e máxima de seu uso durante a execução das aplicações. Para o uso do processador são apresentadas as porcentagens mínima, média e máxima obtidas a partir das 30 execuções de cada mini-aplicação para cada `SO`. A Tabela 24 mostra o uso do processador para a mini-aplicação `Bluetooth`. Pode-se observar que para o `SO Android`, o *framework* `NativeScrip-vue.js` apresentou em média a menor porcentagem (0,5%), embora tenha apresentado o maior pico no uso do processador (43%). Já o `Flutter` aparece com a maior porcentagem média (5%) e também porcentagem máxima elevada (40%). No `SO iOS`, nota-se que o `Ionic` chega a usar 50% do processador, enquanto o `Flutter` apresenta a menor porcentagem média (2%) e máxima (3%).

Tabela 24 – Uso do processador para a mini-aplicação Bluetooth.

Framework	**Mínimo (%)**		**Médio (%)**		**Máximo (%)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	0	0	5	**2**	40	**3**
Ionic	0	0	0,9	8	34	50
NativeScript-Vue.js	0	0	**0,5**	4	43	32
React Native	0	0	0,7	7	**27**	17

Fonte – Própria (2019)

A Tabela 25 apresenta as porcentagens de uso do processador para a mini-aplicação `Câmera`. Para o `SO Android`, pode-se notar que o uso do processador não é maior que 39%, sendo que a porcentagem máxima de uso varia entre 24% a 39%. Embora o uso máximo do processador não ultrapasse 40%, esses valores são considerados altos para aplicações com baixa complexidade. Por outro lado, o `iOS` apresenta a maior variação, possuindo porcentagem máxima entre 5% a 48%, sendo que o `Flutter` apresenta a menor média de uso do processador.

Tabela 25 – Desempenho da mini-aplicação Câmera.

Framework	**Mínimo (%)**		**Médio (%)**		**Máximo (%)**	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Flutter	0	0	7	**2**	30	**3**
Ionic	0	0	5	4	33	28
NativeScript-Vue.js	0	0	**2**	3	39	45
React Native	0	0	4	4	**24**	19

Fonte – Própria (2019)

A Tabela 26 apresenta as porcentagens do uso do processador da mini-aplicação GPS. Para o `SO Android`, o `React Native` apresentou o menor uso do processador, sendo o máximo de uso do processador 8% e uma média de 2%, enquanto os outros *framework* mantiveram o valor máximo maior que 20%. Já para o `iOS`, o `Flutter` mais uma vez apresentou ser o melhor para este tipo de funcionalidade, com a média igual a 2% e a máxima de 5%.

Tabela 26 – Uso do processador para a mini-aplicação GPS.

Framework	Mínimo (%)		Médio (%)		Máximo (%)	
	Android	iOS	Android	iOS	Android	iOS
Flutter	0	0	3	**2**	20	**5**
Ionic	0	0	4	5	30	25
NativeScript-Vue.js	0	0	3	4	48	34
React Native	0	0	**2**	**2**	**8**	14

Fonte – Própria (2019)

A Tabela 27 apresenta as porcentagens de uso do processador para a mini-aplicações `Memória`. Para o `SO Android`, o `React Native` mostrou-se melhor visto que obteve uma porcentagem média menor (2%) e também uma menor porcentagem de uso máximo do processador (15%). As mini-aplicações para `iOS` apresentaram melhores resultados, porém, o `Flutter` ainda apresentou o melhor resultado para este `SO`, apresentando média de 2% e máxima de 5%.

Tabela 27 – Uso do processador para a mini-aplicação Memória.

Framework	Mínimo (%)		Médio (%)		Máximo (%)	
	Android	iOS	Android	iOS	Android	iOS
Flutter	0	0	**2**	**2**	30	**5**
Ionic	0	0	7	6	35	36
NativeScript-Vue.js	0	0	**2**	3	46	33
React Native	0	0	**2**	**2**	**15**	**15**

Fonte – Própria (2019)

Ao somarmos o uso médio do processador obtidos na execução de todas as mini-aplicações para ambos os `SO` (mostrado na Tabela 28), o `NativeScript-vue.js` é o `framework` com o menor uso médio do processador (21,5 %), seguido do `React Native` (23,7 %), do `Flutter` (25 %) e, por fim, do `Ionic` (39,9 %).

Tabela 28 – Soma do uso médio do processador para todas as mini-aplicações.

Framework	Android	iOS	Total
Flutter	17 %	**8** %	25 %
Ionic	16,9 %	23 %	39,9 %
NativeScript-Vue.js	**7,5 %**	14 %	**21,5 %**
React Native	8,7 %	15 %	23,7 %

Fonte – Própria (2019)

6.1.4 Resumo

Com base nos resultados obtidos a partir dos testes, o *framework* `NativeScript-Vue.js` é o que apresentou os melhores resultados em relação ao tempo de execução e uso do processador, porém possui o terceiro maior consumo de memória, apresentando valores bem

superiores aos dois anteriores. Já o `Flutter` apresentou um bom compromisso entre os três critérios analisados, obtendo o menor consumo médio de memória, o segundo menor tempo médio de execução e o terceiro menor uso do processador. O `React Native` obteve o segundo menor consumo de memória e uso do processador, sendo o terceiro em tempo médio de execução. O *framework* `Ionic` obteve o pior desempenho em todos os critérios avaliados.

Adicionalmente, em relação a implementação das mini-aplicações realizadas nos diferentes *frameworks*, alguns aspectos devem ser observados. São eles:

- O `Flutter`, por utilizar a linguagem `Dart`, necessita de algumas configurações extras para sua execução em dispositivos `Android` e `iOS`, sendo mais complexo se tratando do `XCode`, que além de precisar de uma conta de desenvolvedor, é necessário a instalação de plugins manualmente.

- Para a execução das mini-aplicações desenvolvidas no `Ionic` para o `SO iOS`, é necessário realizar a compilação de uma versão para os dispositivos específicos, como por exemplo, compilar uma versão para iPhones, outra para iPads e outra para MacOs.

- Dentre todos os *frameworks*, o `React Native` apresentou o tempo mais longo para geração de arquivos de execução.

- A mini-aplicação `Bluetooth` do `Flutter` somente pode ser compilado em uma versão específica de API do `Android`, sendo essa igual ou superior a versão 23, causando um problema para permitir a criação de uma aplicação com objetivo de atingir versões anteriores.

- Após a atualização do `Ionic`, da versão 3 para a 4, muitos plugins e componentes que estão presentes no `GitHub` ficaram depreciados ou pararam de compilar.

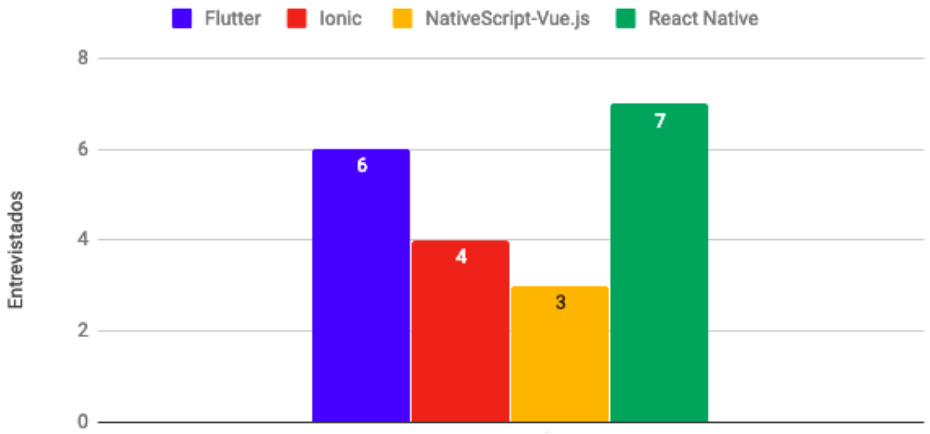
6.2 Questionário

O questionário desenvolvido foi compartilhado com instituições de ensino e indústrias de software, com o objetivo de colher o maior número de opiniões sobre os *frameworks* apresentados neste trabalho. Uma premissa para responder a este questionário é que o respondente tenha experiência prática ou teórica com, no mpinimo, um dos quatro *frameworks*. Com isto, foram obtidas 15 respostas, sendo a identidade dos respondentes mantida em sigilo.

A Figura 15 apresenta a quantidade de respondentes que conhecem cada um dos *frameworks*. Pode-se observar que o `React Native` é o *framework* mais conhecido entre os respondentes, sendo utilizado por quase metade dos respondentes (7). Já o `Flutter`, apesar de ser um *framework* consideravelmente novo, já foi usado por mais profissionais (6) da

área que outros *frameworks* mais consolidados, como o `Ionic` (4) e `NativeScript-Vue.js` (3).

Figura 15 – Quantidade de respondentes que conhecem os *Frameworks*.



Fonte – Própria (2019)

De acordo a Tabela 29, que apresenta as respostas da `QQ1`, nenhum dos *frameworks* descritos neste trabalho possui uma curva de aprendizagem alta, porém, apenas o `Flutter` obteve uma avaliação "Muito Fácil". A resposta "Mediano" foi a mais selecionada pelos respondentes dos *frameworks* com exceção do `NativeScript-Vue.js`, que obteve mais vezes a resposta "Fácil".

Tabela 29 – Nível de dificuldade para aprender o *Framework*.

Framework	Muito Fácil	Fácil	Mediano	Difícil	Muito Difícil
Flutter	1	1	3	0	0
Ionic	0	1	3	0	0
NativeScript-Vue.js	0	2	1	0	0
React Native	0	2	5	0	0

Fonte – Própria (2019)

A Tabela 30 demonstra os resultados da `QQ2`, referente ao nível de satisfação da documentação. O `Flutter`, mesmo sendo considerado novo, e o `Ionic` apresentaram a maioria das respostas como "Muito Satisfatório" e "Satisfatório", pois possuem documentações escritas de forma clara, simples e com apresentação de ótimos exemplos de uso. O `React Native` também apresentou tais respostas, por possuir documentação bem escrita, porém, houve uma resposta "Insatisfatório", pois os exemplos apresentados são muito simples, sendo necessário buscar por informações externas.

A `QQ3` pergunta sobre a velocidade de construção/compilação do *framework* e tem suas respostas apresentadas na Tabela 31. De acordo com os respondentes, nenhum dos *frameworks* é "Lento" ou "Muito Lento" para realizar sua construção ou compilação, porém, o `Flutter` e o `React Native` apresentam os melhores resultados, sendo mais rápidos.

Tabela 30 – Nível de satisfação da documentação dos *Framework*.

Framework	M/ Satis.	Satisfatório	Indiferente	Insatisfatório	M/ Insatis.
Flutter	1	3	1	0	0
Ionic	1	3	0	0	0
NativeScript-Vue.js	0	1	3	0	0
React Native	2	4	0	1	0

Fonte – Própria (2019)

Tabela 31 – Velocidade de construção/compilação do *Framework*.

Framework	Muito Rápido	Rápido	Normal	Lento	Muito Lento
Flutter	2	2	1	0	0
Ionic	0	1	3	0	0
NativeScript-Vue.js	1	0	2	0	0
React Native	1	5	1	0	0

Fonte – Própria (2019)

A Tabela 32 demonstra a opinião dos respondentes sobre a complexidade para acessar as funcionalidades (QQ4). O `React Native` é considerado o *framework* mais simples, porém, ainda tendo uma resposta "Complexo". Em contrapartida, o `Flutter` obteve uma resposta em "Muito Simples" e uma em "Simples", porém, a maioria das respostas foi em "Complexo".

Tabela 32 – Complexidade do *framework* para acessar as funcionalidades.

Framework	Muito Simples	Simples	Normal	Complexo	Muito Complexo
Flutter	1	1	0	3	0
Ionic	0	0	3	1	0
NativeScript-Vue.js	0	0	3	0	0
React Native	1	3	2	1	0

Fonte – Própria (2019)

A Tabela 33 contém as respostas da QQ5, referente as vantagens de cada *framework*. Nota-se algumas vantagens que estão presentes nos *frameworks* apresentados neste trabalho, sendo elas:

- Ser híbrida, possibilitando a geração de aplicações para dois ou mais SO de dispositivos móveis;
- Permitir o desenvolvimento de aplicações através de componentes ou plugins;
- Ter a possibilidade de integrar bibliotecas externas de código aberto ou comerciais.

Os *frameworks* também apresentam algumas vantagens particulares. O `Flutter`, por exemplo, possui componentes próprios baseados no `Material Design` e possibilidade do uso de Programação Orientada a Objetos. `Material Design` é um sistema de guias,

padrões e componentes que auxiliam na construção de telas agradáveis, simples e interativas[2]. Já o `Ionic` oferece a possibilidade de criação de aplicações Web. O `React Native` atualmente é o mais estável no mercado. Já o `NativeScript-vue.js` foi o único *framework* a não apresentar nenhuma vantagem particular.

Tabela 33 – Vantagens dos *frameworks*.

Framework	**Vantagem**
Flutter	Ser híbrido; A que mais se destaca é o desempenho; Possibilidade de melhor uso de Programação Orientada a Objetos; Códigos para criação de telas muito simples, aplicativos de alto desempenho e bem fluidos, animações sem gargalos; Aplicativos robustos e leves, tanto em tamanho de aplicação quanto em fluidez; É capaz de construir componentes visuais , bonitos, leves e com pouco código; Possui componentes próprios baseado em `Material Design`; Compilar aplicação para código nativo para as plataformas `Android` e `iOS`.
Ionic	Construção de aplicação híbrida, possui uma extensa biblioteca; *Framework* baseado em tecnologia web além de criar aplicações para Android e IOS é possível executar as mesmas aplicações para navegador, além de ser vantajoso para desenvolvedores web pois o `Ionic` utiliza `HTML`, `CSS` e `JavaScript`; Nada complexo, quase tudo é encontrado de forma simples na documentação do *framework*
NativeScript-Vue.js	Aplicações hibridas; Utiliza tecnologia Web podendo usar `JavaScript`, `TypeScript`, `Angular2` ou `Vue.js` e conta também com `CSS` para os estilos; Gera aplicações nativas para cada sistema operacional, utiliza linguagem de programação popularmente conhecida.
React Native	Criar aplicativos simples e a consumação de API em `JSON`; Simplicidade no código; Mais estável no mercado, com quantidade maior de componentes e documentação na internet; Aplicações híbridas; Constrói aplicativos de excelente performance; Códigos ficam bem organizados e estruturados; Bibliotecas para acesso de recurso nativos são lançadas/atualizadas em um curto prazo de tempo; `React Native` tem como vantagem a premissa de utilizar a biblioteca de componentes nativa do sistema operacional onde o código é compilado. A documentação, mesmo para alguém sem experiência, é de fácil interpretação.

Fonte – Própria (2019)

As desvantagens relatadas pelos respondentes (`QQ6`) estão sintetizadas na Tabela 34. O `Flutter` e o `React Native` possuem a maioria das desvantagens citadas, destacando-se o fato do `Flutter` ser um *framework* recente, ser obrigatório a criação de

[2] https://material.io/

extensões para as bibliotecas externas e a utilização do DART como linguagem de programação. O React Native também apresenta particularidades, como a configuração necessária nos arquivos nativos de cada SO ao acessar funcionalidades nativas do dispositivo e a tendência ao uso da biblioteca Redux.

Salienta-se que, para todos os *frameworks* é obrigatório o uso de um dispositivo com o SO MacOs para executar a aplicação em simuladores ou no dispositivo físico para iOS.

Tabela 34 – Desvantagens dos *frameworks*.

Framework	**Desvantagem**
Flutter	Extremamente novo; Utilização de algumas bibliotecas internas que necessitam a criação de sua extensão; Por ser muito recente, sua versão 1.0 ainda carece de componentes ao utilizar recursos mais avançados; Poucos membros brasileiros ativos na comunidade; Códigos complexos para acesso de funcionalidades nativas do dispositivo; Códigos mais complexos e pouco conteúdo em Português; *framework* baseado em linguagem de programação DART Atualmente só é possível gerar aplicações para Android e IOS.
Ionic	Bibliotecas para acesso nativo do dispositivo demoram pra ser publicadas/atualizadas; Não é recomendado para aplicativos complexos, *Ionic* roda encima de uma WebView pode deixar o aplicativo mais lento quando for preciso usar recursos nativos; Bibliotecas para acesso de recursos nativos podem demorar algum tempo para serem atualizadas/desenvolvidas; Muitas adaptações para um recurso nativo funcionar em todas as plataformas (Android, e IOS).
NativeScript-Vue.js	Mercado ainda muito pequeno; Pouca documentação e exemplos de códigos; Necessário dispositivo com SO MacOs para emular e gerar aplicativos para iOS; Configuração do projeto por meio de Interface de Linha de Comando (do inglês *Command-Line Interface*) (CLI).
React Native	Acesso as funcionalidades nativas necessitam de edição nos arquivos internos de cada plataforma; Você se sente forçado a criar uma aplicação com Redux para gerenciamento dos estados, mesmo que não for usar, por conta do esforço feito pra implementação no meio do projeto; Um pouco mais complexo para entender a funcionalidade; Códigos para alguns recurso nativos precisam ser adaptados para cada plataforma de maneira diferente; Uma desvantagem é a necessidade de ter um dispositivo com SO MacOs para compilar o código para iOS, e também configurar componentes, como notificação por exemplo; As vezes é corrigido alguns problemas de configuração e os mesmos aparecem constantemente em alguns momento.

Fonte – Própria (2019)

As dificuldades e problemas levantados pelos respondentes na QQ7 podem ser observados na Tabela 35. Dentre todos os *frameworks*, um problema que aparece em comum

entre eles trata-se do uso de bibliotecas. O `Flutter` apresenta algumas bibliotecas, como a utilizada na mini-aplicação Bluetooth, que necessita de instalação e configuração do `SDK` mais recente e a implementação para a API do `Android` igual ou maior que 23, fazendo com que não seja possível executar em versões antigas do `SO`. O `Ionic` possui diversas bibliotecas para o mesmo objetivo, dificultando a decisão do desenvolvedor e contendo bibliotecas que não funcionam em versões mais atuais do *framework*. O `NativeScript-Vue.js` apresenta o inverso do `Ionic`, a ausência de bibliotecas dificulta sua implementação. O `React Native` possui bibliotecas complexas, gerando problemas na hora de instalar e configurá-las.

Nota-se também problemas particulares dos *frameworks*, como por exemplo o uso da linguagem de programação `DART`, a obrigatoriedade do registro dos componentes criados em algum dos módulos da aplicação, a falta de exemplos de códigos fonte e a dependência do uso do *framework* `Expo`[3].

Tabela 35 – Dificuldades dos *frameworks*.

Framework	**Dificuldade**
Flutter	Código muito grande; Falta de alguns componentes e instabilidade na utilização de alguns recursos mais avançados; Algumas bibliotecas te obrigam á ter a `SDK` mais atualizada do `Android`; Documentação somente em inglês; Códigos complexos para acessar recursos nativos; Por ser baseado em `DART` pode dificultar a curva de aprendizagem.
Ionic	Todo componente criado deve ser registrado em algum módulo; Existem muitas bibliotecas para o `Ionic` no qual dificulta um pouco para escolher qual usar, pois algumas podem estar descontinuadas ou podem demorar muito tempo ter alguma atualização e outras são desenvolvidas especificamente para um SO.
NativeScript-Vue.js	Poucos exemplos de códigos faz com que fique mais tempo analisando e compreendendo determinadas funções de bibliotecas; Muita presença de erro comum ao criar e desenvolver diferentes projetos, que poderiam ser auxiliados à solução de uma melhor maneira para quem desenvolve; Não há como testar os aplicativos para `iOS` sem ter um dispositivo com `SO` `MacOs`
React Native	A dependência do uso do *framework* `Expo` em desenvolvimento; Problemas na instalação/configuração de componentes mais complexos Alguns códigos de recurso nativos tem que ser escrito de formas diferentes para cada tipo de `SO`; De modo geral, em nosso ambiente, enfrentamos um pouco de dificuldade na implantação/organização da navegação.

Fonte – Própria (2019)

A Tabela 36 mostra os pontos positivos dos *frameworks* (`QQ8`). O desenvolvimento

[3] https://expo.io/

de aplicações fluídas e curto tempo de desenvolvimento são pontos positivos que foram levantados pelos respondentes sobre todos os *frameworks*.

O `React Native` apresenta um outro ponto positivo: o fato de não ser executado através de `WebView`, aumentando o desempenho de aplicações robustas e possuindo *design* mais próximo ao nativo.

Tabela 36 – Pontos positivos encontrados no uso dos *Frameworks*.

Framework	**Pontos Positivos**
Flutter	Desenvolvimento Rápido; Simples e muito intuitivo; Baseado em `DART`, código simples e de fácil entendimento; Instalação e configuração simples; Baixa curva de aprendizado; Poucas adaptações de códigos para recursos nativos; Usa componentes próprios baseado em Material Design[4]; Fácil instalação e configuração do *Framework*.
Ionic	*Framework* simples para aprender, principalmente se for um desenvolvedor de aplicações; Fácil de aprender principalmente pra quem é desenvolvedor Web; Possível criar a mesma aplicação para navegadores; Documentação com exemplos muito simples, se procurar fazer; algo mais complexo com as funcionalidades nativas, deverá procurar fora da documentação do `Ionic`.
NativeScript-Vue.js	Aplicação fluída e rápida; Rápido desenvolvimento; Rápida geração dos aplicações; Fácil aprendizagem.
React Native	Utilizar `JavaScript` como linguagem e ter uma comunidade bastante ativa; Baixa curva de aprendizado, quem vem do desenvolvimento web não sentira muita dificuldade, constrói aplicações robustas bem fluidas; Grande aumento e aceitação no mercado; Constrói aplicações com códigos nativos; Fácil aprendizagem pra quem já é desenvolvedor web; Uma aplicação desenvolvida com `React Native` tende a ter uma maior qualidade no produto final, até mesmo em desempenho, diferente de outros *frameworks*, como o `Ionic`, que roda a aplicação em uma `WebView`; Reutilização é um ponto muito forte da tecnologia.

Fonte – Própria (2019)

6.2.1 Resumo

Analisando as respostas dos participantes, entende-se que, o `NativeScript-Vue.js`, possui poucos pontos negativos e positivos pelo fato de não ser muito conhecido pelos respondentes. Porém, foi levantado o ponto positivo do suporte as linguagens de programação `Angular2` e `TypeScript` e o negativo de não possuir muitos exemplos, por sua comunidade ser consideravelmente menor. O `Ionic` se destacou positivamente pela baixa curva de aprendizagem e possibilidade de criar aplicações web. Contudo, apresenta des-

vantagens na questão desempenho, em vista que necessita de `WebView` para ser executado e sua grande quantidade de bibliotecas que pode causar confusões na hora da decisão. O `Flutter`, apesar de novo, mostrou-se bem conhecido dentre os *frameworks* apresentados. Os respondentes salientaram o fato da linguagem de programação ser `DART`, o que remete a alta curva de aprendizagem, o ótimo desempenho apresentado por aplicações robustas e simples, também o fato de não possuir muitas bibliotecas e componentes disponíveis. O que apresenta a melhor avaliação dos usuários é o `React Native`. Embora o uso de algumas bibliotecas seja apontada como complexo, ele possui uma ótima documentação, apresenta ótimo desempenho e é um *framework* consolidado e estável no mercado de software.

7 CONSIDERAÇÕES FINAIS

Neste trabalho foram apresentados os conceitos fundamentais sobre os SO Android e iOS e suas características; as aplicações móveis e suas categorias; e os *frameworks* React Native, Ionic, Flutter e NativeScript-Vue.js. Adicionalmente, um mapeamento sistemático foi realizado, selecionando 9 trabalhos que possuem objetivos semelhantes a este. Com base nesses estudos, características e funcionalidades foram identificadas para serem comparadas neste trabalho.

Para realizar a comparação entre os *frameworks*, foram desenvolvidas mini-aplicações que passaram por conjuntos de testes com o objetivo de avaliar os seguintes critérios: tempo de execução, consumo de memória e porcentagem do uso do processador do dispositivo. Ainda, um questionário foi elaborado e distribuído aos estudantes e profissionais da área de desenvolvimento de aplicações móveis com objetivo de identificar características que podem ser observadas com a experiência prática com cada *framework*.

Com os resultados obtidos, pode-se observar que:

- O NativeScript-Vue.js apresentou menor tempo médio de execução para as mini-aplicações de ambos SO. Esse *framework* também apresentou a menor média no uso do processador para algumas mini-aplicações, porém a porcentagem máxima foi a mais elevada entre os *frameworks*, podendo indicar que há presença de altas oscilações. Destaca-se também o fato de ser o *framework* menos conhecido entre os respondentes do questionário.

- O Flutter apresentou os menores valores médios para o consumo de memória. Mesmo este *framework* sendo novo no mercado e ainda não possuindo grande quantidade de recursos, mostrou-se bem conhecido e utilizado. Adicionalmente, alguns pontos foram levantados, como a dificuldade no uso da linguagem de programação DART, a possibilidade de desenvolver utilizando nativamente Programação Orientada a Objetos e um bom desempenho para ambos SOs.

- O React Native é o mais indicado para o desenvolvimento rápido e menos complexo. Ele apresentou as melhores respostas em relação à satisfação da documentação, à velocidade de compilação e à complexidade para acessar funcionalidades nativas do dispositivo. Sendo assim, pode-se inferir que qualquer dúvida que apareça durante o desenvolvimento, poderá ser sanada pela documentação ou não terá uma resolução complexa. Este framework foi o mais conhecido entre os respondentes e de acordo com os resultados dos testes, apresentou o segundo menor consumo médio de memória e uso médio do processador entre os quatro frameworks avaliados.

- O Ionic mostrou ser o framework menos aconselhável para aplicações que venham a acessar funcionalidades nativas do dispositivo ou aplicativos que necessitem de lógicas muito complexas. Este apresentou os piores resultados do uso do processador

nos testes realizados, e apesar de apresentar uma boa documentação para auxiliar os usuários, não possui nenhuma vantagem nos outros quesitos. Destaca-se o fato de ser executado através de `WebView`, o que deixa seus recursos mais lentos.

Para a realização deste trabalho, duas limitações foram encontradas, sendo elas o tempo para encontrar a ferramenta de teste e a baixa quantidade de respostas obtidas pelo questionário. Encontrar a ferramenta de teste mostrou-se um desafio, pois cada ferramenta apresentava algum impedimento que inviabilizava seu uso, como por exemplo: a falta de funcionalidade para executar *scripts*, não ser possível realizar a gravação das ações na tela e repeti-las inúmeras vezes e a falta de compatibilidade com todos os *frameworks*. O questionário foi enviado para 80 pessoas, porém, de acordo com VASCONCELLOS-GUEDES e GUEDES (2007), o baixo número de respostas obtidos pelo questionário foi ocasionado pela forma de sua distribuição, pois o envio por e-mail possui algumas limitações, como por exemplo: dificuldade de incluir incentivos para envio da resposta; respondentes podem considerar o recebimento da mensagem de e-mail não desejada como uma invasão de privacidade ou "lixo eletrônico"; baixa confiabilidade nos dados, uma vez que muitos respondentes podem falsificar informações.

Para trabalhos futuros, há a motivação para a realizar avanços em usabilidade e desempenho. O trabalho com viés em usabilidade se baseia em comparar os componentes nativos de *frameworks* identificando quais estão disponíveis, a complexidade para personalização das telas, validação com usuários para identificar diferenças em tempo de resposta da tela e possíveis comparações nas animações. O outro trabalho tem como objetivo comparar o desempenho dos `frameworks` quando se trata de aplicações complexas e de larga escala, contendo grande número de funcionalidades e testes em ambientes reais.

REFERÊNCIAS

AHTI, V.; HYRYNSALMI, S.; NEVALAINEN, O. An evaluation framework for cross-platform mobile app development tools: A case analysis of adobe phonegap framework. In: **Proceedings of the 17th International Conference on Computer Systems and Technologies 2016**. New York, NY, USA: ACM, 2016. (CompSysTech '16), p. 41–48. ISBN 978-1-4503-4182-0. Disponível em: <http://doi.acm.org/10.1145/2983468.2983484>. Citado 4 vezes nas páginas 44, 46, 49 e 50.

ALCANTARA, C. **React Patterns — Começando pelo Render Props**. 2018. Disponível em: <https://medium.com/collabcode/react-patterns-come%C3%A7ando-pelo-render-props-e0040ef723ce>. Acesso em: 19-11-2018. Citado na página 38.

ANDROID, D. **Kotlin and Android**. 2018. Disponível em: <https://developer.android.com/kotlin>. Acesso em: 11-11-2018. Citado na página 30.

APPLE. **Apple**. 2018. Disponível em: <https://www.apple.com>. Acesso em: 12-11-2018. Citado na página 29.

APPLE, I. **XNU**. 2018. Disponível em: <https://opensource.apple.com/tarballs/xnu/>. Acesso em: 12-11-2018. Citado na página 31.

APPLE inc. **Apple Developer**. 2018. Disponível em: <https://developer.apple.com>. Acesso em: 12-11-2018. Citado na página 31.

BALASUBRAMANIAN, N.; BALASUBRAMANIAN, A.; VENKATARAMANI, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In: ACM. **Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement**. [S.l.], 2009. p. 280–293. Citado na página 32.

BERGHER, R. **Saiba tudo sobre sistema operacional de celular**. 2012. Disponível em: <https://www.zoom.com.br/celular/deumzoom/saiba-tudo-sobre-sistema-operacional-de-celular>. Acesso em: 15-11-2018. Citado 2 vezes nas páginas 29 e 30.

BOTELLA, F.; ESCRIBANO, P.; PEÑALVER, A. Selecting the best mobile framework for developing web and hybrid mobile apps. In: **Proceedings of the XVII International Conference on Human Computer Interaction**. New York, NY, USA: ACM, 2016. (Interacción '16), p. 40:1–40:4. ISBN 978-1-4503-4119-6. Disponível em: <http://doi.acm.org/10.1145/2998626.2998648>. Citado 7 vezes nas páginas 36, 44, 46, 47, 48, 49 e 50.

BRISTOWE, J. **What is a Hybrid Mobile App?** 2017. Disponível em: <https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app>. Acesso em: 18-11-2018. Citado na página 36.

BRITO, H. et al. Javascript in mobile applications: React native vs ionic vs nativescript vs native development [javascript em aplicações móveis: React native vs ionic vs nativescript vs desenvolvimento nativo]. In: A. COTA M.P., L.-T. A. G. R. R. (Ed.). IEEE Computer Society, 2018. v. 2018-June, p. 1–6. ISBN 9789899843486. ISSN 21660727. Cited By 0. Disponível em: <https:

//www.scopus.com/inward/record.uri?eid=2-s2.0-85049887829&doi=10.23919%2fCISTI.2018.8399283&partnerID=40&md5=40f91e5588d7b53618ec058e5479b88b>. Citado 10 vezes nas páginas 33, 36, 40, 44, 46, 47, 48, 49, 50 e 51.

BUDGEN, D.; BRERETON, P. Performing systematic literature reviews in software engineering. In: ACM. **Proceedings of the 28th international conference on Software engineering**. [S.l.], 2006. p. 1051–1052. Citado na página 39.

CALLAHAM, J. **Google made its best acquisition 13 years ago: Can you guess what it was?** 2018. Disponível em: <https://www.androidauthority.com/google-android-acquisition-884194>. Acesso em: 10-11-2018. Citado na página 30.

CAMBRIDGE, U. P. **Aplicação Móvel (*Mobile Application*)**. 2018. Disponível em: <https://dictionary.cambridge.org/dictionary/english/mobile-application>. Acesso em: 17-11-2018. Citado na página 31.

CAPUTO, V. **Relembre cada modelo de iPhone lançado pela Apple**. 2017. Disponível em: <https://exame.abril.com.br/tecnologia/relembre-cada-modelo-de-iphone-lancado-pela-apple/>. Acesso em: 12-11-2018. Citado na página 31.

CHARLAND, A.; LEROUX, B. Mobile application development: web vs. native. **Queue**, ACM, v. 9, n. 4, p. 20, 2011. Citado na página 32.

CIMAN, M.; GAGGI, O.; GONZO, N. Cross-platform mobile development: A study on apps with animations. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2014. (SAC '14), p. 757–759. ISBN 978-1-4503-2469-4. Disponível em: <http://doi.acm.org/10.1145/2554850.2555104>. Citado 7 vezes nas páginas 36, 37, 44, 46, 47, 48 e 50.

DALMASSO, I. et al. Survey, comparison and evaluation of cross platform mobile application development tools. In: . [s.n.], 2013. p. 323–328. ISBN 9781467324793. Cited By 47. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84883695902&doi=10.1109%2fIWCMC.2013.6583580&partnerID=40&md5=1159bf2cf9bc06d1b24ea42bb06321d8>. Citado 6 vezes nas páginas 44, 45, 46, 47, 50 e 51.

FACEBOOK, O. S. **React**. 2018. Disponível em: <https://reactjs.org>. Acesso em: 18-11-2018. Citado na página 38.

FERREIRA, C. M. et al. An evaluation of cross-platform frameworks for multimedia mobile applications development. **IEEE Latin America Transactions**, IEEE, v. 16, n. 4, p. 1206–1212, 2018. Citado na página 36.

FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. [S.l.]: Bookman, 2009. Citado na página 60.

GASPAROTTO, H. M. **Aplicações Móveis: Nativas ou Web?** 2014. Disponível em: <https://www.devmedia.com.br/aplicacoes-moveis-nativas-ou-web/30392>. Acesso em: 05-05-2019. Citado na página 53.

GAžO, F. **Native apps with Flutter and React Native?** 2018. Disponível em: <https://medium.com/inloopx/native-apps-with-flutter-and-react-native-8d300805b3c6>. Acesso em: 01-01-2019. Citado na página 37.

GOODWILL, C. F. **O que é um sistema operacional?** 2018. Disponível em: <https://edu.gcfglobal.org/pt/informatica-basica/o-que-e-um-sistema-operacional/1>. Acesso em: 15-11-2018. Citado na página 29.

GOOGLE. **Android**. 2018. Disponível em: <https://www.android.com>. Acesso em: 10-11-2018. Citado na página 29.

GOOGLE. **Material Design**. 2018. Disponível em: <https://material.io>. Acesso em: 17-11-2018. Citado na página 33.

GUPTA, A. **Part 1: The road to cross platform frameworks**. 2018. Disponível em: <https://medium.com/coding-blocks/part-1-the-road-to-cross-platform-frameworks-d6a193b9ce2d>. Acesso em: 06-06-2019. Citado na página 27.

HEITKÖTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Evaluating cross-platform development approaches for mobile applications. In: SPRINGER. **International Conference on Web Information Systems and Technologies**. [S.l.], 2012. p. 120–138. Citado 2 vezes nas páginas 36 e 40.

HEITKöTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. Comparing cross-platform development approaches for mobile applications. In: . [s.n.], 2012. p. 299–311. ISBN 9789898565082. Cited By 27. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84864885189&partnerID=40&md5=63f96e62bbff982667de56578306b125>. Citado 8 vezes nas páginas 36, 44, 46, 47, 48, 49, 50 e 51.

IONIC. **Ionic**. 2018. Disponível em: <https://ionicframework.com/docs/intro/concepts/>. Acesso em: 20-11-2018. Citado na página 37.

JOBE, W. Native apps vs. mobile web apps. **International Journal of Interactive Mobile Technologies (iJIM)**, v. 7, n. 4, p. 27–32, 2013. Citado na página 34.

KASUNIC, M. **Designing an effective survey**. [S.l.], 2005. Citado na página 54.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1–26, 2004. Citado na página 39.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering–a systematic literature review. **Information and software technology**, Elsevier, v. 51, n. 1, p. 7–15, 2009. Citado na página 39.

KITCHENHAM, B. A. et al. Lessons from applying the systematic literature review process within the software engineering domain. **Journal of Systems and Software**, v. 80, n. 4, p. 571 – 583, 2007. ISSN 0164-1212. Software Performance. Disponível em: <http://www.sciencedirect.com/science/article/pii/S016412120600197X>. Citado na página 39.

KORPIPÄÄ, P.; MÄNTYJÄRVI, J. An ontology for mobile device sensor-based context awareness. In: SPRINGER. **International and Interdisciplinary Conference on Modeling and Using Context**. [S.l.], 2003. p. 451–458. Citado na página 32.

KYFONIDIS, C.; MOUMOUTZIS, N.; CHRISTODOULAKIS, S. Block-c: a block-based programming teaching tool to facilitate introductory c programming courses. In: IEEE. **Global Engineering Education Conference (EDUCON), 2017 IEEE**. [S.l.], 2017. p. 570–579. Citado na página 31.

LINUX, O. K. **The Linux Kernel Archives**. 2018. Disponível em: <https://www.kernel.org>. Acesso em: 10-11-2018. Citado na página 30.

MALAVOLTA, I. et al. Hybrid mobile apps in the google play store: An exploratory investigation. In: IEEE PRESS. **Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems**. [S.l.], 2015. p. 56–59. Citado na página 36.

MANCHANDA, A. **Native Vs Hybrid Apps Development – Finding Clarity in Confusion**. 2018. Disponível em: <https://www.netsolutions.com/insights/native-vs-hybrid-apps-from-confusion-to-clarity/>. Acesso em: 28-11-2019. Citado na página 33.

MERCADO, I. T.; MUNAIAH, N.; MENEELY, A. The impact of cross-platform development approaches for mobile applications from the user's perspective. In: **Proceedings of the International Workshop on App Market Analytics**. New York, NY, USA: ACM, 2016. (WAMA 2016), p. 43–49. ISBN 978-1-4503-4398-5. Disponível em: <http://doi.acm.org/10.1145/2993259.2993268>. Citado 5 vezes nas páginas 44, 45, 46, 49 e 50.

MORE, K. A.; CHANDRAN, M. P. Native vs hybrid apps. **Proceeding of International Journal of Current Trends in Engineering & Research**, p. 563–572, 2016. Citado na página 33.

NELSON, R. **Apple's App Store Will Hit 5 Million Apps by 2020, More Than Doubling Its Current Size**. 2016. Disponível em: <https://sensortower.com/blog/app-store-growth-forecast-2020>. Acesso em: 06-06-2019. Citado na página 27.

NIELD, D. **Conheça todos os sensores do seu smartphone e como eles funcionam**. 2017. Disponível em: <https://gizmodo.uol.com.br/sensores-smartphones-guia>. Acesso em: 17-11-2018. Citado na página 32.

OCCHINO, T. **React Native: Bringing modern web techniques to mobile**. 2015. Disponível em: <https://code.fb.com/android/react-native-bringing-modern-web-techniques-to-mobile/>. Acesso em: 19-11-2018. Citado na página 38.

ORACLE. **Java Platform Standard Edition 8 Documentation**. 2018. Disponível em: <https://docs.oracle.com/javase/8/docs>. Acesso em: 12-11-2018. Citado na página 30.

OVERFLOW, S. **Most Loved, Dreaded, and Wanted Languages**. 2018. Disponível em: <https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted>. Acesso em: 2-12-2018. Citado na página 37.

PALMIERI, M.; SINGH, I.; CICCHETTI, A. Comparison of cross-platform mobile development tools. In: . [s.n.], 2012. p. 179–186. ISBN 9781467315265. Cited By 51. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84871874191&doi=10.1109%2fICIN.2012.6376023&partnerID=40&md5=98f50b7842b8eea9c86fdd5738864e08>. Citado 6 vezes nas páginas 36, 44, 46, 48, 49 e 50.

PATIL, R. **Attractive Features Of Ionic App Development Framework**. 2017. Disponível em: <https://www.mytechlogy.com/IT-blogs/15907/attractive-features-of-ionic-app-development-framework/#.W_Qrn5NKjOQ>. Acesso em: 20-11-2018. Citado na página 37.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição**. [S.l.]: Editora Feevale, 2013. Citado na página 53.

PUREDARWIN. **PureDarwin**. 2018. Disponível em: <http://www.puredarwin.org>. Acesso em: 12-11-2018. Citado na página 31.

RAWLINGS, R. Objective-c: an object-oriented language for pragmatists. In: IET. **Applications of Object-Oriented Programming, IEE Colloquium on**. [S.l.], 1989. p. 2–1. Citado na página 31.

REBOUÇAS, M. et al. An empirical study on the usage of the swift programming language. In: IEEE. **Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on**. [S.l.], 2016. v. 1, p. 634–638. Citado na página 31.

RUIQING, D.; XIAOHUI, L. Discussions on the teaching of c++ programming language in colleges. In: IEEE. **Artificial Intelligence and Education (ICAIE), 2010 International Conference on**. [S.l.], 2010. p. 579–581. Citado na página 31.

SANTOS, C. M. da C.; PIMENTA, C. A. de M.; NOBRE, M. R. C. A estratégia pico para a construção da pergunta de pesquisa e busca de evidências. **Revista Latino-Americana de Enfermagem**, SciELO Brasil, v. 15, n. 3, p. 508–511, 2007. Citado na página 39.

SCHUMAN, D. **História do Android**. 2018. Disponível em: <https://www.androidpit.com.br/historia-do-android>. Acesso em: 17-11-2018. Citado na página 30.

SILVEIRA, J. A. et al. Effectiveness of school-based nutrition education interventions to prevent and reduce excessive weight gain in children and adolescents: a systematic review. **Jornal de pediatria**, SciELO Brasil, v. 87, n. 5, p. 382–392, 2011. Citado na página 39.

SOFTWARE, C. P. **Create Native iOS and Android Apps With JavaScript**. 2018. Disponível em: <https://www.nativescript.org/>. Acesso em: 3-12-2018. Citado na página 38.

SOFTWARE, C. P. **Getting Started With NativeScript and Angular**. 2018. Disponível em: <https://docs.nativescript.org/angular/start/introduction>. Acesso em: 3-12-2018. Citado na página 38.

SOFTWARE, C. P. **Truly native apps using Vue.js and NativeScript**. 2018. Disponível em: <https://nativescript-vue.org>. Acesso em: 3-12-2018. Citado na página 38.

STATE, L. **Cross-Platform App Development: Ending the iOS vs. Android Debate**. 2018. Disponível em: <https://liquid-state.com/cross-platform-app-development/>. Acesso em: 23-05-2019. Citado na página 27.

STATISTA. **Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018**. 2018. Disponível em: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems>. Acesso em: 15-11-2018. Citado na página 29.

STATISTA. **Number of available applications in the Google Play Store from December 2009 to March 2019**. 2019. Disponível em: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acesso em: 06-06-2019. Citado na página 27.

TRIPATHI, A. et al. **Optimizing data traffic and power consumption in mobile unified communication applications**. [S.l.]: Google Patents, 2011. US Patent 7,974,194. Citado na página 32.

UNFOLDLABS. **2019 Mobile App Development — Is Cross Platform still a Fad?** 2019. Disponível em: <https://medium.com/@Unfoldlabs/2019-mobile-app-development-is-cross-platform-still-a-fad-2bb26fc798ef>. Acesso em: 29-06-2019. Citado na página 27.

VASCONCELLOS-GUEDES, L.; GUEDES, L. F. A. E-surveys: Vantagens e limitações dos questionários eletrônicos via internet no contexto da pesquisa científica1. **X SemeAd-Seminário em Administração FEA/USP (São Paulo, Brasil)**, 2007. Citado na página 84.

VENDORS, R. . W.; RESOURCES. **difference between native app, web app and hybrid app | native app vs web app vs hybrid app**. 2018. Disponível em: <http://www.rfwireless-world.com/Terminology/Native-App-vs-Web-App-vs-Hybrid-App.html>. Acesso em: 07-02-2019. Citado 2 vezes nas páginas 34 e 35.

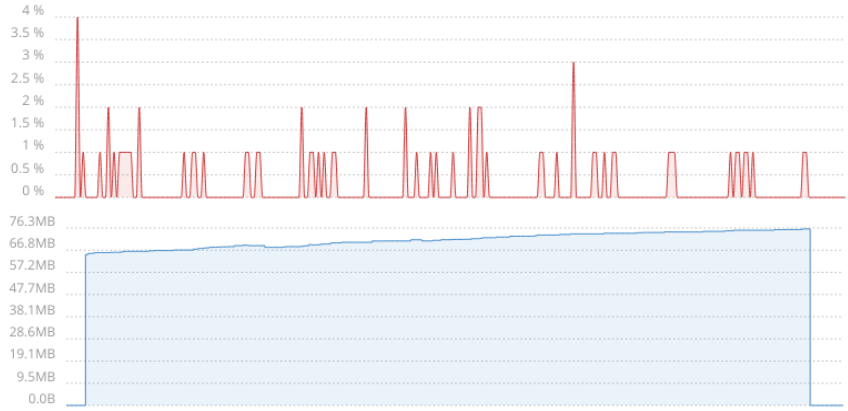
WASSERMAN, A. I. Software engineering issues for mobile application development. In: ACM. **Proceedings of the FSE/SDP workshop on Future of software engineering research**. [S.l.], 2010. p. 397–400. Citado 2 vezes nas páginas 31 e 32.

XANTHOPOULOS, S.; XINOGALOS, S. A comparative analysis of cross-platform development approaches for mobile applications. In: **Proceedings of the 6th Balkan Conference in Informatics**. New York, NY, USA: ACM, 2013. (BCI '13), p. 213–220. ISBN 978-1-4503-1851-8. Disponível em: <http://doi.acm.org/10.1145/2490257.2490292>. Citado 6 vezes nas páginas 27, 34, 44, 45, 46 e 50.

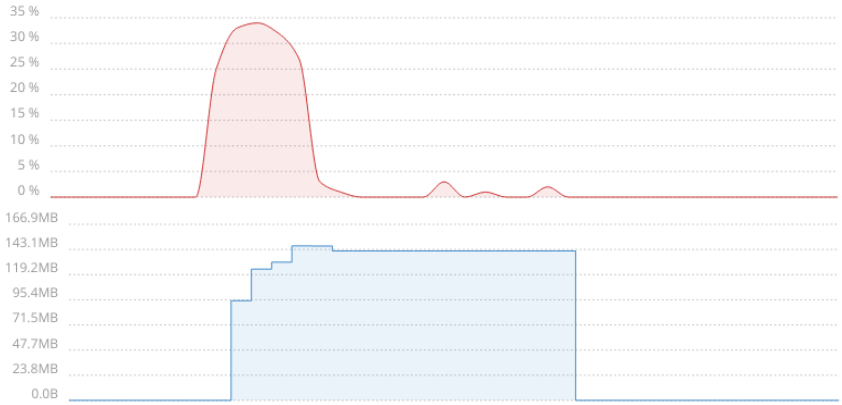
Apêndices

APÊNDICE A – RESULTADOS BLUETOOTH ANDROID

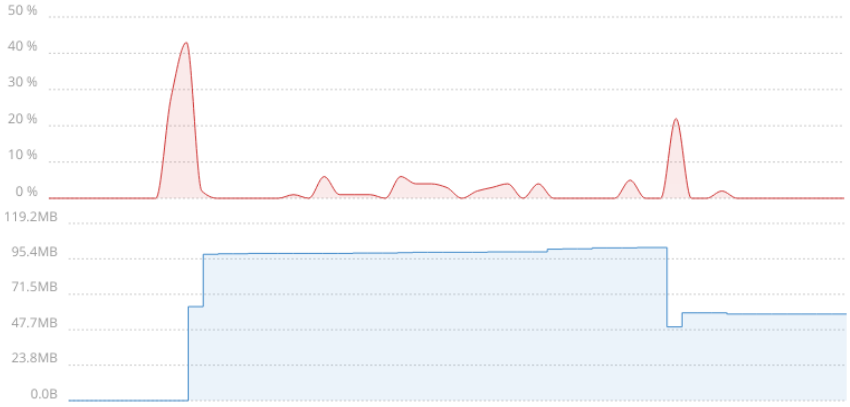
Figura 16 – Relatório do desempenho e uso de memória Bluetooth/Android.



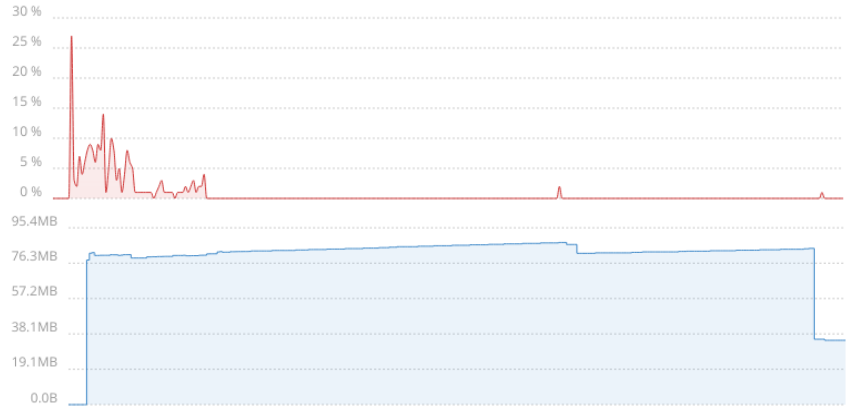
(a) Flutter



(b) Ionic



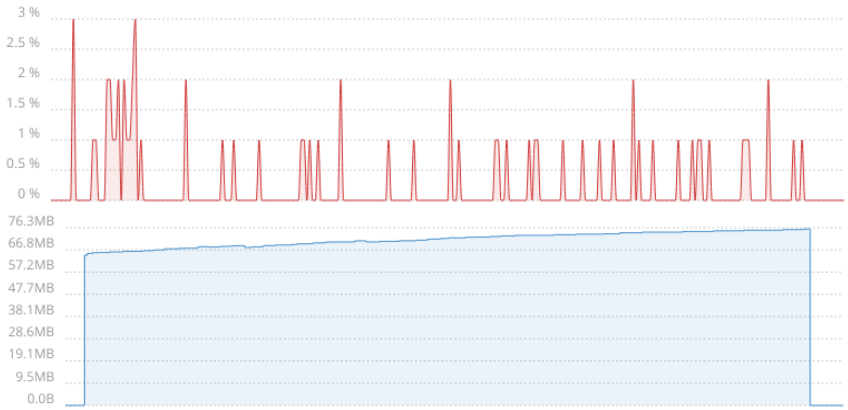
(c) NativeScript



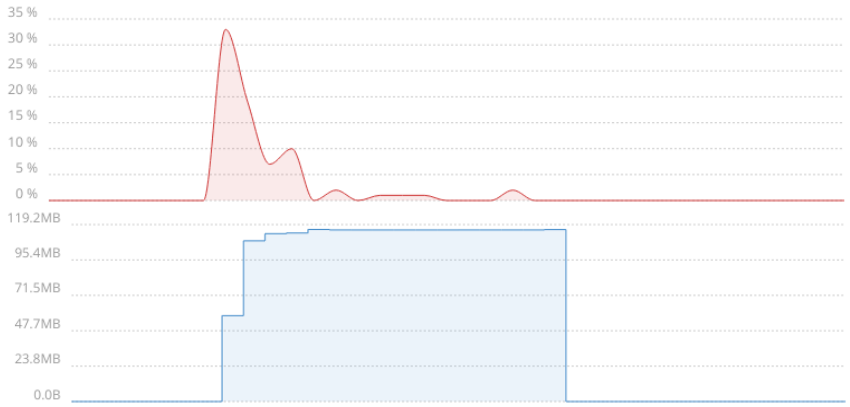
(d) React Native

APÊNDICE B – RESULTADOS CÂMERA ANDROID

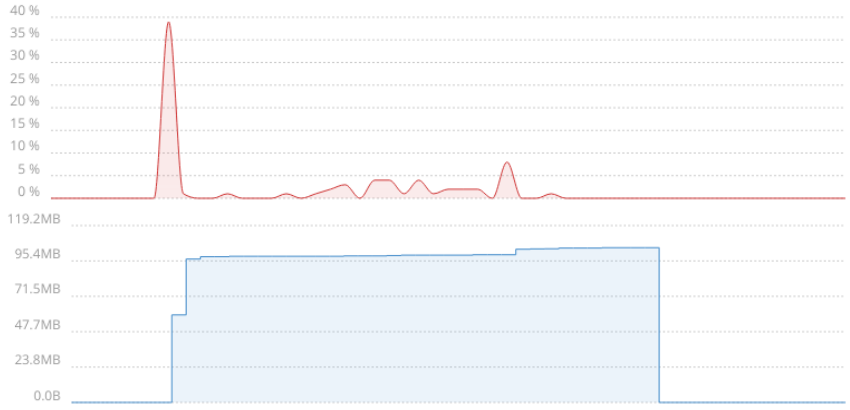
Figura 17 – Relatório do desempenho e uso de memória Camera/Android.



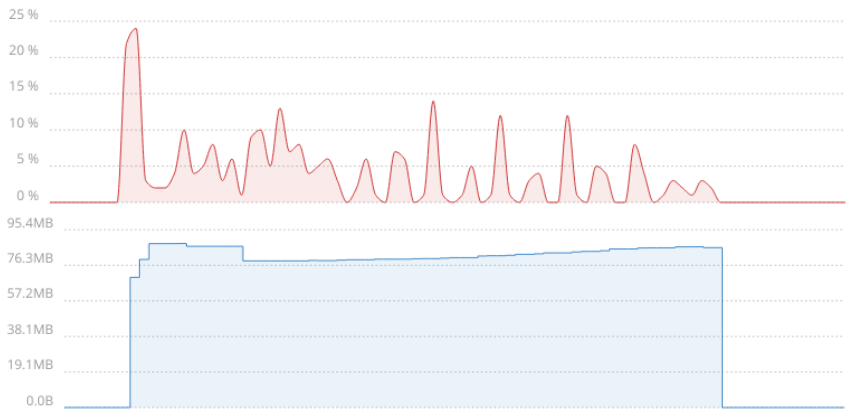
(a) Flutter



(b) Ionic



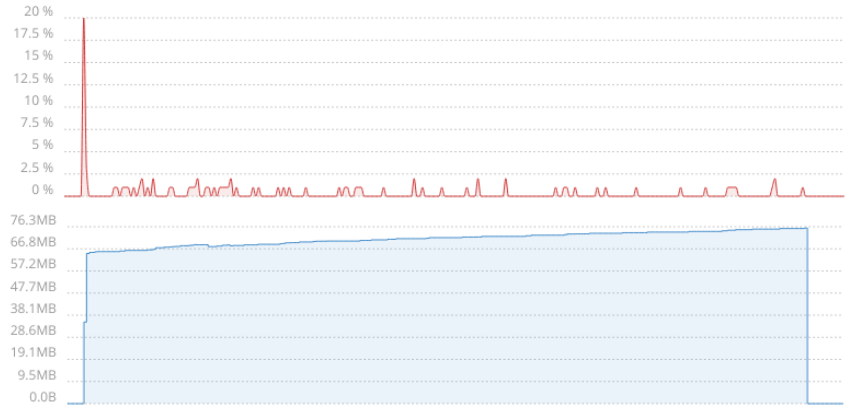
(c) NativeScript



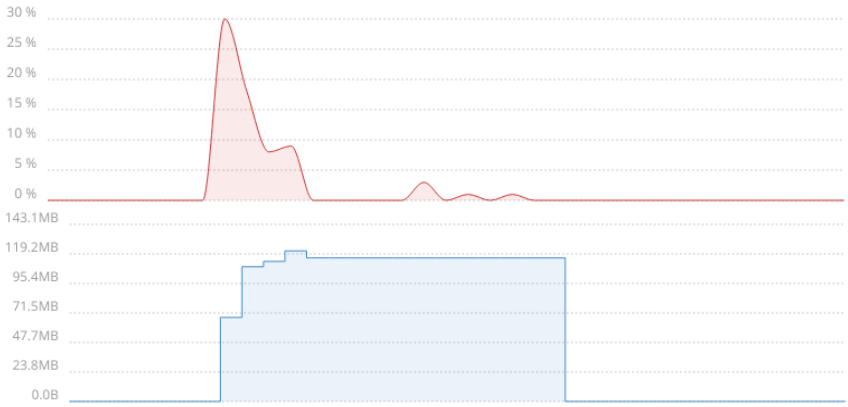
(d) React Native

APÊNDICE C – RESULTADOS GPS ANDROID

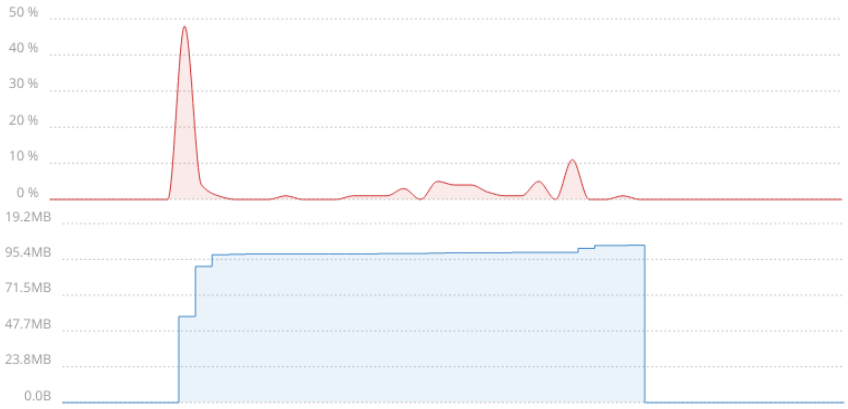
Figura 18 – Relatório do desempenho e uso de memória GPS/Android.



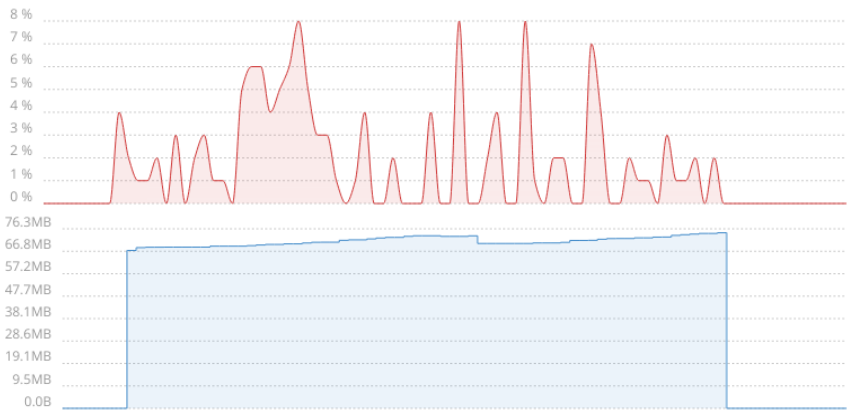
(a) Flutter



(b) Ionic



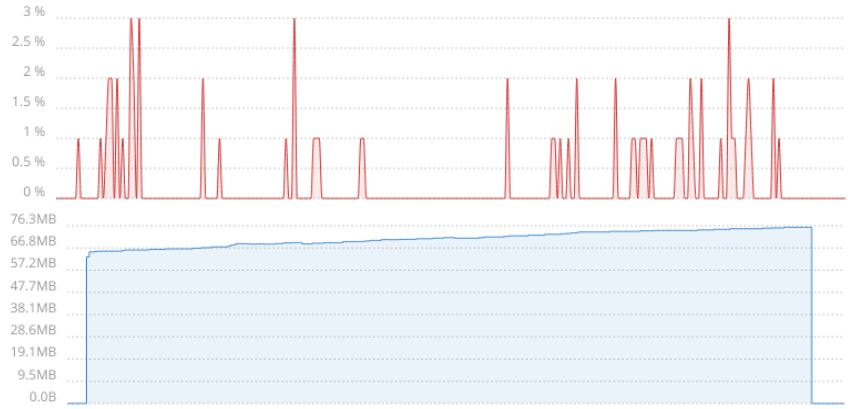
(c) NativeScript



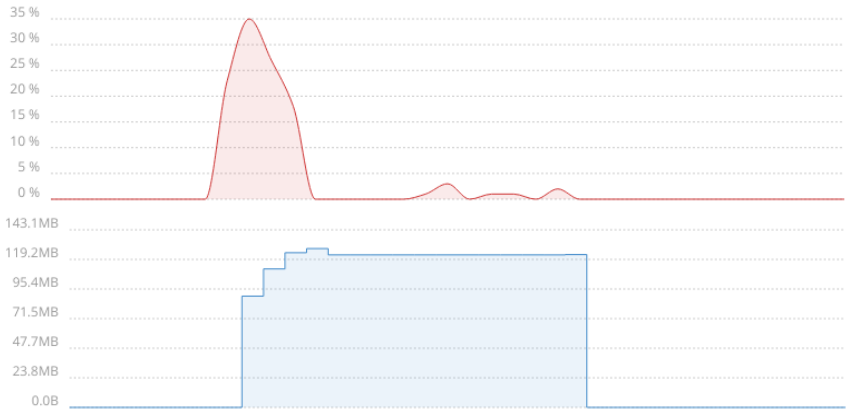
(d) React Native

APÊNDICE D – RESULTADOS ARMAZENAMENTO INTERNO ANDROID

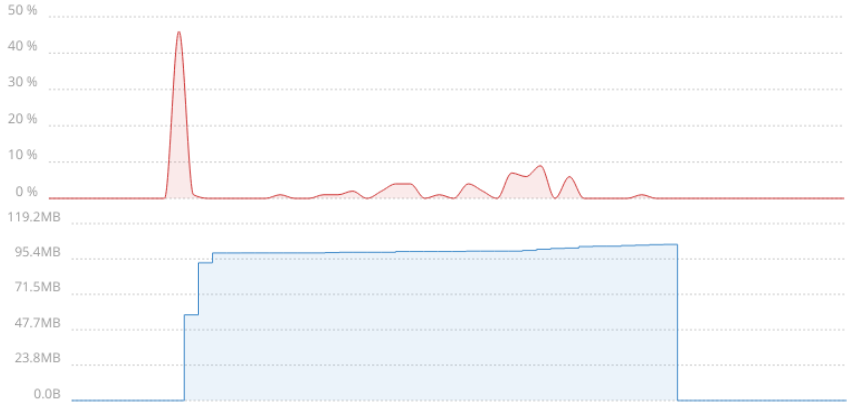
Figura 19 – Relatório do desempenho e uso de memória Armazenamento/Android.



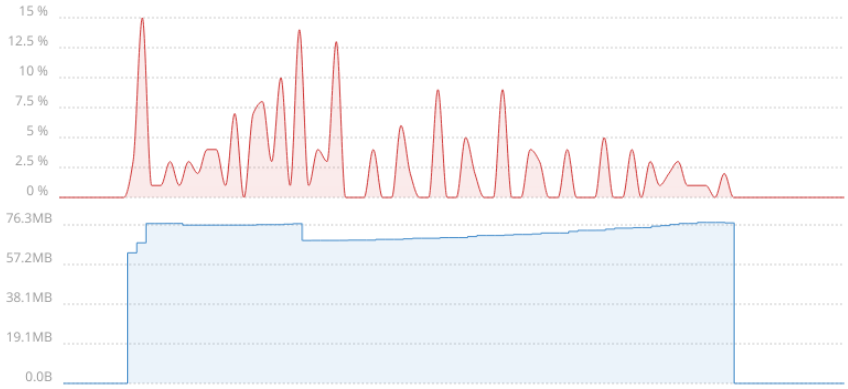
(a) Flutter



(b) Ionic



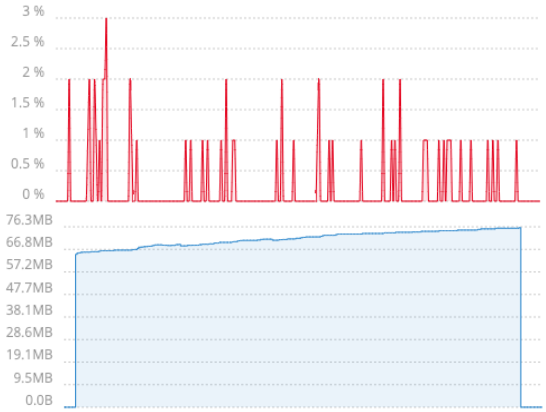
(c) NativeScript



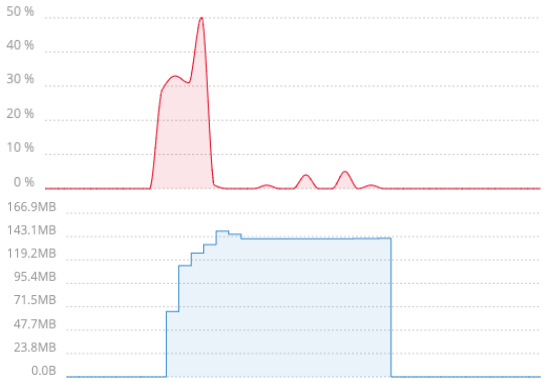
(d) React Native

APÊNDICE E – RESULTADOS BLUETOOTH IOS

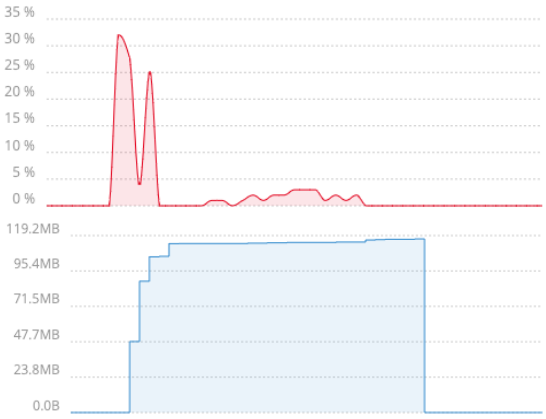
Figura 20 – Relatório do desempenho e uso de memória Bluetooth/iOS.



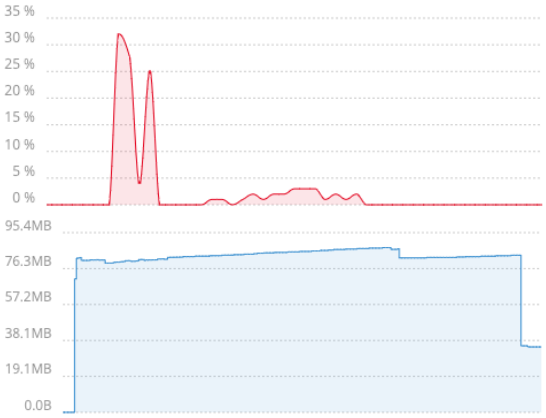
(a) Flutter



(b) Ionic



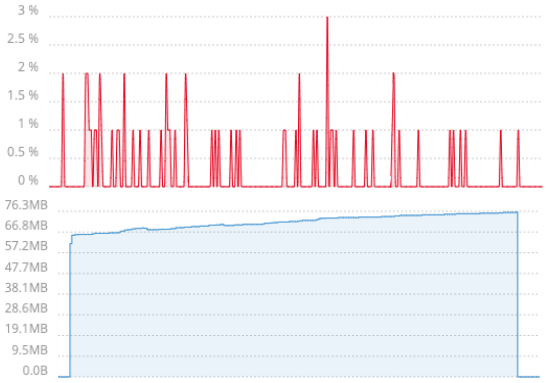
(c) NativeScript



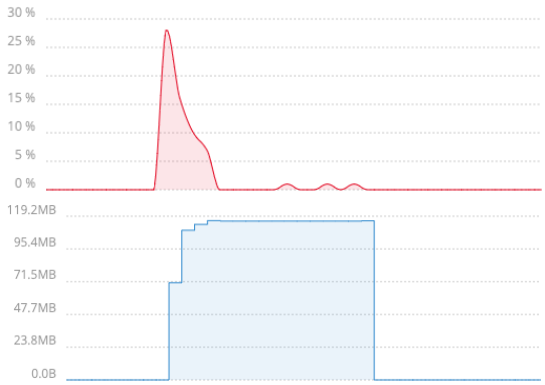
(d) React Native

APÊNDICE F – RESULTADOS CÂMERA IOS

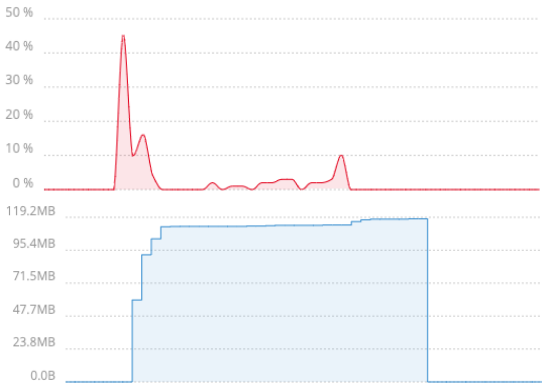
Figura 21 – Relatório do desempenho e uso de memória Câmera/iOS.



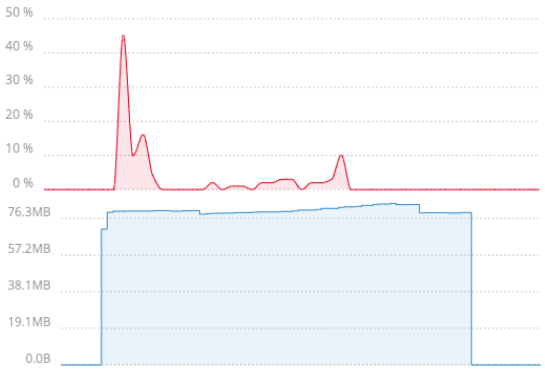
(a) Flutter



(b) Ionic



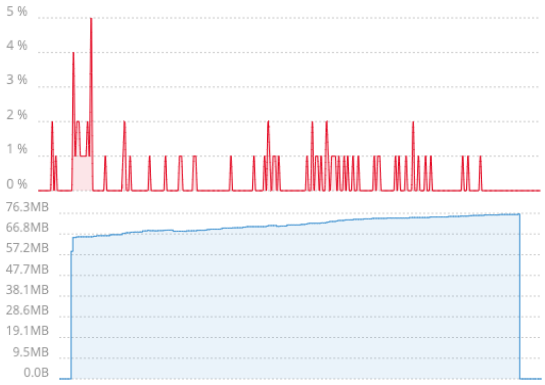
(c) NativeScript



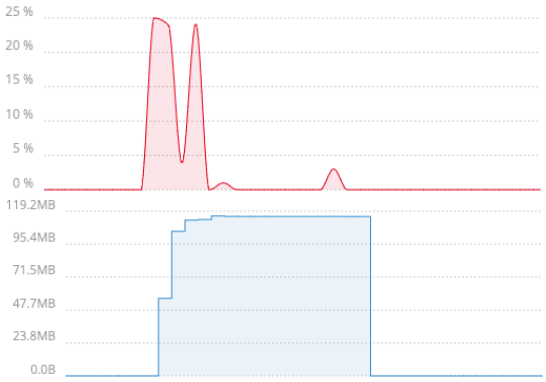
(d) React Native

APÊNDICE G – RESULTADOS GPS IOS

Figura 22 – Relatório do desempenho e uso de memória GPS/iOS.



(a) Flutter



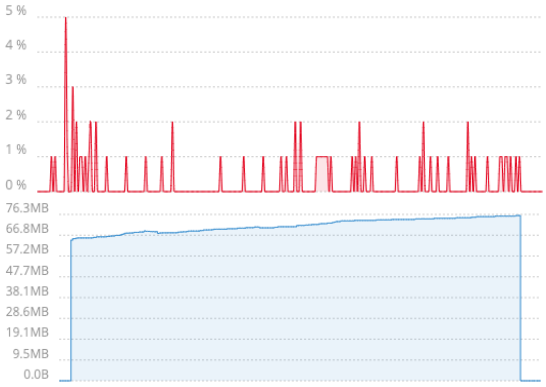
(b) Ionic

(c) NativeScript

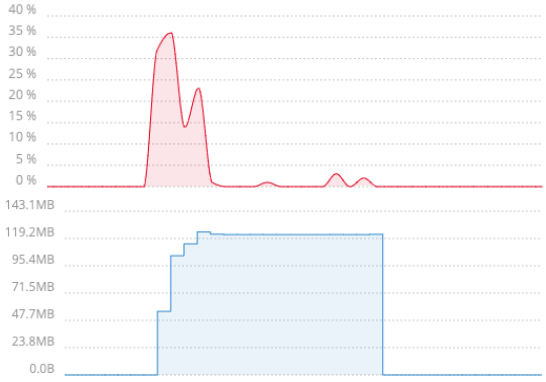
(d) React Native

APÊNDICE H – RESULTADOS ARMAZENAMENTO INTERNO IOS

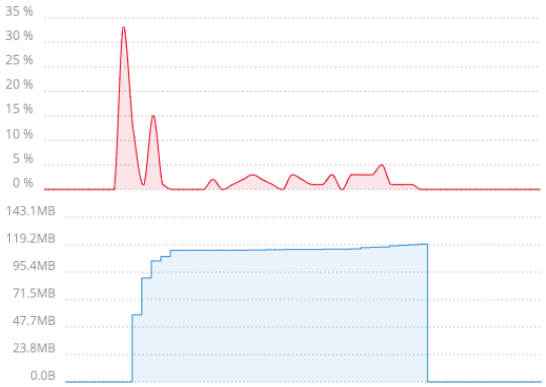
Figura 23 – Relatório do desempenho e uso de memória Armazenamento Interno/iOS.



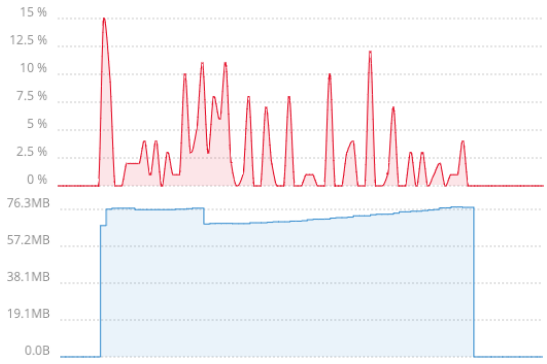
(a) Flutter



(b) Ionic



(c) NativeScript



(d) React Native

APÊNDICE I – QUESTIONÁRIO

Questionário sobre Frameworks de desenvolvimento de aplicações híbridas

Este formulário tem como objetivo colher informações sobre as Frameworks de desenvolvimento de aplicações híbridas. Tais dados são necessários para realizar a comparação entre as Frameworks para o trabalho de conclusão de curso na área de Engenharia de Software da Universidade Federal do Pampa.

OBS.: Os dados obtidos nesta pesquisa serão mantidos no anonimato, sendo necessário informar somente o email para futuras pesquisas ou envio dos Termos e Condições de Uso para aceite caso sofram alterações.

* Required

1. **Email address** *

2. **Termos e Condições de Uso** *
   Você permite que as respostas informadas neste questionário sejam usadas para a presente pesquisa?
   Check all that apply.

   - [ ] Sim, aceito disponibilizar minhas respostas para uso acadêmico.
   - [ ] Não, recuso disponibilizar minhas respostas para qualquer finalidade.

Flutter

3. **Tem experiência acadêmica ou profissional com o framework Flutter?** *
   Mark only one oval.

   - ◯ Sim *Skip to question 3.*
   - ◯ Não *Skip to question 11.*

Questões Flutter

4. **Qual a dificuldade encontrada para o aprendizado do Flutter?** *
   Mark only one oval.

   - ◯ Muito Fácil
   - ◯ Fácil
   - ◯ Mediano
   - ◯ Difícil
   - ◯ Muito Difícil

5. **Quão satisfatória é a didática da documentação do Flutter?** *
Mark only one oval.

- ◯ Muito Satisfatória
- ◯ Satisfatória
- ◯ Indiferente
- ◯ Insatisfatória
- ◯ Muito Insatisfatória

6. **Quão veloz é a construção/compilação da aplicação?** *
Mark only one oval.

- ◯ Muito Rápida
- ◯ Rápida
- ◯ Normal
- ◯ Lento
- ◯ Muito Lento

7. **Quão complexos são os códigos fontes necessários para o Flutter acessar as funcionalidade nativas do dispositivo?** *
Mark only one oval.

- ◯ Muito Simples
- ◯ Simples
- ◯ Normal
- ◯ Complexo
- ◯ Muito Complexo

8. **Quais as vantagens o Flutter proporciona?**

9. **Quais as desvantagens encontradas no Flutter?**

10. **Problemas ou dificuldades encontrados no desenvolvimento de aplicações utilizando Flutter?**

11. **Pontos positivos encontrados no desenvolvimento de aplicações utilizando Flutter?**

Ionic

12. **Tem experiência acadêmica ou profissional com o framework Ionic?** *
Mark only one oval.

- Sim *Skip to question 12.*
- Não *Skip to question 20.*

Questões Ionic

13. **Qual a dificuldade encontrada para o aprendizado do Ionic?** *
Mark only one oval.

- Muito Fácil
- Fácil
- Mediano
- Difícil
- Muito Difícil

14. **Quão satisfatória é a didática da documentação do Ionic?** *
Mark only one oval.

- Muito Satisfatória
- Satisfatória
- Indiferente
- Insatisfatória
- Muito Insatisfatória

15. **Quão veloz é a construção/compilação da aplicação?** *

Mark only one oval.

- Muito Rápida
- Rápida
- Normal
- Lento
- Muito Lento

16. **Quão complexos são os códigos fontes necessários para o Ionic acessar as funcionalidade nativas do dispositivo?** *

Mark only one oval.

- Muito Simples
- Simples
- Normal
- Complexo
- Muito Complexo

17. **Quais as vantagens o Ionic proporciona?**

18. **Quais as desvantagens encontradas no Ionic?**

19. **Problemas ou dificuldades encontrados no desenvolvimento de aplicações utilizando Ionic?**

20. **Pontos positivos encontrados no desenvolvimento de aplicações utilizando Ionic?**

NativeScript

21. **Tem experiência acadêmica ou profissional com o framework NativeScript?** *
Mark only one oval.

- Sim *Skip to question 21.*
- Não *Skip to question 29.*

Questões NativeScript

22. **Qual a dificuldade encontrada para o aprendizado do NativeScript?** *
Mark only one oval.

- Muito Fácil
- Fácil
- Mediano
- Difícil
- Muito Difícil

23. **Quão satisfatória é a didática da documentação do NativeScript?** *
Mark only one oval.

- Muito Satisfatória
- Satisfatória
- Indiferente
- Insatisfatória
- Muito Insatisfatória

24. **Quão veloz é a construção/compilação da aplicação?** *
Mark only one oval.

- Muito Rápida
- Rápida
- Normal
- Lento
- Muito Lento

25. **Quão complexos são os códigos fontes necessários para o NativeScript acessar as funcionalidade nativas do dispositivo?** *

Mark only one oval.

- ◯ Muito Simples
- ◯ Simples
- ◯ Normal
- ◯ Complexo
- ◯ Muito Complexo

26. **Quais as vantagens o NativeScript proporciona?**

27. **Quais as desvantagens encontradas no NativeScript?**

28. **Problemas ou dificuldades encontrados no desenvolvimento de aplicações utilizando NativeScript?**

29. **Pontos positivos encontrados no desenvolvimento de aplicações utilizando NativeScript?**

React Native

30. **Tem experiência acadêmica ou profissional com o framework React Native?** *

Mark only one oval.

- ◯ Sim *Skip to question 30.*
- ◯ Não *Skip to "Obrigado pela Atenção."*

Questões React Native

31. **Qual a dificuldade encontrada para o aprendizado do React Native?** *
Mark only one oval.

- ◯ Muito Fácil
- ◯ Fácil
- ◯ Mediano
- ◯ Difícil
- ◯ Muito Difícil

32. **Quão satisfatória é a didática da documentação do React Native?** *
Mark only one oval.

- ◯ Muito Satisfatória
- ◯ Satisfatória
- ◯ Indiferente
- ◯ Insatisfatória
- ◯ Muito Insatisfatória

33. **Quão veloz é a construção/compilação da aplicação?** *
Mark only one oval.

- ◯ Muito Rápida
- ◯ Rápida
- ◯ Normal
- ◯ Lento
- ◯ Muito Lento

34. **Quão complexos são os códigos fontes necessários para o React Native acessar as funcionalidade nativas do dispositivo?** *
Mark only one oval.

- ◯ Muito Simples
- ◯ Simples
- ◯ Normal
- ◯ Complexo
- ◯ Muito Complexo

35. **Quais as vantagens o React Native proporciona?**

36. **Quais as desvantagens encontradas no React Native?**

37. **Problemas ou dificuldades encontrados no desenvolvimento de aplicações utilizando React Native?**

38. **Pontos positivos encontrados no desenvolvimento de aplicações utilizando React Native?**

Obrigado pela Atenção

Agradeço pelo tempo disponibilizado para responder a este questionário.

Para fins de demonstrar o uso das respostas que foram autorizadas, a pesquisa será enviada em seu endereço de email no final do processo.

Fernando.