UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ANTONIO EUDÁLIO DE SOUSA DA SILVA

ANÁLISE COMPARATIVA ENTRE OS FRAMEWORKS DE DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

QUIXADÁ

2021

ANTONIO EUDÁLIO DE SOUSA DA SILVA

ANÁLISE COMPARATIVA ENTRE OS FRAMEWORKS DE DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

Monografia apresentada ao curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientadora: Profa. Ma. Antonia Diana Braga Nogueira

QUIXADÁ

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S578a Silva, Antonio Eudálio de Sousa da.
Análise comparativa entre os frameworks de desenvolvimento de aplicativos móveis multiplataforma. / Antonio Eudálio de Sousa da Silva. – 2021.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2021.
Orientação: Profª. Ma. Antonia Diana Braga Nogueira.

1.Multiplataforma-Desenvolvimento. 2. Framework (Arquivo de computador). 3. JavaScript (Linguagem de programação de computador). 4. Aplicativos móveis.

CDD 005

ANTONIO EUDÁLIO DE SOUSA DA SILVA

ANÁLISE COMPARATIVA ENTRE OS FRAMEWORKS DE DESENVOLVIMENTO MÓVEL MULTIPLATAFORMA

Monografia apresentada ao curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em: ____/____/_____.

BANCA EXAMINADORA

_____
Profª. Ma. Antonia Diana Braga Nogueira (Orientadora)
Universidade Federal do Ceará (UFC)

_____
Profª. Dra. Paulyne Matthews Jucá
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará (UFC)

Dedico este trabalho primeiramente à Deus, minha mãe e minha esposa por terem me dado todo o incentivo e ajuda para que eu chegasse aqui, contribuindo com o meu crescimento e aprendizado.

AGRADECIMENTOS

Agradeço primeiramente a Deus meu Senhor e dono de tudo, que com seu amor e misericórdia me concedeu o dom da vida e que em todos os momentos da minha existência, me segura pela mão e demonstra de várias maneiras que me ama e está comigo.

Minha eterna gratidão a minha mãe Eudália de Sousa, pelo amor, educação, lições, tempo e por ter sido minha mãe e meu pai e acima de tudo, sempre me incentivou a estudar e buscar um futuro melhor para que pudesse realizar todos os meus sonhos.

Agradeço a esta universidade, seu corpo docente, direção e administração que oportunizaram este momento, e de forma indiscutível foram co-responsáveis pelo crescimento intelectual, profissional e pessoal de tantos alunos.

Ao Prof. Me. Antonia Diana Braga Nogueira, pela excelente orientação. Sou grato porque ao longo do trabalho se mostrou sempre disposta a ajudar e encontrar a melhor solução para os problemas.

Aos professores participantes da banca examinadora Diana Braga, Jefferson Carvalho e Paulyne Jucá pelo tempo, pelas valiosas colaborações e sugestões.

Aos professores Marcos Dantas e Marcio Maia por terem sido bons orientadores no período em que fui bolsista, aprendi muito através dos senhores.

Agradeço de forma especial a minha esposa Leiliane Silva, que me acompanha desde o início desta jornada, sempre me incentivando, apoiando e cobrando para que eu fizesse o meu melhor a cada dia. Seu apoio foi e é de fundamental importância em minha vida, obrigado meu amor.

“Dificuldades preparam pessoas comuns para destinos extraordinários.” (CS. LEWIS, 1898 – 1963)

RESUMO

Com o avanço nas tecnologias móveis, em paralelo ocorreu uma demanda crescente por aplicativos móveis, obrigando o mercado a oferecer produtos de boa qualidade, com o menor tempo e custo possível. Atualmente existem duas plataformas principais, o Android e o iOS. Neste trabalho, o comparativo foi feito em relação ao desenvolvimento com Android, com dois dos frameworks mais modernos: o React Native e o Flutter. Essa comparação tem como objetivo ajudar os desenvolvedores a escolherem o framework mais adequado para utilizar em seus projetos, bem como resolver o seu problema. Para fazer a comparação foi levantado uma série de critérios a partir de trabalhos relacionados, e, para cada um dos critérios, foi aplicado um peso, a fim de determinar o nível de relevância para a comparação. Os critérios foram escolhidos com o intuito de ter uma noção geral da ferramenta em aspectos como: infraestrutura, desenvolvimento, performance e usabilidade. A comparação mostra que os dois frameworks são capazes de atender as necessidades dos mais diversos projetos, que possuem uma comunidade engajada e a tendência é que esses frameworks se aperfeiçoem e cresçam ainda mais no curto/médio prazo. Além disso, entre os resultados está o fato de que o Flutter se destacou em termos de performance e velocidade de renderização, enquanto o React Native se sobressai em termos de tecnologia por utilizar tecnologias populares como HTML, CSS e JavaScript que favorecem a velocidade do aprendizado e desenvolvimento com o framework.

Palavras-chave: Multiplataforma-Desenvolvimento. Framework (Arquivo de computador). JavaScript (Linguagem de programação de computador). Aplicativos móveis.

ABSTRACT

With the advancement in mobile technologies, in parallel there has been an increasing demand for mobile applications, forcing the market to offer good quality products, in the shortest possible time and cost. There are currently two main platforms, Android and iOS. In this work, the comparison was made in relation to the development with Android, with two of the most modern frameworks: React Native and Flutter. This comparison aims to help developers choose the most appropriate framework to use in their projects, as well as solve their problem. In order to make the comparison, a series of criteria was raised based on related works, and, for each of the criteria, a weight was applied in order to determine the level of relevance for the comparison. The criteria were chosen in order to have a general notion of the tool in aspects such as: infrastructure, development, performance and usability. The comparison shows that the two frameworks are capable of meeting the needs of the most diverse projects, which have an engaged community and the tendency is for these frameworks to improve and grow even more in the short / medium term. In addition, among the results is the fact that Flutter stood out in terms of performance and rendering speed, while React Native stands out in terms of technology for using popular technologies such as HTML, CSS and JavaScript that favor the speed of learning and development with the framework.

Keywords: Multiplatform-Development. Framework (Computer file). JavaScript (Computer programming language). Mobile apps.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com a constante expansão da internet e dos dispositivos a ela conectados, o mercado tem buscado diversas maneiras de lançar aplicações que atendam a demanda crescente de soluções digitais. Em se tratando de dispositivos móveis, é de conhecimento popular que a presença no mercado global de sistemas operacionais móveis é dominada principalmente por dois sistemas operacionais: Android, fabricado pelo Google; e iOS, fabricado pela Apple. Estes dois sistemas juntos tinham mais de 99% de participação de mercado até abril de 2020, de acordo com (STATCOUNTER, 2020).

Devido às diferenças entre esses dois sistemas operacionais, o desenvolvimento de aplicativos nativos deve ser feito separadamente. Para atender tal demanda, as empresas enfrentam o desafio de encontrar equipes de desenvolvimento flexíveis, com desenvolvedores habilitados ao menos em duas linguagens de programação e com conhecimentos aprofundados em diversas características específicas de cada plataforma. Estas divergências, além de dificultarem o desenvolvimento de aplicações compatíveis em ambas as plataformas, aumentam a probabilidade de erros. Também é de se destacar a provável elevação de custos para o desenvolvimento de duas soluções distintas, bem como o custo para manutenção e solução de *bugs* em bases de código diferentes, mas com um mesmo objetivo final. (CRUZ; PRETUCELLI, 2018).

Neste contexto, surgem as aplicações híbridas, em que se desenvolve apenas uma aplicação, que pode ser utilizada em vários dispositivos com diferentes sistemas operacionais. Isso é possível devidos aos *frameworks* para desenvolvimento de aplicações híbridas, que são responsáveis por empacotar o código-fonte para as diferentes plataformas, permitindo que elas sejam instaladas no dispositivo e possam acessar os seus recursos, como câmera, GPS e contatos (PREZOTTO; BONIATI, 2017).

Na sequência do trabalho é apresentado, respectivamente, objetivos, trabalhos relacionados, fundamentação teórica, metodologia e pesquisa e, no fim, os resultados obtidos acerca deste estudo.

2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos que almeja-se atingir ao fim deste trabalho.

2.1 Objetivos gerais

Realizar uma análise comparativa entre os principais frameworks de desenvolvimento móvel multiplataforma disponíveis no mercado, a fim de apresentar a comunidade de desenvolvedores, bem como as fábricas de software, alternativas para o desenvolvimento móvel e para auxiliar no processo de escolha de qual framework utilizar em seus projetos.

2.2 Objetivos específicos

- Definir os frameworks a serem estudados;
- Levantar os critérios e métricas a serem utilizados para a comparação;
- Implementar uma aplicação prática;
- Apresentar o resultado obtido com a comparação a partir dos critérios estabelecidos.

3 TRABALHOS RELACIONADOS

Nesta seção, serão apresentados estudos correlatos, na quais serviram como base para consolidar a proposta deste estudo.

3.1 Evaluation of React Native and Flutter for cross-platform mobile application development

O estudo de Hjort (2020) faz uma análise dos frameworks React Native e Flutter, com a intenção de avaliar e determinar qual deles é o mais adequado para ser utilizado na empresa de desenvolvimento de software Gambit. Como o trabalho foi realizado em parceria com a empresa Gambit, se fez necessário que o framework escolhido atenda os necessidades da empresa, entre elas: ser altamente personalizável, ter acesso a uma grande quantidade de funcionalidades, apresentar viabilidade no longo prazo e ter uma curva de aprendizagem baixa, visto que a equipe é familiarizada com desenvolvimento web e móvel.

A fim de realizar a avaliação, Hjort (2020) desenvolveu um aplicativo utilizando os dois frameworks, onde o aplicativo precisaria ser capaz de acessar a câmera, gravar um áudio usando o microfone, identificar a localização e exibir isso em um mapa, além de ser capaz de detectar movimentos usando o acelerômetro e o giroscópio, para testar a capacidade dos frameworks de utilizarem os recursos do dispositivo. Os critérios para comparação dos frameworks foram selecionados em conjunto com a empresa Gambit, e foram divididos em duas categorias: Perspectiva de desenvolvimento e perspectiva de aplicação, onde a primeira se atém a aos aspectos presentes no processo de desenvolvimento e a segunda categoria vai analisar os aspectos que impactam o produto final. Para cada critério de avaliação, foi estipulado um peso de um a cinco, onde um é “muito ruim” e cinco é “muito bom”, a ideia de colocar pesos é para definir o nível de relevância de cada critério para a avaliação.

Dentre os resultados obtidos pelo estudo feito por Hjort (2020), estão: as duas ferramentas se provaram ser opções perfeitamente viáveis para o desenvolvimento nativo, os frameworks multiplataforma entregam um processo de desenvolvimento mais simples se comparado com os frameworks nativos, e por fim, o principal resultado de seu trabalho foi que o React Native se apresentou como a alternativa mais adequada, pois, obteve uma pontuação final maior, se mostrando segundo os critérios e ponderações levantadas, a melhor opção para ser usada na empresa Gambit.

3.2 Estudo de Frameworks Multiplataforma para desenvolvimento de aplicações mobile híbridas

O trabalho de Prezotto e Boniati (2017) demonstra os diferentes tipos de desenvolvimento de aplicações para dispositivos móveis (com enfoque em aplicações híbridas), expondo sempre as características para cada tipo de desenvolvimento. Os autores desenvolveram um aplicativo utilizando a ferramenta multiplataforma Apache Cordova e concluíram que a abordagem multiplataforma não é melhor que a nativa, mas que há situações onde cada abordagem se encaixa melhor, deixando claro dessa forma, que a responsabilidade de escolher qual alternativa usar está sob o desenvolvedor.

Outra conclusão do trabalho foi a de que as aplicações híbridas, podem, através de um mesmo código fonte, originar aplicações para diferentes plataformas. O autor também constatou que uma aplicação híbrida possui acesso a todos recursos do dispositivo e pode também ser disponibilizada juntamente com as aplicações nativas nas lojas de aplicativos. Contudo relatou que elas não se comportam da mesma maneira, ressaltando a importância de analisar minuciosamente os requisitos para identificar se a plataforma dará ou não suporte a eles.

3.3 Tecnologias web para o desenvolvimento mobile nativo

Em Cruz e Pretucelli (2018), há um estudo comparativo teórico sobre desenvolvimento multiplataforma, utilizando três tecnologias: React Native, NativeScript e Weex.

No estudo, foram analisados através de uma revisão bibliográfica artigos acadêmicos e livros sobre as tecnologias em questão. Além disso, devido a carência de material publicado, foram analisadas também as documentações oficiais disponibilizadas pelas equipes responsáveis pelas tecnologias. Para analisar cada tecnologia, foram elencados alguns pontos, como: Breve introdução sobre a ferramenta, as características gerais de uso durante o desenvolvimento, o funcionamento interno da comunicação com o sistema operacional móvel, as características de conectividade à Internet, os conhecimentos prévios necessários para adoção da ferramenta e uma reflexão sobre possíveis vantagens e desvantagens.

Ao final do estudo comparativo, observou-se que os desenvolvedores realmente podem usufruir das soluções para compilação de aplicativos móveis nativos a partir do desenvolvimento em plataforma Web, porém, deixou claro que a maturidade destas

ferramentas ainda pode ser questionada, e que a utilização de tais ferramentas em projetos reais para ambientes em produção devem ser criticamente analisadas antes de começar, pois apresentam uma carga de risco alta, devido as mesmas ainda estarem em franco desenvolvimento e a ainda há incertezas quanto à continuidade dos projetos.

3.4 Comparativo entre frameworks para desenvolvimento multiplataforma

No trabalho realizado por Basseto (2019), é apresentado uma comparação entre os frameworks de desenvolvimento multiplataforma: React Native, Flutter e Xamarim. Onde o principal objetivo do autor era apresentar à comunidade as características, vantagens e desvantagens, e testar a viabilidade de se utilizar um desses frameworks para desenvolver aplicativos móveis.

Como o trabalho se propunha a analisar os principais frameworks, a escolha seguiu algumas características que o autor considerava obrigatórias, bem como pesquisa de popularidade em sites como: Stackoverflow e Github. Para realizar a comparação, foi desenvolvido um aplicativo com as mesmas funcionalidades em cada framework. Para comparação foram utilizados alguns critérios inspirados por outros autores e para efeito de classificação, para cada métrica analisada foi aplicado uma nota de 1 a 3, sendo 1 o melhor e 3 o pior.

Ao fim do trabalho o autor não evidenciou dentre os três qual seria o melhor, basicamente ele concluiu que a escolha do framework a ser utilizado vai depender muito do propósito e/ou do objetivo do projeto. Contudo, se observarmos a pontuação definida para os framework em cada uma das métricas observadas na pesquisa, pode-se ver que o Flutter obteve a menor quantidade de pontos, colocando-o como o melhor, seguido do React Native e do Xamarim, respectivamente.

4 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, alguns aspectos teóricos relacionados ao trabalho são apresentados. Na Seção 3.1, são descritos conceitos iniciais a respeito do desenvolvimento móvel. Na Seção 3.2, são apresentadas algumas ferramentas que permitem e dinamizam o desenvolvimento de aplicativos móveis multiplataforma.

4.1 Desenvolvimento Móvel

Segundo Ferreira et al. (2018), é chamado aplicação móvel todo e qualquer software destinado a ser executado em um dispositivo móvel, normalmente obtido em lojas de aplicativos existentes para cada plataforma. Ao iniciar um novo projeto o desenvolvedor precisa determinar qual estratégia de desenvolvimento vai seguir. Os aplicativos móveis podem ser criados de várias maneiras. Em Xanthopoulos e Xinogalos (2013), as abordagens são divididas em: Nativa, Web, Híbrida, Interpretada e Gerada. Já em Hjort (2020), as abordagens são classificadas em: Aplicações nativas e aplicações multiplataforma, onde a última ainda é dividida em quatro categorias (web, híbrido, interpretado e compilado cruzado).

4.1.1 Desenvolvimento Nativo

Para El-Kassas (2017), os aplicativos nativos “são desenvolvidos usando as ferramentas e linguagens de programação fornecidas para uma determinada plataforma móvel. Esses aplicativos são executados apenas em celulares com a plataforma de destino”. Em resumo, os aplicativos móveis desenvolvidos de maneira nativa, são construídos usando a estrutura oficial e o Software Development Kit (SDK) do sistema operacional. Eles seguem as instruções específicas do sistema operacional em relação à aparência, estrutura e fluxo de navegação. Os aplicativos nativos normalmente são rápidos, responsivos, oferecem melhor desempenho, experiência e são capazes de fazer uso completo dos recursos de hardware e software, uma vez que se comunicam diretamente com os componentes de interface de usuário e tais componentes são personalizados especificamente para sua plataforma (HJORT, 2020.

O ambiente de desenvolvimento integrado (IDE) oficial do Android é o Android Studio (ANDROID, 2020), e a linguagem de desenvolvimento é Java ou Kotlin, onde o último tem obtido maior destaque recentemente. Os desenvolvedores são incentivados a seguir as diretrizes de material design do Google para obter uma aparência e um

comportamento familiar aos usuários do Android. No caso do iOS a IDE oficial é denominada Xcode, e as linguagens com suporte para desenvolvimento são Swift e Objective-C. No caso do iOS, a Apple é quem define as diretrizes de interface (UI), responsáveis por definirem a aparência de um aplicativo iOS (CUNHA, 2018). As duas IDE's auxiliam ajudam e proporcionam maior velocidade no processo de desenvolvimento, além disso, devido a integração de todo o ecossistema dos produtos e serviços da Apple, a sua IDE oferece a possibilidade de publicar aplicativos na loja de aplicativos diretamente da interface de desenvolvimento (FENTAW, 2020).

4.1.2 Desenvolvimento Multiplataforma

Os meios de desenvolvimento multiplataforma só existem por conta dos problemas encontrados no desenvolvimento nativo, entre os principais: o fato de ser caro, precisa manter duas bases de código e requer tempo e habilidades diferentes para cada plataforma (HJORT, 2020).

- **Aplicações web (Web App):** Basicamente são aplicações que rodam no *browser* do dispositivo. Dessa forma, não é necessário se preocupar com as diferentes linguagens, já que utiliza padrões web como HTML, CSS e JavaScript para o lado do cliente e PHP, Java ou outras linguagens para o lado servidor (PREZOTTO; BONIATI, 2017). HTML define o conteúdo e monta a estrutura de uma página da web. Ele usa tags predefinidas para definir o layout e os elementos da página (MOZILLA, 2021). O CSS é usado para definir o estilo de um documento HTML, declarando regras sobre como os elementos selecionados devem ser exibidos (W3C, 2021). JavaScript é uma linguagem de script para páginas web do lado do cliente, porém, pode ser usada em outros contextos sem um browser, como em Node.js. Assim o código-fonte pode ser processado em um servidor web, bem como pelo navegador do cliente (MOZILLA, 2021). Sua maior vantagem é a portabilidade, já que todas as plataformas (tanto móvel quanto desktop) possuem um browser compatível com padrões do W3C (FERREIRA et al., 2018). Uma outra vantagem, segundo (EL-KASSAS et al. 2017), é que essa abordagem é fácil de aprender, desenvolver e manter por se tratar de tecnologias web e suas atualizações serem feitas no lado do servidor. Dentre as desvantagens estão: o acesso limitado a APIs nativas e quaisquer recursos do dispositivo, exigência de

conexão com a internet para o funcionamento e não podem ser instalados no dispositivo (HJORT, 2020).

- **Progressive Web App:** São web apps aprimorados com recursos semelhantes aos nativos. Um PWA é criado adicionando um manifesto de aplicativo da web, incluindo o nome do aplicativo, url e ícones, entre outras informações. Os PWA trouxeram recursos que antes eram exclusivos de aplicativos nativos, como: Adicionar a tela inicial e notificações push, além disso, eles foram projetados para funcionar independentemente da situação da conexão do dispositivo, notificações push estão disponíveis por meio de APIs da web. Os PWAs procuram proporcionar aos usuários a mesma experiência com a qual estão acostumados a verem nos aplicativos nativos, o que significa que eles podem imitar a funcionalidade nativa, mas não se comunicam com as APIs nativas do Android ou iOS, tudo acontece no navegador (MOZILLA, 2021).
- **Aplicações híbridas:** A abordagem híbrida mescla as abordagens nativa e web. Em essência são aplicativos web acoplados dentro de uma webview. Uma webview pode ser associada a um navegador, em outras palavras o app híbrido incorpora um navegador com conteúdo HTML5 dentro de um contêiner nativo, seja o contêiner um UIWebView no iOS ou WebView no Android. Para o usuário final, elas se comportam como aplicações nativas, podendo ser baixadas via loja de aplicativos, instaladas no dispositivo e o acesso a hardware e aos dados é possível por meio de APIs especializadas. Um exemplo de frameworks que utilizam essa abordagem é o Ionic (CUNHA, 2018). Apesar dos aplicativos híbridos serem criados utilizando tecnologias web, estas aplicações não são Web Apps, elas ficam instaladas no dispositivo e ao serem acionadas chamam o *browser*, onde serão executadas, independente de estarem conectadas à internet (PREZOTTO; BONIATI, 2017). Contudo, a execução do aplicativo é feita no navegador, portanto, o desempenho pode ser ruim (HJORT, 2020).
- **Aplicações interpretadas:** Comumente conhecida como aplicativo nativo da web (HJORT, 2020). Nessa abordagem a exemplo da abordagem híbrida, a ferramenta conta com uma ponte que permite a geração de código nativo a partir de uma codificação inicial em linguagens/tecnologias Web, resultando em uma versão específica para cada plataforma (FERREIRA et al., 2018). Diferente dos aplicativos híbridos, a ponte é usada para renderizar

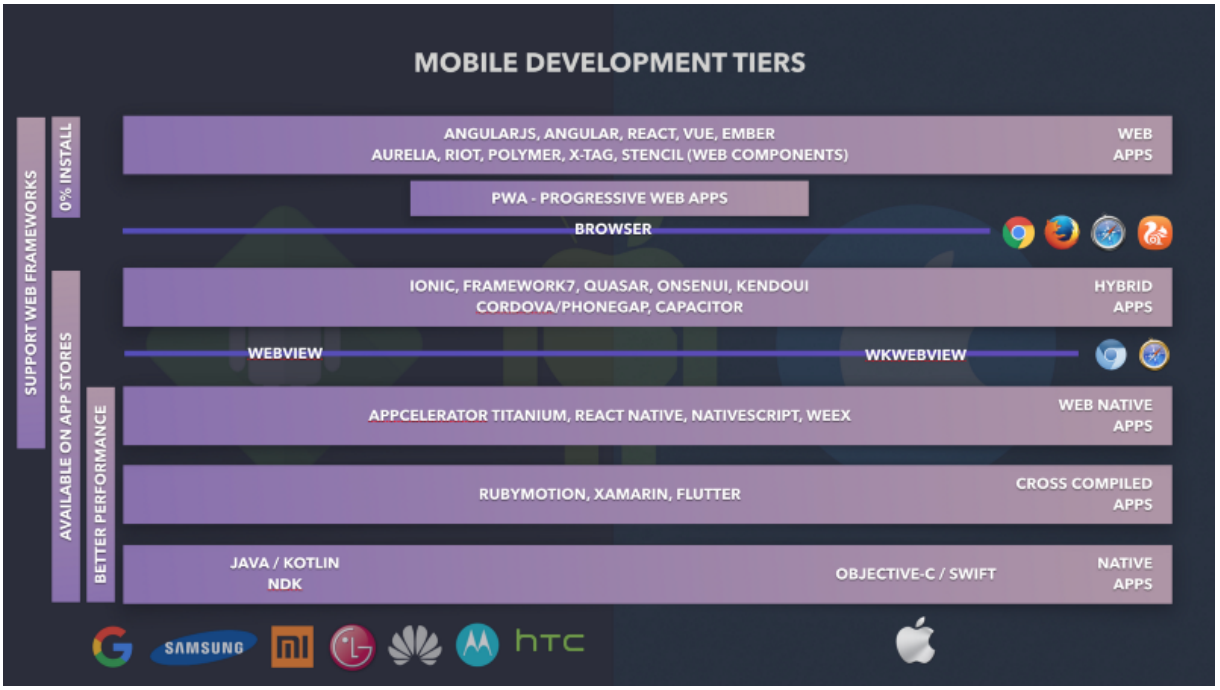
componentes de interface nativos, com isso, nenhuma visualização web é utilizada para renderizar ou executar o aplicativo (HJORT, 2020). Na prática, ocorre uma transpilação (compilação de uma linguagem para outra) do código comumente escrito em Javascript para as linguagens nativas (Java e Swift), de forma que ocorre um empacotamento, minificação, fatiamento e ofuscação do código fonte, tudo simultâneo. Essa abordagem tende a alcançar melhor desempenho de UI por conta do código nativo gerado, onde todos os componentes são iguais aos usados por aplicativos nativos, o que proporciona uma melhor experiência aos usuários finais (CUNHA, 2018).

- **Aplicações compilado cruzado:** Nessa abordagem, os aplicativos gerados são compilados como um aplicativo nativo e com isso uma versão específica da plataforma do aplicativo é criada para cada plataforma de destino (XANTHOPOULOS, XINOGALOS, 2013). Por conta disso, essa abordagem tem o mesmo tempo de execução de aplicações nativas. Normalmente isso proporciona um desempenho melhor do que outras abordagens, porém como o código nativo é gerado automaticamente, pode acontecer alguma complicação ao acessar uma API nativa específica para um determinado recurso (HJORT, 2020). Atualmente as principais ferramentas que se encaixam nessa abordagem são: o Flutter e o Xamarim (CUNHA, 2018).

Em resumo, todas essas abordagens para o desenvolvimento de multiplataforma possuem o mesmo objetivo: criar aplicativos para distintas plataformas, sem as dificuldades já conhecidas do desenvolvimento nativo. Cada abordagem possui seus prós e contras, bem como sua própria maneira de lidar com o problema. O fato é que todas são capazes de produzir aplicativos multiplataforma a partir de uma única base de código, apesar de que os desenvolvedores e as casas de software estarão sempre limitados aos recursos fornecidos pela ferramenta de desenvolvimento. O que significa que conforme forem surgindo novos recursos nativos, pode acontecer da ferramenta não implementar ou levar um tempo para integrar os novos recursos com o framework. E isso pode acontecer por diversos motivos, desde uma limitação de linguagem ou ferramenta até a própria descontinuação ou estagnação do suporte da ferramenta.

A Figura 1, ilustra um resumo das distintas formas de desenvolvimento de aplicações móveis multiplataforma.

Figura 1 – Abordagens de desenvolvimento móvel e suas principais características.



Fonte: (CUNHA, 2018)

4.2 Ferramentas de desenvolvimento multiplataforma

Os frameworks multiplataforma permitem a criação de aplicações para múltiplas plataformas, a partir de uma única base de código e uma série de ferramentas que visam ajudar e otimizar o tempo de desenvolvimento, bem como reduzir custos e outros benefícios (HJORT, 2020). No decorrer desta seção, foram apresentados alguns dos principais *frameworks* para desenvolvimento multiplataforma existentes no mercado.

4.2.1 Flutter

Flutter é um SDK de código aberto desenvolvido pelo Google, escrito em C, C ++ e Dart, que permite o desenvolvimento de aplicativos móveis multiplataforma. A priori é usado para desenvolver aplicativos para plataformas móveis, como Android e iOS. Além disso, o Flutter pode ser usado para desenvolver aplicativos da Web e de desktop a partir de uma única base de código (FLUTTER1; FLUTTER2, 2020). O Flutter foi lançado inicialmente em 2017, mas a primeira versão estável veio em dezembro de 2018 (FLUTTER1, 2020). Em pouco tempo, tornou-se uma das alternativas mais populares para o desenvolvimento multiplataforma. Os aplicativos Flutter são escritos em Dart, uma linguagem de programação também desenvolvida pelo Google. Como o código-fonte é compilado na linguagem nativa, o Flutter tem um ótimo desempenho. Ele não usa componentes de IU

nativos, mas renderiza seus próprios componentes, widgets de material e widgets Cupertino (estilo iOS), que têm aparência nativa. Uma vez que os componentes já estão estilizados, o framework tem menos trabalho para obter uma aparência nativa para o aplicativo. Uma desvantagem do Flutter é que os desenvolvedores precisam aprender uma nova linguagem para usá-lo. Por outro lado, o Dart possui semelhanças com C++, Java e JavaScript, por isso deve parecer familiar para a maioria dos programadores (HJORT, 2020).

- **Dart:** É uma linguagem otimizada para o cliente para o desenvolvimento de aplicativos rápidos em qualquer plataforma (Dart, 2021). O Dart foi projetado para ser uma solução ponta-a-ponta, ou seja, pode ser usado para desenvolvimento front-end e back-end, uma vez que pode ser executado em um navegador da web e também em um servidor (MITCHEL et al., 2017).
- **Widgets:** Um widget é uma parte de um código reutilizável que descreve como a interface de usuário de um aplicativo é construída. Os widgets são usados como os blocos de construção fundamentais dos aplicativos Flutter. Eles definem as interfaces de usuário por seu estado e criação de interface de usuário do aplicativo, criando uma estrutura semelhante a uma árvore chamada árvore de widgets. sempre que houver uma mudança no estado de um widget, o Flutter irá renderizar novamente a interface do usuário, aplicando as mudanças na árvore do widget desde a última renderização. A nova renderização ocorre de forma eficiente porque visa apenas aqueles com alterações na árvore de widgets. O Flutter vem com muitos widgets integrados que podem ser facilmente customizados para aprimorar o desenvolvimento rápido de aplicativos. Um widget pode ser composto de vários widgets pequenos e de propósito único, que se combinam para produzir efeitos significativos e poderosos, o que significa que os widgets podem ser aninhados. Tudo em aplicativos Flutter é um widget, um widget pode ser, um elemento estrutural como um botão ou menu, um elemento estilístico como uma fonte, um aspecto de layout como margem, padding, etc (FLUTTER2. 2020).
- **Layout e Estilo:** O layout de um aplicativo Flutter é criado inteiramente a partir de widgets. A maioria dos widgets são componentes invisíveis como linhas, colunas e grades que determinam a colocação de outros widgets visíveis. Os widgets podem ser customizados definindo margem, cor ou outros atributos. Um exemplo de widget altamente customizável é a classe Container.

Um widget filho geralmente é empacotado em um widget de contêiner para obter uma determinada aparência. CSS não é usado no Flutter, todo o estilo é feito definindo os atributos apropriados para os widgets. Um widget de Texto pode ser estilizado dando-lhe um filho TextStyle especificando a fonte e a cor (FLUTTER2, 2020).

4.2.2 React Native

React Native é um framework desenvolvido pelo Facebook e sua primeira versão foi lançada em 2015. O React Native possui o código aberto, portanto além do Facebook, o código fonte do framework recebeu participação de desenvolvedores individuais e até mesmo empresas interessadas na evolução e desenvolvimento da ferramenta (EISENMAN, 2015; REACT NATIVE, 2020).

O React Native é classificado como um framework multiplataforma do tipo interpretado, pelo fato de renderizar componentes de interface usando a API de renderização nativa padrão da plataforma de destino. Um aplicativo React Native é capaz de utilizar recursos específicos da plataforma, como o microfone ou a câmera do telefone, acessando a API da plataforma por meio de interfaces JavaScript (EISENMAN, 2015). Não há dúvidas que o uso do JavaScript é uma das grandes vantagens do React Native, visto que a linguagem é comumente difundida e amplamente utilizada por desenvolvedores web. Com isso torna-se prático construir aplicativos móveis com técnicas já familiares em vez de ter que aprender uma nova linguagem e ambiente de desenvolvimento para cada plataforma (HJORT, 2020).

- **Componentes:** Um aplicativo React Native é construído com componentes dinâmicos. Tudo no layout é um componente. Os componentes são escritos em JSX, que é uma extensão de sintaxe para JavaScript e significa JavaScript XML (JSX, 2021). XML é a abreviação de Extensible Markup Language (BRAY, 2021) e é uma linguagem de marcação mais genérica e flexível do que HTML, onde os usuários podem definir suas próprias tags. JSX combina a lógica de controle, marcação e estilo em um arquivo, mesmo na mesma linguagem. Um dos componentes mais básicos é o View, um elemento versátil da interface do usuário que pode ser comparado ao elemento HTML . Como a maioria dos componentes do React Native, ele pode ser usado para Android e iOS. O componente renderiza para uma visualização Android ou uma UIView iOS, dependendo da plataforma. Existem também componentes específicos da plataforma, como DatepickerIOS que só pode ser usado para

iOS e renderiza para o selecionador de data iOS típico (HJORT, 2020).

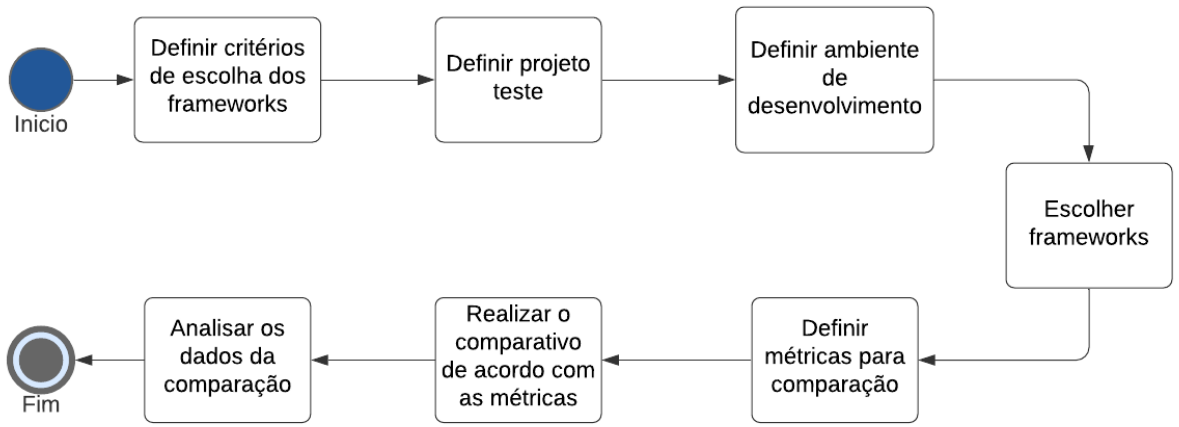
- **Layout e Estilo:** Ao trabalhar com aplicativos móveis nativos, cada plataforma tem sua própria maneira de lidar com o layout e o estilo. Para a web, CSS é mais comumente usado para estilizar elementos. O React Native não pode usar CSS diretamente, mas usa um subconjunto limitado de estilos CSS disponíveis. O objetivo é manter um estilo simples, mas ainda expressivo. Esta implementação CSS faz parte da ponte entre o React e a plataforma de destino, de forma que pode ser traduzida para o estilo dos componentes nativos (EISENMAN, 2015).

5 MATERIAIS E MÉTODOS

O presente trabalho tem o caráter de uma pesquisa comparativa, que, segundo Heidenheimer (1983), é o ato de comparar duas ou mais coisas, tendo como objetivo descobrir algo sobre uma ou todas as coisas envolvidas na comparação. O objetivo desta comparação é analisar os principais *frameworks* disponíveis no mercado para desenvolvimento multiplataforma. Ao fim do trabalho, será exposto de forma comparativa, as diferenças entre elas, sejam positivas ou negativas.

O processo de comparação foi baseado em métricas utilizadas em estudos similares e algumas propostas pelo autor, para que com base nas métricas se torne possível expor detalhes de cada ferramenta. A Figura 2 ilustra os procedimentos metodológicos deste trabalho. Nas seções subsequentes, é apresentado a descrição de como foram realizadas cada etapa.

Figura 2 – Procedimentos Metodológicos



Fonte: Elaborado pelo autor.

5.1 Critérios de escolha dos frameworks/ferramentas

Para a realização deste estudo comparativo, faz-se necessária a definição e explicação no que se refere aos critérios de escolha dos *frameworks*. Contudo, a priori, precisa-se estabelecer quantos e quais *frameworks* serão objetos deste estudo. Por decisão dos autores, foi definido que este estudo seria aplicado em dois *frameworks*, para que se tivesse tempo hábil para desenvolver ferramentas completamente distintas.

Os critérios utilizados para determinar os *frameworks* a serem analisados foram:

- Ser multiplataforma (*Cross Platform*);

- Ter a visualização reativa (*Hot Reload*);
- Permitir acesso aos recursos nativos;
- Popularidade;
- Engajamento da comunidade.

Para conseguir os dados referentes aos dois últimos critérios definidos, buscou-se em sites especializados em controle de versão e ajuda com desenvolvimento de *software*. E os escolhidos foram:

- GitHub;
- Stack Overflow.

5.2 Definir projeto teste

O aplicativo foi pensado de forma específica para cumprir os objetivos desta pesquisa. Portanto foram definidas algumas funcionalidades fundamentais, sendo a maioria delas planejadas e realizadas para o aplicativo ter a capacidade de ter acesso aos recursos nativos dos aparelho, as funcionalidades implementadas na aplicação foram:

- **Acesso a câmera do dispositivo:** Muitos aplicativos fazem uso deste recurso, como: redes sociais, editores de foto, soluções de pagamento, entre vários outros tipos de aplicações. Portanto é relevante analisar este tipo de acesso. Neste trabalho foi testada a capacidade de acessar a câmera do dispositivo e exibição da fotografia em tela.
- **Acesso a localização:** A localização é outro recurso com grande relevância para o usuário e para as aplicações. Neste trabalho foi feita a utilização da API do Google Maps, onde foi testada a capacidade de conseguir extrair a localização do dispositivo, e se possível identificar em que ponto do mapa está localizado.
- **Consumo de uma API - REST:** Para que um aplicativo consiga enviar ou requisitar dados de um servidor, é necessário o consumo de uma API que propicie a realização de operações de CRUD de forma instantânea/online. As operações de CRUD são: escrita, leitura, atualização ou exclusão em um banco de dados (para que os dados não fiquem armazenados na memória do dispositivo). Para este trabalho, foi utilizada a API pública do GitHub, requisitando os dados de um dado usuário informado via *input*. Os dados

retornados pela API vêm por padrão em JSON, que é o padrão utilizado em arquiteturas REST. A requisição foi realizada no seguinte endpoint https://api.github.com/users/{Login do Usuário}.

5.2.1 Especificação do projeto teste

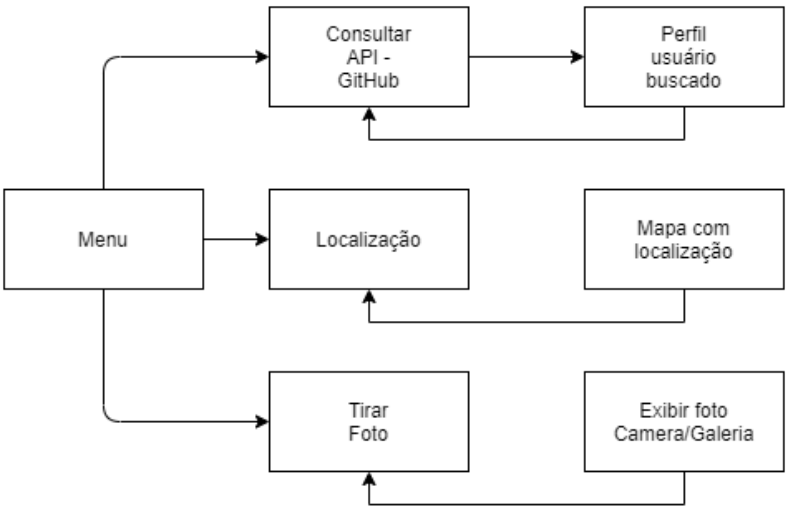
O aplicativo contém uma tela inicial com função de menu que possui três funcionalidades. A primeira é "Consultar API – GitHub", que leva até a tela que é possível buscar um usuário cadastrado no GitHub. Essa ação é armazenada na memória do celular, permanecendo os dados até que o usuário exclua o aplicativo da pilha de aplicativos em execução.

A segunda funcionalidade do menu da tela inicial é que através de um botão com o texto "Tirar uma Foto", é possível acessar uma outra tela específica para tirar fotos que contém um botão na parte superior da tela, com o texto "Abrir Câmera". Com a câmera aberta, tem-se a possibilidade de realizar três ações: ver a imagem da câmera, tirar uma foto e alternar entre os tipos de câmera (frontal e traseira). Caso tire uma foto, a mesma será exibida na tela anterior. Por último, o botão escrito "Localização" acessa a página que contém um mapa e a posição onde o dispositivo se encontra destacado no mapa.

5.2.2 Diagrama de transição do projeto teste

A Figura 3 apresenta o diagrama de transição entre telas do aplicativo desenvolvido como teste.

Figura 3 – Diagrama de transição do aplicativo



Fonte: Elaborado pelo autor.

5.3 Definir ambiente de desenvolvimento

Na Tabela 1, apresenta-se a descrição das ferramentas utilizadas para o desenvolvimento do projeto teste.

Tabela 1 – Ambiente de desenvolvimento - Ferramentas

Ferramentas			
Nome	**Tipo**	**Versão**	**Descrição**
Notebook Acer Aspire 5	Computador utilizado	-	8GB de memória RAM; processador Core i5 de 1.6GHz e 1TB de HD.
Visual Studio Code	IDE	1.37.1	IDE utilizada para a codificação do projeto teste
Google Chrome	Navegador	76.0.3809.132	Navegador usado para efetuar debug
Git	Sistema de Controle de Versão	2.23.0	Ferramenta utilizada para controlar/gerenciar as mudanças do projeto teste.
Windows	Sistema Operacional	10 Pro (x86/64)	Sistema Operacional utilizado para executar as ferramentas de desenvolvimento.
React Native	Ferramenta de Desenvolvimento Mobile	0.62.2	Ferramenta que propicia desenvolver apps em javascript e converter para código Android e iOS.
Genymotion	Emulador	3.0.2	Ferramenta usada para emular o ambiente Android.
Javascript	Linguagem de Programação	ES6	Linguagem por trás do framework React Native.

Flutter	Ferramenta de Desenvolvimento Mobile	1.9	Ferramenta que propicia desenvolver apps e convertê-los para código Android e iOS.
Android	Sistema Operacional Alvo	>= 6	Sistema Operacional utilizado para testar o app desenvolvido
iOS	Sistema Operacional Alvo	iPhone 6/7/8 Plus	Sistema Operacional utilizado para testar o app desenvolvido

Fonte: Elaborado pelo autor.

5.4 Escolha dos frameworks

Definido os critérios de escolhas dos frameworks, foi feita uma pesquisa nos sites escolhidos na Seção 5.1, são eles: *Github* e *Stack Overflow*. Eles serviram como base para dois dos cinco critérios definidos: Popularidade e Engajamento da comunidade. No *Github*, foi levado em consideração o número de estrelas e contribuidores no repositório do *framework*, já no *Stack Overflow*, foi levado em consideração os dados sobre os *frameworks* mais amados.

A Tabela 2 contém o resultado obtido na plataforma de controle de versão *Github*. É possível observar que os *frameworks* que apresentam os melhores índices no que se refere a popularidade e engajamento na comunidade são o Flutter e o React Native, seguidos por Ionic e Xamarin.

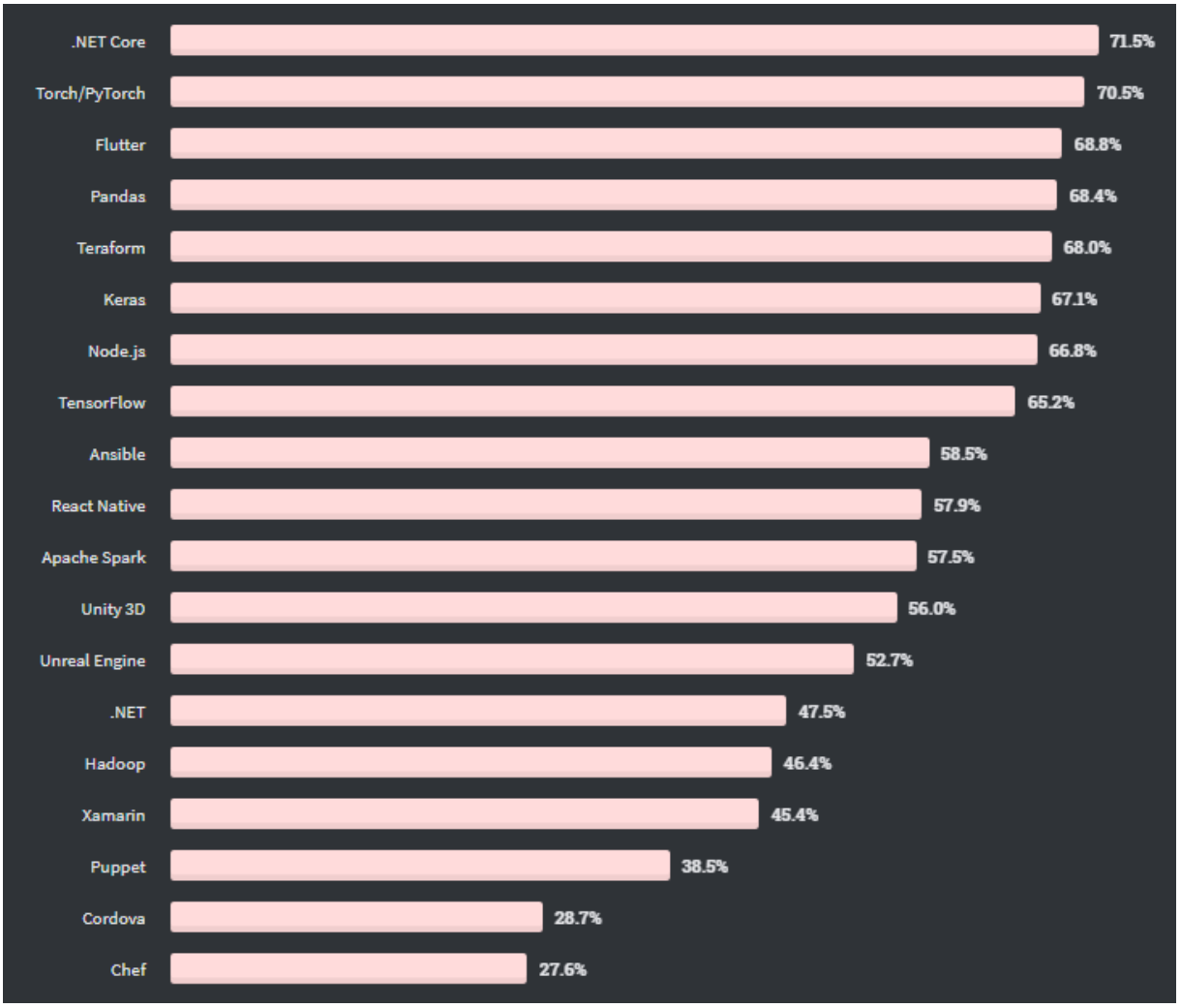
Tabela 2 – Top 4 frameworks no Github

Frameworks - Github			
Nome	**Nº de Estrelas**	**Nº Colaboradores**	**Data de Acesso**
Flutter	117.297	835	03 de Abril de 2021
React Native	94.500	2.266	03 de Abril de 2021
Ionic	43.341	404	03 de Abril de 2021
Xamarin	5.216	316	03 de Abril de 2021

Fonte: Elaborado pelo autor.

No ano de 2020, ilustrado pela Figura 5, pode-se observar que o Flutter e o React Native obtiveram alta expressividade, se mantendo à frente do Xamarin e do Ionic que nem obteve posição nesta pesquisa.

Figura 4 – Frameworks mais amados em 2020



Fonte: Stackoverflow (2020).

A partir da Tabela 3, foi possível notar que os *frameworks* que mais se encaixaram nos critérios definidos foram o Flutter e o React Native, seguido pelo Xamarin. Tendo como base esses dados, este estudo considerou o Flutter e o React Native como objetos de estudo. Em circunstâncias com mais recursos disponíveis, seria interessante adicionar o framework Xamarin.

Tabela 3 – Características fundamentais dos Frameworks escolhidos

Framework	**Multiplataforma (Cross Platform)**	**Visualização reativa (Hot Reload)**	**Acesso aos recursos nativos**
Flutter	X	X	X
React Native	X	X	X
Ionic	X	X	
Xamarin	X	X	X

Fonte: Elaborado pelo autor.

5.5 Definir critérios para a comparação

Os critérios para a avaliação do presente trabalho de comparação de *frameworks* multiplataformas entre o React Native e Flutter foram selecionados a partir da inspiração em trabalhos como: Heitkötter et al. (2012), Rieger e Majchrzak (2019), Basseto (2019) e Castro (2016).

As métricas escolhidas foram:

- Obtenção, instalação e configuração (Heitkötter, 2012 e Rieger e Majchrzak, 2019);
- Características (Rieger e Majchrzak, 2019);
- Viabilidade no longo prazo (Heitkötter, 2012 e Rieger e Majchrzak, 2019);
- Eficiência em acessar os recursos nativos (Heitkötter, 2012 e Rieger e Majchrzak, 2019);
- Tamanho do arquivo da aplicação para distribuição (Castro, 2016);
- Tempo de carregamento(Rieger e Majchrzak, 2019);
- Manutenibilidade (Heitkötter, 2012 e Rieger e Majchrzak, 2019);
- Tempo de preparação (Rieger e Majchrzak, 2019);
- Tempo de desenvolvimento (Rieger e Majchrzak, 2019);
- Distribuição (Heitkötter, 2012 e Rieger e Majchrzak, 2019).

5.5.1 Obtenção, instalação e configuração

A obtenção, instalação e configuração da ferramenta de desenvolvimento não são rotinas críticas, pois não tem um impacto suficiente no objetivo da ferramenta, que é criar um aplicativo, mas de algum modo pode vir a criar uma barreira para o desenvolvedor. Esse passo é inevitável e todos os desenvolvedores passarão por isso algum dia.

De forma resumida, o que será considerado para análise neste critério é:

- Tempo total para instalação;
- Tamanho do pacote;
- Processo de instalação;
- Processo de configuração framework;
- Tempo para criar um projeto.

5.5.2 Características

A proposta deste critério é enxergar dentro do contexto prático aspectos relevantes inerentes ao desenvolvimento de aplicativos móveis, como:

- Licenciamento;
- Ambiente de desenvolvimento integrado;
- Linguagem de programação;
- Escrita e desenho da *interface*

5.5.3 Viabilidade no longo prazo

A viabilidade é um critério para validar se é seguro migrar ou aderir a um framework. É complexo avaliar esse critério pois trata-se de projetar a continuidade de um framework para o momento presente, o problema é porque não tem como prever o futuro. Por isso, faz-se necessário utilizar indicadores que ajudem a projetar a viabilidade no longo prazo. Segundo Heitkötter (2012), os fatores que podem ajudar estimar a viabilidade são: Possuir atualizações constantes, oferecer suporte a versões recentes de sistemas operacionais móveis, a idade do framework e uma comunidade ativa com um número considerável de desenvolvedores e apoiadores que contribuem de forma frequente para a evolução do framework. Para analisar esse critério, no que se refere a comunidade, foi avaliado o tamanho de cada comunidade dentro do GitHub, mostrando o número de *stars* e de *issues*, além de analisar a quantidade de vezes que cada ferramenta foi citada no *Stackoverflow.*

5.5.4 Eficiência em acessar os recursos nativos

Os aplicativos móveis frequentemente usam o hardware do dispositivo, como a câmera e o GPS, portanto, é imprescindível para um framework multiplataforma oferecer suporte a essa funcionalidade. O foco desse critério é identificar se o framework consegue fazer acesso aos recursos nativos do celular de modo satisfatório. Para ser capaz de medir, foi mostrado cada passo para conseguir acessar o recurso da câmera, em cada framework.

5.5.5 Tamanho do arquivo da aplicação para distribuição

O tamanho da aplicação foi analisado ao final da construção do código e realizado o *build*. O tamanho do arquivo a ser distribuído nas lojas de aplicativos não tem tanta importância para o desenvolvedor, esse critério tem um impacto na experiência do usuário, visto que o tamanho está diretamente relacionado com o tempo de *download* e consumo de dados no dispositivo.

5.5.6 Tempo de carregamento

A proposta deste critério foi de mensurar o tempo que o aplicativo leva para carregar toda sua interface no dispositivo, dessa forma pode-se avaliar o desempenho do framework e a experiência do usuário. Para isso cronometramos o tempo desde o clique para iniciar o aplicativo até a renderização completa do mesmo. Quanto menor o tempo, melhor a experiência. A fim de termos uma informação imparcial, foi removido todos os aplicativos em execução e então realizado três vezes este teste em cada aplicativo, e o resultado a ser apresentado foi a média de tempo das três execuções.

5.5.7 Manutenibilidade

Os aplicativos geralmente precisam ser atualizados e mantidos para acompanhar a evolução da tecnologia ou para adicionar novos recursos. Uma maneira simples de medir a sustentabilidade é por linhas de código (RIEGER e MAJCHRZAL, 2019). O objetivo deste critério foi medir a quantidade de linhas de código fonte necessários para o framework montar o aplicativo, desconsiderando linhas de arquivos de configuração de extensões/bibliotecas externas e tudo que não foi o próprio desenvolvedor que escreveu. Esse critério influencia apenas os desenvolvedores e permite avaliar a manutenção do projeto, visto que um código fonte com menos linhas é mais fácil de entender e manter.

5.5.8 Tempo de preparação

Este critério examinou o tempo e o esforço necessários para começar a usar a estrutura. Dentre os aspectos analisados estão:

- Requer conhecimento prévio de linguagens de programação ou ferramentas adicionais;
- A curva de aprendizado.

5.5.9 Tempo de desenvolvimento

O tempo de desenvolvimento é um critério muito importante, principalmente quando observamos sob a perspectiva de desenvolvimento multiplataforma, onde o mesmo código fonte será executado para geração e distribuição de aplicativos para múltiplas plataformas. Mensurar o tempo para desenvolvimento da aplicação ou de uma funcionalidade específica é relevante nesse contexto para saber se realmente está ocorrendo ganhos em termos de velocidade de desenvolvimento.

A análise deste critério abrange:

- Tempo de desenvolvimento de funcionalidades específicas: consumo de *web-service*, acesso a localização, câmera;
- Tempo de desenvolvimento global.

5.5.10 Distribuição

Independente de como foi criado, um aplicativo deve ser distribuído por meio das lojas de aplicativos padrão para cada plataforma, contudo, cada *framework* possui procedimentos que precisam ser executados para a geração do aplicativo, e só então, disponibilizado nas lojas de aplicativos.

Dentre os pontos analisados para esse critério estão:

- Tempo até ter o arquivo para distribuição;
- Complexidade do processo e execução de comandos para geração do aplicativo;
- Integração com as lojas de aplicativos.

5.6 Pesos

Para cada critério foi aplicado um peso entre 1 e 5 para indicar o grau de relevância do critério para o estudo, onde 1 é menos relevante e 5 é mais relevante. A atribuição de peso para cada critério foi inspirada por trabalhos já mencionados sobre o tema, principalmente Rieger e Majchrzak (2019). Além do peso que cada critério receberá, também foi atribuída uma avaliação textual.

A maior ênfase é dada ao suporte da plataforma, distribuição e aparência. O peso mais baixo é dado aos critérios de instalação/configuração, tamanho do arquivo final para distribuição e manutenção, pois, não se apresentam como fatores obrigatórios para um bom resultado final ou porque são muito subjetivos e dificultam obter uma avaliação precisa.

A Tabela 4, apresenta a distribuição de pesos para cada critério avaliado.

Tabela 4 – Critérios ponderados

Critério	Peso
Obtenção, instalação e configuração	2
Características	5
Viabilidade no longo prazo	4
Eficiência em acessar os recursos nativos	5

Tamanho do arquivo da aplicação para distribuição	2
Tempo de carregamento	3
Manutenibilidade	4
Tempo de preparação	2
Tempo de desenvolvimento	3
Distribuição	5

Fonte: Elaborado pelo autor.

5.7 Realizar o comparativo de acordo com as métricas

Após concluir o projeto do aplicativo teste, iniciou-se a fase onde foi realizada a comparação e avaliação das aplicações desenvolvidas de acordo com as métricas definidas na Seção 5.5.

5.8 Analisar os dados da comparação

Por fim, após as comparações devidamente realizadas, inicia-se o passo cujo o objetivo é consolidar e unificar as informações obtidas ao final do passo 5.6, que basicamente constitui o objetivo principal do presente trabalho. Foram geradas tabelas e gráficos contendo os valores obtidos em cada aplicação tendo como base cada métrica definida na Seção 5.5.

6 DESENVOLVIMENTO DO PROTÓTIPO

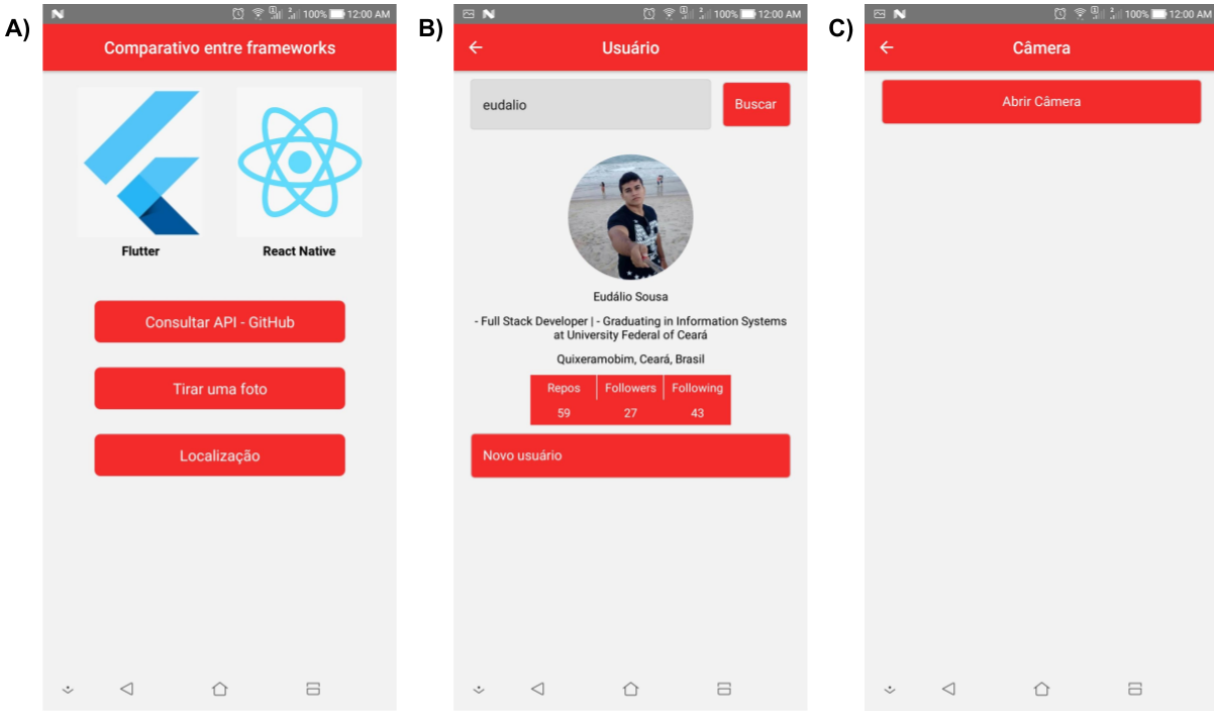
Nesta seção será apresentado o protótipo das telas utilizadas como base para desenvolver a aplicação (bem como a descrição das telas) e a experiência com o desenvolvimento em cada framework apresentando entre outras coisas suas vantagens e desvantagens.

Na descrição do desenvolvimento das aplicações, foi possível analisar apenas os resultados na plataforma Android, visto que a análise no iOS demanda de recursos financeiros não disponíveis para a execução desta pesquisa.

6.1 Protótipo das telas do aplicativo

Conforme é ilustrado na Figura 5, é possível observar na letra A, a tela inicial da aplicação, onde estão dispostos os menus de navegação. Na letra B, observa-se que tem um campo de texto para pesquisa por algum usuário do GitHub, que busca o usuário eudalio e mostra alguns dados de seu perfil. Na letra C, pode-se observar a tela inicial da função tirar uma foto, onde tem um botão que levará o usuário à abertura da câmera.

Figura 5 – A) Tela inicial. B) Consultar perfil de usuário do GitHub. C) Selecionar câmera

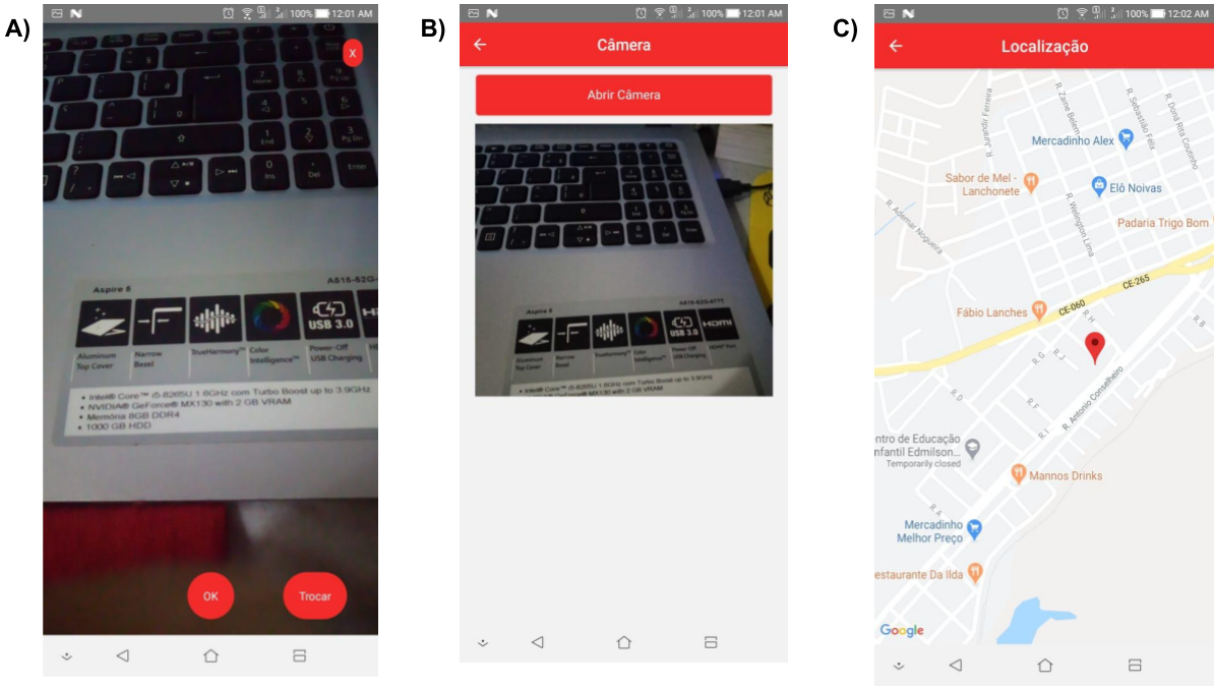


Fonte: Elaborado pelo autor.

Na Figura 6, na letra A, a câmera do dispositivo aberta e 3 funções disponíveis (tirar foto, trocar tipo de câmera e fechar câmera), na letra B, observa-se quando a foto é

tirada/selecionada, a mesma é exibida nesta tela, e, na letra C, temos a tela que captura a localização do usuário e mostra a posição onde está situado no mapa.

Figura 6 – A) Câmera. B) Exibir foto tirada. C) Localização (mapa)



Fonte: Elaborado pelo autor.

6.2 Desenvolvimento do aplicativo

Neste tópico será descrito como foi o desenvolvimento em cada uma das ferramentas escolhidas.

6.2.1 React Native

Foi o primeiro framework utilizado para desenvolvimento do projeto. O React Native é baseado em um framework que tem como foco a construção de *interfaces web*, o ReactJs. Ambos foram criados pelo Facebook e são mantidos pela empresa e comunidade que dão suporte a manutenção e evolução dos *frameworks*. O React Native funciona de forma semelhante ao ReactJs, sendo mais exato toda a estrutura lógica é igual, a diferença pode ser identificada na forma como essas ferramentas funcionam para construir as telas. Enquanto o ReactJs faz uso de elementos do HTML (como: div, p, a, etc.) para construir os componentes, no React Native foram criadas *tags* específicas (como: View, Text).

Como era de se esperar, foi observado que desenvolvedores que trabalham com o ReactJs tem maior facilidade em caso de migração, enquanto para quem nunca teve contato a migração se torna mais árdua, pois, se faz necessário entender bem sua estrutura, como se

organiza e seu funcionamento, para fazer melhor uso da ferramenta.

A seguir foram descritas algumas características identificadas no React Native.

- **Instalação**: Quando busca-se a documentação oficial para a instalação e configuração do ambiente de desenvolvimento da ferramenta, é possível encontrar as informações necessárias rapidamente. É importante ressaltar que toda informação disponibilizada pela empresa por seus meios oficiais estão disponíveis em inglês. Existem bons materiais em português, contudo, até o presente momento, o passo a passo de instalação com linguagem mais simples foi desenvolvido pela empresa de educação Rocketseat (2020).
- **Estilização**: A criação do *layout* das telas se torna simples devido a utilização do CSS3, a mesma ferramenta que é utilizada para estilizar páginas *web*. Com isso, desenvolvedores habituados e com conhecimento em CSS3, quando forem construir suas telas no React Native, terão uma experiência muito parecida com a forma em que se estrutura e trabalha em projetos *web*. No React Native. tem três formas de estilizar os elementos da interface. A primeira é através da propriedade *"styles*" encontrada na grande maioria dos componentes fornecidos pela própria ferramenta (Ex.: <View style={{color: 'red'}}>). A segunda é através de uma ferramenta especial: StyleSheet disponibilizada para definir estilos fora do JSX e, em seguida, referenciando dentro da propriedade "styles" mencionada anteriormente. E a terceira forma de estilizar componentes no React Native é utilizando uma biblioteca externa chamada *styled-components*, através dela é possível de forma sucinta escrever CSS3 dentro do código JS. Esse método é denominado *CSS-in-JS*. No entanto, existem também bibliotecas que propiciam uma maior velocidade na criação de interfaces, pois permitem utilizar elementos previamente estilizados e oferecem a possibilidade de fazer alguns ajustes como cores e posicionamento.
- **Acesso aos recursos nativos**: Para acessar um determinado recurso nativo em React Native, é necessário o uso de bibliotecas auxiliares. Existem em alguns casos mais de uma biblioteca que pode auxiliar no acesso a um recurso específico, todas são bem documentadas e de fácil utilização. Para utilizar essas bibliotecas, é necessário fazer algumas alterações em código nativo, no entanto, nada muito extenso ou complexo, apenas pequenas adaptações. As documentações das bibliotecas explicam de forma clara e objetiva como fazer estas adaptações em código nativo. Não houve dificuldades na instalação.

- **Navegação e rotas**: Para fazer a navegação entre as telas é necessário uma biblioteca de terceiro, chamada *react navigation*, já que o React Native não possui esse recurso. Com ela é possível criar vários tipos de navegação. O *react navigation* é um *framework* a parte, que ele oferece todo um ambiente de como lidar com as transições de telas, estruturar o tipo de navegação e *layout* da aplicação.
- **Versão da aplicação**: A documentação oficial disponibiliza de forma ilustrativa, clara e precisa o passo a passo a ser seguido para gerar a *release*. É um pré-requisito para isso saber para qual plataforma deseja gerar, pois, os passos diferem entre o Android e iOS, que são os possíveis.

A seguir, são listadas as vantagens identificadas no framework React Native:

- Permite a criação de apps para Android e iOs a partir de uma mesma base de código fundamentada em JavaScript;
- Aproveitamento do conhecimento previamente adquirido do ReactJs, pois a estrutura lógica é a mesma;
- Utiliza de CSS3 para fazer a estilização dos elementos e componentes;
- Excelente documentação das bibliotecas;
- Comunidade muito forte, participativa e engajada em contribuir para o crescimento da ferramenta e suas bibliotecas auxiliares;
- *Hot Reload* rápido e funcional;
- Permite manipular cada plataforma de forma diferente.

Já as desvantagens identificadas no *framework* React Native são:

- Dependência extrema de bibliotecas externas para quase todas as funcionalidades;
- Defasagem das bibliotecas auxiliares em relação a velocidade das evoluções do *framework*;
- Documentação oficial insatisfatória, visto que se faz necessário realizar buscas e recorrer a comunidade para tirar dúvidas.

6.2.2 Flutter

O Flutter traz um conceito de componentização parecido com o do React Native, porém esses “componentes” têm o nome de *Widgets*. Quase tudo no Flutter é um *Widget*. Quando trata-se do Flutter temos uma maneira de montar a interface baseada em classes, totalmente distinta React Native, Xamarin e outros, pois não utiliza-se tags nem a sintaxe do

XML e ou HTML comumente encontrada nos frameworks concorrentes. No início do desenvolvimento, há certa dificuldade. É possível compreender que precisa-se de um pouco mais de tempo para entender e lidar com a ferramenta, pois, além do framework Flutter tem-se de aprender os conceitos introdutórios da linguagem Dart.

A seguir foram descritas algumas características identificadas no Flutter:

- **Instalação**: A documentação oficial disponibilizada e mantida pela ferramenta é muito precisa no que se refere a como iniciar a utilização do *framework*. Em nenhum momento foi preciso consultar alguma fonte externa para conseguir fazer a instalação e configuração. Dentro da SDK do Flutter acompanha uma ferramenta chamada *flutter doctor*. Basicamente, com um único comando essa ferramenta realiza um diagnóstico das dependências que estão instaladas e auxiliam na visualização, atualização e instalação de tudo que o framework precisa para funcionar. Então se ao utilizar essa ferramenta, ela não apontar nenhuma pendência significa que seu ambiente está pronto para iniciar.
- **Estilização**: Existem componentes prontos extremamente bonitos, pois são herdados do *Material Design* e por esse motivo boa parte dos elementos traz uma estilização padrão utilizável.
- **Acesso aos recursos nativos**: Esse quesito foi sem dúvida o mais surpreendente no Flutter. O acesso à câmera é extremamente fácil e intuitivo, foi necessário apenas instalar um pacote complementar, não foi necessário alteração alguma em código nativo. Contudo, para acessar a localização houve a necessidade de poucas alterações em arquivos fontes nativos. O Flutter foi a ferramenta que apresentou maior facilidade para fazer o acesso aos recursos nativos.
- **Navegação e Rotas**: As rotas são muito fáceis de se aplicar. Os recursos disponibilizados de forma nativa da ferramenta atendem todas as necessidades sem necessitar de bibliotecas externas nesta funcionalidade. O material disponível na documentação é suficiente para mesmo com pouca experiência conseguir fazer aplicativos com mais de uma tela.
- **Versão da aplicação**: Seguir os passos disponíveis na documentação da ferramenta faz de sua utilização algo intuitivo e eficaz. Nesse ponto o processo para realizar o *build* é semelhante ao que ocorre no React Native.

Abaixo, são listadas as vantagens identificadas no *framework* React Native:

- Documentação detalhada e eficiente;

- Possibilidade de executar o código em desenvolvimento a partir da integração via *plugin* com o *VS Code*;
- Acesso aos recursos nativos de forma fácil e intuitiva. Basta instalar uma biblioteca, salvar e estará pronto para usar, sem muitas configurações;
- *Hot Reload* consideravelmente rápido;

A seguir, são elencadas as desvantagens identificadas no *framework* Flutter:

- Organização do código fonte é de difícil legibilidade, a árvore de elementos se torna extensa com pouco código e dificulta a indentação;
- Não é muito estável devido ser um *framework* recente, lançado em 2018;
- Requer o aprendizado do Dart;
- Devido ser recente, pode ainda haver funcionalidades em desenvolvimento, o que pode aumentar o tempo necessário de desenvolvimento.

7 RESULTADOS E DISCUSSÃO

Nesta seção, serão apresentados os resultados deste trabalho de acordo com o que foi estabelecido na Seção 4.

7.1 Comparativo das métricas

Neste tópico, são apresentados os resultados adquiridos a partir dos testes propostos na Seção 4.5. O ambiente de desenvolvimento está especificado na Seção 4.3. O aparelho utilizado para realizar os testes foi: Asus ZenFone Max Pro (M1), memória interna de 32GB, processador Octa-core 1.8 GHz Kryo 260 + 1.6 GHz Kryo 260, chipset Snapdragon 636 Qualcomm SDM636, 3 GB de memória RAM. O código das aplicações estão disponíveis em https://github.com/Eudalio.

7.1.1 Obtenção, instalação e configuração

As ferramentas possuem diferenças quanto à forma como são obtidas. O Flutter exige o *download* do *kit* de desenvolvimento (SDK), descompactação e alguns passos de configuração, como adicionar o caminho do SDK do Flutter nas variáveis de ambiente. Já o React Native precisa ter o Node e consequentemente o NPM instalado na máquina. Entretanto, antes mesmo de dar os primeiros passos com as ferramentas, é preciso baixar, instalar e configurar o java, bem como os SDKs do Android estejam instalados e configurados, para fazer isso, neste estudo instalamos o Android Studio e a partir dele instalamos as SDKs necessárias.

Quanto a instalação, o React Native oferece 3 possíveis maneiras para se criar um projeto, são eles: a primeira maneira é instalando um guia de interface de linha de comando (CLI), e a partir dele criar os projetos, a segunda maneira é utilizando o NPX que é uma alternativa disponibilizada pelo NPM para evitar instalações no dispositivo, e a terceira maneira é através do EXPO que é uma ferramenta que permite acessar todos os recursos dos demais tipos, porém, sem precisar instalar nada no computador, dá para criar e escrever aplicativos apenas com o navegador e um dispositivo móvel. O Flutter, disponibiliza uma interface semelhante através do seu *kit* de desenvolvimento, sendo possível a partir dele realizar ações como criar projeto, atualizar a versão, definir configurações da ferramenta, entre outras opções. Para o desenvolvimento do aplicativo teste, utilizamos o NPX e o *kit* de desenvolvimento para criar os projetos nos *frameworks*.

Na Tabela 5, é apresentado um comparativo entre as ferramentas, que foca os

principais pontos e fatores que impactam no primeiro passo do desenvolvimento do projeto, que é justamente, a instalação e configuração das ferramentas.

Tabela 5 – Comparativo do processo de instalação das ferramentas

	Tempo de instalação	**Tamanho do pacote**	**Tipo de Instalação**	**Configuração**	**Tempo para criar projeto**	**Suporte para Android e iOS**
Flutter	~ 20 minutos	590 MB	Manual / Complexidade baixa	Manual / Complexidade média	< 2 minutos	Possui
React Native	Não requer	Não requer	Automática / Complexidade baixa	Automatizada via NPM / Complexidade baixa	~ 6 minutos	Possui

Fonte: Elaborado pelo autor.

Pode-se observar que neste ponto, os dois *frameworks* não se assemelham, pois, enquanto o React Native não quer instalação do *framework* em si, o Flutter até ser utilizado precisa passar todo o processo da obtenção, passando pela descompactação e configuração, levando nesse caso aproximadamente 20 minutos até o ambiente estar pronto para iniciar um projeto. Um aspecto relevante a ser destacado é o tempo para criar um projeto após a ferramenta instalada. O Flutter levou menos de 2 minutos para se criar a estrutura básica de um projeto e possui uma necessidade de armazenamento drasticamente menor se comparado ao React Native.

Peso React Native: 4

Peso Flutter: 3

7.1.2 Características

As ferramentas possuem grandes diferenças em relação às suas características, principalmente, no modo como definem a interface da aplicação. Na Tabela 6, estão destacadas as que possuem maior relevância no que tange ao desenvolvimento de aplicativos multiplataforma. Os dois *frameworks* possuem o licenciamento gratuito. A linguagem de programação utilizada, bem como a maneira como é feita a definição de interface são completamente diferentes. Enquanto o Flutter usa Dart e constrói a UI com *Widgets*, o React Native utiliza em essência Javascript e monta a UI com JSX. Quanto ao ambiente de desenvolvimento, o React Native não recomenda oficialmente nenhuma IDE, diferente do Flutter que recomenda o uso do Android Studio ou Intellij. Para o desenvolvimento do

aplicativo teste, foi utilizado o Visual Studio Code, que apesar de não possuir recomendação oficial, atende muito bem a demanda fazendo uso de extensões que dão suporte ao desenvolvimento.

Tabela 6 – Características das ferramentas

	Licenciamento	**IDE Nativo / IDE utilizado**	**Linguagem de programação**	**Definição de interface**
Flutter	Grátis - Código aberto	Android Studio ou Intellij / Visual Studio Code	Dart	Widgets
React Native	Grátis - Código aberto	Não possui / Visual Studio Code	Javascript	JSX

Fonte: Elaborado pelo autor.

Os resultados obtidos nesse critério são análogos aos trabalhos de Hjort (2020) e Rieger e Majchrzak (2019). Diferindo apenas no ambiente de desenvolvimento utilizado, onde Hjort (2020) utiliza o Android Studio como IDE para construir um aplicativo com Flutter.

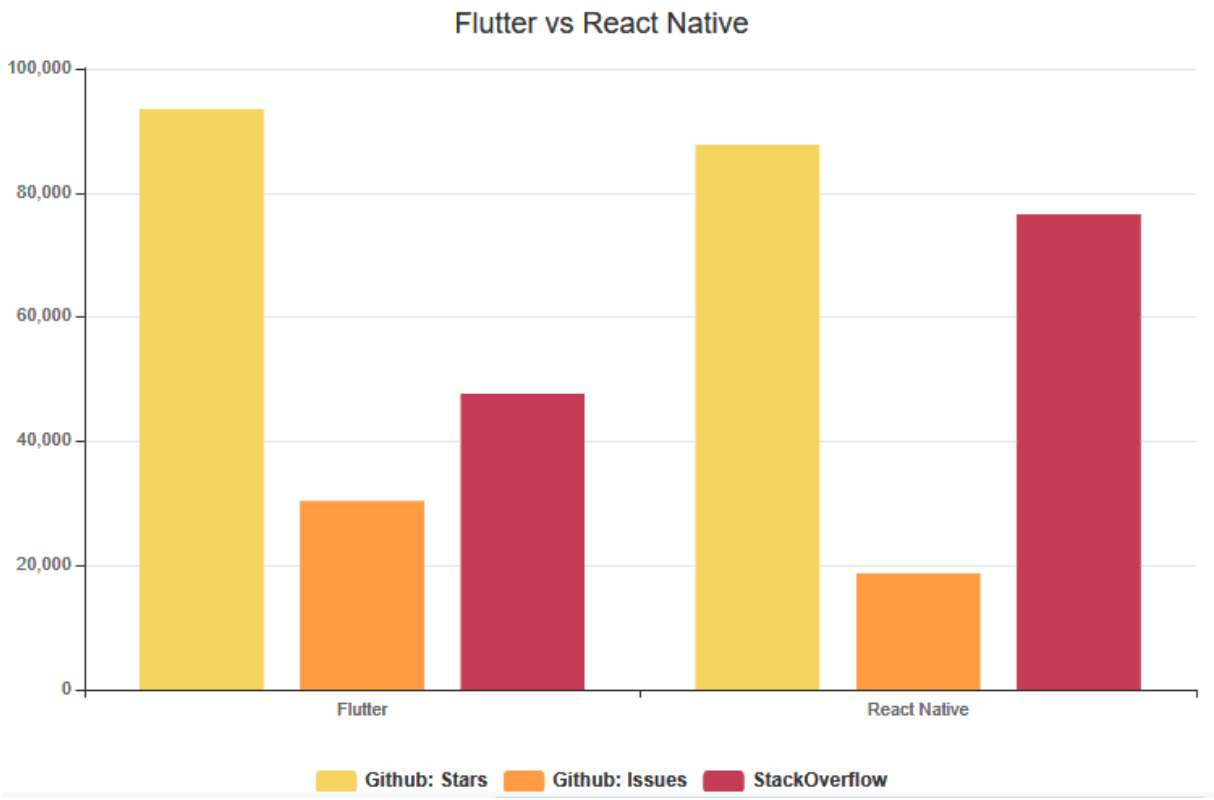
Peso React Native: 4

Peso Flutter: 4

7.1.3 Viabilidade no longo prazo

Neste quesito foi analisado se a comunidade dos frameworks conseguiram atender de forma satisfatória as necessidades em relação aos possíveis erros e problemas ocorridos durante o desenvolvimento. Primeiramente foram apresentados os números de *starts* no GitHub de cada *framework*, *issues* fechadas, e quantidade de vezes que cada um foi citado no Stackoverflow, conforme é ilustrado na Figura 7.

Figura 7 – Comunidade ativa dos frameworks

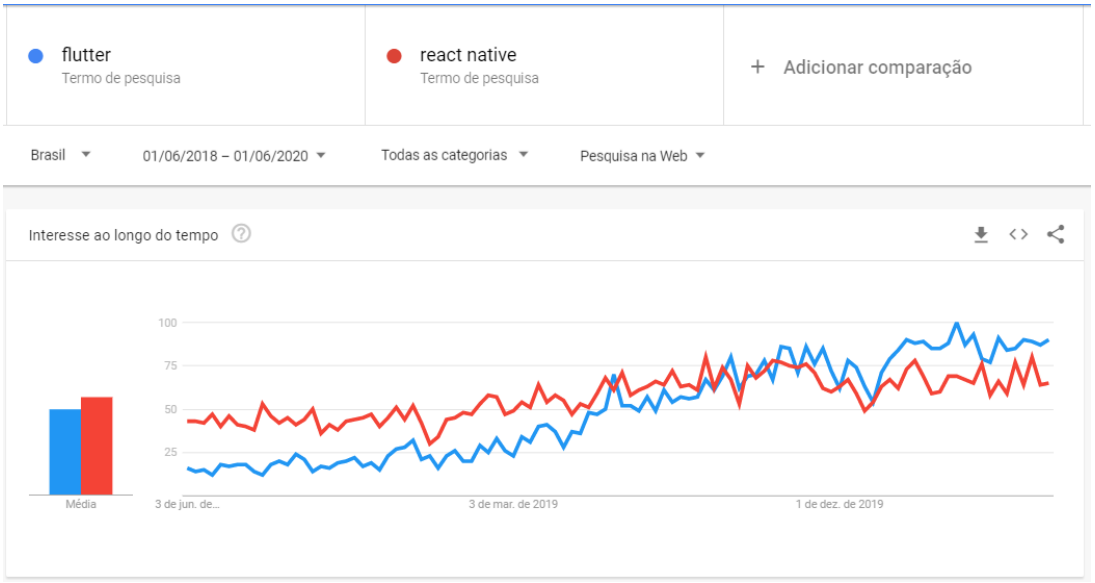


Fonte: Elaborado pelo autor.

No momento que este estudo está sendo escrito o React Native conta com 2266 contribuidores e atualizações frequentes de no máximo 70 dias (GITHUB1, 2021). Enquanto o Flutter conta com 836 contribuidores e atualizações frequentes de no máximo 25 dias (GITHUB2, 2021).

Também foi levado em consideração os dados do *Google Trends*, que é uma ferramenta gratuita disponibilizada pelo Google que permite acompanhar a evolução do número de buscas por uma determinada palavra-chave ou tópico ao longo do tempo, no caso em questão, definimos o tempo em exatos dois anos, conforme é apresentado na Figura 8.

Figura 8 – Gráfico Google Trends



Fonte: Elaborado pelo autor.

Ao observar as Figuras 7 e 8, pode-se visualizar que o Flutter tem maior popularidade e aparenta maior crescimento em números de buscas, o que indica que esse *framework* está em alta no momento ganhando muitos adeptos. Contudo, a quantidade de citações do React Native no Stackoverflow é realmente relevante, o que dimensiona o envolvimento da comunidade para ajudar na remoção de dúvidas. Além disso, o fato de o Flutter ser uma solução em ascensão não torna seu antagonista obsoleto, o React Native está no mercado a mais tempo, utiliza tecnologias populares, e na média a quantidade de buscas por React Native ainda é maior, como podemos ver na Figura 8.

No estudo realizado por (HJORT, 2020), chama a atenção o uso da expressão “risco de bloqueio tecnológico" ao falar do Flutter em comparação com o React Native. Onde o autor afirma que o React Native pode ser considerado maduro e estável pelo seu tempo de existência, comunidade grande e ativa, estrutura apoiada pelo Facebook, e, principalmente, por utilizar o javascript que é uma linguagem amplamente utilizada em diferentes estruturas (web, mobile), o torna mais viável para utilização em produção comercial. Todos esses elementos reforçam o que o autor chama de risco de bloqueio tecnológico.

Peso React Native: 5

Peso Flutter: 3

7.1.4 Eficiência em acessar os recursos nativos

Neste parâmetro, foi analisado se as aplicações teriam a capacidade de acessar os recursos nativos, observando principalmente o grau de dificuldade e esforço. Como resultado final, todas conseguiram cumprir esse parâmetro, entretanto, o Flutter se sobressaiu neste

item, por conta da facilidade de implantação. Todavia será mostrado o passo a passo para configurar o uso da câmera nos *frameworks*, para que seja possível evidenciar o grau de dificuldade em acessar tal recurso.

No React Native é necessário a instalação de uma biblioteca *react-native-camera*, para fazer a instalação basta utilizar o comando *yarn add react-native-camera* ou *npm i react-native-camera --save.* Feito a instalação da biblioteca, é preciso dar as permissões para o aplicativo ser capaz de acessar a câmera, isso é feito no arquivo *android/app/src/main/AndroidManifest.xml*, este passo é ilustrado na Figura 9, a seguir.

Figura 9 – Permissão para acessar câmera React Native.

```
<!-- Required -->
<uses-permission android:name="android.permission.CAMERA" />

<!-- Include this only if you are planning to use the camera roll -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- Include this only if you are planning to use the microphone for video recording -->
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Fonte: REACT NATIVE CAMERA (2020).

Após inserir essas permissões, ainda precisa adicionar um linha no arquivo *android/app/build.gradle*, conforme é ilustrado na Figura 10.

Figura 10 – Alteração no arquivo *build.gradle* no React Native

```
android {
  ...
  defaultConfig {
    ...
    missingDimensionStrategy 'react-native-camera', 'general' // <--- insert this line
  }
}
```

Fonte: REACT NATIVE CAMERA (2020).

Estes são os passos a seguir para instalar e configurar a biblioteca para uso da câmera, concluído estes passos basta recompilar o projeto para utilizar.

O processo para se fazer o mesmo com o Flutter é mais simples, precisa apenas fazer a instalação na biblioteca e utilizar. Basta entrar no arquivo *pubspec.yaml*, adicionar na Seção *dependencies* a biblioteca *image-picker*, salvar o arquivo, e já está pronto para utilizar.

A disparidade entre os frameworks, quanto a complexidade para utilizar um pacote externo, fica evidenciado também no trabalho de (STENDER; ÅKESSON, 2020), onde é relatado que uma das principais diferenças entre os frameworks é a complexidade e o

tempo gasto com configurações para se utilizar bibliotecas externas, na qual o React Native destoa negativamente em relação ao Flutter. Há também uma diferença considerável em termos de desempenho como é explicado no trabalho de Hjort (2020), onde o React Native consome mais recursos de CPU, memória e energia do que o Flutter.

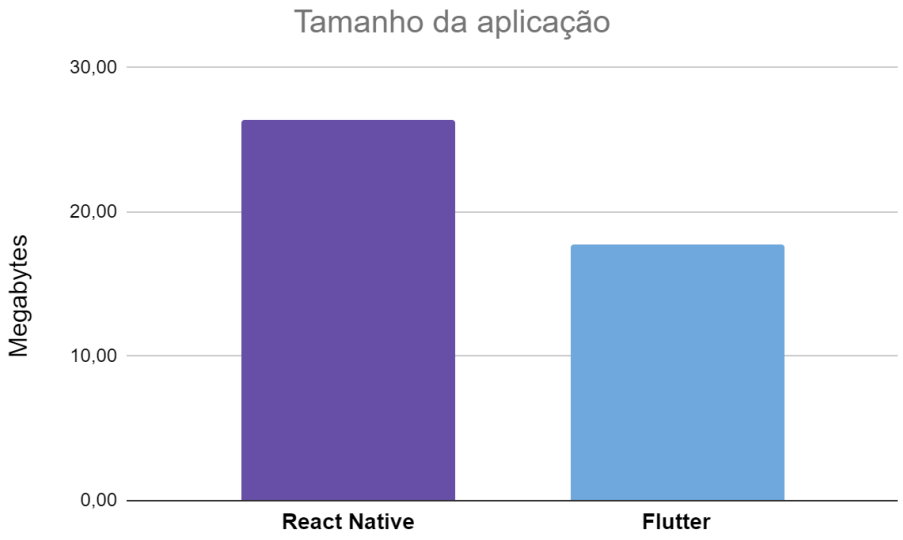
Peso React Native: 3

Peso Flutter: 5

7.1.5 Tamanho do arquivo da aplicação para distribuição

Após finalizar o desenvolvimento do aplicativo, foi realizado o *build* da aplicação, que nada mais é do que a compilação de todos os arquivos em um único executável que pode ser disponibilizado em massa por lojas de aplicativos. Conforme é ilustrado na Figura 11, os arquivos finais gerados para distribuição nas lojas de aplicativos possuem 26.4 MB e 17.8 MB, para o React Native e o Flutter, respectivamente.

Figura 11 – Comparativo tamanho da aplicação



Fonte: Elaborado pelo autor.

A exemplo deste trabalho, nos estudos de (STENDER; ÅKESSON, 2020) e (BASSETO, 2019), houve uma pequena diferença no arquivo final gerado para distribuição nas lojas de aplicativos, onde, no trabalho de Stender e Âkesson (2020), o React Native teve um tamanho de 22.7 MB e o Flutter teve um tamanho de 18.8 MB. No trabalho de Basseto (2019), o React Native resultou um tamanho de 22.8 MB e o Flutter resultou em uma tamanho de 13.3 MB. Não precisamente igual, porém, afirmando que existe uma diferença entre os arquivos para distribuição gerados pelos frameworks.

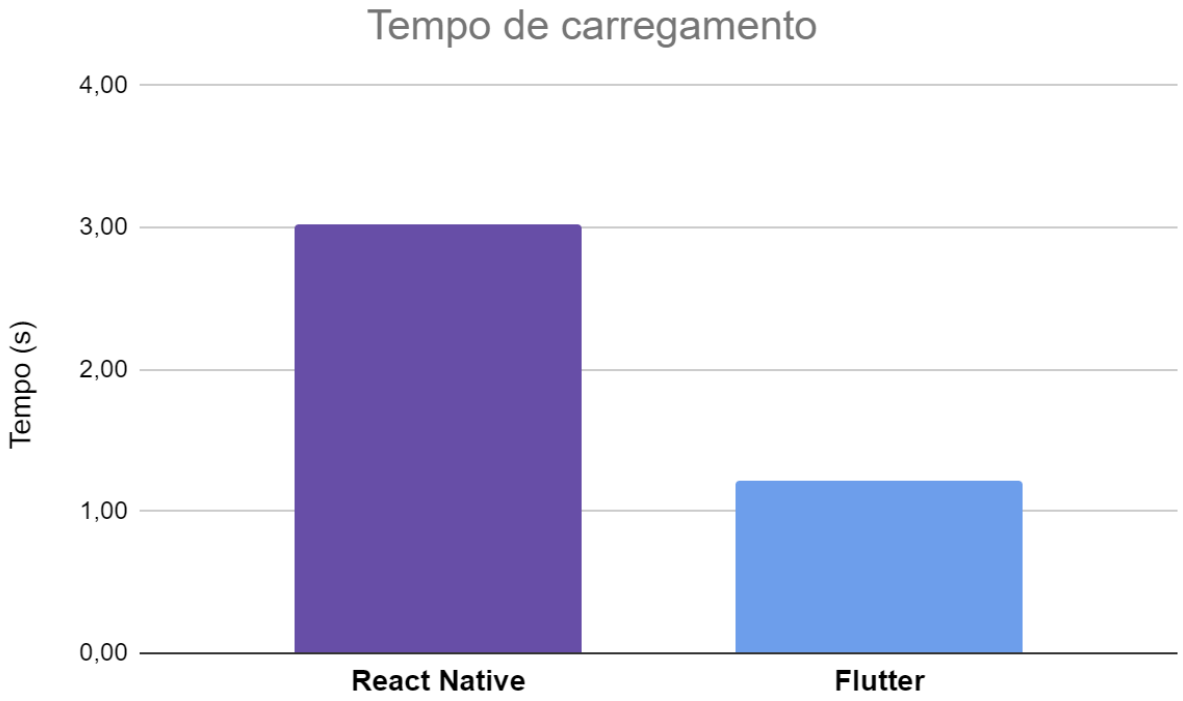
Peso React Native: 3

Peso Flutter: 4

7.1.6 Tempo de carregamento

O tempo de carregamento é o único critério de desempenho deste estudo e, conforme é ilustrado na Figura 12, o Flutter possui o tempo de carregamento notavelmente menor que o React Native. Isso implica em considerável vantagem para o framework Flutter, devido a maior velocidade ao abrir a aplicação e renderizar os componentes de interface em tela.

Figura 12 – Comparativo tempo de carregamento



Fonte: Elaborado pelo autor.

Em (STENDER; ÅKESSON, 2020) e (BASSETO, 2019), os autores tiveram resultados similares em sua pesquisa. No trabalho de Basseto (2019), o React native carregou em aproximadamente 1.3 segundos e o aplicativo Flutter carregou em aproximadamente 0.9 segundos. Enquanto, no trabalho de Stender e Âkesson (2020), foi identificado uma intensidade maior no consumo de recursos do dispositivo por parte do React Native. Todas essas informações indicam uma tendência de que a estrutura interna do React native não tem uma boa harmonia com o sistema operacional, levando ao maior consumo de energia e recursos do dispositivo para conseguir executar e manter a aplicação, quando comparado ao Flutter.

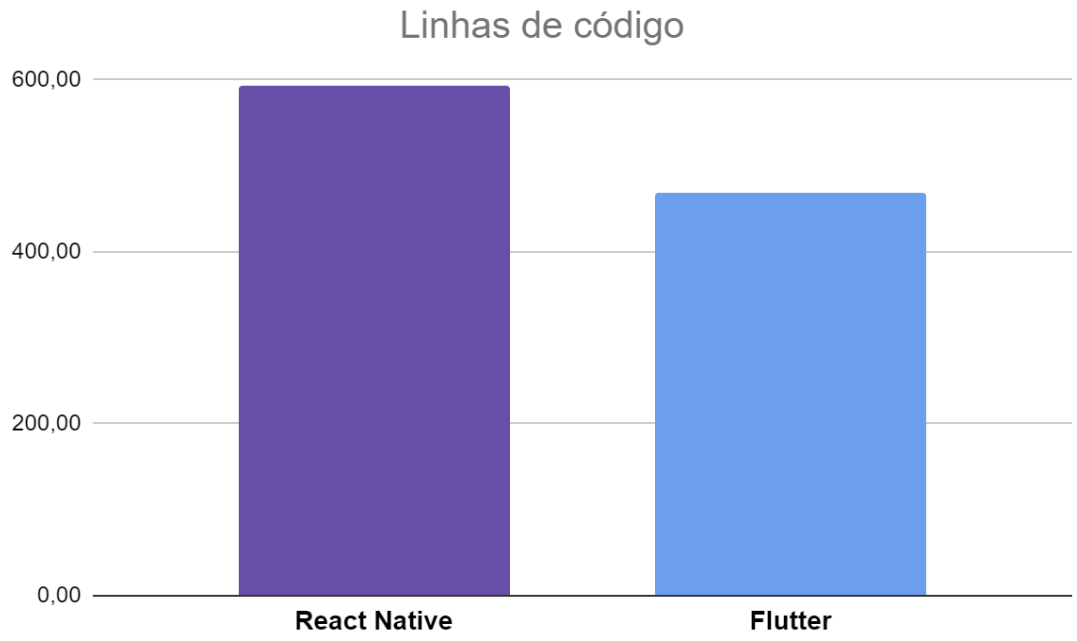
Peso React Native: 4

Peso Flutter: 5

7.1.7 Manutenibilidade

O React Native alcançou os maiores valores para linhas de código com 594 linhas escritas, e o Flutter ficou com 469 linhas de código, conforme ilustrado na Figura 13. Essa característica mensura e resulta qual das ferramentas oferece maior esforço para o desenvolvedor comparando com os demais frameworks. A partir destes dados é possível afirmar que o React Native é a ferramenta mais verbosa, contudo deve-se salientar que muito disso deve-se a escrita de código CSS. Por conta disso, foi levado em consideração esse fato na avaliação do React Native nesse critério, descontando da avaliação o código de estilização, tornando os dois frameworks proporcionalmente iguais.

Figura 13 – Comparativo linhas de código



Fonte: Elaborado pelo autor.

Os resultados obtidos no estudo realizado por (HJORT, 2020) foram semelhantes ao desenvolvido aqui quanto a essa característica, diferindo da quantidade (até porque foi desenvolvida uma aplicação totalmente diferente), mas concordando na diferença da quantidade de linhas totais de código entre os *frameworks*. O que confirma que de modo geral o React Native requer mais código do que o Flutter e precisa de atenção especial para organização e documentação, para posteriormente evitar transtornos em manutenções.

Peso React Native: 4

Peso Flutter: 4

7.1.8 Tempo de preparação

Para começar a desenvolver aplicativos são necessárias duas etapas: que o ambiente esteja pronto (isso inclui ferramentas, kit de desenvolvimento, download, configuração), e precisa conhecer o framework (estrutura, organização, conceitos, documentação, linguagem de programação, extensões). No que se refere a parte de montar o ambiente, o processo difere um pouco, requer a instalação do editor de código, do Java (no caso do Android), adicionar variáveis de ambiente entre outras ações já mencionados na seção 7.1.1.

O ponto principal neste critério está em conhecer o framework. O grande diferencial do React Native é o uso de tecnologias populares como: HTML, CSS e principalmente o JavaScript. Isso torna a curva de aprendizagem baixa, proporciona ao desenvolver familiaridade com a sintaxe e um progresso rápido no domínio do framework. Por outro lado, o Flutter tem uma curva de aprendizagem alta porque requer o conhecimento na linguagem Dart, que não é uma linguagem comum e portanto exige uma dedicação extra para aprender os aspectos da linguagem.

Peso React Native: 5

Peso Flutter: 2

7.1.9 Tempo de desenvolvimento

O desenvolvimento do código-fonte de um aplicativo é parte fundamental, tanto no processo de construção, como nas funcionalidades disponíveis para o usuário quando estiver em produção. Entretanto, esse critério possui um grau de subjetividade alto, por envolver diversos fatores humanos que podem influenciar o resultado final. Claramente, o fato de que cada ferramenta emprega diferentes linguagens de programação interfere neste processo. No decorrer da utilização de um aplicativo móvel, o fato de ter sido usada uma ou outra linguagem de programação não representa nada, contudo, enquanto estiver no ambiente de desenvolvimento este fator tem alta importância, isso se dá devido a qualidade e velocidade da implementação, bem como ao bom desempenho e funcionamento do produto entregue aos clientes.

Ao mensurar o desenvolvimento do aplicativo com as diferentes ferramentas, alguns aspectos foram considerados, conforme resultados expressos na Tabela 7.

Tabela 7 – Tempo de desenvolvimento

Código-fonte	React Native	Flutter
Tempo de desenvolvimento de uma tela com consumo de web service	12 horas	15 horas
Tempo de desenvolvimento de uma tela com acesso à câmera	5 horas	3 horas
Tempo de desenvolvimento de uma tela com acesso à localização	4 horas	2,5 horas
Tempo de desenvolvimento total	~30 horas	~25 horas

Fonte: Elaborado pelo autor.

Ao observar a Tabela 7, tem-se os seguintes resultados: não houve uma diferença expressiva, os resultados são bem parecidos o que demonstra equiparação entre os frameworks neste critério. O React Native apresentou-se mais intuitivo devido o disseminamento da linguagem base Javascript, porém, devido ao surgimento de alguns erros durante o desenvolvimento, o tempo para criar funcionalidades acabou se estendendo. Quanto ao Flutter houve uma surpresa positiva, pois, apesar da falta de conhecimento da linguagem *Dart* ser uma barreira no começo, a documentação da ferramenta atende bem às necessidades, apresentando exemplos práticos e ilustrativos. É possível concluirmos a partir desses dados que em situações onde precisa usar bibliotecas externas o desenvolvimento com React Native foi mais ineficiente. Entretanto, o fato de o desenvolvimento do aplicativo ter sido feito primeiramente com o React Native, pode ter um impacto direto no tempo final de desenvolvimento das funcionalidades utilizando o Flutter, isso se dá por conta da experiência adquirida na resolução de problemas em um framework em relação ao outro. Todavia, essa situação não foi levada em consideração no escopo da avaliação desse critério.

Peso React Native: 3

Peso Flutter: 4

7.1.10 Distribuição

Neste quesito, o tempo para gerar um arquivo de versão utilizando o *framework* React Native foi prejudicado devido a alguns erros ocorridos após a execução dos comandos, porém, esse prejuízo não será considerado na avaliação, porque não foram encontrados

problemas do tipo em outros estudos, corroborando com a idéia de que não foram problemas do framework e sim do ambiente. O tempo de configuração é basicamente o mesmo, pois requerem as mesmas obrigatoriedades, sendo necessário gerar uma chave identificadora única para esse aplicativo e colocar essas configurações e apontamentos em arquivos dentro do código fonte do sistema operacional desejado. Com as configurações feitas, em poucos minutos o arquivo de distribuição estava pronto. Um ponto negativo, nesse critério, é que nenhum dos frameworks possui integração com as lojas de aplicativos para automatizar o processo de envio e hospedagem para a loja.

Peso React Native: 5

Peso Flutter: 5

7.2 Avaliação Geral

A fim de consolidar os resultados da avaliação, a média ponderada foi calculada para cada um dos frameworks. O resultado é apresentado na Tabela 8. O React Native obteve uma pontuação de 4,0 e Flutter também teve a pontuação 4,0, de um total de 5,0. No contexto geral os dois frameworks obtiveram um bom desempenho, em vários critérios chegaram a se equiparar ou ficar próximo a isso. Apesar do empate técnico no total, ficou evidente em quais critérios ocorre a distinção entre os frameworks. O React Native se sobressaiu na viabilidade de longo prazo e no tempo de preparação, prevalecendo ao seu favor a maturidade e popularidade das tecnologias utilizadas. O Flutter se destacou nos critérios de desempenho como: o tempo de carregamento e o acesso aos recursos nativos.

O trabalho de Hjort (2020), fez uma comparação entre os mesmos frameworks, com critérios em maioria diferentes e também ponderando com pesos cada critério. Ao final seu resultado foi que o React Native recebeu uma pontuação de 4,5 e o Flutter recebeu uma pontuação de 4,1. Ao analisar os resultados no trabalho de Basseto (2019), pode-se ver que o Flutter obteve uma pontuação menor que o React Native, 11 e 14, respectivamente.

Tabela 8 – Classificação dos frameworks em cada métrica

Métricas	Peso	React Native	Flutter
Obtenção, instalação e configuração	2	4	3
Características	5	4	4
Viabilidade no longo prazo	4	5	2
Eficiência em acessar	5	3	5

recursos nativos			
Tamanho do arquivo (apk) para distribuição	2	3	4
Tempo de carregamento	3	4	5
Manutenibilidade	4	4	4
Tempo de preparação	2	5	2
Tempo de desenvolvimento	3	3	4
Distribuição	5	5	5
TOTAL	**-**	**4.0**	**4.0**

Fonte: Elaborado pelo autor.

8 CONSIDERAÇÕES FINAIS

A análise do desenvolvimento de aplicações utilizando os frameworks React Native e Flutter tornou possível a identificação de desafios reais ao utilizar estas ferramentas para construir uma aplicação. Propiciando, dessa forma, a criação de um relato para que outros desenvolvedores possam utilizá-lo como base para auxiliar a decidirem qual ferramenta escolher para usar em seus projetos, destacando as principais vantagens e desvantagens de cada ferramenta, bem como o desempenho prático das mesmas.

Diante dos resultados apresentados, é possível observar que embora muito recente, o Flutter mostrou ser bem estruturado, rápido e leve. Enquanto isso, o React Native apresentou-se maduro. Pode-se dizer que definir qual das duas ferramentas utilizar é relativo, e vai depender da proposta e objetivo do projeto. Os dois *frameworks* são capazes de entregar resultados de alta qualidade. O peso maior na escolha de um ou outro está sobre os requisitos e o contexto em que será aplicado. Contudo, de acordo com a experiência adquirida neste trabalho, ressalto que a utilização do React Native é indicada quando se é aplicado em um contexto onde há uma forte presença de tecnologias web, seja nas habilidades dos desenvolvedores do projeto ou inerente à regra de negócios. Sem dúvida, nesses casos, a utilização do React Native trará mais benefícios do que dificuldades. Em contrapartida, se o contexto é um projeto novo sem ligações com ambiente web, ou existe um requisito incontestável de garantir uma performance muito próxima à nativa, é indiscutível a recomendação para considerar o uso do Flutter em tais situações.

Como possíveis trabalhos futuros podem ser realizados testes visando levantar mais informações acerca da plataforma iOS. Outra possível linha de pesquisa seria realizar uma análise de desempenho entre aplicações criadas a partir de ferramentas multiplataforma em comparação com aplicações nativas.

REFERÊNCIAS

BASSETTO, G. **Comparativo entre frameworks para desenvolvimento multiplataforma.** Paraná: Universidade Estadual do Norte do Paraná, 2019. Disponível em: http://200.201.11.152/handle/123456789/392. Acesso em: 15 mar. 2020.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C.M., MALLER, E., YERGEAU, F. **XML Specification**. [S.l], [2008?]. Disponível em: https://www.w3.org/TR/xml. Acesso em: 04 mar. 2021.

CASTRO, Daniel Wolney Mendonça de. **Comparação de arcabouços para desenvolvimento de aplicações móveis multiplataforma no ambiente de TI do Ceará**. 2016. TCC (Graduação em Sistemas de Informação) - Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2016.

CRUZ, V.; PRETUCELLI, E. TECNOLOGIAS WEB PARA O DESENVOLVIMENTO MOBILE NATIVO. **SIMTEC - Simpósio de Tecnologia da Fatec Taquaritinga**, Taquaritinga, v. 4, n. 1, p. 15, 2018. Disponível em: https://simtec.fatectq.edu.br/index.php/simtec/article/view/269. Acesso em: 13 mai. 2020.

CUNHA, J. B, **Mobile apps runtime architectures**, 2018. Disponível em: https://www.freecodecamp.org/news/a-deeply-detailed-but-neverdefinitive-guide-to-mobile-development-architecture-6b01ce3b1528/. Acesso em: 19 mar. 2021.

DART. **Dart overview**. [Sl], [2018?] Disponível em: https://dart.dev/overview. Acesso em: 19 mar. 2021.

EISENMAN, B. **Learning react native:** Construindo aplicativos móveis nativos com JavaScript. [S.l]: O'Reilly Media, Inc., 2015. Acesso em: 25 set. 2020.

EL-KASSAS, Wafaa S. et al. Taxonomia de abordagens de desenvolvimento de aplicativos móveis multiplataforma. **Ain Shams Engineering Journal**, [S.l], v. 8, n. 2, p. 163-190, 2017. Disponível em: https://www.sciencedirect.com/science/article/pii/S2090447915001276. Acesso em: 13 mai. 2020.

FENTAW, A. E. **Desenvolvimento de aplicativos móveis multiplataforma:** um estudo comparativo de react native vs flutter. Jyväskylä: University of Jyväskylä, 2020. Disponível em: http://urn.fi/URN:NBN:fi:jyu-202006295155. Acesso em: 20 fev. 2021.

FERREIRA, Cristiane MS et al. Uma avaliação de frameworks multiplataforma para desenvolvimento de aplicativos móveis multimídia. **IEEE LATIN AMERICA TRANSACTIONS**, [S.I], v. 16, n. 4, 2018. Disponível em: https://ieeexplore.ieee.org/abstract/document/8362158. Acesso em: 15 fev. 2020.

FLUTTER1. **FAQ**. [S.l], [2018?]. Disponível em: https://flutter.dev/docs/resources/faq. Acesso em: 16 mar. 2020.

FLUTTER2. **Flutter architectural overview.** [S.l], [2018?]. Disponível em: https://flutter.dev/docs/resources/architectural-overview. Acesso em: 22 mar. 2020.

FLUTTER3. **Flutter documentation.** [S.l], [2018?]. Disponível em: https://flutter.dev/docs. Acesso em: 22 mar. 2020.

GITHUB1. **Page of React Native on Github**. [S.l], [2020?]. Disponível em: https://github.com/facebook/react-native. Acesso em: 06 abr. 2021.

GITHUB2. **Page of Flutter on Github**. [S.l], [2020?]. Disponível em: https://github.com/flutter/flutter. Acesso em: 06 abr. 2021.

GITHUB3. **Page of Ionic on Github**. [S.l], [2020?]. Disponível em: https://github.com/ionic-team/ionic-framework. Acesso em: 06 abr. 2021.

GITHUB4. **Page of Xamarin on Github**. [S.l], [2020?]. Disponível em: https://github.com/xamarin/Xamarin.Forms. Acesso em: 06 abr. 2021.

HEIDENHEIMER, A. J.; HECLO, H., ADAMS, C. T. **Comparative Public Policy**. [Sl]: St. Martin's Press, 1983.

HEITKÖTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Evaluating cross-platform development approaches for mobile applications. In: **International Conference on Web Information Systems and Technologies**. Springer, Berlin, Heidelberg, 2012. p. 120-138. Disponível em: https://doi.org/10.1007/978-3-642-36608-6_8. Acesso em: 26 mar. 2020.

HJORT, E. **Evaluation of React Native and Flutter for cross-platform mobile application development.** Åbo Akademi University, 2020. Disponível em: http://urn.fi/URN:NBN:fi-fe2020112392758. Acesso em: 10 jan. 2021.

JSX. **Introduzindo JSX.** [S.l], [2018?]. Disponível em: https://pt-br.reactjs.org/docs/introducing-jsx.html. Acesso em: 05 mar. 2021.

KUITUNEN, M. **Cross-Platform Mobile Application Development with React Native**. Tampere University, 2019. Disponível em: http://urn.fi/URN:NBN:fi:tty-201902111234. Acesso em: Abril/2020.

MITCHEL, D., AKOPKOKHYANTS, S., & BALBAERT, I. **Dart:** Scalable Application Development. [S.l]: Packt Publishing Ltd. 2017.

MOZILLA, Developer. **HTML: Linguagem de Marcação de Hipertexto.** [S.l], [2021?]. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 11 mar. 2021.

MOZILLA, Developer. **JavaScript tutoriais.** [S.l], [2021?]. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 11 mar. 2021.

MOZILLA, Developer. **Manifestos de aplicativos da web.** [S.l], [2021?]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/Manifest. Acesso em: 11 mar. 2021.

PREZOTTO, E. D.; BONIATI, B. B. **Estudo de frameworks multiplataforma para desenvolvimento de aplicações mobile híbridas.** [S.l: sn], 2017. Acesso em: 07 jan. 2020

REACT NATIVE. **React Native official documentation**. [S.l], [2018?]. Disponível em: https://reactnative.dev. Acesso em: 26 fev. 2020.

REACT NATIVE CAMERA. **Installation**, [S.l], [2020?]. Disponível em: https://react-nativecommunity.github.io/react-native-camera/docs/installation. Acesso em: 12 mai. 2020.

RIEGER, C.; MAJCHRZAK, T. A. Towards the definitive evaluation framework for cross-platform app development approaches. **Journal of Systems and Software**, v. 153, p. 175-199, 2019. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121219300743. Acesso em: 23 mar. 2020.

SCHMITZ, L. **Análise de ferramentas de desenvolvimento multiplataforma para criação de aplicativos móveis**. Universidade de Santa Cruz do Sul, 2016. Disponível em: https://repositorio.unisc.br/jspui/handle/11624/2149. Acesso em: 25 mar. 2020.

STACKOVERFLOW. **Developer Survey 2020**. [S.l], [2020?]. Disponível em: https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-other-frameworks-libraries-and-tools-loved3. Acesso em: 07 abr. 2021.

STACKOVERFLOW. **Newest 'flutter' Questions - Stack Overflow**. [S.l], [2020?]. Disponível em: https://stackoverflow.com/questions/tagged/flutter. Acesso em: 15 mai. 2020.

STACKOVERFLOW. **Newest 'react-native' Questions - Stack Overflow**. [S.l], [2020?]. Disponível em: https://stackoverflow.com/questions/tagged/react-native. Acesso em: 15 mai. 2020.

STATCOUNTER. **Mobile Operating System Market Share Worldwide**, [S.l], [2020?]. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2020-2020-bar. Acesso em: 27 mai 2020.

STENDER, S .; ÅKESSON, H. **Comparação de frameworks de plataforma cruzada**: flutter & react native. Blekinge: Blekinge Institute of Technology, 2020. Disponível em: http://urn.kb.se/resolve?urn= urn:nbn:se:bth-19749. Acesso em: 10 jan. 2021

XANTHOPOULOS, S .; XINOGALOS, S. Uma análise comparativa de abordagens de desenvolvimento de plataforma cruzada para aplicativos móveis. In: **Atas da 6ª Conferência dos Balcãs em Informática**. [Sl], pág. 213-220, 2013. Acesso em: 08 fev. 2021.

W3C. **CSS standard**. [S.l], [2020?] Disponível em: https://www.w3.org/Style/CSS. Acesso em: 03 mar. 2021.