UNIVERSIDADE FEEVALE

VINÍCIUS WARKEN

ANÁLISE COMPARATIVA DE FRAMEWORKS DE DESENVOLVIMENTO MULTIPLATAFORMA PARA DISPOSITIVOS MÓVEIS

Novo Hamburgo
2021

VINÍCIUS WARKEN

ANÁLISE COMPARATIVA DE FRAMEWORKS DE DESENVOLVIMENTO MULTIPLATAFORMA PARA DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do grau de Bacharel em
Ciência da Computação pela
Universidade Feevale

Orientador: Juliano Varella de Carvalho

Novo Hamburgo
2021

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

À minha namorara Franciele, pelo auxílio, atenção, incentivo e compreensão enquanto estive ausente para a realização deste trabalho.

À minha mãe Sonia, que me educou e sempre incentivou a dedicação aos estudos e trabalho.

Aos meus sogros Fabiane e Marcelo, que dedicaram seu tempo e carinho, como verdadeiros pais.

Ao professor Juliano, pela dedicação, empenho e cobrança em minha orientação.

RESUMO

O fragmentado mercado de dispositivos móveis com sua enorme variedade de modelos e sistemas operacionais, torna o desenvolvimento nativo de aplicações um esforço desafiador e custoso aos desenvolvedores. Ao longo do tempo, foram surgindo *frameworks* de desenvolvimento multiplataforma que amenizam e vêm resolvendo esse problema. Estas ferramentas possibilitam que as aplicações sejam desenvolvidas apenas uma vez e executadas em dispositivos com arquiteturas e tamanhos diferentes, reduzindo o tempo de desenvolvimento, manutenção e entregando aos desenvolvedores uma interface facilitada de comunicação com os dispositivos. Ao mesmo tempo que facilitam algumas tarefas dos desenvolvedores de *software*, acabam trazendo algumas decisões a serem tomadas. Com o desenvolvimento deste tipo de ferramenta, vários *frameworks* surgiram e cada um possui algumas particularidades que devem ser levadas em consideração antes de serem utilizados. A principal questão está em definir qual *framework* utilizar quando já decidido pelo desenvolvimento multiplataforma. Neste trabalho, foram avaliados e comparados dois *frameworks* de desenvolvimento multiplataforma, React Native e Flutter, a fim de auxiliar na decisão de qual ferramenta deve ser utilizada em um projeto. Para isso, foi desenvolvida uma aplicação nos *frameworks* propostos e com base nesta experiência foram discutidas e apresentadas suas características, bem como construídas recomendações de uso para determinados tipos de aplicações. Através do estudo foi possível identificar que o React Native possui uma breve vantagem frente ao Flutter quando levados em consideração os quesitos levantados por este trabalho. O React Native é mais recomendado para a construção de aplicações seguras e se os desenvolvedores procuram uma comunidade grande e difundida. Já o Flutter recomenda-se para aplicações que se sabe antecipadamente que receberá frequentes manutenções e novas implementações pela sua alta capacidade de manutenibilidade.

Palavras-chave: *cross-platform development*, desenvolvimento de *software*, aplicativos móveis, *framework* de desenvolvimento, tomada de decisão.

ABSTRACT

The fragmented mobile device market with its huge variety of models and operating systems makes native application development a challenging and costly effort for developers. Over time, cross-platform development frameworks have emerged that mitigate and are solving this problem. These tools allow applications to be developed only once and run on devices with different architectures and sizes, reducing development and maintenance time and providing developers with an easy interface to communicate with the devices. While facilitating some tasks for software developers, they end up bringing some decisions to be made. With the development of this type of tool, several frameworks emerged and each one has some peculiarities that must be considered before being used. The main issue is to define which framework to use when already decided for cross-platform development. In this work, two cross-platform development frameworks, React Native and Flutter, were evaluated and compared, in order to assist in the decision of which tool should be used in a project. For this, an application was developed in the proposed frameworks and based on this experience, its characteristics were discussed and presented, as well as building recommendations for use for certain types of applications. Through the study, it was possible to identify that React Native has a brief advantage over Flutter when considering the questions raised by this work. React Native is recommended for building secure applications and if developers are looking for a large and widespread community. Flutter, on the other hand, is recommended for applications that it is known in advance that it will receive frequent maintenance and new implementations due to its high maintainability.

Keywords: cross-platform development, software development, mobile application, development framework, decision-making.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ACM	Association for Computing Machinery
ADB	Android Debug Bridge
API	Application Programming Interface, Interface de Programação de Aplicação
ART	Android Runtime
CSS	Cascading Style Sheets, Folhas de Estilo em Cascata
GCS	Gerência de Configuração de Software
GPS	Global Positioning System, Sistema de Posicionamento Global
HAL	Hardware Abstraction Layer, Camada de Abstração de Hardware
HTML	Hypertext Markup Language, Linguagem de Marcação de Hipertexto
IEEE	Institute of Electrical and Electronics Engineers
IDE	Integrated Development Environment, Ambiente de Desenvolvimento Integrado
JDK	Java Development Kit, Kit de Desenvolvimento Java
PIB	Produto Interno Bruto
ROI	Return On Investment, Retorno Sobre Investimento
SDK	Software Development Kit, Kit de Desenvolvimento de Software

SUMÁRIO

1 INTRODUÇÃO

Smartphones combinam uma variedade de funções como reprodução de mídia, câmera e GPS (Sistema de Posicionamento Global), com avançadas capacidades computacionais. Essas funções possibilitam inovações em sistemas de informação, que frequentemente são chamados de aplicativos. O fragmentado mercado de smartphones que possui uma enorme gama de modelos e importantes plataformas mobile (iOS, Android), torna o desenvolvimento nativo de aplicativos um esforço desafiador e custoso. Logo, o desenvolvimento multiplataforma pode aliviar essa situação. (HEITKÖTTER; HANSCHKE; MAJCHRZAK, 2013)

As limitações das lojas de distribuição dos aplicativos, o investimento de tempo e custo de desenvolvimento, a complexidade das linguagens e tecnologias utilizadas em sua elaboração e manutenção são questões importantes quando se inicia um novo projeto de criação de um aplicativo (SILVA; SANTOS, 2014).

Neste cenário nascem as aplicações chamadas de híbridas/multiplataforma, onde se desenvolve a aplicação apenas uma única vez, podendo aproveitá-la em dispositivos e sistemas operacionais variados. O que torna essa tarefa possível são os *frameworks* de desenvolvimento multiplataforma, que realizam um arranjo do código fonte desenvolvido e o torna capaz de ser instalado e executado em diferentes plataformas, disponibilizando ainda ao desenvolvedor uma interface de comunicação facilitada aos recursos dos dispositivos (PREZOTTO; BONIATI, 2014).

Existem no mercado diversas ferramentas para o desenvolvimento multiplataforma, sendo Xamarin, React Native, Flutter, Ionic, Phonegap e Cordova apenas alguns deles. Estes por sua vez, utilizam-se de linguagens como C#, JavaScript, Dart e/ou HTML e CSS para construir as aplicações. A maior parte destes, focam principalmente na interface gráfica e na interação do usuário com a mesma, além de disponibilizar bibliotecas que facilitam a utilização dos recursos nativos dos dispositivos (BASSETTO, 2019).

Para Mercado et. al (2016), as equipes de desenvolvimento devem estar cientes das capacidades, ou falta delas, sobre as várias alternativas de desenvolvimento híbrido, a fim de identificar a mais apropriada para seu projeto. Porém, avaliar cada uma destas alternativas não é uma tarefa simples. Quem opta pelo desenvolvimento nativo pode dizer que sua aplicação é superior quando comparada às opções de desenvolvimento multiplataforma, utilizando como base a experiência do usuário e características de performance. Por outro lado,

desenvolvedores que empregam práticas de desenvolvimento multiplataforma, irão argumentar que o retorno de investimento nesse tipo de estratégia é consideravelmente maior. Mesmo com as duas abordagens tendo sua relevância, é preciso avaliar qual, entre as duas, compensa em determinado tipo de projeto ou qual deve ser utilizada por uma equipe de desenvolvimento (MERCADO; MUNAIAH; MENEELY, 2016).

Mesmo optando pelo desenvolvimento multiplataforma, ainda há uma questão em aberta, definir qual *framework* de desenvolvimento utilizar para conseguir tirar proveito dos ganhos que se esperam nesse tipo de ferramenta. Existem diversas tecnologias desse tipo disponíveis no mercado, criando inclusive uma certa confusão entre os desenvolvedores em relação a qual delas considerar o uso e quais descartar.

A popularidade destas ferramentas demanda uma pesquisa mais profunda nessa área e os desenvolvedores esperam informação clara a respeito delas antes de optarem pela utilização de alguma delas. É o que ocorre com boa parte das aplicações que necessitam estar disponíveis em qualquer dispositivo e não se pode investir no desenvolvimento nativo para cada um deles.

Dhillon et al. (2014) apresenta um método de comparação de performance e um processo reutilizável para implementar testes e avaliar *frameworks* de desenvolvimento multiplataforma e as aplicações produzidas por eles. Majchrzak et al. (2017) desenvolveu um estudo a respeito de três destas ferramentas, React Native, Ionic e Fuse, apresentando uma análise de fácil compreensão. A análise se baseou em um caso real de desenvolvimento, que possibilitou ao estudo identificar e construir recomendações a respeito das mesmas.

Para a realização destas avaliações e comparações entre *frameworks*, Rieger et al. (2019) produziu um modelo base que pode ser utilizado nos novos estudos para avaliar diversos critérios. Estes critérios podem receber pesos ponderados a fim de disponibilizar uma personalização ao modelo e ao mesmo tempo atender a variedade de estruturas possíveis nas aplicações onde os *frameworks* serão utilizados. Em seu estudo, os autores também apontam que o desenvolvimento multiplataforma teve muito progresso nos últimos anos, mas que, ao mesmo tempo, estão sempre inovando e encontrando novos desafios a serem resolvidos.

Este trabalho, portanto, propôs, por meio da utilização de um projeto real de desenvolvimento de uma aplicação multiplataforma, avaliar e comparar dois dos *frameworks* populares, com o objetivo de auxiliar equipes de desenvolvimento de *software* a decidirem qual ferramenta utilizar em cada um dos projetos que iniciarem. Foram definidos parâmetros e

características que poderão ser utilizados pelas equipes como base e guia em sua tarefa de decisão.

Este trabalho está dividido em oito capítulos. No primeiro capítulo é apresentada a introdução ao assunto do trabalho, assim como a motivação que levou ao seu desenvolvimento. No capítulo dois são apresentados os objetivos gerais e específicos definidos para o projeto. No terceiro capítulo é realizada uma contextualização a respeito do desenvolvimento de aplicativos para dispositivos móveis. No quarto capítulo são elaboradas as referências bibliográficas. No capítulo seguinte é desenvolvida, aplicada e analisada uma pesquisa quantitativa para apoiar o desenvolvimento deste trabalho. Nos capítulos 5 e 7 é apresentado o processo de desenvolvimento dos aplicativos com cada *framework* e sua análise, com base nos parâmetros e métricas definidas anteriormente no trabalho. No oitavo e último capítulo, são desenvolvidas algumas considerações acerca do desenvolvimento deste estudo, sintetizando todo o desenvolvimento já efetuado, resultados, limitações e trabalhos relacionados futuros.

2 OBJETIVOS

Abaixo são apresentados o objetivo geral e os objetivos específicos que se espera alcançar com o desenvolvimento deste trabalho.

2.1 OBJETIVO GERAL

Avaliar e comparar dois *frameworks* de desenvolvimento multiplataforma, a fim de auxiliar na decisão de qual ferramenta utilizar em cada projeto.

2.2 OBJETIVOS ESPECÍFICOS

- Pesquisar trabalhos relacionados, a fim de auxiliar na definição dos parâmetros de comparação;
- Entrevistar desenvolvedores de *software*, a fim de auxiliar na definição dos parâmetros de comparação;
- Definir os parâmetros de comparação que serão avaliados nos *frameworks*;
- Definir os *frameworks* que serão avaliados;
- Construir uma aplicação *mobile* utilizando os *frameworks* definidos;
- Discutir as características positivas e negativas em relação a cada um dos parâmetros definidos;
- Demonstrar um comparativo técnico entre os *frameworks*, utilizando como métricas os parâmetros estipulados;
- Recomendar com base nas análises e comparações feitas, o *framework* ideal em determinados tipos de aplicações;
- Apresentar as aplicações concluídas demonstrando as particularidades notadas durante o processo de desenvolvimento.

3 DESENVOLVIMENTO MOBILE

Este capítulo apresenta uma introdução a história dos dispositivos portáteis, expondo a maneira com que foram popularizando-se ao longo do tempo, os motivos pelos quais estes evoluíram tão rapidamente e mostrando números que demonstram esse crescimento acelerado.

3.1 HISTÓRIA

A comunicação é um elemento essencial para a sobrevivência humana dentro de uma comunidade, tanto na comunicação individual quanto na comunicação social. As mídias de comunicação tentam estimular os eventos de socialização entre as pessoas através da imprensa, do rádio, da televisão, do cinema, entre outros, nas últimas décadas do século XX. Nesse progresso da socialização, o telefone também passou a estar presente, primeiramente o telefone fixo e após isso o telefone móvel e multimídia que se torna cada vez mais popular nos dias de hoje. Assim sendo, novas tecnologias e aplicações para este tipo de dispositivo surgem para que os mesmos continuem em ascensão dentro da comunicação pessoal (BARNES; MEYERS, 2011).

Barnes e Meyers (2011) apontam ainda que a principal função de um telefone seja a comunicação por voz, como no telefone convencional primeiramente inventado, embora o desenvolvimento rápido dessa ferramenta fez com que diversas outras funções fossem adicionadas a ele, como a câmera, o acesso à internet, a reprodução de vídeos e músicas e o GPS. Além da evolução tecnológica, também deve-se levar em consideração a evolução física destes dispositivos, onde o peso é um dos principais fatores. Essa evolução em conjunto com a miniaturização dos componentes eletrônicos fez com que esse dispositivo se tornasse um objeto de uso pessoal, inclusive podendo hoje ser utilizado em forma de um relógio no pulso (BARNES; MEYERS, 2011).

A partir de então os dispositivos móveis passaram a crescer num ritmo alucinante. Na última década, eles alcançaram a incrível marca da indústria com o desenvolvimento mais rápido do mundo, e em 2009, a indústria de celulares ultrapassou a barreira de US$ 1 trilhão em receita anual, chegando também a se tornar uma das maiores indústrias do planeta. A indústria de dispositivos móveis já é maior que as mídias de transmissão (rádio e televisão juntos), largamente na frente também das indústrias de informática e tecnologia de informação, música, filmes, videogames e revistas, livros e jornais. O mercado movimentado pelos celulares se encarrega por cerca de 2% do total do PIB do mundo, igualando-se ao

seleto grupo do US$ 1 trilhão de receita anual, que comporta setores gigantes como o automotivo, habitacional e de construção, alimentício, militar, e bancário (BRUCK; RAO, 2013).

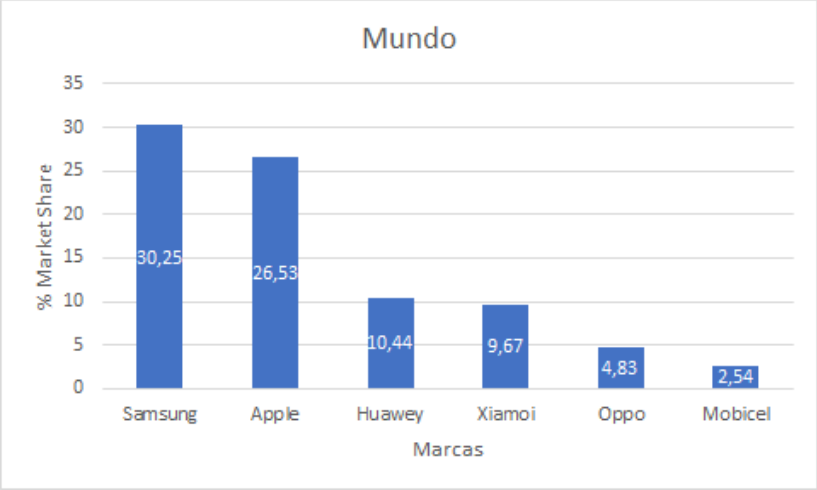
Contudo, Bruck e Rao (2013) observam que enquanto as indústrias do clube de US$ 1 trilhão são centenárias - ou até milenares - a indústria dos dispositivos móveis tem menos de 50 anos de existência. O primeiro movimento comercial de telecomunicações e telefonia do mundo surgiu em Tóquio no Japão em 1979, quando a Nippon Telegraph and Telephone lançou seu serviço de telecomunicação celular ao mercado. Este mercado foi o que cresceu mais rápido na história quando levada em consideração a receita anual. Esse movimento impressiona e não é de se espantar que conheçamos histórias de sucesso avassaladoras nesse meio como a de empresas como a Apple e a Google.

3.2 MERCADO ATUAL

Segundo a BankMyCell (2020) e a Statista (2019), o número de smartphones no mundo inteiro ultrapassa 3,5 bilhões de dispositivos, o que equivale a 44,87% da população mundial. O número de usuários de smartphones no mundo aumentou em 40% de 2016 a 2020, sendo que em 2016 havia 2,5 bilhões de dispositivos e em 2020, atingindo a marca dos 3,5 bilhões. A Statista também prevê que o crescimento seguirá substancial, estimando que até 2021 este número já poderá alcançar os 3,8 bilhões de smartphones ativos. O Brasil, no mês de setembro de 2020, conta com 228,3 milhões de celulares ativos, o que corresponde a uma densidade de 107,53 celulares/100 habitantes. O número de adições de novos celulares nesse montante chega a 1 milhão (TELECO, 2020).

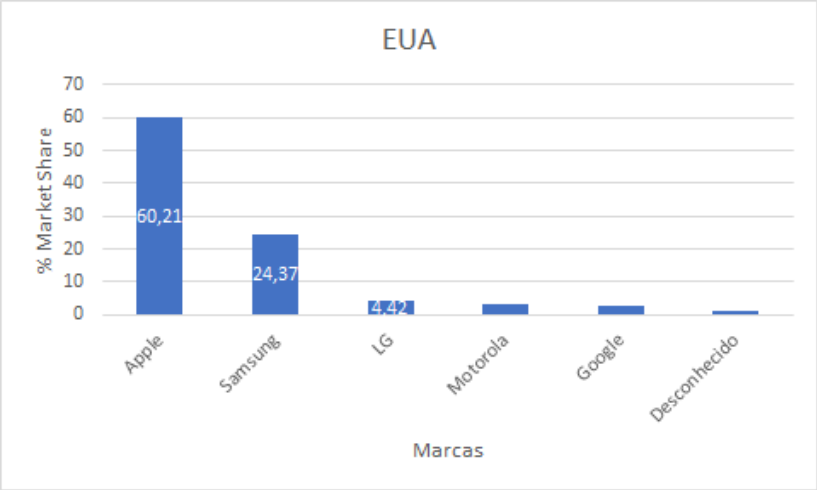
Nas Figuras 1, 2 e 3 constam o percentual de *market share* referente as maiores marcas de *smartphones* no mundo, nos Estados Unidos e no Brasil, respectivamente.

Figura 1 – *Market Share* das maiores marcas de *smartphones* do mundo



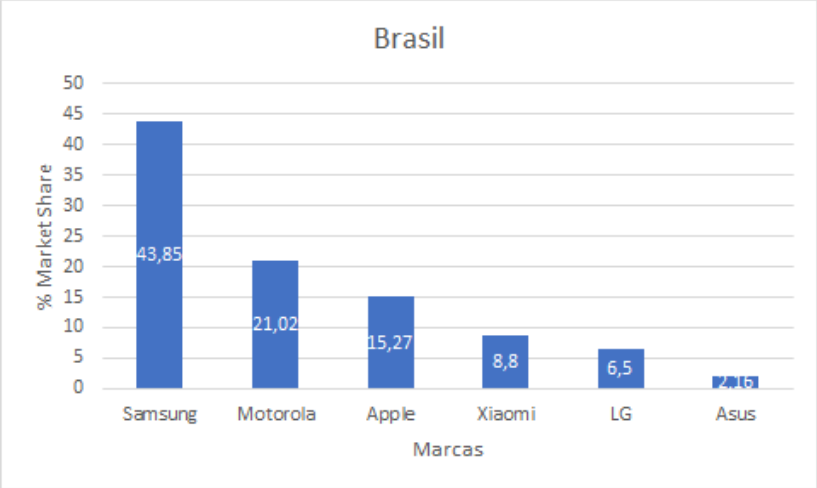
Fonte: StatCounter (2020)

Figura 2 – *Market Share* das maiores marcas de *smartphones* do EUA



Fonte: StatCounter (2020)

Figura 3 – *Market Share* das maiores marcas de *smartphones* do Brasil



Fonte: StatCounter (2020)

3.3 ARQUITETURAS DE DESENVOLVIMENTO

O crescente desenvolvimento da indústria de dispositivos móveis após o ano 2000, passou a exigir um desempenho mais elevado dos sistemas operacionais e as grandes empresas do ramo de tecnologia tiveram uma importante atribuição. Por ser um mercado novo e em alta ascensão, Apple, Microsoft, Nokia, Symbian e Google investiram no desenvolvimento de seus sistemas operacionais móveis para se encaixar no novo mercado. Na primeira década dos anos 2000, os principais sistemas operacionais eram Android, BlackBerry, iOS, Symbian e o Windows Fone (PALMIERI et al., 2012).

O número de sistemas operacionais passou a cair drasticamente nos últimos anos, resultando num duopólio, dividido entre Apple com o iOS e Google com o Android. Segundo o New York Times (THE NEW YORK TIMES, 2016), a participação de mercado da Blackberry caiu para a casa de um dígito na América do norte e na Europa há bastante tempo. Em 2013 a empresa anunciou inclusive que introduziria o BlackBerry 10, uma nova versão do sistema operacional, mas que não emplacou, e a utilização do Android em alguns dos novos dispositivos da marca.

Alguns anos depois, foi a vez da Microsoft anunciar a descontinuidade do Windows 10 Mobile. Com apenas atualizações de segurança sendo lançadas desde 2017, a empresa informou que a última versão seria introduzida em dezembro de 2019 e que encerraria o suporte ao sistema. A Microsoft inclusive passou a sugerir a substituição para dispositivos embarcados com os sistemas com iOS e Android (THE VERGE, 2019).

3.3.1 Apple iOS

O iOS é o sistema operacional mobile desenvolvido pela Apple Inc., e que apenas é utilizado nos dispositivos da mesma empresa. Ele é baseado no Unix, assim como o Mac OS X, outro sistema operacional da marca. Um dos grandes diferenciais deste *software* é que, diferentemente da maior parte dos concorrentes atuais e anteriores, é de que o sistema e o produto que o embarca são vendidos e mantidos pela mesma empresa. Não há a possibilidade de utilizar nem instalar o iOS em um dispositivo diferente do qual foi projetado (NOVAC et al., 2017).

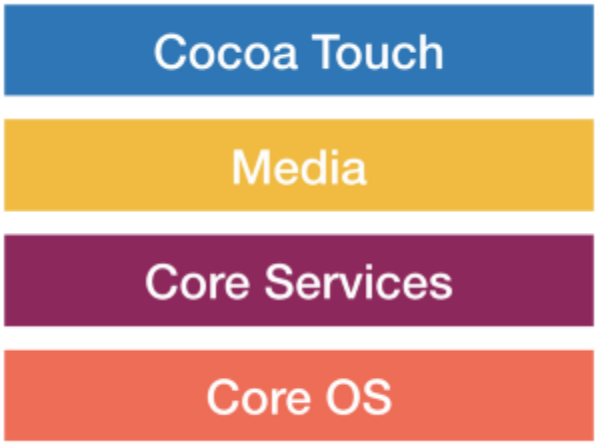
Novac et al. (2017) explicam que o iOS possui quatro níveis de abstração, sendo: o *kernel* do sistema operacional, o núcleo de serviços, o nível de multimídia e a interface de usuário. O *kernel* do sistema é o que mais se aproxima do *hardware* e do *kernel* do Unix.

Neste nível são utilizadas APIs em C, portanto, não orientadas a objetos. Já no segundo nível, o núcleo de serviços, há APIs orientadas a objetos. Ela é responsável por disponibilizar serviços básicos como a manipulação de textos, contatos e funcionalidades do sistema, GPS, acelerômetro e giroscópio. O terceiro nível é o responsável por transferências multimídias: som e vídeo.

Por último, o Cocoa Touch, é o nível de interface de usuário. É neste nível da arquitetura em que 90% do trabalho dos desenvolvedores acontece quando desenvolvem uma aplicação. Também é neste nível que são disponibilizadas as infraestruturas básicas das aplicações como: paralelismo, touchscreen e notificações. Para o desenvolvimento das aplicações são utilizadas as linguagens Objective-C e Swift (linguagem desenvolvida pela própria Apple em 2014, exclusivamente para o iOS). A nova linguagem foi criada para proteger e tornar o código escrito pelos desenvolvedores mais limpo e sucinto. Por ter controle sobre o sistema operacional e sobre os dispositivos que o executam, não é necessária a execução sobre uma máquina virtual, assim como em outros sistemas operacionais (NOVAC et al., 2017).

Na Figura 4 é exemplificado em um esquema básico, a estrutura de 4 níveis do sistema operacional da Apple:

Figura 4 – Estrutura de 4 níveis do sistema operacional iOS da Apple



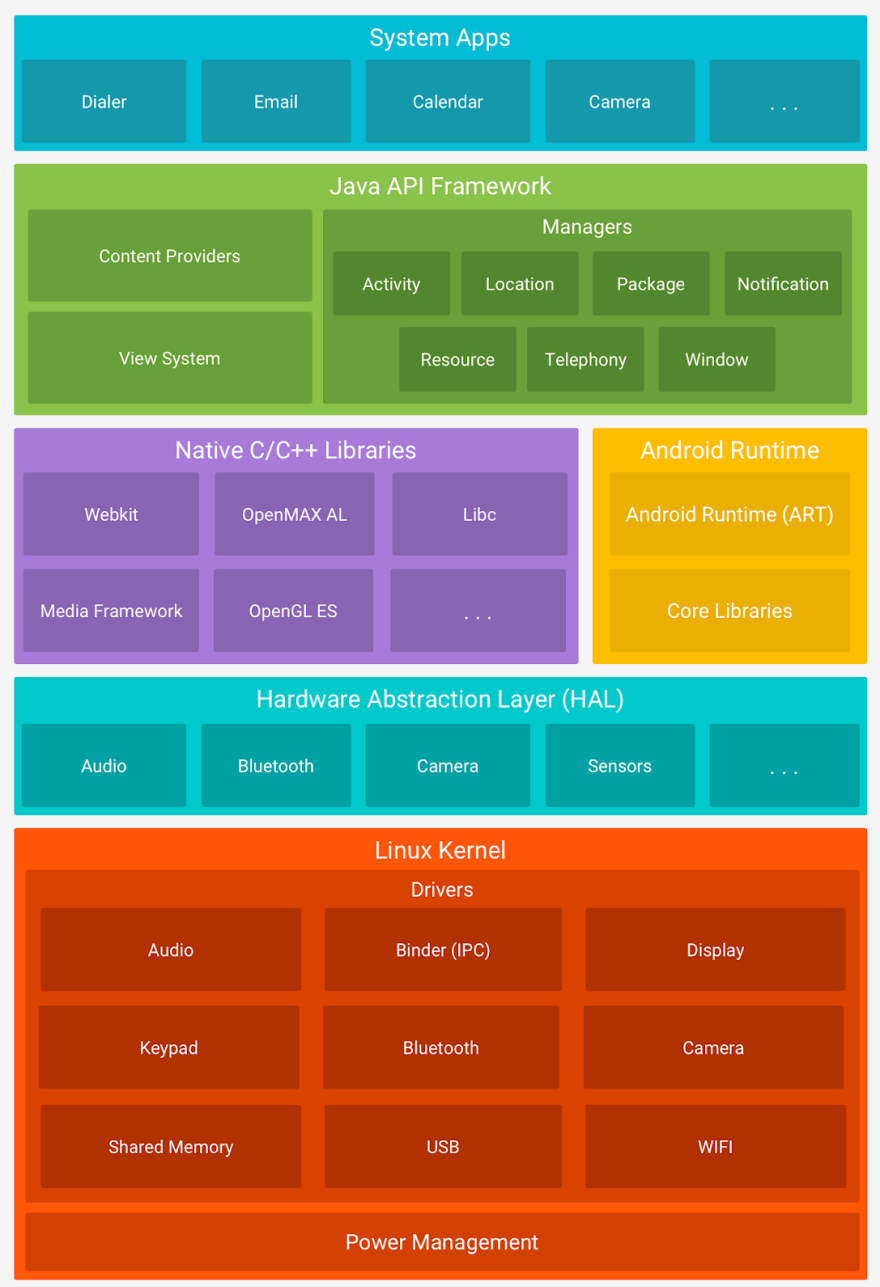
Fonte: Medium (2018)

3.3.2 Google Android

O Android é o sistema operacional mantido pela Google e foi criado com base em Linux, de código aberto, que é desenvolvido para diversos tipos, tamanhos e marcas de

dispositivos. A Figura 5 apresenta um esquema que mostra de maneira geral as camadas de *software* do Android.

Figura 5 – Estrutura de camadas do sistema operacional Android da Google



Fonte: Google (2020)

A base do sistema é o *kernel* do Linux, e com isso, os módulos superiores como o *Android Runtime* (ART) confiam ao Linux a execução de funcionalidades de baixo nível. Outra vantagem na utilização do *kernel* do Linux é a questão de segurança, pois os fabricantes desenvolvem drivers para um *kernel* já conhecido. A camada de abstração de *hardware* (HAL) fornece uma interface padronizada e facilitada a API de mais alto nível do Java para acessar as funcionalidades do *hardware* dos dispositivos. Os recursos do sistema operacional estão disponíveis para uso através das linguagens Java e mais recentemente, o Kotlin. Através

delas é possível o desenvolvimento de interfaces de usuário, gerenciamento de recursos, notificações e atividades, além de permitir o acesso a aplicativos nativos do próprio sistema.

3.4 MULTIPLATAFORMA X NATIVO

Nesta seção são apresentados os conceitos de desenvolvimento nativo e de desenvolvimento multiplataforma. Além de conceituadas as duas abordagens de desenvolvimento, são expostas as características relacionadas ao desenvolvimento e usabilidade, assim como suas vantagens e desvantagens.

3.4.1 Desenvolvimento Nativo

O conceito de um aplicativo nativo pode ser definido como uma aplicação que foi pensada e desenvolvida para apenas um tipo específico de plataforma. As plataformas são um conjunto de tecnologias que formam uma determinada arquitetura como: o sistema operacional, as linguagens de programação e as IDEs (*Integrated Development Environment*) (SILVA; SANTOS, 2014). Como mostrado na seção acima, as duas principais plataformas são o Android (Google) e o iOS (Apple Inc.) e estas utilizam uma linguagem de programação específica para a construção de suas aplicações. Através de APIs disponibilizadas pelos próprios idealizadores das plataformas, os desenvolvedores são capazes de acessar funcionalidades que são concedidas pelo sistema operacional, como o GPS, armazenamento, câmera, áudio, entre outros (SAMBASIVAN et al., 2011).

Para que se possa construir uma aplicação nativa, é necessário que se possua conhecimento específico a respeito da linguagem e arquitetura para a qual se deseja construir e executar a aplicação. Existem vantagens e desvantagens nesse tipo de abordagem, e a principal desvantagem em relação a ela é o fato de a mesma somente poder ser executada na plataforma para a qual foi concebida. Isto acarreta mais alguns pontos negativos que são o aumento do custo, tempo e esforço para conseguir disponibilizar o mesmo serviço em plataformas diferentes (SAMBASIVAN et al., 2011).

Em contrapartida, os aplicativos nativos apresentam uma ótima experiência de uso aos utilizadores, pois disponibilizam acesso a todos os recursos nativos disponíveis nos aparelhos, e assim, permitem uma experiência mais profunda por parte dos seus usuários. As interfaces construídas nativamente, são iguais as utilizadas pelo próprio sistema operacional, tornando a navegação e o uso padronizados, apresentando um ambiente intuitivo aos usuários. Além disso, aplicativos nativos apresentam bom desempenho em suas plataformas, pois não

requerem qualquer tipo de interpretação de código durante o seu uso. Este talvez seja um dos pontos mais fortes deste tipo de abordagem. Finalmente, outra vantagem é que as comunidades responsáveis pelas diferentes versões dos *softwares*, ferramentas e linguagens são os próprios fornecedores do sistema operacional e portanto, assim que lançadas, as novas funcionalidades disponibilizadas podem ser imediatamente utilizadas, sem nenhum tempo de espera para que tecnologias paralelas se adequem as novidades (SILVA; SANTOS, 2014).

3.4.2 Desenvolvimento Multiplataforma

Considerando que hoje existem dispositivos variados com sistemas operacionais distintos e arquiteturas de desenvolvimento diferentes entre si, possuir uma maneira de desenvolver uma aplicação para qualquer dispositivo, ou para uma boa parte deles, é um dos maiores desafios da computação móvel atual. Várias ferramentas de desenvolvimento multiplataforma surgiram nos últimos anos, juntamente com a grande variedade de sistemas e dispositivos diferentes, cada um com seus propósitos e características próprios (SILVA; SANTOS, 2014).

Para Palmieri et al. (2012), os principais benefícios trazidos por essas ferramentas foram, redução de código e sua complexidade, redução do tempo de desenvolvimento e custo de manutenção das aplicações e consequentemente a facilidade de desenvolvimento e participação no mercado. A maior das vantagens trazidas pela abordagem multiplataforma é a possibilidade de haver um ROI (retorno sobre investimento) mais elevado, pois desenvolvendo a aplicação uma única vez, é possível empacotar a aplicação como uma aplicação nativa e disponibilizá-la para todas as plataformas que o *framework* suporta.

Hartmann et al. (2011) conceituam os principais tipos de ferramentas de desenvolvimento multiplataforma, apontando que em cada uma delas podem existir limitações e atendimento a diferentes demandas. As categorias de ferramentas de desenvolvimento multiplataforma são:

3.4.2.1 Aplicativos Web Puros

Estratégia que se torna-se cada vez mais popular e que executa suas aplicações nos próprios navegadores dos dispositivos para quais foram desenvolvidos.

3.4.2.2 Aplicativos Web Híbridos

Abordagem que explora uma funcionalidade da maior parte das plataformas existentes, onde se acessa através de programação por código uma instância nativa do

navegador do dispositivo em questão, chamada de *WebView*. Esta alternativa não difere muito dos aplicativos Web puros, mas ocultam os menus e abas padrão dos navegadores, dando a impressão de uma execução mais aproximada da nativa.

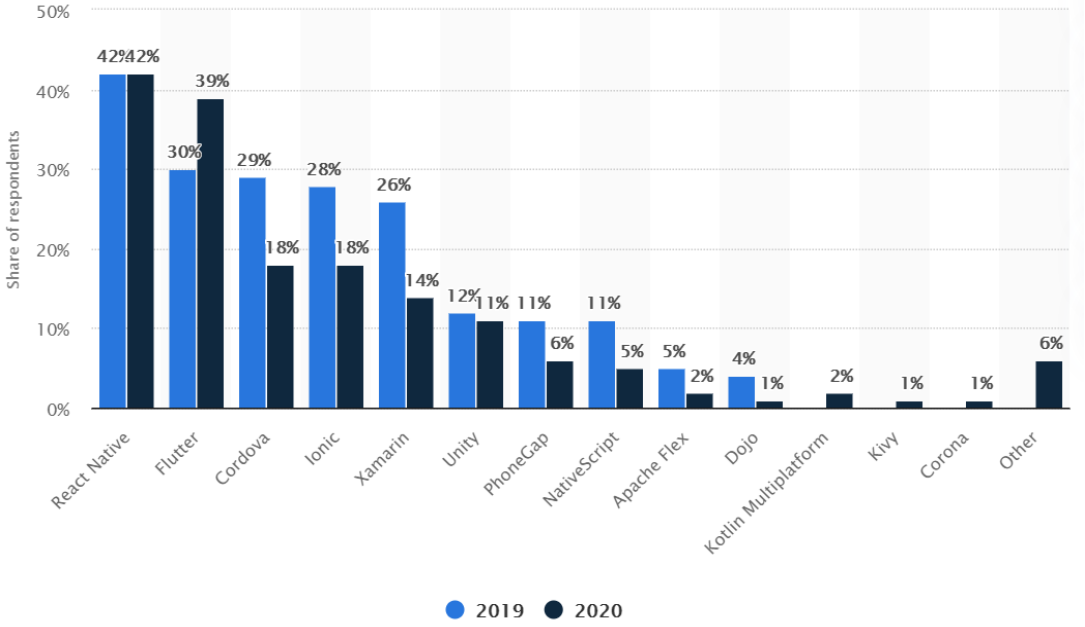
3.4.2.3 Aplicativos de Compilação Cruzada

A última alternativa consiste na conversão de código escrito na linguagem específica do *framework* que está sendo utilizado, para código nativo. As aplicações podem ser construídas do início ao fim, incluindo interfaces de interação com o usuário, armazenamento e manipulação de dados e aplicação das regras de negócio e, ao final, serem empacotadas em tempo de compilação para as plataformas integradas pelo *framework.* A conversão de código da ferramenta para código nativo dá o nome ao Compilador Cruzado.

3.5 FRAMEWORKS DE DESENVOLVIMENTO MULTIPLATAFORMA

Com a popularização dos dispositivos móveis e com o entendimento de que o desenvolvimento nativo não era a melhor solução para todas as aplicações móveis, foram surgindo diversos *frameworks* de desenvolvimento multiplataforma. Cada um desses *frameworks* se dispôs a resolver determinados problemas e de maneiras distintas. De acordo com a pesquisa realizada pela Statista (2019), a maior parte dos desenvolvedores de aplicações *mobile* que utilizam *frameworks* multiplataforma, desenvolvem suas aplicações em sua maioria com React Native, Flutter, Cordova, Ionic e/ou Xamarin, respectivamente. Nesta pesquisa foram entrevistados ao todo, 19.696 pessoas do mundo todo, e na Figura 6 é apresentado um gráfico onde consta o percentual de desenvolvedores que utilizam cada um dos *frameworks*.

Figura 6 – Percentual de desenvolvedores X *Framework* de desenvolvimento multiplataforma



Fonte: Statista (2020)

A pesquisa também revelou que um terço dos desenvolvedores entrevistados utiliza tecnologias de desenvolvimento multiplataforma, enquanto o restante apenas utilizou as abordagens nativas disponibilizadas pelas plataformas. Para complementar e validar a pesquisa feita pela Statista, a Figura 7 mostra um gráfico que reforça que o interesse nos últimos 12 meses foi mais relevante para Flutter e React Native no Brasil.

Figura 7 – Interesse de pesquisa no Brasil nos últimos 12 meses no buscador da Google

Fonte: Google (2020)

3.5.1 Flutter

O Flutter é o kit de ferramentas para interfaces de usuário desenvolvido pela Google para a construção de aplicações nativas *mobile, web* e *desktop,* a partir de um único código base. A empresa disponibiliza tutoriais que auxiliam desenvolvedores advindos de outras tecnologias do mercado, o que facilita e diminui a curva de aprendizado. Caracteriza-se especialmente por permitir a construção de interfaces nativas de maneira rápida e flexível, com renderizações velozes, podendo ser comparadas a performance de aplicações nativas. O *framework* incorpora todas as características específicas que diferenciam as plataformas de desenvolvimento, como rolagem, navegação, ícones e fontes, através da linguagem Dart, também desenvolvida e disponibilizada pela Google. A ferramenta, mesmo sendo nova, já se integra ao mercado, sendo utilizada inclusive por grandes empresas como a própria Google em seus produtos, BMW, Grupo Alibaba, entre outras (FLUTTER, 2020).

3.5.2 React Native

O React Native possibilita a criação de aplicações nativas para Android e iOS utilizando o React, uma biblioteca JavaScript para a construção de interfaces já consolidada na comunidade de desenvolvimento *web.* Aplicações React Native podem ser construídas do zero ou integradas a aplicações já existentes, facilitando a iniciação com a plataforma. Da mesma forma que o Flutter, as primitivas React renderizam interfaces de usuário nativas, o que significa que utilizam as mesmas APIs que aplicações totalmente nativas. A ferramenta permite a criação de componentes que podem ser compartilhados entre distintas plataformas, fazendo com que a mesma equipe possa manter aplicações diferentes, que utilizam os mesmos componentes. Há pouco mais tempo no mercado, o React Native também já ocupa seu lugar em empresas influentes na construção de tecnologias de *software* como, o Facebook (seu desenvolvedor), Instagram, Skype, Pinterest, Tesla, Wix e outros (REACT NATIVE, 2020).

3.5.3 Xamarin

Além dos dois *frameworks* já apresentados, há ainda o Xamarin, desenvolvido e mantido pela Microsoft, que o define como uma plataforma para construção de aplicações Android e iOS com .NET e C#. Além destas plataformas, o Xamarin também possibilita construir aplicações para watchOS, macOS, aplicativos Windows com acesso nativo aos recursos dos dispositivos. A Microsoft disponibiliza treinamentos, com vídeos cursos para o aprendizado da plataforma. Grandes empresas como a UPS, BBVA e a Alaska Airlines

optaram por utilizar a ferramenta, que garante performance e acesso nativo aos recursos dos dispositivos de maneira facilitada. O projeto é *open-source*, e portanto, já conta com uma forte comunidade formada por mais de 60000 contribuidores de mais de 3700 empresas. Assim como o .NET, o Xamarin é gratuito e por isso, não exige que sejam adquiridas licenças e podem ser usadas para uso comercial, diferentemente da maior parte das ferramentas da Microsoft (XAMARIN, 2020).

3.5.4 Considerações Finais

Os três *frameworks* apresentados acima são desenvolvidos por empresas sólidas e com forte participação no mercado de desenvolvimento de *software*. Portanto, as ferramentas dispõem de segurança e garantia de que continuarão recebendo atualizações, correções e novas funcionalidades. Além de serem desenvolvidas por grandes empresas, elas também já são utilizadas por gigantes do mercado mundial, como BMW, UPS e Tesla, o que assegura e incentiva a manutenção destas ferramentas.

As plataformas suportadas por eles, compreendem a maior parte do mercado de dispositivos móveis, os três têm a capacidade de acessar os recursos nativos dos aparelhos e de entregar performance aproximada da nativa, fazendo com que estes sejam substitutos a altura da abordagem nativa. Os três enquadram-se na categoria de aplicativos de compilação cruzada, pois tem seus códigos escritos originalmente em uma linguagem, e transformados após, pelos *frameworks*, para a linguagem nativa da plataforma para a qual se está compilando.

4 TRABALHOS RELACIONADOS

Foi elaborada uma pesquisa bibliográfica para sustentar a pesquisa científica realizada, trazendo dados e informações a respeito de comparações já efetuadas entre *frameworks* de desenvolvimento, bem como a definição de conceitos chave e referências que pudessem ser utilizadas como norteadoras no desenvolvimento deste projeto. A pesquisa buscou artigos, publicações periódicas, teses, trabalhos de conclusão de curso, livros e websites que trouxessem conhecimento a respeito de *frameworks* de desenvolvimento multiplataforma, concepções e comparações entre eles e que, principalmente, pudessem servir como guias para encontrar uma lacuna de pesquisa sobre o mesmo assunto.

As questões de interesse definidas para esta pesquisa bibliográfica foram:

1. Comparações já efetuadas entre *frameworks* de desenvolvimento multiplataforma;
2. Características e propriedades a respeito de *frameworks* de desenvolvimento multiplataforma;
3. Identificação de *frameworks* de desenvolvimento multiplataforma com alta popularidade;
4. Identificação de estudos práticos a respeito do assunto;
5. Encontrar material de recomendação de ferramentas de desenvolvimento multiplataforma, a fim de identificar métricas utilizadas nas recomendações.

As palavras-chaves que foram utilizadas na investigação bibliográfica foram: *cross-platform development*, desenvolvimento de *software*, aplicativos móveis, *framework* de desenvolvimento, tomada de decisão. Após a pesquisa, obteve-se um embasamento teórico sólido para a elaboração de um novo estudo da mesma área de interesse.

As fontes de dados selecionadas são bases de dados tradicionais das áreas de Computação e Engenharias. As bases utilizadas foram Association for Computing Machinery (acm.org) e IEEE Xplore (ieeexplore.ieee.org), que oferecem acesso a pesquisas confiáveis, íntegras e multidisciplinares. Foram consideradas publicações na língua inglesa e portuguesa. A string de busca utilizada foi criada com o seguinte elemento obrigatório da pesquisa: “cross-platform development” ou “desenvolvimento multiplataforma”, que são termos similares nas línguas inglesa e portuguesa. Também foram utilizadas na string de busca os

termos “software”, “mobile” e “framework”. A string foi aplicada nas bases de dados definidas anteriormente através da sintaxe:

("CROSS PLATFORM DEVELOPMENT" OR "DESENVOLVIMENTO MULTIPLATAFORMA") AND "SOFTWARE" AND "MOBILE" AND "FRAMEWORK".

Para a seleção dos estudos que foram utilizados como referencial teórico foram definidos alguns critérios para a inclusão/exclusão dos resultados obtidos nas bases de dados. As publicações enquadraram-se nos seguintes critérios:

1. Ser um artigo científico publicado;
2. Estar escrito em inglês ou português;
3. Estar disponível integralmente na internet ou através de convênio com alguma instituição de ensino;
4. Apresentar validação ou implementação, ou;
5. Demonstrar uma técnica ou a utilização de uma estratégia de comparação entre ferramentas/*frameworks* de desenvolvimento multiplataforma ou a comparação entre *frameworks* e o desenvolvimento nativo.

Os procedimentos utilizados para a seleção dos estudos estão definidos abaixo. Inicialmente foi utilizada a string de busca nas bases de dados apresentadas anteriormente. Todos os resultados foram exportados no formato BibTex para serem utilizados no *software* StArt, e posteriormente serem aceitos ou não através dos critérios definidos. Na próxima etapa realizou-se a leitura dos títulos, das palavras-chave e resumos. Na sequência, foram validadas as introduções e conclusões e, por fim, a leitura integral das publicações selecionadas. Assim sendo, ficaram definidas as fases de seleção de artigos como segue:

Fase 1 - Validar os critérios de inclusão e exclusão;

Fase 2 - Leitura do título, palavras-chave e resumo;

Fase 3 - Leitura da introdução e conclusão;

Fase 4 - Leitura integral dos artigos e validação quanto a qualidade dos mesmos em relação ao assunto deste projeto;

Para validar os estudos selecionados da fase 4, foram levados em consideração alguns critérios de qualidade, sendo eles:

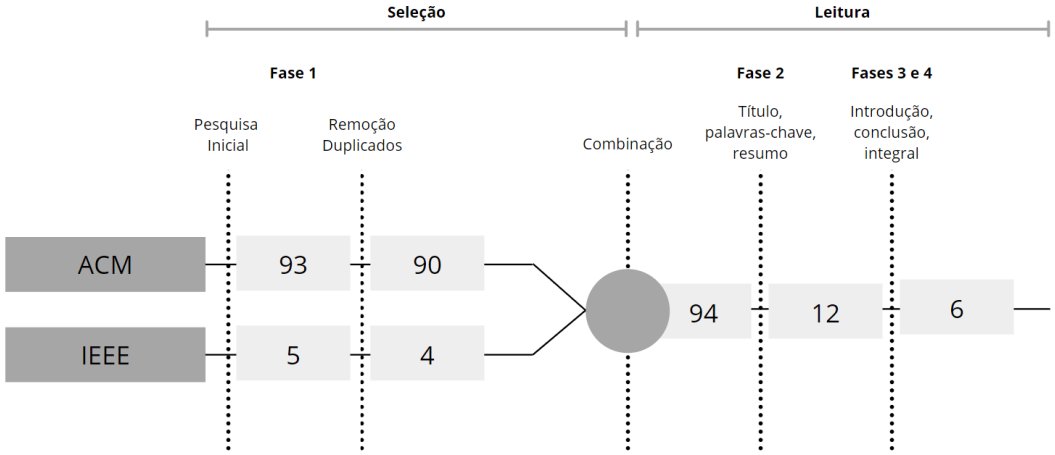
1. Há um modelo de comparação de *frameworks* de desenvolvimento multiplataforma?
2. O modelo foi testado/aplicado a algum *framework*?
3. Quais os critérios de avaliação dos *frameworks*?
4. Quais os resultados obtidos?
5. Foram feitas recomendações com base no resultado da comparação dos critérios?

Os critérios de seleção da fase 1 foram revisitados em todas as etapas sempre que necessário para garantir que estes continuavam sendo respeitados.

Após a aplicação da string de busca nas bases de dados definidas acima, obteve-se 98 trabalhos como resultado, 93 na base da ACM e 5 na da IEEE. O resultado foi exportado das bases para um formato específico que pudesse ser importado em um *software* para auxiliar na seleção dos estudos. Destes, 4 foram identificados como duplicados e portanto já foram descartados. Assim sendo, para o início da seleção, restaram 94 trabalhos para serem avaliados.

Após a conclusão da fase 2, onde foram lidos os títulos, palavras-chave e resumos, a maior parte dos estudos foram desqualificados pois nestes trechos já era possível identificar que não tinham relação ou não se encaixavam nos critérios definidos até esta fase, restando apenas 12 para as fases seguintes. Concluídas as fases 3 e 4, restaram 6 trabalhos que atenderam aos critérios de qualidade definidos e que foram utilizados como auxiliares no decorrer de todo o desenvolvimento deste trabalho. A Figura 8 mostra o processo de filtragem dos trabalhos de maneira sintetizada.

Figura 8 – Síntese do processo de filtragem dos trabalhos relacionados



Fonte: elaborado pelo autor

Estão dispostas na Figura 9, as respostas de cada um dos trabalhos selecionados para os critérios de qualidade, sintetizando o resultado obtido após a conclusão de todas as fases.

Figura 9 – Respostas para os critérios de qualidade dos trabalhos selecionados como referência bibliográfica

	HÁ UM MODELO DE COMPARAÇÃO DE FRAMEWORKS DE DESENVOLVIMENTO MULTIPLATAFORMA?	O MODELO FOI TESTADO/APLICADO A ALGUM FRAMEWORK?	QUAIS OS CRITÉRIOS DE AVALIAÇÃO DOS FRAMEWORKS?	QUAIS OS RESULTADOS OBTIDOS?	FORAM FEITAS RECOMENDAÇÕES COM BASE NO RESULTADO DA COMPARAÇÃO DOS CRITÉRIOS?
FRAMEWORK CHOICE CRITERIA FOR MOBILE APPLICATION DEVELOPMENT	✗	✗	Acesso a recursos nativos, , performance, plataformas alvo, aparência nativa, consumo de energia, segurança, escalabilidade	Apresenta um modelo de recomendação de qual abordagem de framework multiplataforma utilizar	✗
COMPARING PERFORMANCE PARAMETERS OF MOBILE APP DEVELOPMENT STRATEGIES	✓	✓	Tempo de resposta, consumo de memória, uso de CPU, espaço em disco	Apresenta uma análise de uma aplicação desenvolvida em 10 frameworks, auxiliando na escolha de um tipo de framework	✓
AN EVALUATION FRAMEWORK FOR CROSS-PLATFORM MOBILE APP DEVELOPMENT TOOLS: A CASE ANALISYS OF ADOBE PHONEGAP FRAMEWORK	✓	✓	Tempo de inicialização, uso de memória RAM, tamanho do aplicativo, experiência do usuário, aparência, facilidade de desenvolvimento	Apresenta um modelo de avaliação de aplicações multiplataforma, apresentando as vantagens e desvantagens	✗
A CASE STUDY ON CROSS-PLATFORM DEVELOPMENT FRAMEWORKS FOR MOBILE APPLICATIONS AND UX	✓	✓	Performance, experiência do usuário	Apresenta um estudo de caso avaliando a usabilidade utilizando frameworks multiplataforma	✗
CROSS-PLATFORM MOBILE DEVELOPMENT: A STUDY ON APPS WITH ANIMATIONS	✗	✗	Licenças, API, comunidade, tutoriais, complexidade, IDE, dispositivos, GUI, conhecimento	Apresenta uma comparação entre aplicativos desenvolvidos em diferentes frameworks multiplataforma	✗
RECOMMENDATION SYSTEM FOR CROSS-PLATFORM MOBILE DEVELOPMENT FRAMEWORK	✓	✗	Plataformas suportadas, linguagens, licenças, disponibilização nas lojas, recursos nativos, IDE	Apresenta um sistema de recomendação de frameworks multiplataforma, utilizando critérios pesquisados	✓

Fonte: Elaborado pelo autor

A maior parte dos trabalhos relacionados apresenta um modelo de comparação de *frameworks* de desenvolvimento multiplataforma, portanto definem critérios de avaliação para os mesmos. Além de definir um modelo de comparação, três dos trabalhos aplicam o modelo sobre algum *framework*, mostrando-o em funcionamento e testando os critérios que haviam sido definidos.

Todos os estudos estabeleceram uma série de critérios para avaliação de *frameworks*, mesmo os que não definiram um modelo e nem realizaram uma comparação de fato. A maior parte destes se ateve a avaliar questões de usabilidade e experiência de usuário, estipulando como critérios, acesso a recursos nativos, performance, fluidez, aparência, consumo de energia e segurança. Embora sejam quesitos importantes e frequentemente lembrados, outros estudos buscaram considerar questões ligadas ao desenvolvimento e gestão das aplicações, usando como critérios, as plataformas suportadas, linguagens, licenças, comunidade de desenvolvedores, entre outros.

Todos os estudos resultaram na apresentação de uma avaliação/análise sobre o desenvolvimento multiplataforma. Os que definiram um modelo de comparação e avaliação o apresentaram, mostrando suas vantagens e desvantagens, os que não construíram um modelo, apenas compararam aplicações desenvolvidas comentando suas impressões ou apresentaram os resultados de sua pesquisa apontando suas contribuições. Dois trabalhos foram capazes de efetuar recomendações baseados nas comparações dos critérios que por eles foram definidos. Um destes trabalhos subdividiu suas recomendações em duas subseções, analisando parâmetros baseados em performance e não baseados em performance separadamente. O outro trabalho desenvolveu um sistema de recomendação, onde os próprios usuários podem definir que características são importantes em seu projeto e receber as recomendações com base nelas.

4.1 CONSIDERAÇÕES FINAIS

Com a pesquisa bibliográfica realizada, definição de sua string de busca e de seus critérios de seleção, foi possível selecionar inicialmente 98 trabalhos relacionados. Após a filtragem das seis fases desenvolvidas, e consequentemente, eliminação dos trabalhos que não se aderiram ao assunto deste trabalho, restaram seis referências. Os trabalhos selecionados estabeleceram critérios de avaliação de *frameworks*, mas nem todos apresentaram comparação de fato ou a criação de um modelo de comparação.

Nos trabalhos finalistas, foram definidos tanto critérios do ponto de vista dos usuários quanto critérios do ponto de vista de gestão e desenvolvimento. Estes trabalhos apresentaram os pontos positivos e negativos de seus modelos e comparações e alguns demonstraram recomendações com base em seus resultados.

5 PESQUISA QUANTITATIVA

Neste capítulo será apresentado como se deu o desenvolvimento do questionário da pesquisa quantitativa, quais foram os resultados obtidos, assim como uma análise dos mesmos, com o intuito de selecionar os critérios de comparação e selecionar os *frameworks* que foram utilizados neste trabalho.

A pesquisa quantitativa foi escolhida para abordar o problema pois, considera o que pode ser quantificável, traduzindo a opinião e as informações coletadas dos entrevistados classificando-as e analisando-as. Para a análise dos resultados obtidos, deve-se utilizar recursos e artifícios estatísticos, como a: percentagem, média, moda, mediana, desvio-padrão, coeficiente de correlação, análise de regressão (PRODANOV; FREITAS, 2013).

5.1 DESENVOLVIMENTO DO QUESTIONÁRIO

Para entender qual o posicionamento e opinião de desenvolvedores e gestores de equipes de desenvolvimento a respeito do desenvolvimento de aplicativos multiplataforma, desenvolveu-se um questionário com 23 perguntas, todas de preenchimento obrigatório. Para a realização da pesquisa, não foram solicitados dados a respeito do entrevistado, apenas da sua opinião a respeito do tema. A pesquisa foi dividida em quatro seções, sendo:

1. Identificação: seção com apenas uma pergunta, direcionada a todos os entrevistados, com o objetivo de identificar se o respondente já desenvolveu ou geriu uma equipe que desenvolveu algum aplicativo móvel, e que com a resposta negativa leva o questionado ao final da pesquisa;
    - Você já desenvolveu ou geriu uma equipe que desenvolveu aplicativos *mobile*?
2. Experiência: seção com 8 perguntas, aplicada a todos os questionados, com o intuito de obter informação a respeito da experiência dos entrevistados e classificá-los como desenvolvedor ou como líder de equipe. Esta seção bifurca a pesquisa, direcionando desenvolvedores à seção 3 e os líderes/gestores à seção 4, com a intenção de obter opiniões a respeito de assuntos diferentes de cada um destes;

- Que abordagens você ou sua equipe já utilizaram para o desenvolvimento de aplicativos? (Nativo, Multiplataforma, Nativo e Multiplataforma)
- Qual sua experiência com Flutter? (1 - inexperiente a 5 - experiente)
- Qual sua experiência com React Native? (1 - inexperiente a 5 - experiente)
- Qual sua experiência com Cordova? (1 - inexperiente a 5 - experiente)
- Qual sua experiência com Xamarin? (1 - inexperiente a 5 - experiente)
- Qual sua experiência com Phonegap? (1 - inexperiente a 5 - experiente)
- Na sua opinião, o desenvolvimento de aplicativos utilizando *frameworks* multiplataforma ainda fazem sentido, sendo que o mercado é "duopolizado" por Apple (iOS) e Google (Android)?
- Atualmente, você é um desenvolvedor de *software* ou um líder/gestor de equipes? (Desenvolvedor ou Líder/Gestor)

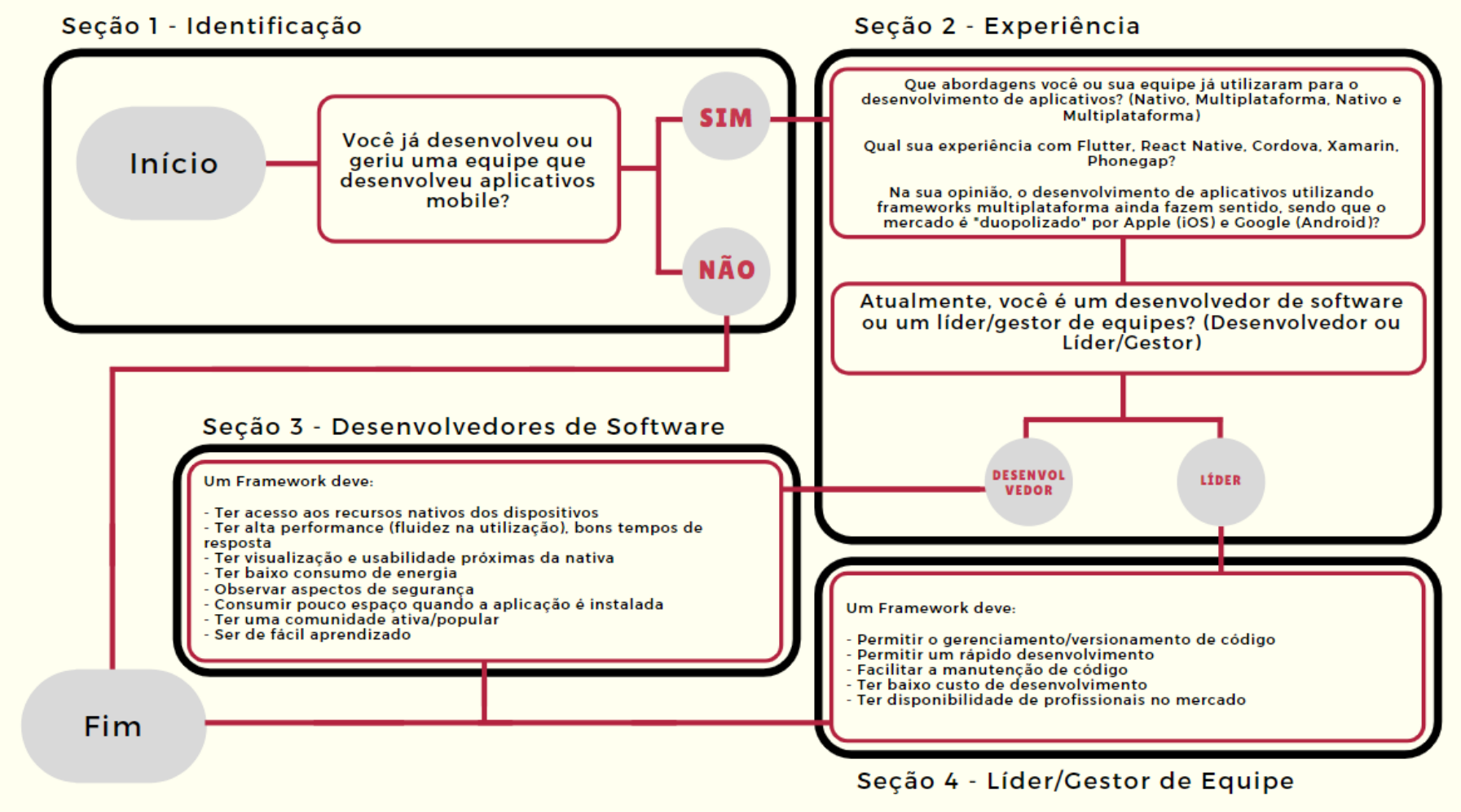
3. Desenvolvedores de *Software*: seção com 8 perguntas, com objetivo de entender quais características os desenvolvedores classificam como importantes num *framework*;

    - Na sua opinião, um *framework* deve (1 - pouco importante a 5 - muito importante):
        - Ter acesso aos recursos nativos dos dispositivos;
        - Ter alta performance (fluidez na utilização), bons tempos de resposta;
        - Ter visualização e usabilidade próximas da nativa;
        - Ter baixo consumo de energia;
        - Observar aspectos de segurança;
        - Consumir pouco espaço quando a aplicação é instalada;
        - Ter uma comunidade ativa/popular;
        - Ser de fácil aprendizado.

4. Líder/Gestor de Equipe: seção com 5 perguntas, com objetivo de entender quais características os líderes e gestores de equipes de desenvolvimento classificam como importantes num *framework*;
    - Na sua opinião, um *framework* deve (1 - pouco importante a 5 - muito importante):
        - Permitir o gerenciamento/versionamento de código;
        - Permitir um rápido desenvolvimento;
        - Facilitar a manutenção de código;
        - Ter baixo custo de desenvolvimento;
        - Ter disponibilidade de profissionais no mercado.

Na Figura 10, estão sumarizadas num fluxograma as seções, perguntas e seus desvios.

Figura 10 – Fluxograma sintetizado das seções, perguntas e desvios do questionário da pesquisa quantitativa



Fonte: Elaborado pelo autor

5.2 RESULTADOS

A seguir, são apresentados os resultados que foram obtidos com a aplicação do questionário, bem como uma análise de cada uma das perguntas aplicadas aos entrevistados. A pesquisa foi aplicada em grupos de alunos, ex-alunos e professores vinculados a Universidade Feevale. O questionário manteve-se ativo do dia 13/10/2020 a 07/11/2020 e, ao todo, obtiveram-se 84 respostas para a pesquisa, sendo que o questionário foi disponibilizado apenas para desenvolvedores e gestores de equipes de desenvolvimento vinculados ou que já tiveram algum vínculo com a Universidade Feevale, sendo um nicho mais específico de entrevistados.

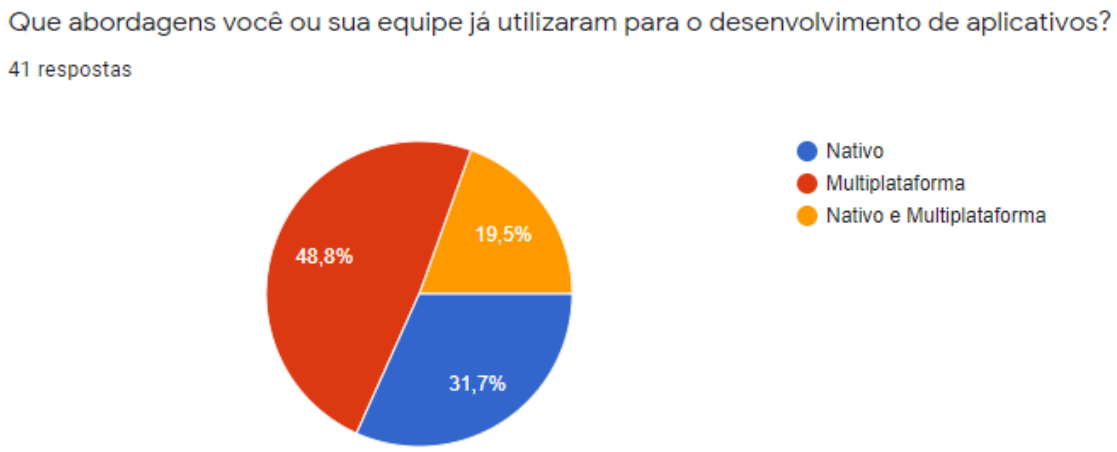
A primeira seção, que contava com apenas uma pergunta, mostrada na Figura 11, filtrou consideravelmente os entrevistados que passaram para a próxima seção, pois nela classificaram-se para continuar respondendo a pesquisa, apenas quem já havia desenvolvido/gerido um aplicativo *mobile*. Responderam positivamente à primeira questão, 48,8% dos respondentes, limitando o número de entrevistados a 41 para as próximas seções.

Figura 11 – Pergunta da pesquisa quantitativa: "Você já desenvolveu ou geriu uma equipe que desenvolveu aplicativos mobile"

Fonte: elaborado pelo autor

A segunda questão, tinha o objetivo de medir qual abordagem havia sido usada pela maior parcela dos entrevistados. Observou-se que, em sua maioria, a abordagem multiplataforma é a mais utilizada, como mostrado na Figura 12. O que pode explicar esse comportamento, é a conveniência que este método proporciona, principalmente a desenvolvedores *web*, que já possuem conhecimento prévio em algumas linguagens e ferramentas, que em muitos casos, são utilizados pelos *frameworks.*

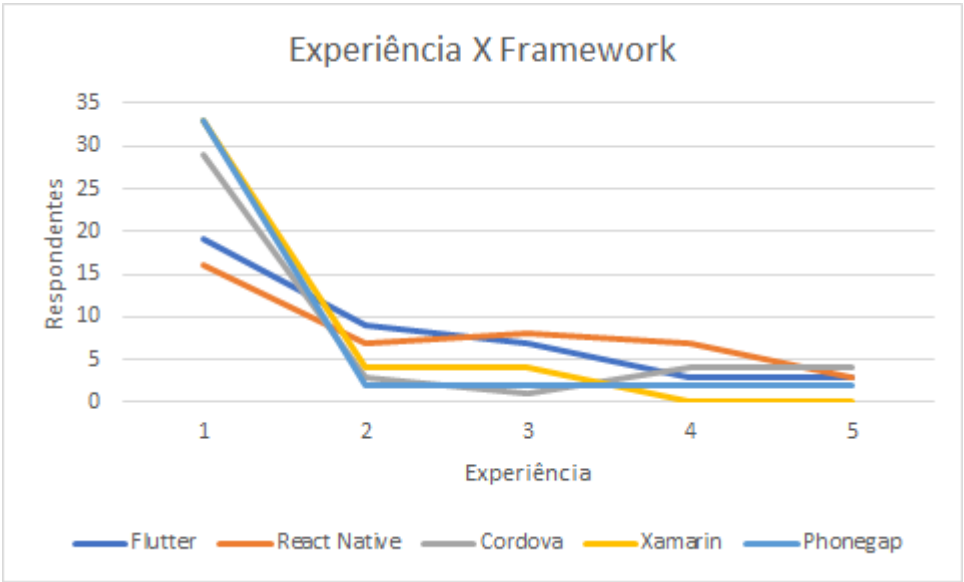
Figura 12 – Pergunta da pesquisa quantitativa: "Que abordagens você ou sua equipe já utilizaram para o desenvolvimento de aplicativos"



Fonte: elaborado pelo autor

As próximas cinco questões eram referentes à experiência dos entrevistados, a respeito de 5 *frameworks* de desenvolvimento multiplataforma. Por haver 31,7% dos entrevistados que apenas desenvolveram aplicativos nativos, esperava-se que a experiência deles com *frameworks* multiplataforma fosse baixa, e foi o que ocorreu. Foi construído um gráfico (Figura 13) com as respostas para a experiência com cada um dos *frameworks*, mostrando uma comparação entre eles.

Figura 13 – Comparativo entre *frameworks* em relação a experiência dos entrevistados



Fonte: elaborado pelo autor

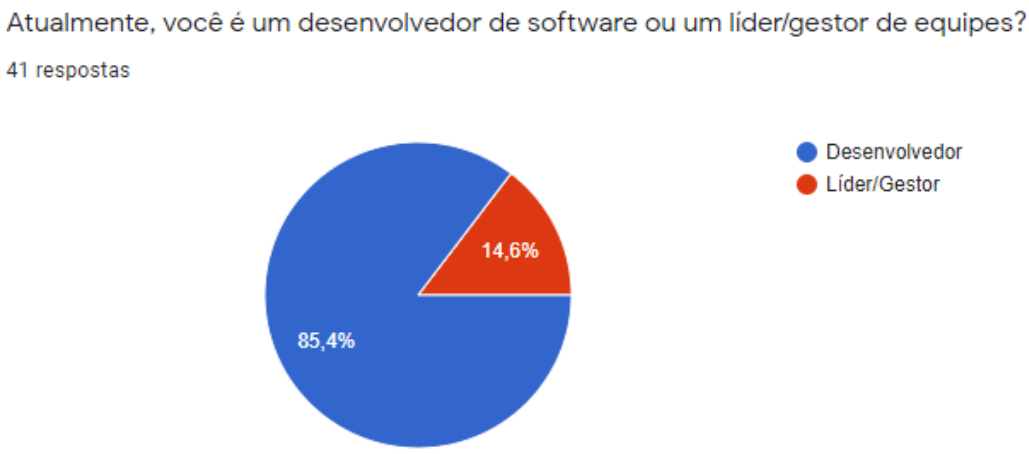
Percebe-se que Flutter e React Native são os *frameworks* com menor número de entrevistados com pouca experiência. Embora isso aconteça com estes *frameworks*, nenhum

dos 5 analisados possuem um número considerável de respondentes com muita experiência. Isso mostra que mesmo conhecendo melhor Flutter e React Native, uma parcela pequena se considera especialista em seu uso.

A pergunta “Na sua opinião, o desenvolvimento de aplicativos utilizando *frameworks* multiplataforma ainda fazem sentido, sendo que o mercado é "duopolizado" por Apple (iOS) e Google (Android)?”, obteve em sua maioria respostas afirmando que o desenvolvimento multiplataforma ainda faz sentido, atingindo 93% dos entrevistados. Constatou-se que as justificativas para as respostas apresentavam uma preocupação grande em relação ao trabalho duplo, no caso de desenvolvimento para as duas maiores plataformas do mercado, assim como a preocupação com a curva de aprendizado maior, custo mais elevado e maior tempo de desenvolvimento utilizando a abordagem nativa.

Na Figura 14, está a questão que divide os entrevistados em dois grupos distintos, desenvolvedores e gestores, e os encaminha a seções diferentes, encaminhou para a seção 4, 35 respondentes, e para a seção 5, 6 respondentes.

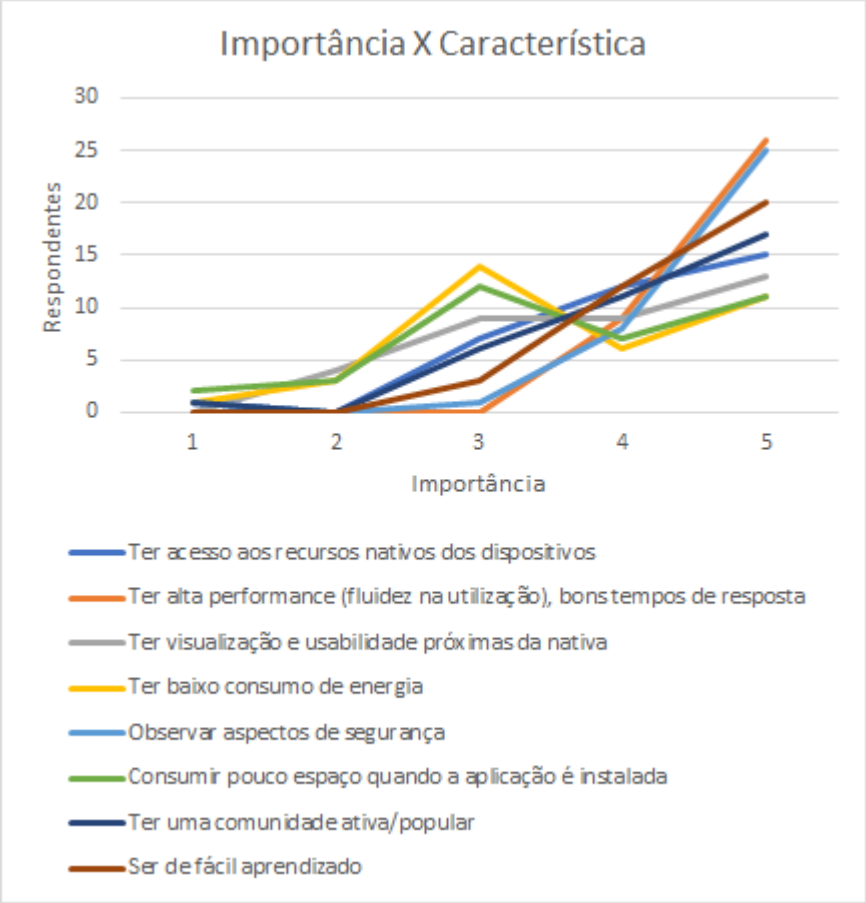
Figura 14 – Pergunta da pesquisa quantitativa: “Atualmente, você é um desenvolvedor de *software* ou um líder/gestor de equipes?”



Fonte: elaborado pelo autor

Na seção destinada a desenvolvedores de *software* (seção 4), os entrevistados foram questionados a respeito de características que *frameworks* devem possuir. De acordo com a Figura 15, os respondentes demonstraram que consideram todas as características importantes. Os critérios que tiveram maior número de respostas com importância menor ou igual a 3 foram, respectivamente, o “baixo consumo de energia”, “consumir pouco espaço quando a aplicação é instalada” e “ter visualização e usabilidade próximas da nativa”.

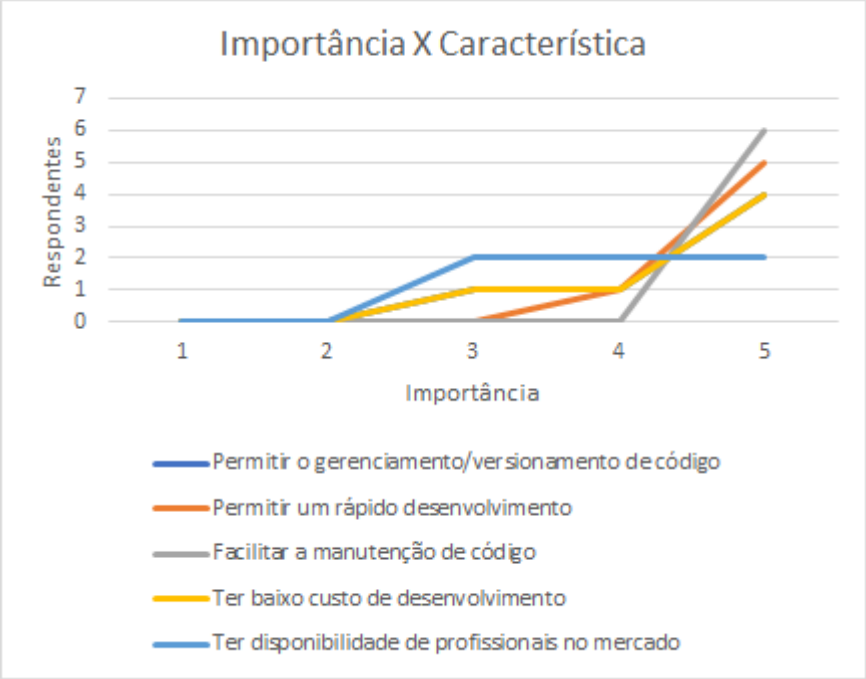
Figura 15 – Comparativo entre *frameworks* em relação a importância de suas características para os desenvolvedores



Fonte: elaborado pelo autor

A quinta e última seção do questionário foi destinada a líderes e gestores de equipes de desenvolvimento de *software*, os entrevistados foram questionados a respeito de características que *frameworks* devem possuir, levando em consideração aspectos gerenciais e de liderança. Assim como na seção destinada aos desenvolvedores, o mesmo comportamento se repetiu e apenas uma das características se mostrou com uma leve redução na importância, segundo os respondentes. A Figura 16 exibe como se comportaram as características definidas para os líderes e gestores e mostra que a disponibilidade de profissionais no mercado não tem a mesma importância que as demais características, segundo os entrevistados. Também pode ser visto que “facilitar a manutenção de código” é a característica com opinião unânime entre os entrevistados, com importância máxima.

Figura 16 – Comparativo entre *frameworks* em relação a importância de suas características para os líderes e gestores de equipes de desenvolvimento de *software*



Fonte: elaborado pelo autor

5.3 SELEÇÃO DE FRAMEWORKS E CRITÉRIOS PARA A COMPARAÇÃO

Nesta subseção são escolhidos os *frameworks* que foram utilizados na comparação que ocorreu neste trabalho, bem como os critérios de comparação que foram utilizados. A escolha se deu com base na bibliografia encontrada na seção anterior e com a ajuda dos resultados da pesquisa quantitativa desenvolvida.

5.3.1 Frameworks

Para a seleção dos *frameworks* que foram comparados foi levado em consideração, sua popularidade na comunidade, a experiência dos entrevistados na pesquisa deste trabalho, nos resultados das consultas nas pesquisas no Google Trends (GOOGLE, 2020) e na pesquisa realizada pela Statista (STATISTA, 2019).

De acordo com o resultado apresentado pela Statista (STATISTA, 2019) na Figura 6 dentre os *frameworks* que aparecem como mais utilizadas pelos respondentes da pesquisa estão: React Native, Flutter, Cordova, Ionic, Xamarin, Unity, Phonegap, entre outros com menos expressividade. Deste resultado, foram extraídos alguns dos *frameworks* com maior expressividade de popularidade, observando que a margem de distância entre React Native e Flutter dos demais *frameworks* indica uma forte diferença.

Em seguida, estes *frameworks* de maior expressividade foram consultados no Google Trends (mostrado na Figura 7), a fim de consolidar a descoberta da pesquisa da Statista e avalizar o resultado por ela obtido. Na consulta efetuada ao Google Trends, foram dispostas na comparação: React Native, Flutter, Cordova, Ionic e Xamarin, que compunham a massiva superioridade na pesquisa da Statista. Novamente se comprovou que Flutter e React Native ficam em um patamar superior de intenções de pesquisa e popularidade quando comparados aos demais consultados.

Na pesquisa quantitativa deste trabalho, foi perguntado aos entrevistados a respeito de sua experiência com determinados *frameworks* de desenvolvimento multiplataforma, com o intuito de observar, com ênfase em uma localização mais aproximada de onde o trabalho estava sendo desenvolvido, qual a percepção de popularidade da comunidade e sua experiência em relação a estas ferramentas. Mais uma vez repetiu-se o comportamento apresentado nos parágrafos anteriores, onde React Native e Flutter se mantiveram mais sólidos e estáveis, como mostra a Figura 13, e apontando superioridade em relação aos demais *frameworks*, quando levado em consideração sua popularidade entre os pesquisados.

Portanto, com as justificativas apresentadas, foram selecionados para a continuidade deste e trabalho e comparados os *frameworks* Flutter e React Native.

5.3.2 Critérios de Comparação

Com os *frameworks* já definidos, foram também justificados e decididos nesta seção, os critérios de comparação que foram utilizados no decorrer deste trabalho. No desenvolvimento da pesquisa quantitativa, mais especificamente das questões relacionadas aos critérios de comparação, foram utilizados como base para a escolha dos critérios, os trabalhos relacionados do capítulo anterior.

Os critérios que mais foram mencionados na bibliografia foram também selecionados para compor a lista de questões, e o resultado da pesquisa norteou a escolha dos critérios que foram definidos para a sequência deste trabalho. Como a pesquisa subdividiu os critérios em seções destinadas a desenvolvedores de *software* e líderes/gestores de equipes de desenvolvimento, o mesmo ocorreu para a definição dos critérios. Foram escolhidos critérios destinados a avaliar os *frameworks* do ponto de vista de desenvolvimento e do ponto de vista de gestão.

Assim sendo, foram selecionados 5 dos 8 critérios existentes do ponto de vista dos desenvolvedores e 4 dos 5 critérios existentes do ponto de vista dos gestores e líderes. Estes critérios foram escolhidos segundo sua importância, de acordo com o que mostram as Figuras 15 e 16, sendo:

- Critérios de desenvolvimento
  - Ter baixo consumo de energia;
  - Observar aspectos de segurança;
  - Ser de fácil aprendizado;
  - Ter uma comunidade ativa/popular;
  - Ter acesso aos recursos nativos dos dispositivos.
- Critérios de gestão
  - Facilitar a manutenção de código;
  - Permitir um rápido desenvolvimento;
  - Ter baixo custo de desenvolvimento;
  - Permitir o gerenciamento/versionamento de código.

6 DESENVOLVIMENTO DO APLICATIVO

Neste trabalho, foi desenvolvido um projeto real de uma aplicação utilizando um *framework* multiplataforma, avaliando e comparando os *frameworks* React Native e Flutter, com o objetivo de auxiliar equipes de desenvolvimento de *software* a decidirem qual destas ferramentas utilizar em cada um dos projetos que iniciarem. O processo de desenvolvimento abordou os parâmetros e características definidos anteriormente para que estes pudessem ser avaliados.

6.1 APRENDIZADO

Para a realização do desenvolvimento do projeto nos *frameworks* definidos, efetuou-se o estudo deles através de suas documentações e cursos em vídeo. Buscou-se este aprendizado para que ficassem entendidos seus conceitos, regras, definições, assim como suas boas práticas.

As documentações utilizadas encontram-se disponíveis em https://reactnative.dev/ - no caso do React Native – e em https://flutter.dev/ - para o Flutter. Os cursos em vídeo foram adquiridos e realizados na plataforma da Udemy[1]. Os cursos definidos foram escolhidos com base em avaliações da própria plataforma e que contemplassem amplamente os conceitos básicos e avançados das ferramentas. Além disso, foram adquiridos cursos que foram disponibilizados pela Cod3r, e que foram ministrados pelo mesmo instrutor, para que houvesse homogeneidade na aprendizagem dos dois *frameworks*.

6.2 O APLICATIVO

O aplicativo escolhido para o desenvolvimento deste projeto foi o FitCash, ideia esta que foi apresentada no evento PitNight da Universidade Feevale. A ideia consiste na construção de uma ferramenta que tem como objetivo auxiliar na diminuição do sedentarismo, melhorar a saúde e qualidade de vida de seus usuários e ajudar na mobilidade urbana.

A solução foi resumida através da seguinte frase, “um aplicativo onde usuários podem trocar seu tempo praticando esportes por recompensas, utilizando-o como incentivo para melhora de sua saúde, qualidade de vida e mobilidade urbana”.

---

[1] https://www.udemy.com/

O funcionamento da ferramenta consiste no *tracking* de atividades físicas realizadas pelos usuários por meio de aplicativos destinados a este fim, obtenção de pontos através deles e a troca desses pontos em loja parceiras, com o intuito de incentivar e promover continuamente a prática de exercícios.

6.3 ANÁLISE E PROTOTIPAÇÃO

Antes de iniciar o desenvolvimento do aplicativo, foi elaborada uma análise e prototipação das telas que foram construídas. Esta análise baseou-se sempre nos critérios de comparação definidos, para que os *frameworks* pudessem ser confrontados posteriormente.

6.3.1 Análise e definição de requisitos

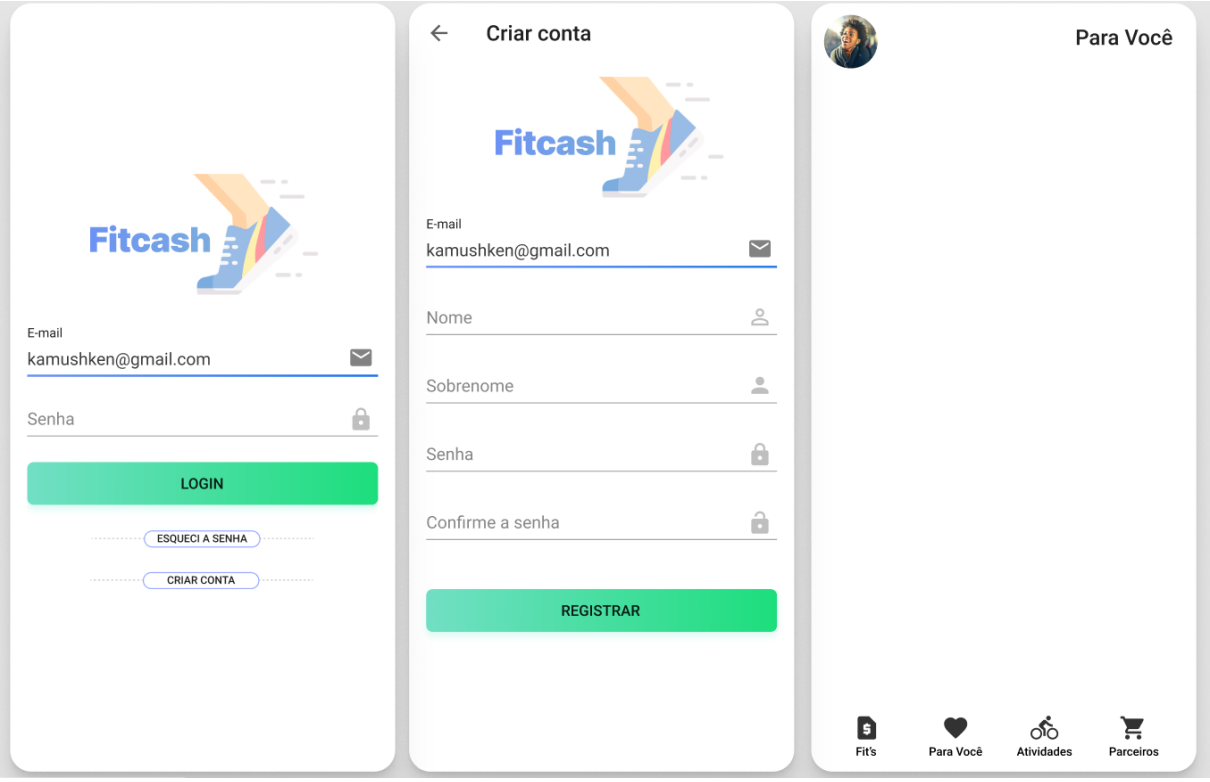
Os requisitos estão definidos abaixo e são brevemente explanados. Estes requisitos contemplam alguns dos critérios de comparação definidos. Nem todos os critérios de comparação puderam ser vinculados diretamente a um requisito e estes foram abordados mais adiante.

1. Tela de Login;
   Tela de autenticação para acesso a área restrita do aplicativo;
2. Tela de Cadastro;
   Tela de criação de usuário para autenticação na tela de login;
3. Tela de Edição de Perfil;
   Tela para edição dos dados de um usuário já criado e autenticado;
4. Tela Inicial;
   Primeira tela acessível após feita a autenticação;
5. Acesso a câmera;
   Disponibilização do recurso nativo para captura de fotografias;
6. Acesso a galeria;
   Disponibilização do recurso nativo para obtenção de fotografias do armazenamento do dispositivo;
7. Acesso ao GPS;
   Disponibilização de recurso nativo de geolocalização para obtenção de locais próximos ao dispositivo.

6.3.2 Prototipação

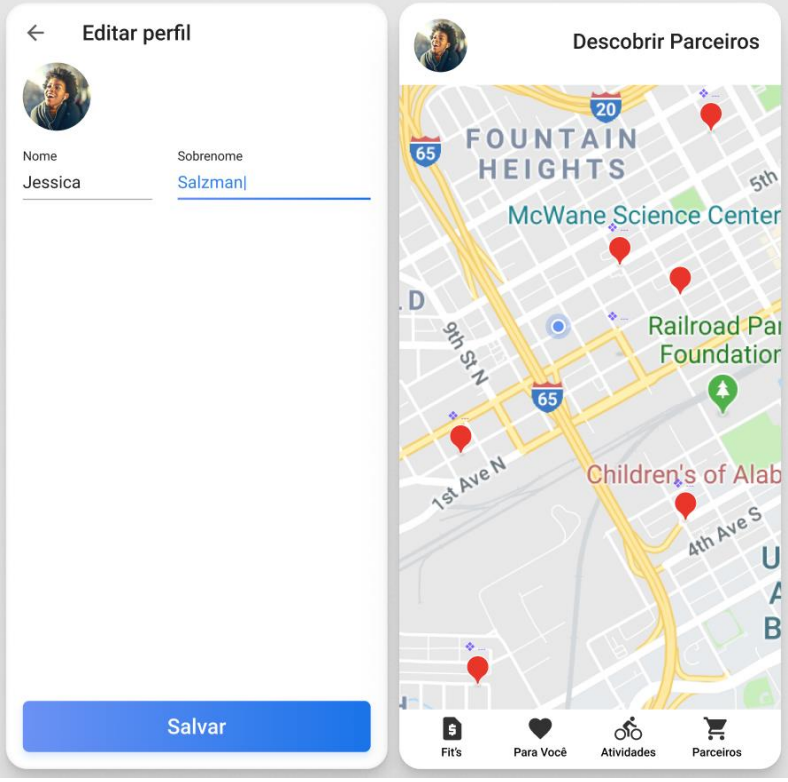
Além da definição dos requisitos para o desenvolvimento do aplicativo, foram prototipadas as telas que teriam que ser desenvolvidas. Nem todos os elementos prototipados foram construídos de fato no desenvolvimento pois estes não eram necessários para contemplar os critérios de comparação definidos. Foram construídas 5 telas, apresentadas na Figura 17 e Figura 18.

Figura 17 – Prototipação das três primeiras telas do aplicativo



Fonte: elaborado pelo autor

Figura 18 – Prototipação das duas últimas telas do aplicativo

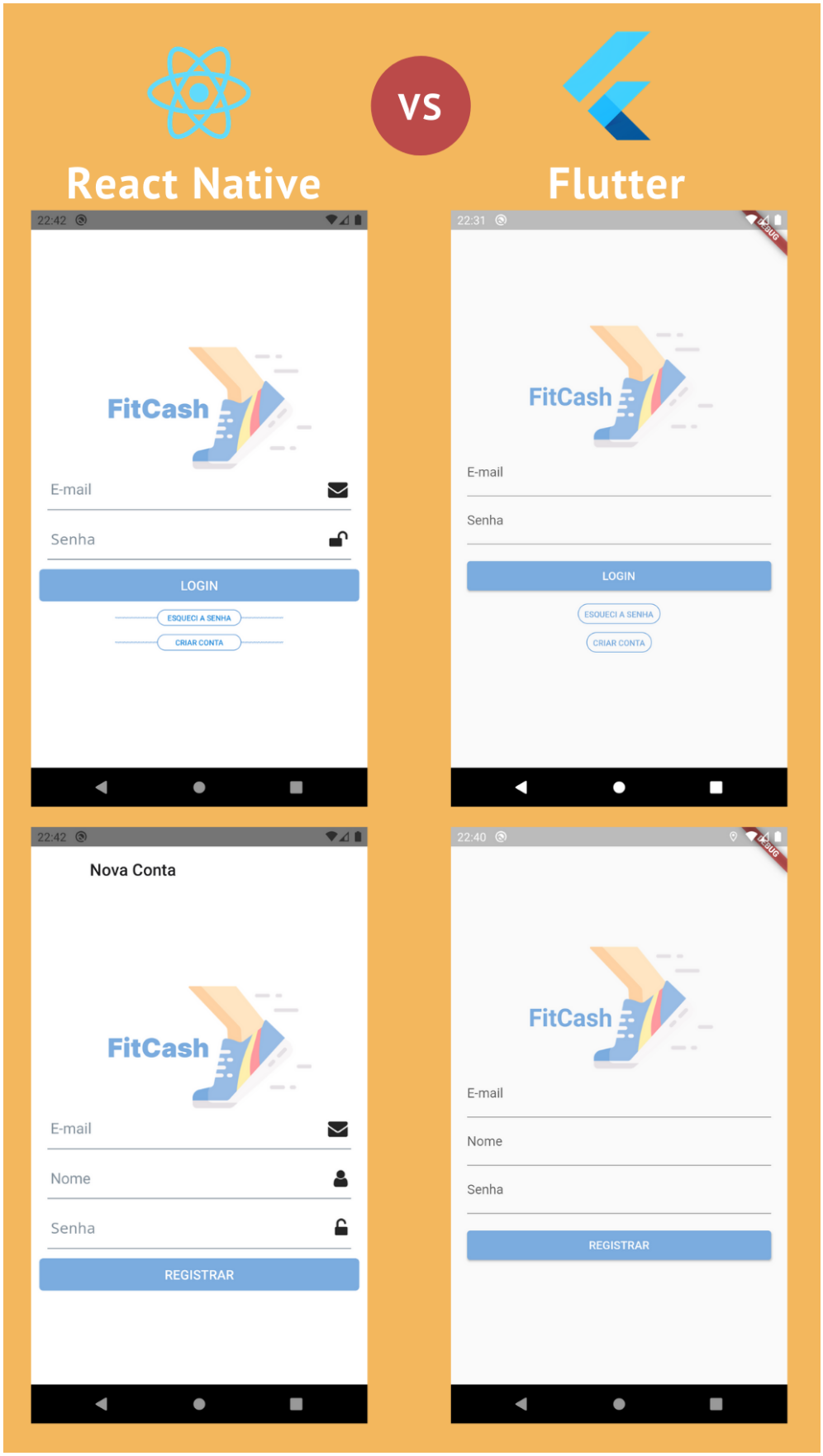


Fonte: elaborado pelo autor

6.4 APLICATIVO FINALIZADO EM REACT NATIVE E FLUTTER

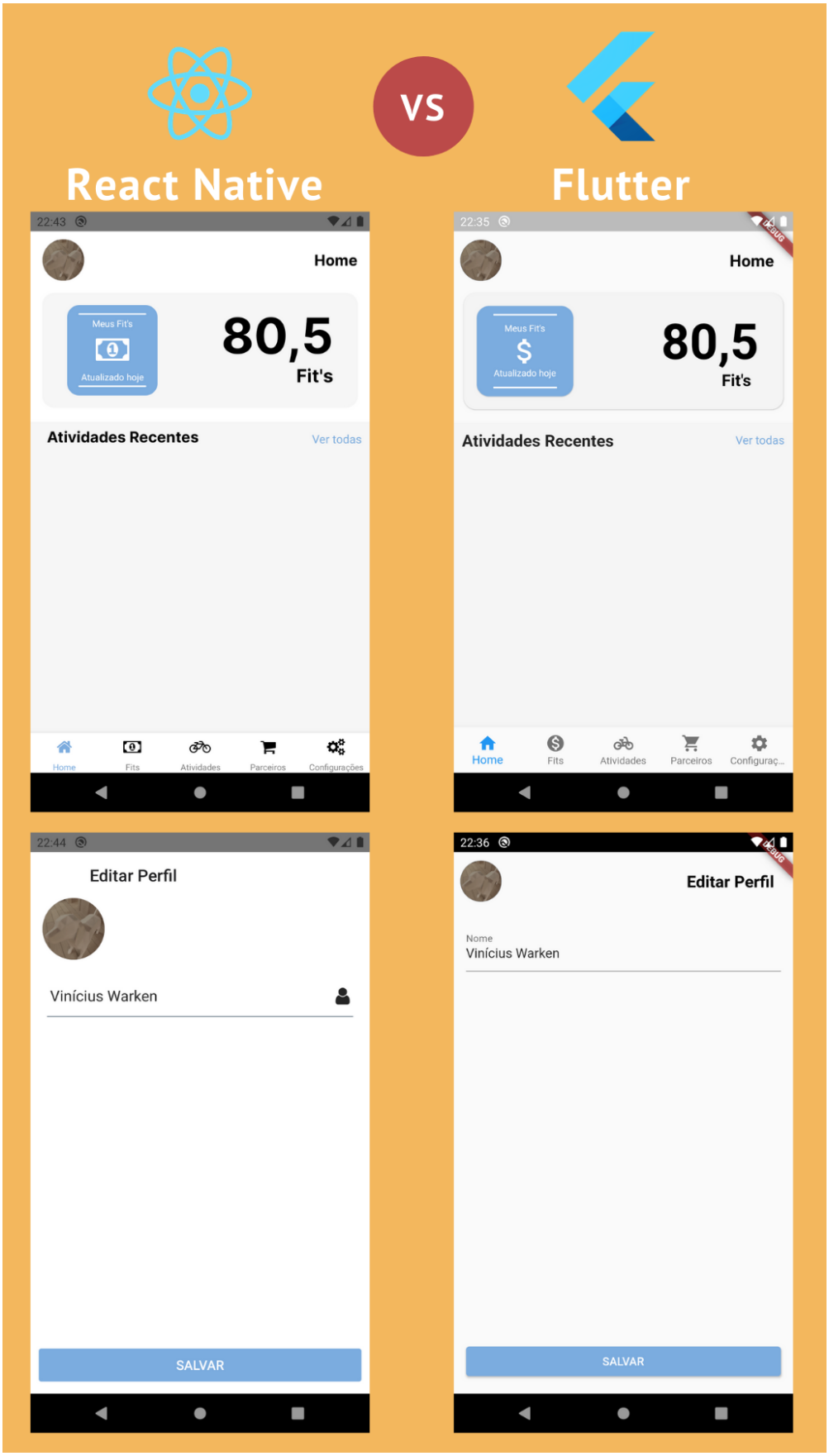
Nesta seção são demonstradas as telas dos aplicativos nas Figuras 19, 20 e 21, lado a lado para comparação. As telas são definidas respectivamente como, *login*, cadastro, *home*, edição de perfil e parceiros. Nestas telas é possível notar a personalização dos elementos, de tal maneira que os aplicativos construídos, utilizando os dois *frameworks*, podem ser praticamente idênticos. O *layout* não fazia parte dos critérios de comparação.

Figura 19 – Comparativo dos aplicativos finalizado, telas 1 e 2



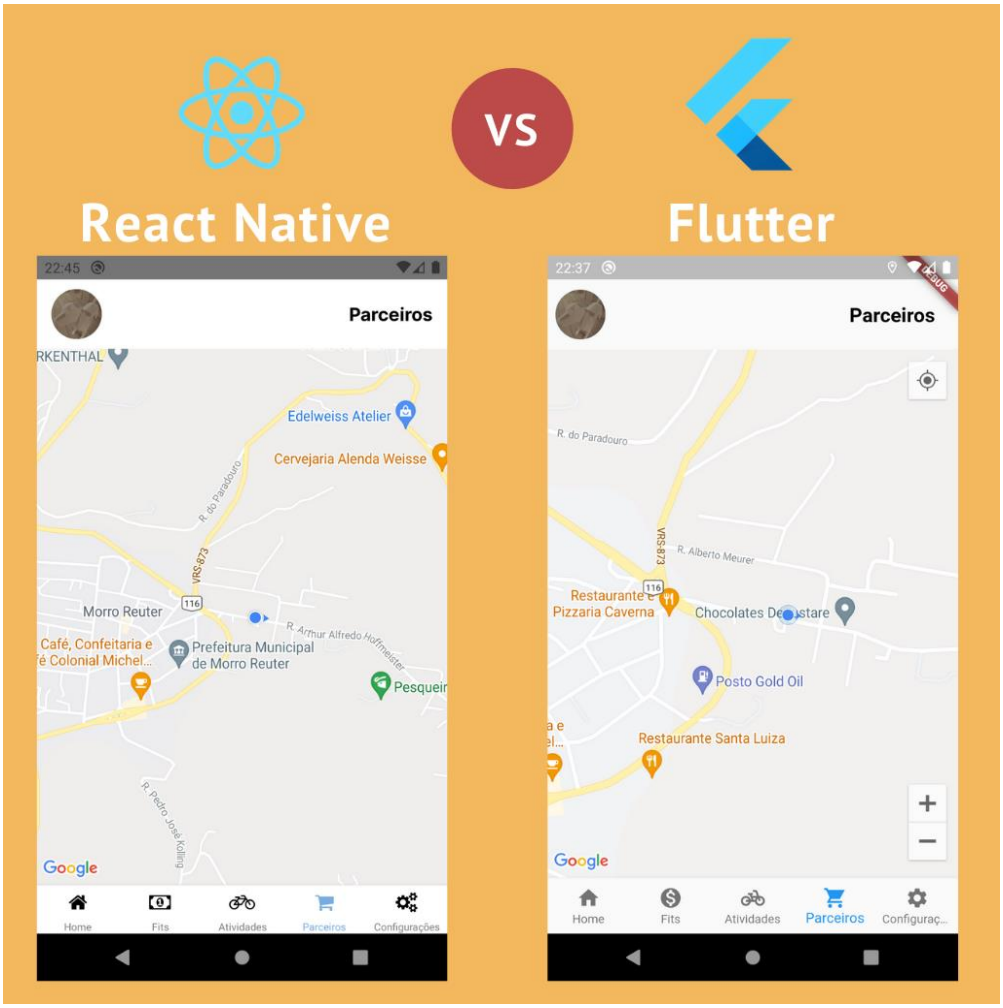
Fonte: elaborado pelo autor

Figura 20 – Comparativo dos aplicativos finalizado, telas 3 e 4



Fonte: elaborado pelo autor

Figura 21 – Comparativo dos aplicativos finalizado, tela 5



Fonte: elaborado pelo autor

7 ANÁLISE COMPARATIVA DOS *FRAMEWORKS*

Neste capítulo estão dispostos os detalhes da comparação entre os *frameworks* multiplataforma. O capítulo aborda a forma como foram comparados, suas particularidades e resultados encontrados. Finalmente, apresentam-se recomendações de utilização de cada um dos *frameworks* e as limitações apresentadas por esse estudo.

7.1 PARÂMETROS COMO MÉTRICAS DE COMPARAÇÃO

Nesta seção se estabeleceram os métodos e como estes foram aplicados para a comparação de cada uma das métricas definidas, assim como os motivos pelos quais elas foram feitas desta maneira.

7.1.1 Consumo de Energia

O consumo de energia nos *smartphones* tornou-se um dos gargalos da computação nos últimos anos. Neste quesito, se questiona frequentemente o consumo excessivo de energia por parte dos aplicativos (COUTO et al., 2016). Para que esta comparação possa ser efetuada de maneira igualitária entre os *frameworks* que estão sendo avaliados, foram utilizados nas comparações, o mesmo sistema operacional, o mesmo *hardware* nos dispositivos e o tempo em que o consumo será avaliado. Estes são fatores cruciais na manutenção e preservação do recurso energético dos dispositivos móveis (PINTO; CASTOR, 2017). Para Pinto e Castor (2017), é uma prática muito comum delegar o gerenciamento do consumo de energia às camadas de baixo nível dos sistemas operacionais para o qual a aplicação está sendo desenvolvida. Apesar de comum, a participação dos desenvolvedores neste processo pode melhorar consideravelmente a eficiência energética das aplicações construídas.

Os *frameworks* também participam desse processo de conservação da bateria, pois em diversos casos eles abstraem e já implementam tarefas que, se não fossem utilizados, seriam delegadas ao desenvolvedor que está implementando. Portanto, *frameworks* distintos realizando as mesmas tarefas, podem consumir energia de maneira completamente diferente (CRUZ, 2019). Existem algumas ferramentas que auxiliam no processo de monitoramento e análise do consumo de energia em dispositivos móveis, entre eles o Gator, o SonarQube, BatteryStats e o Android Profiler. Para a comparação entre React Native e Flutter neste trabalho, foi utilizado o BatteryStats em conjunto com o Battery Historian, que são ferramentas recomendadas na própria documentação do Android, na seção de desempenho e

otimização da duração da bateria. Para Wu, Yang e Rountev "O padrão de diretrizes para a criação de aplicativos que utilizam GPS, alertam os desenvolvedores para sempre estarem atentos ao fato de que utilizar o *listener* de localização por um longo período de tempo consome muita energia da bateria."(2016 apud RIBEIRO, 2019, p. 27). Ahmad et al. (2017) apontam que aplicativos que utilizam vídeo sob demanda, jogos *multiplayer*, GPS ou outros sensores, estão entre os maiores consumidores de energia.

Assim sendo, a comparação se deu através da implementação de um *listener* constante de GPS, responsável por monitorar a geolocalização do dispositivo enquanto este se locomove por 30 minutos e assim extraiu-se os resultados disponibilizados pelo BatteryStats para a comparação.

O BatteryStats é uma ferramenta que está incluída no próprio *framework* do Android. Ele monitora e coleta dados a respeito da bateria durante a utilização do dispositivo. Através do *Android Debug Bridge* (ADB), os dados da bateria podem ser despejados para o computador de desenvolvimento e relatórios de uso e consumo gerados a partir do Battery Historian. O Battery Historian é uma ferramenta auxiliar de visualização que recebe os dados do BatteryStats e os converte em uma visualização HTML amigável que pode ser aberta em qualquer navegador. Através desta análise podem ser identificados processos e tarefas que estão consumindo bateria demasiadamente, assim como monitorar comandos que possam ser adiados ou até removidos dos aplicativos (GOOGLE, 2021).

O dispositivo utilizado na comparação foi um Motorola Moto G6 Plus, que possui o Android com sistema operacional na versão 9. Antes do início de cada teste, a bateria do dispositivo foi totalmente carregada e os dados fornecidos pelo ADB através do BatteryStats, foram resetados, conforme orienta a documentação do Android para este tipo de análise. O percurso percorrido nos testes com as aplicações criadas com o React Native e Flutter foi exatamente o mesmo, em torno de 10 km e percorridos de bicicleta, numa velocidade média de 17 km/h. Além disso, o dispositivo permaneceu com a tela ligada e o brilho no nível máximo durante todo o teste para acompanhamento.

O resultado demonstrou que o aplicativo construído com React Native ao final do teste permanecia ainda com 89% da bateria e teve uso estimado da bateria de 1,27%. Durante o teste, o sensor de GPS permaneceu ativo por 36 minutos e 55 segundos, enquanto o acelerômetro por 25 minutos e 36 segundos. Já o aplicativo construído com Flutter, ao final do teste, ainda permanecia com 87% e teve uso estimado da bateria de 0,98%. Durante o teste

o sensor de GPS ficou ativo por 28 minutos e 11 segundos e diferentemente do app React Native, não utilizou o acelerômetro.

Neste teste inicial os consumos de bateria dos dois aplicativos equivalem-se e que, com o simples uso de um *listener* de GPS, o consumo de bateria dos aplicativos foi baixo. Foi possível identificar que a maior parte da bateria foi consumida pela tela, pois permaneceu em nível alto de brilho durante todo o tempo nos dois testes.

7.1.2 Segurança

Assim como o consumo de energia dos *smartphones*, outro assunto que se tornou frequente em fóruns e portais de tecnologia é a relação destes dispositivos com os dados dos seus usuários. A preocupação com o aumento de invasões e vazamento de dados pessoais não é mais somente dos desenvolvedores, e sim dos próprios usuários (ANSONG; SYNAEPA-ADDISION, 2019).

O quesito segurança na construção de aplicações móveis é frequentemente relegada a segundo plano por parte de desenvolvedores e *software houses*. Isso acontece na maior parte das vezes, pois manter aplicações seguras demandam altos investimentos financeiros e de recursos. Vale lembrar que, mesmo disponibilizando equipes a esta função, criar *software* totalmente impenetrável é impossível. Os bancos financeiros são o exemplo prático disso, pois dedicam muito esforço nessa tarefa e ainda assim acabam sendo alvos de invasões. Ainda assim, a probabilidade de ser vítima de um ataque malicioso é inversamente proporcional ao esforço dedicado à proteção do aplicativo em questão, como mostra a Figura 22 (REACT NATIVE, 2021).

Figura 22 – Relação Esforço Investido em Segurança X Probabilidade de uma Violação de Segurança

Fonte: elaborado pelo autor (adaptado de REACT NATIVE, 2021)

A comparação entre React Native e Flutter no quesito segurança foi realizada através da disponibilização de informações e práticas a respeito do tema em suas documentações e comunidades de desenvolvedores. Avaliou-se também a disponibilização de recursos básicos de segurança como métodos para o armazenamento de informações sensíveis, autenticação, segurança em rede, além de ferramentas que possam auxiliar na segurança dos aplicativos. Esta abordagem foi utilizada na comparação pois, através da facilitação ao acesso de mecanismos de proteção, é possível fazer com que desenvolvedores e líderes de equipes passem a observar com mais atenção os aspectos de segurança.

Em relação ao tema segurança, em suas documentações, tanto React Native quanto Flutter possuem uma página dedicada ao assunto. A página do React Native possui mais conteúdo e fornece recomendações mais bem embasadas. Ao contrário da documentação do Flutter, React Native orienta a utilização de pacotes de ferramentas já consolidados para a realização de tratativas de segurança para vários segmentos da área, como: armazenamento de informações sensíveis, autenticação e *deep linking* e segurança de rede. Já a página do Flutter apenas descreve que o time de desenvolvimento do *framework*, trabalha constantemente em atualizações de segurança e que devem permanecer sempre atualizados os SDKs do Flutter, dependências de pacotes externos e dispor sempre de uma cópia estável de sua aplicação, além de um canal facilitado para reportar falhas de segurança detectados.

Esta comparação não prova que o React Native é de fato o *framework* mais seguro, mas aponta que ele é mais maduro e consolidado em relação a este tema. Como descrito no primeiro parágrafo desta seção por Ansong e Synaepa-Addision (2019), a segurança ainda é deixada em segundo plano por ser um aspecto que demanda recursos capacitados e tempo, mas que com as informações disponibilizadas de maneira facilitada nas documentações o processo de adoção destas práticas torne-se cada vez mais recorrente. A vantagem apresentada pelo React Native em relação ao Flutter pode ser justificada também pelo seu maior tempo de mercado e maior número de usuários, assunto que será debatido na subseção 7.1.4.

7.1.3 Aprendizado

A facilidade de aprendizagem dos *frameworks* é muito relativa e está diretamente relacionada à pré-existência de conhecimento nas linguagens de programação que são base destas ferramentas. Além disso, vale ressaltar que podem cooperar com a facilidade no aprendizado questões referentes aos conceitos básicos de cada um dos *frameworks* e sua configuração, bem como toda a história de desenvolvimento do programador.

Na comparação deste ponto, foram levados em consideração os tópicos definidos por Gupta (2004) como determinantes para a construção de um aprendizado acessível e descomplicado de uma nova linguagem de programação. Os quesitos que foram utilizados são os seguintes: simplicidade, ortogonalidade, cobertura e suporte a debug.

Vale lembrar que esta avaliação foi realizada sob a perspectiva do autor, mas utilizando os critérios de Gupta (2004). Também é importante ressaltar que o autor já tinha conhecimento prévio básico da linguagem JavaScript e nenhum conhecimento na linguagem Dart.

Abaixo estão descritas e exemplificadas as comparações dos critérios definidos acima. Em alguns dos quesitos foram observados aspectos e particularidades referentes a cada um dos *frameworks* para o melhor compreendimento da comparação efetuada.

- Simplicidade:

Iniciantes em um *framework* ou linguagem de programação podem facilmente ser sufocados pelo código quando ele não é intuitivo ou difícil de ler e consequentemente, linguagens que possuem sintaxes não usuais e controles de fluxo complexos tornam-se difíceis de se aprender. Linguagens funcionais portanto tornam-se mais complexas de se aprender quando comparadas a linguagens procedurais (GUPTA, 2004).

Tanto o Javascript quanto o Dart, linguagens do React Native e Flutter, respectivamente, são linguagens de *script* multiparadigma. A principal diferença está na tipagem, que no caso do Javascript é dinâmica e fraca e no Dart é estática, forte e inferida.

De acordo com a definição de simplicidade de Gupta (2004), ambas não se encaixariam nas mais recomendadas para se iniciar um aprendizado de programação, mas em contrapartida, ambas possuem comunidades grandes e difundidas e que possibilitam e facilitam a procura por exemplos de implementação de código. Para programadores que já tiveram contato prévio com linguagens como Java e C#, não encontrarão dificuldades no aprendizado dos principais conceitos destas linguagens. Assim sendo, neste quesito, nenhuma das linguagens se destaca como sendo de maior simplicidade.

- Ortogonalidade

Por ortogonalidade, entende-se que são as diferentes maneiras de se executar uma tarefa em uma linguagem de programação. Linguagens com ortogonalidade extremamente alta ou extremamente baixa não são recomendáveis para iniciantes (GUPTA, 2004). Para

usuários que já possuem experiência com linguagens semelhantes a Javascript e Dart não terão dificuldades em relação a esse quesito. Mas como apenas foi desenvolvida a mesma aplicação com estas linguagens, este critério torna-se pessoal, ou seja, a dificuldade pode variar de acordo com cada utilizador.

- Cobertura

Assim como na ortogonalidade, em relação ao tópico cobertura também não há como definir uma linguagem mais conveniente. Cobertura refere-se a linguagem possuir as mais comuns construções sintáticas e semânticas, dentre elas, estruturas condicionais, iterativas, estruturais e de tratamento de exceções (GUPTA, 2004). Estruturas estas, disponibilizadas tanto pelo Javascript quanto pelo Dart.

- Suporte a debug

Nos processos de aprendizagem, programadores costumam realizar pequenas sequências de compilações e/ou execuções, pois cometem pequenos erros constantemente. Para tanto, é preferível que os ambientes de desenvolvimento disponibilizem curtos ciclos entre execuções e novas compilações, para que a cada novo ajuste rapidamente possa-se corrigir os problemas encontrados. Outros pontos relevantes a serem considerados são o suporte ao processo de *debug* nas IDEs ou editores utilizados e as mensagens de erros retornados pelas linguagens nos momentos em que ocorrerem erros (GUPTA, 2004).

Tanto o React Native quanto o Flutter apresentam um rápido recarregamento das telas que são alteradas, simplesmente salvando o arquivo alterado. Em relação a este quesito, o Flutter se sobressai pois mostra-se mais estável, conseguindo renderizar as telas com as modificações na maior parte das alterações que são feitas. No React Native, em uma quantidade consideravelmente maior de vezes é necessária a completa compilação e/ou reinicialização da tela ou até da aplicação como um todo.

O desenvolvimento dos apps foi realizado com o mesmo editor, Visual Studio Code, amplamente recomendado e difundido em ambas as comunidades dos *frameworks*. Neste editor as duas linguagens e *frameworks* disponibilizam extensões que facilitam o *debug* e a execução dos aplicativos em emuladores e dispositivos físicos. Para o Flutter, destacam-se as extensões “dart-code.flutter” e “dart-code.dart-code”, e para o React Native a “msjsdiag.vscode-react-native”.

Finalmente, em relação às mensagens de erro apresentadas pelas ferramentas, as disponibilizadas pelo Flutter se mostram mais esclarecedoras, apontam com mais precisão

onde os problemas se encontram e tornam o processo de desenvolvimento mais fluido e dinâmico. Esta afirmação será esclarecida com maior ênfase na subseção 7.1.6 pelas Figuras 28 e 29. Assim sendo, o Flutter mostra-se mais amigável no quesito suporte a *debug*, principalmente para principiantes e programadores que venham a realizar manutenções em aplicações existentes.

Portanto, em relação ao aprendizado, não há grande vantagem para nenhum dos *frameworks*, exceto pela pequena superioridade do Flutter no suporte à depuração de código, o que pode acelerar brevemente o desenvolvimento das aplicações. Para desenvolvedores experientes ou que já possuem conhecimento em linguagens semelhantes, não haverá maiores dificuldades no aprendizado, mas para iniciantes na programação, não se recomenda a utilização destes *frameworks*, nem suas linguagens.

7.1.4 Comunidade

Ter uma comunidade ativa e popular é tido como um requisito quando se analisa a implementação de uma nova tecnologia ou ferramenta de desenvolvimento de *software.* A comparação realizada entre os *frameworks* neste tema levou em consideração o número de questões criadas e vinculadas às tags *react-native* e *flutter* no StackOverflow, além da posição das linguagens Javascript e Dart nos índices TIOBE[2] e PYPL[3].

O StackOverflow é hoje uma das plataformas mais populares de perguntas e respostas a respeito de tecnologia e programação. Ela é mantida pela rede Stack Exchange que possui diversos outros canais sobre outros assuntos. Esta rede disponibiliza uma plataforma online chamada de Stack Exchange Data Explorer[4], que possibilita fazer consultas em canais específicos de toda a rede. Nesta avaliação foram desenvolvidas consultas, a fim de obter dados a respeito da popularidade do React Native e do Flutter. Na plataforma, os assuntos das perguntas são definidos por *tags*, identificadores pelos quais os usuários podem procurar por respostas ou responder questões sobre as quais possui conhecimento.

Foram executadas nesta plataforma duas vezes a mesma *query*, alterando em cada uma delas apenas o filtro efetuado, onde na primeira execução foi filtrado o nome da *tag* como “react-native” e na segunda como “flutter”. O objetivo da consulta é avaliar o número

[2] https://www.tiobe.com/tiobe-index/
[3] https://pypl.github.io/PYPL.html
[4] https://data.stackexchange.com/stackoverflow/queries

de postagens sobre cada *framework*, agrupando o resultado por mês, desde a primeira pergunta criada para cada uma das *tags*. A Figura 23, demonstra a consulta efetuada na plataforma para a obtenção dos resultados.

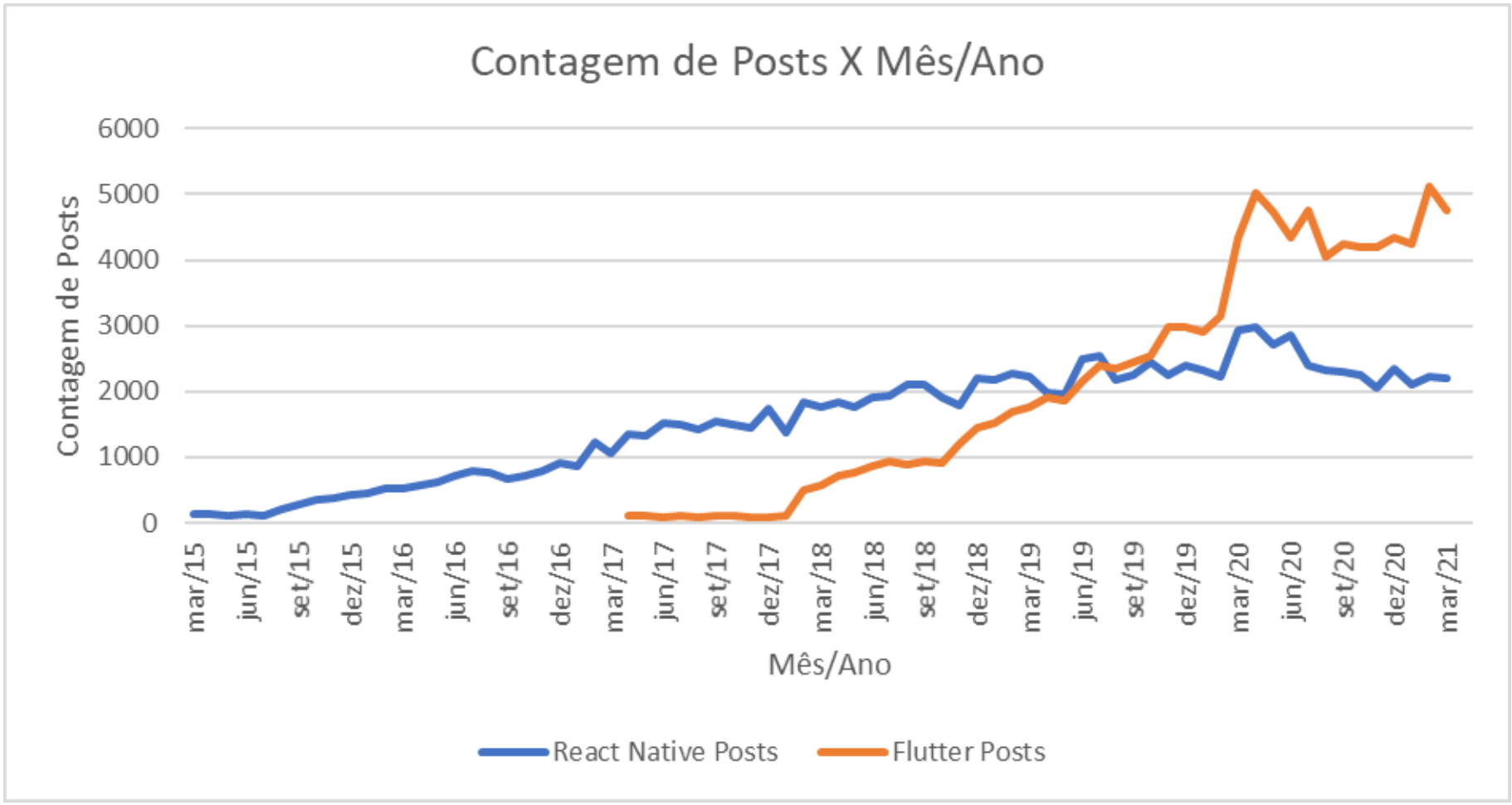
Figura 23 – *Query* utilizada na plataforma Stack Exchange Data Explorer

```
SELECT
    FORMAT(Posts.CreationDate, 'MM/yyyy') CREATION_DATE,
    COUNT(Posts.Id) AS POSTS
FROM Tags
    INNER JOIN PostTags ON PostTags.TagId = Tags.id
    INNER JOIN Posts ON Posts.ParentId = PostTags.PostId
WHERE
    Tags.TagName = :TAG_NAME
GROUP BY FORMAT(Posts.CreationDate, 'MM/yyyy')
```

Fonte: Elaborado pelo autor

O resultado da consulta é mostrado pela Figura 24, que tem sua linha do tempo iniciada em março de 2015, com o lançamento do React Native, e ganha mais uma linha a partir de maio de 2017, no lançamento do Flutter. Os dados apresentam um crescimento linear do número de questões criadas pelos usuários do React Native e uma visível estabilização neste número nos últimos meses. Já a linha que representa o número de postagens para a *tag* do Flutter tem um período de pouco mais de meio ano de estabilidade após seu lançamento, mas após esse período cresceu exponencialmente, inclusive ultrapassando em número de postagens o seu concorrente. Finalmente, também apresenta uma tendência de estabilização nos últimos meses.

Figura 24 – Resultado da consulta efetuada na plataforma Stack Exchange Data Explorer



Fonte: Elaborado pelo autor

Nos índices Tiobe e PYPL de maio/2021, Javascript encontra-se respectivamente na 7ª colocação (com 2.45% dos *ratings*) e em 3º (com 8.31% do *market share*). Já a linguagem Dart se encontra na 27 ª colocação no Tiobe (com 0.53% dos *ratings*) e na 21ª colocação no PYPL (com 0.55% do *market share*).

Conclui-se que quando analisados apenas os *frameworks*, as comunidades já se equivalem, mesmo com o tempo de vida bem menor do Flutter. Mas quando comparadas as linguagens de programação utilizadas, o Javascript ainda possui grande vantagem pois é utilizado em diversas tecnologias diferentes atualmente, diferentemente do Dart que é somente implementado nas aplicações Flutter.

7.1.5 Recursos Nativos

Como Palmieri et al. (2012) mencionam, há diversos pontos que devem ser considerados quando uma ferramenta para o desenvolvimento multiplataforma é considerada para a criação de uma aplicação. Dentre os pontos definidos está o número de recursos nativos que é possível acessar com cada um dos *frameworks*.

Foi comparado se é possível acessar os principais recursos nativos disponibilizados por um dispositivo móvel, assim como a facilidade com que essa tarefa pode ser executada em cada uma das ferramentas. Os recursos nativos que foram avaliados nesta comparação: a câmera, o armazenamento interno e a geolocalização, pois são as funcionalidades mais básicas e populares entre os aplicativos do mercado.

Primeiramente, ressalta-se que em ambos os *frameworks* os três recursos nativos avaliados puderam ser acessados sem muita dificuldade. Para que estes recursos pudessem ser acessados, foi necessário o ajuste dos arquivos de configuração do Android, liberando o acesso aos mesmos, mas este fato também ocorreu com os desenvolvimentos dos apps nas duas plataformas. Vale assinalar ainda que, tanto no React Native, quanto com o Flutter, foram utilizadas bibliotecas já desenvolvidas para estes fins e que estas são mencionadas e recomendadas pelas documentações dos *frameworks.* Abaixo estão relacionadas as bibliotecas utilizadas para a implementação dos recursos nativos com ambos os *frameworks:*

- React Native
    - GPS: *react-native-community/geolocation;*
    - Câmera: *react-native-camera;*

- Armazenamento de Imagens: *react-native-image-picker;*

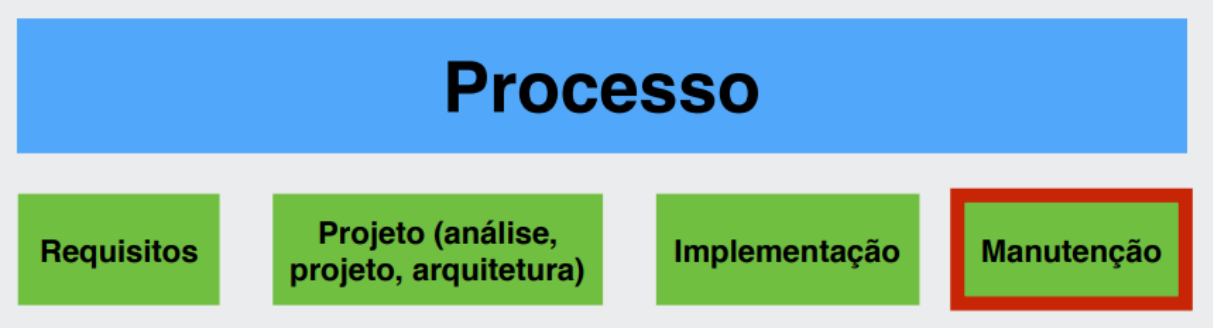
- Flutter:
  - GPS: *location*;
  - Câmera: *image_picker;*
  - Armazenamento de Imagens: *image_picker*.

Conclui-se que o objetivo de acessar os recursos nativos foram concluídos com sucesso e, além disso, alcançados de maneira facilitada através da documentação clara e acessível de ambos *os frameworks*.

7.1.6 Manutenção de Código

É importante ressaltar que a facilidade de manutenção de código está fortemente ligada a boa construção dele, com a utilização de padrões de projeto, boas práticas, entre outras características. Hora (2019) indica que a própria literatura referente a engenharia de *software* destaca a manutenção no processo de construção das aplicações, como mostra a Figura 25.

Figura 25 – Processo de construção de *software*



Fonte: Hora (2019)

A manutenção de *software* é a disciplina responsável por tratar de todas as tarefas que são executadas após a entrega das aplicações, ou seja, define-se como o ato de manter uma entidade em bom estado de reparo, eficiência ou validade, para evitar falhas ou declínio.

Como a manutenibilidade ou facilidade de manutenção de um sistema é de difícil quantificação, e as métricas utilizadas para essa medição normalmente quantificam questões voltadas às boas práticas mencionadas anteriormente, para a comparação deste quesito foram avaliadas as possibilidades e a capacidade de *debug* das rotinas desenvolvidas com estas ferramentas (HORA, 2019). A possibilidade de depuração de uma aplicação facilita o trabalho

e a compreensão das lógicas e código já produzido, portanto torna-se uma importante funcionalidade no auxílio da manutenção de código.

Como já foi explicado durante a comparação da facilidade de aprendizado, na subseção 7.1.3 sobre suporte a *debug*, ambos os *frameworks* disponibilizam ferramentas através de extensões ao editor de texto Visual Studio Code, que possibilitam a depuração do código produzido, implementando as principais funcionalidades disponíveis nas IDEs para desenvolvimento de *software* do mercado.

A principal diferença entre os dois *frameworks* e o que torna o Flutter ligeiramente mais amigável neste sentido é a pilha de erros quando uma exceção é lançada. Também chamada de *stacktrace*, a pilha de erros mostrada pelo Flutter facilita aos desenvolvedores na procura pelos problemas e o local/linha onde ela está sendo lançada. Abaixo estão colocadas, uma abaixo da outra, as pilhas de erros do React Native e do Flutter para um mesmo lançamento de exceção. A exceção lançada foi provocada intencionalmente nos dois aplicativos para a demonstração do *stacktrace.* As Figuras 26 e 27 apresentam o erro sendo provocado, ambos na função executada no clique do botão de *login*.

Figura 26 – Método de login no React Native com exceção na linha 34

```
33    function login() {
34      throw new Error("Exceção intencional.");
35
36      auth()
37        .signInWithEmailAndPassword(email, senha)
38        .then(() => {
39
40        })
41        .catch(error => {
42          if (error.code === 'auth/email-already-in-use') {
43            console.log('That email address is already in use!');
44          }
45
46          if (error.code === 'auth/invalid-email') {
47            console.log('That email address is invalid!');
48          }
49
50          console.error(error);
51        });
52    }
```

Fonte: Elaborado pelo autor

Figura 27 – Método de login no Flutter com exceção na linha 66

```
65  Future<void> login() async {
66    throw new Exception("Exceção intencional.");
67
68    formState.currentState.save();
69    print(await currentUser());
70
71    try {
72      final AuthResult authResult = await _auth.signInWithEmailAndPassword(
73        email: email,
74        password: senha,
75      );
76
77      if (authResult.user != null) {
78        Navigator.pushReplacement(
79          context,
80          MaterialPageRoute(builder: (context) => LoggedPage()),
81        );
82      }
83    } catch (e) {
84      showErrorDialog(e.message);
85    }
86  }
```

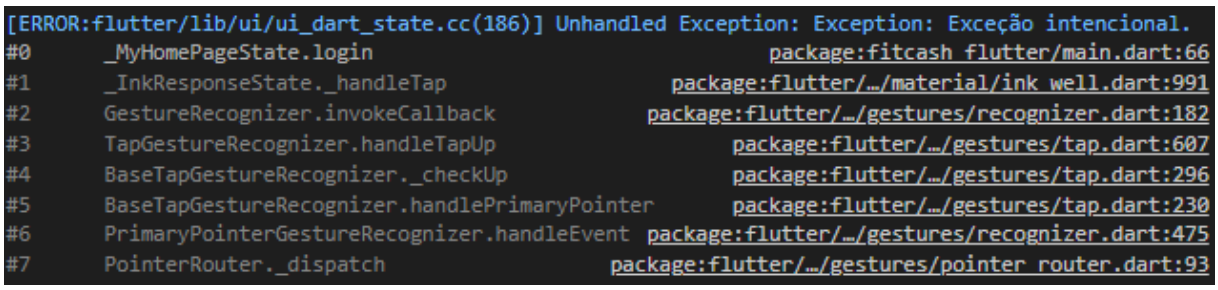
Fonte: Elaborado pelo autor

Como as Figuras 28 e 29 mostram, pode-se notar que o Flutter mostra simplificadamente, onde o erro foi lançado, o que não ocorre com o React Native. Nota-se que apenas na exceção apresentada pelo Flutter, na Figura 29, o método de *login* aparece no topo da pilha, facilitando a depuração do código com erro.

Figura 28 –Pilha de exceções apresentadas no React Native

Fonte: Elaborado pelo autor

Figura 29 – Pilha de exceções apresentadas no Flutter



Fonte: Elaborado pelo autor

7.1.7 Velocidade de Desenvolvimento

Assim como nas seções a respeito da facilidade de aprendizado e manutenção de código, a velocidade de desenvolvimento pode ser afetada por diversos fatores que não são inerentes especificamente aos *frameworks* de desenvolvimento. A velocidade com que as

aplicações são construídas utilizando as ferramentas, pode sofrer impacto dependendo do conhecimento obtido pelos usuários a respeito delas. Além da velocidade de desenvolvimento de fato, podem contribuir significativamente com o tempo para a realização das tarefas, as configurações necessárias para preparar um ambiente de desenvolvimento para cada ferramenta. Por este motivo, esta seção abordará a velocidade de desenvolvimento de maneira diferente, não de fato pelo tempo utilizado em tarefas de construção do *software* propriamente dito, e sim na preparação do ambiente para o início do desenvolvimento.

Na comparação deste quesito foram construídas máquinas virtuais, uma para cada *framework*, com o intuito de mapear as tarefas, assim como o tempo necessário para a geração de um ambiente totalmente configurado, projeto criado, primeiro *build* executado e pronto para o desenvolvimento. A configuração das máquinas virtuais partiu de um mesmo ponto, iniciadas com Windows 10, versão 20H2 (10.0.19042.0) disponibilizada pela Microsoft[5] com algumas ferramentas já instaladas. Dentre estas ferramentas, constam Visual Studio 2019, Visual Studio Code e o modo de desenvolvedor habilitado. Além das ferramentas já instaladas, também foi instalado o Android Studio, que é pré-requisito tanto do React Native quanto do Flutter. Com esta configuração posta, as análises e tempos para a configuração específica de cada *framework* se iniciou.

Para esta configuração de ambiente, foram seguidas à risca as recomendações das documentações[6] dos *frameworks*, portanto além de comparar tempos necessários para o *setup* do ambiente, foram colocadas à prova, a capacidade de explanação a respeito de como os passos deviam ser seguidos, fato que também pode proporcionar uma melhor experiência e velocidade quando bem especificados. Neste quesito, o React Native se mostrou mais performático, levando 26 minutos para ter um projeto compilado e executando em um dispositivo físico, enquanto o Flutter levou 42 minutos. Dentre as tarefas das documentações, elas se equiparam, por exemplo, no *build* final para o dispositivo físico. A grande diferença neste tempo explica-se pelo *download* e descompactação do SDK (*Software Development Kit*) do Flutter, que se mostra muito mais demorado do que o *download* do Node e Java JDK requisitados pelo React Native.

---

[5] https://developer.microsoft.com/pt-br/windows/downloads/virtual-machines/
[6] https://reactnative.dev/docs/environment-setup (React Native)
https://flutter.dev/docs/get-started/install/windows (Flutter)

Em contrapartida, com o ambiente já configurado, para a criação de novos projetos o oposto ocorreu. O React Native levou 2 minutos para realizar o *download* de dependências e finalizar a criação do projeto, enquanto o Flutter levou apenas 20 segundos.

7.1.8 Baixo Custo de Desenvolvimento

O custo de desenvolvimento está diretamente vinculado ao tempo utilizado para a realização das tarefas, mas para a otimização da produtividade no desenvolvimento de *software*, uma série de fatores têm influência. Adotar processos bem definidos de desenvolvimento, melhorar ambiente e infraestrutura, ter uma boa gestão de pessoas e conhecer e aprimorar as tecnologias empregadas são alguns dos princípios definidos pelo *Checkpoint.* Esta é uma ferramenta baseada em conhecimento, que estima tempos para projetos de *software* e apoia-se nos pilares apresentados anteriormente (BOEHM et al., 2000).

Assim como na subseção 7.1.7, aspectos de custo de desenvolvimento podem ser afetados por diversos fatores distintos e que poderiam ser trabalhados em um trabalho exclusivamente voltado para este fim. Como este não é o foco único deste trabalho, na comparação de React Native e Flutter em relação ao custo de desenvolvimento, foi realizado um estudo de mercado buscando avaliar as remunerações de desenvolvedores destes dois *frameworks*.

Para a comparação dos salários dos desenvolvedores foi utilizada a base de salários disponibilizada pela Glassdoor[7]. O resultado apresentado pela base de dados do Glassdoor aponta que a média salarial de desenvolvedores React Native e Flutter são bastante parecidas. Para desenvolvedores de React Native, a média salarial é de R$ 4.677,00/mês enquanto de desenvolvedores Flutter é de R$ 4.176,00/mês. Em ambos os casos os salários iniciam em 1 mil – 2 mil reais para estagiários e profissionais iniciantes e chegam até 10 mil – 11 mil reais para os desenvolvedores mais experientes. Os salários dos profissionais React Native encontram-se mais distribuídos, enquanto os de Flutter estão em sua maior parte distribuídos dos 5 mil reais para baixo, como pode ser visto nas Figuras 30 e 31.

[7] https://www.glassdoor.com.br/

Figura 30 – Salário médio de desenvolvedores React Native

Fonte: Glassdoor (2021)

Figura 31 – Salário médio de desenvolvedores Flutter

Fonte: Glassdoor (2021)

7.1.9 Versionamento de Código

O versionamento de código é um processo indispensável no desenvolvimento de *software*, e por isso torna-se um ponto de suma importância na comparação das ferramentas de desenvolvimento multiplataforma. Freitas (2010) aponta que a Gerência de Configuração de *Software* (GCS), área importante da engenharia de *software*, vem sendo cada vez mais empregada no desenvolvimento das aplicações. Ela é definida como a disciplina que permite manter sob controle a evolução de sistemas, gerenciando e rastreando mudanças e é subdividida em três pilares, controle de modificações, controle de versões e gerenciamento de construção.

Para esta comparação foi utilizado o sistema de versionamento GIT, criado por Linus Torvalds, que é classificado como um sistema de controle de versão descentralizado e um dos mais utilizados atualmente. A ferramenta Visual Studio Code, que foi utilizada no desenvolvimento dos aplicativos em React Native e Flutter possui um módulo já integrado ao editor para a facilitação na utilização do sistema. Assim sendo, foram comparadas as características que possam se diferenciar no versionamento dos códigos dos *frameworks*.

Neste quesito, como foi utilizado o mesmo sistema de versionamento no desenvolvimento dos dois aplicativos, não houve nenhuma particularidade notada durante

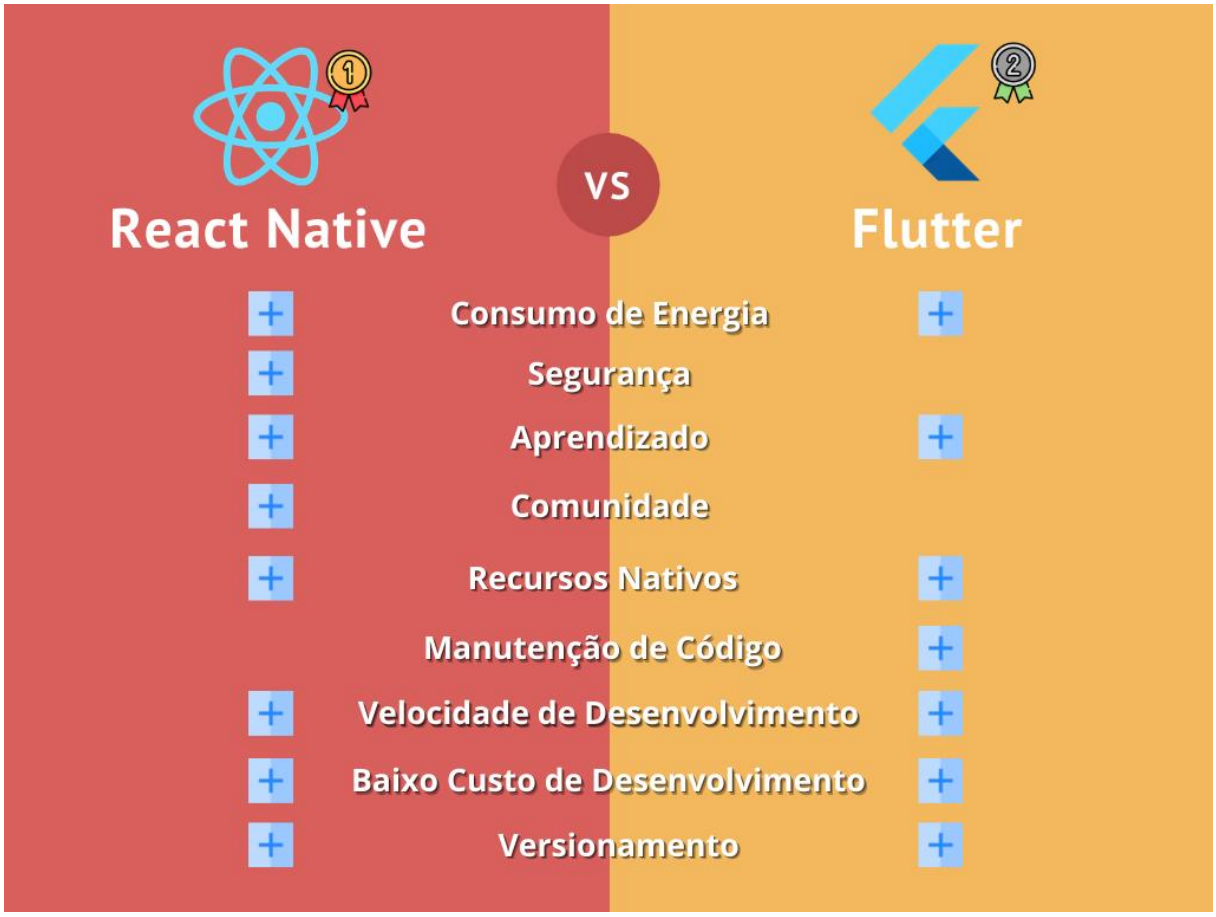
todo o processo. Tanto no React Native quanto no Flutter, por utilizarem apenas arquivos de texto em seu código fonte, o versionamento pode ser feito de maneira bem-sucedida.

Como adendo a este quesito, também foram comparados os tamanhos dos arquivos de instalação dos dois aplicativos – os arquivos .apk – e o tamanho de todo o diretório dos projetos em cada *framework*. O .apk do React Native possui 45.396 KB e o do Flutter, 50.063 KB. Já em relação ao tamanho do projeto como um todo o React Native acumula 959 MB enquanto o projeto Flutter possui 922 MB.

7.2 RESULTADO

Nesta seção será apresentada a Figura 32, sintetizando os resultados da comparação da seção 7.1. Dos nove pontos avaliados neste trabalho, na maior parte deles, os *frameworks* foram equivalentes, mas o aplicativo criado com o *framework* criado em React Native se sobressaiu em relação ao criado com Flutter. Na Figura 32, estão representados ao centro cada um dos quesitos comparados, e baixo de cada *framework,* demarcados pelo símbolo de “+”, qual a ferramenta que se destacou mais ou, como na maior parte dos casos, onde os dois *frameworks* estão demarcados, representando a equivalência.

Figura 32 – Sintetização do comparativo dos *frameworks*



Fonte: elaborado pelo autor

7.3 RECOMENDAÇÕES

Nesta seção são apresentadas recomendações baseadas na experiência de desenvolvimento dos apps com os dois *frameworks.* Além do que foi possível notar ao realizar o desenvolvimento dos aplicativos, também foram utilizados como referência para as recomendações a bibliografia deste trabalho, as pesquisas em documentações e opiniões da comunidade de desenvolvedores do React Native e do Flutter.

Se a segurança é um quesito considerado de extrema importância na aplicação que será construída, e principalmente, se o tempo e recursos dedicados para que este torne-se seguro é escasso, o *framework* recomendado é o React Native. Como demonstrado ao longo do trabalho, a documentação desta ferramenta auxilia seus desenvolvedores na construção de um app seguro, apontando recursos e bibliotecas que facilitam e agilizam o desenvolvimento de aplicativos menos vulneráveis. Ainda assim, como já pontuado em outra ocasião, aplicações impenetráveis não existem.

Assim como no quesito segurança, a comunidade do React Native é amplamente maior e mais difundida do que a do Flutter, pelo menos até o momento. Em parte essa popularidade se dá pois a tecnologia e linguagem utilizada por ele é o Javascript, muito conhecido e difundido no desenvolvimento de aplicações *web*. Portanto se a rápida resolução de problemas, diversificados exemplos de implementações e variados tipos de dúvidas tenham que ser encontrados de maneira facilitada, novamente o React Native é o *framework* mais recomendado. O quesito "Comunidade" deve ser tratado com atenção, visto que a comunidade Flutter cresce exponencialmente a cada dia. Assim sendo, a cada nova aplicação que será construída, este tema poderia ser novamente consultado para verificar se as comunidades já se equivalem.

Finalmente, quando se sabe antecipadamente qual aplicação que será construída, se terá um porte grande, e/ou será integrada com outras aplicações, o que pode aumentar consideravelmente o número de manutenções necessárias, a melhor escolha é o Flutter. Essa escolha se dá pela capacidade deste *framework* de auxiliar na depuração de código, facilitando o trabalho dos desenvolvedores na procura e conserto dos problemas que venham a surgir. Além da facilidade na depuração, ele também auxilia na construção das telas da aplicação, apontando falhas e carregamentos desnecessários ou excessivos.

De maneira geral, a escolha do *framework* ideal é dependente da preferência dos desenvolvedores em alguns quesitos. Após estas recomendações e ainda não tendo clara qual escolha realizar, recomenda-se o desenvolvimento de pequenas aplicações, utilizando o básico de cada um deles para entender o funcionamento e seus conceitos. Após esta experiência definir qual será utilizado. O que pode ser afirmado é que ambos *os frameworks* atenderam às necessidades do desenvolvimento da aplicação escolhida para este trabalho.

8 CONCLUSÃO

A análise comparativa dos *frameworks* de desenvolvimento multiplataforma buscou responder questões frequentes da comunidade de desenvolvimento de aplicações *mobile*. Através destes questionamentos, formularam-se os objetivos deste trabalho, que partiram do objetivo principal - avaliar e comparar dois frameworks de desenvolvimento multiplataforma, a fim de auxiliar na decisão de qual ferramenta utilizar em cada projeto. Os objetivos específicos do trabalho, listavam pesquisar trabalhos relacionados, entrevistar desenvolvedores, definir parâmetros de comparação, definir os *frameworks* avaliados, construção de aplicações como casos de uso, e finalmente, discutir, comparar, recomendar e apresentar as aplicações finalizadas.

Para atingir os objetivos propostos, foram seguidos alguns passos. Inicialmente se desenvolveu uma pesquisa a respeito do desenvolvimento *mobile*, sua história, como encontra-se o mercado atual e as principais arquiteturas de desenvolvimento. Após esta introdução, voltou-se a pesquisa para os *frameworks* de desenvolvimento multiplataforma. Comparou-se os conceitos de desenvolvimento nativo em relação à multiplataforma e foram apresentados os principais *frameworks* utilizados no mercado.

Além da pesquisa bibliográfica, foi realizada uma pesquisa quantitativa buscando conhecer as necessidades e características que desenvolvedores e líderes de equipe consideram essenciais em *frameworks* de desenvolvimento multiplataforma. Desta forma, foi possível desenvolver um aplicativo nos *frameworks* definidos e compará-los levando estes pontos em consideração. Após desenvolver os aplicativos, implementando as funcionalidades que explorassem as características necessárias, foi possível compará-los e recomendá-los baseado na experiência obtida e com base na pesquisa realizada.

Como resultado, conclui-se que quando comparados, React Native e Flutter, quem se sai melhor nos quesitos definidos é o React Native. Sob os 9 temas comparados, ele mostrou-se superior em 2, enquanto o Flutter em apenas 1. No restante dos quesitos as ferramentas se equipararam. Assim sendo, as diferenças entre eles foram mínimas, podendo ser afirmado que ambos são ótimas opções para o desenvolvimento multiplataforma. As recomendações dadas baseiam-se nos quesitos que cada *framework* se mostrou superior. Portanto, o React Native foi recomendado para a construção de aplicações seguras e para desenvolvedores/equipes que buscam uma comunidade ampla e consistente. Já o Flutter, recomendado para aplicações onde se sabe previamente que haverá constantes manutenções no código produzido.

Como trabalhos futuros, podem ser avaliados qualquer um dos quesitos desse trabalho individualmente. Ao longo do desenvolvimento deste em diversas situações entendia-se que os assuntos poderiam ser tratados com muito mais detalhe, mas para que o objetivo fosse seguido acabava-se apenas observando o necessário e seguindo o trabalho. Um exemplo deste cenário é o quesito consumo de energia, que pode ser observado sob diversos aspectos e neste trabalho apenas pode-se analisar enquanto o GPS era utilizado. Cada um dos quesitos definidos também poderia ser comparado exclusivamente com o desenvolvimento nativo com a intenção de avaliar as diferenças entre os paradigmas de desenvolvimento.

Outro potencial trabalho futuro seria a realização de uma comparação de velocidade e complexidade no desenvolvimento de algumas tarefas básicas e frequentes na criação das aplicações.

O estudo desenvolvido possui limitações, pois todos os testes e comparações foram efetuadas sobre a plataforma Android. Esse fato confere-se como limitação pois os resultados em alguns quesitos poderiam ser afetados caso os testes fossem realizados em cada plataforma disponível no mercado.

REFERÊNCIAS BIBLIOGRÁFICAS

AHMAD, R. W. et al. **A survey on energy estimation and power modeling schemes for smartphone applications**. International Journal of Communication Systems, 2017.

ANSONG, Edward Danso; Synaepa-Addision, Thomas Quansah. **A Comparative Study Of User Data Security and Privacy in Native and Cross Platform Android Mobile Banking Applications.** Kwane Nkrumah University of Science and Technology, 2019 International Conference on Cyber Security and Internet of Things (ICSIoT), Kumasi, Ghana, 2019.

BANKMYCELL. **How many smartphones are in the world? November 2020 mobile user statistics: discover the number of phones in the world & smartphone penetration by country region**, 2020. Disponível em: <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world#1579705085743-b3697bdb-9a8f/>. Acesso em: 17 nov. 2020.

BARNES, Micaela C.; MEYERS, Neil P. **Mobile Phones**: Technology, Networks and User Issues. Nova Science Publishers, Inc, 2011. 268 p.

BASSETTO, Giovanni. **Comparativo entre frameworks para desenvolvimento multiplataforma.** Universidade Estadual do Norte do Paraná, 2019.

BOEHM, Barry; ABTS, Chris; CHULANI, Sunita. **Software develoment cost estimation approaches – A survey**. University of Southern California, IBM Research, Annals of Software Engineering, 2000.

BRUCK, Peter A.; RAO, Madanmohan. **Global Mobile**: Applications and Innovations for the Worldwide Mobile Ecosystem., 2013.

COUTO, M. et al. **Analyzing and Classifying Energy Consumption in Android Applications**., 2019.

CRUZ, L. M. d. **Tools and Techniques for Energy-Efficient Mobile Application Development.** Universidade do Porto., 2019.

DHILLON, Sunny; MAHMOUD, Qusay H. **An evaluation framework for cross-platform mobile application development tools.** School of Computer Science, University of Guelph, Guelph, ON, Canada, 2014.

FLUTTER. **Flutter Dev**., 2020. Disponível em:<https://flutter.dev/>. Acesso em: 17 nov. 2020.

FREITAS, Daniel Tannure Menandro De. **Análise comparativa entre sistemas de controle de versões.** Universidade Federal de Juiz de Fora, Juiz de Fora, 2010.

GOOGLE. **Android Developers**: Arquitetura da plataforma., 2020. Disponível em: <https://developer.android.com/guide/platform>. Acesso em: 17 nov. 2020.

GOOGLE. **Google Trends**., 2020. Disponível em: <https://trends.google.com.br/trends/explore?geo=BR&q=react%20native,flutter,cordova,ionic,xamarin>. Acesso em: 17 nov. 2020.

GUPTA, Diwaker. **What is a good first programming language?.** Crossroads, The ACM Student Magazine, 2011.

HARTMANN, Gustavo; STEAD, Geoff; DEGANI, Asi. **Cross-platform mobile development.** Medical Mobile Development Project, 2011.

HEITKÖTTER, Henning; HANSCHKE, Sebastian; MAJCHRZAK, Tim A. **Evaluating Cross-Platform Development Approaches for Mobile Applications.** WEBIST, 2013.

HORA, André. **Manutenção de Software: Introdução.** Departamento de Ciência da Computação, UFMG, 2019.

MAJCHRZAK, Tim A.; BIØRN-HANSEN, Andreas; GRØNLI, Tor-Morten. **Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks.** Proceedings of the 50th Hawaii International Conference on System Sciences, 2017.

MEDIUM; CALDERA, Anuradh. **IOS Architecture**: iOS is based on Mac OS X. development languages are swift, objective-c and c. and it's not an open source., 2018. Disponível em: <https://medium.com/@anuradhs/ios-architecture-a2169dad8067>. Acesso em: 17 nov. 2020.

MERCADO, Ván Tactuk; MUNAIAH, Nuthan; MENEELY, Andrew. **The Impact of Cross-Platform Development Approaches for Mobile Applications from the User's Perspective.** Department of Software Engineering Rochester Institute of Technology, Rochester, New York, USA, 2016.

NOVAC, Ovidiu Constantin; NOVAC, Mihaela; GORDAN, Cornelia; BERCZES, Tamas. **Comparative Study of Google Android, Apple iOS and Microsoft Windows Phone Mobile Operating Systems**. 2017 14th International Conference on Engineering of Modern Electric Systems (EMES), 2017.

PALMIERI, Manuel; SINGH, Inderjeet; CICCHETTI, Antonio. **Comparison of Cross-Platform Mobile Development Tools.** 2012 16th International Conference on Intelligence in Next Generation Networks, 2012.

PINTO, G.; CASTOR, F. **Energy Efficiency: A New Concern for Application Software Developers.** Communications of the ACM, 2017.

PREZOTTO, Ezequiel Douglas; BONIATI, Bruno Batista. **Estudo de Frameworks Multiplataforma Para Desenvolvimento de Aplicações Mobile Híbridas.** Anais do EATI - Encontro Anual de Tecnologia da Informação, 2014.

PRODANOV, Cleber. **Manual de Metodologia Científica.** 2ª ed. Novo Hamburgo: FEEVALE, 2013.

REACT NATIVE. **React Native**: Learn once, write anywhere., 2020. Disponível em: <https://reactnative.dev/>. Acesso em: 17 nov. 2020.

REACT NATIVE. **React Native**: Security., 2021. Disponível em: <https://reactnative.dev/docs/security>. Acesso em: 26 abr. 2021.

RIEGER, Christoph; MAJCHRZAK, Tim A. **Towards the definitive evaluation framework for cross-platform app development approaches.** The Journal of Systems and Software, 2019.

SAMBASIVAN, Divya; JOHN, Nikita; UDAYAKUMAR, Shruthi; GUPTA, Rajat. **Generic Framework for mobile application development**. IEEE, 2011.

SILVA, Marcelo Moro da; SANTOS, Marilde Terezinha Prado. **Os Paradigmas de Desenvolvimento de Aplicativos para Aparelhos Celulares.** Departamento de Computação - Universidade Federal de São Carlos (UFSCar), 2014.

STATCOUNTER. **Mobile Vendor Market Share Worldwide - October 2020**., 2020. Disponível em: <https://gs.statcounter.com/vendor-market-share/mobile/worldwide>. Acesso em: 17 nov. 2020.

STATCOUNTER. **Mobile Vendor Market Share in United States of America - October 2020**., 2020. Disponível em: <https://gs.statcounter.com/vendor-market-share/mobile/united-states-of-america>. Acesso em: 17 nov. 2020.

STATCOUNTER. **Mobile Vendor Market Share in Brazil - October 2020**., 2020. Disponível em: <https://gs.statcounter.com/vendor-market-share/mobile/brazil>. Acesso em: 17 nov. 2020.

STATISTA, **Number of smartphone users from 2016 to 2021.** [*S. l.*], 2019. Disponível em: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>. Acesso em: 17 nov. 2020.

STATISTA; LIU, Shanhong. **Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020**., 2011. Disponível em: <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>. Acesso em: 17 nov. 2020.

TELECO. **Estatísticas de Celulares no Brasil**., 2020. Disponível em: <https://www.teleco.com.br/ncel.asp>. Acesso em: 17 nov. 2020.

THE NEW YORK TIMES; AUSTEN, Ian. **BlackBerry Abandons Its Phone**., 2016. Disponível em: <https://www.nytimes.com/2016/09/29/technology/blackberry-phones-earnings-q2.html>. Acesso em: 17 nov. 2020.

THE VERGE; WARREN, Tom. **Microsoft to end Windows 10 Mobile updates and support in December**: Microsoft recommends iOS or Android devices as replacements., 2019. Disponível em: <https://www.theverge.com/2019/1/18/18188054/microsoft-windows-phone-windows-10-mobile-end-of-support-updates>. Acesso em: 17 nov. 2020.

XAMARIN. **Xamarin**: Free. Cross-platform. Open source. An app platform for building Android and iOS apps with .NET and C#., 2020. Disponível em: <https://dotnet.microsoft.com/apps/xamarin>. Acesso em: 17 nov. 2020.

WU, H.; YANG, S.; ROUNTEV, A. **Static Detection of Energy Defect Patterns in Android Applications**. International Conference on Compiler Construction, 2016.