ESTUDO DAS FERRAMENTAS GOOGLE FLUTTER E REACT NATIVE NO DESENVOLVIMENTO MULTIPLATAFORMA

Aline Mizumukai[1], Jade Simões de Castro[1], Anna Patricia Zakem China[1]

[1]Faculdade de Tecnologia de FATEC Ribeirão Preto (FATEC)

Ribeirão Preto, SP – Brasil

Resumo. *O desenvolvimento multiplataforma de aplicações móveis possibilita que sejam criadas aplicações para mais de um sistema operacional a partir de uma única base de código, alcançando o maior número de usuários em menor tempo. Dentre os frameworks atuais, o Google Flutter e o React Native são os mais atrativos e populares. Esta pesquisa apresenta as principais características destas ferramentas com o objetivo de auxiliar na tomada de decisão sobre qual tecnologia adotar.*

Abstract. *The cross-platform development of mobile applications makes it possible to create applications for more than one operating system from a single code base, reaching more users in less time. Among the current frameworks, Google Flutter and React Native are the most attractive and popular. This research presents the main characteristics of these tools in order to assist in deciding which technology to adopt.*

1. Introdução

O mercado de dispositivos móveis está cada vez mais aquecido. Quando se analisa o *market share* de dispositivos desktop e mobile a nível global nos últimos cinco anos, observa-se que o número de usuários desktop era maior até o final de 2016, quando essa situação se inverteu e desde então o número de usuários de celulares mantém-se sutilmente maior (STATCOUNTER, 2021).

Também é possível verificar o crescimento no mercado de aplicativos móveis ao analisar o número de *downloads* de aplicativos nos *App Stores*. De acordo com o estudo publicado recentemente pela plataforma (STATISTA, 2021), o número de *downloads* de aplicativos móveis em todo o mundo tem aumentado constantemente, ultrapassando 218 bilhões em 2020, um aumento de mais de 50 por cento em comparação ao número de downloads em 2016.

Em se tratando de desenvolvimento de aplicativos móveis, a opção pelo desenvolvimento nativo é vantajosa pelo seu desempenho e pela proximidade do fabricante, facilitando o acesso às APIs, especialmente as mais novas. Entretanto, esta alternativa torna necessário o uso de ferramentas e linguagens de programação específicas para cada plataforma escolhida, além de uma mão de obra extremamente especializada. Somado a isto, considerando que atualmente as aplicações móveis são disponibilizadas para os principais sistemas operacionais Android e iOS, o processo de desenvolvimento nativo também implica em maior custo e tempo de produção.

Como alternativa ao desenvolvimento nativo existe a opção do desenvolvimento multiplataforma, que permite desenvolver softwares em múltiplas arquiteturas de computadores ou sistemas operacionais utilizando o mesmo código base para as plataformas suportadas por cada framework, diminuindo o esforço necessário para

desenvolver os aplicativos. Dentre as soluções disponíveis, o Flutter e o React Native são os pioneiros na construção de aplicativos com desempenho nativo.

O objetivo desta pesquisa é apresentar as principais características das ferramentas Flutter e React Native para auxiliar na tomada de decisão sobre qual tecnologia escolher ao trabalhar no mercado de desenvolvimento multiplataforma de aplicativos móveis.

2. Referencial Teórico

Dentre as formas existentes atualmente para se desenvolver aplicações móveis, a opção pelo uso de frameworks multiplataforma vem ganhando destaque e popularidade entre os desenvolvedores. O desenvolvimento multiplataforma possibilita que sejam criadas aplicações para mais de uma plataforma nativa a partir de um único código de programação, tendo como resultado a produção de aplicativos de forma mais rápida e econômica (KUMAR, 2020).

Além de praticamente tornar dispensável o estudo detalhado de cada plataforma nativa e a criação de um código específico para cada uma, a rapidez no desenvolvimento do projeto também pode ser justificada pela possibilidade de se trabalhar com equipes especializadas. Já a produção em menor tempo permite que as equipes se concentrem mais nas regras de negócio do projeto, aumentando assim as chances de sucesso do aplicativo e reduzindo o risco de retrabalho.

Quando se fala em ferramentas multiplataforma, os *frameworks Flutter*, desenvolvido pelo Google, e *React Native*, criado pelo Facebook, são as soluções de código aberto mais utilizadas na construção de aplicativos móveis para os sistemas operacionais Android e iOS. Ambos os *frameworks* geram aplicativos nativos, ou seja, possuem estruturas que viabilizam a comunicação entre o código gerado no framework escolhido e o código nativo – no iOS, são utilizadas as linguagens de programação Objective-C ou Swift e no Android, Java ou Kotlin (WU, 2018).

De acordo com o estudo Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021 divulgado na plataforma Statista em julho de 2021, o Flutter é a ferramenta multiplataforma para desenvolvimento de aplicativos móveis mais popular usada pelos desenvolvedores. Para a realização da pesquisa, foram entrevistados 31.743 desenvolvedores do mundo todo durante os anos de 2019 a 2021(LIU, 2021).

Na figura 1, observa-se que 42% dos desenvolvedores de aplicações móveis usaram o Flutter como ferramenta em 2021. Já o React Native, que foi o mais popular entre os desenvolvedores nos anos anteriores, atualmente ocupa o segundo lugar de acordo com a pesquisa (LIU, 2021).

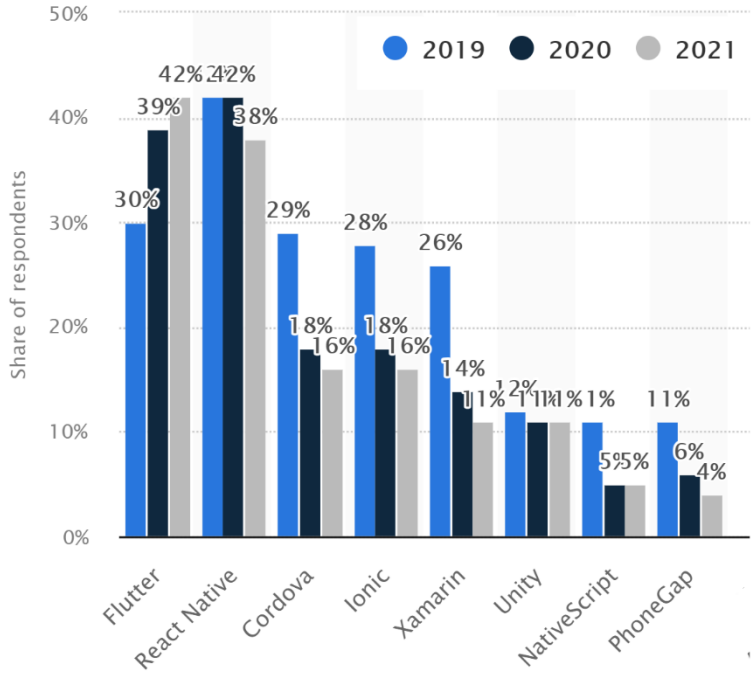

Figura 1. Frameworks multiplataforma para desenvolvimento mobile usado por desenvolvedores do mundo todo de 2019 a 2021.
Fonte: (Statista, 2021)

2.1. React Native: Aprenda uma vez, escreva em qualquer lugar

O React Native foi lançado em 2015, inicialmente como uma extensão do React, que é uma biblioteca JavaScript utilizada para criar interfaces para aplicativos web. Este framework adota a linguagem de programação JavaScript e utilizava a estrutura de *brigde* (ponte) para se comunicar com o código nativo (ANJOS, 2020) - recentemente a ferramenta disponibilizou uma atualização para habilitar o uso do JSI (JavaScript Interface), que elimina a necessidade do uso da *bridge* e permite a comunicação síncrona entre o lado JavaScript e o lado nativo.

O React Native permite o desenvolvimento de aplicativos móveis verdadeiramente nativos, sem comprometer a experiência dos usuários. Isso é possível, pois ele fornece um conjunto de componentes essenciais previamente mapeados para corresponder diretamente aos componentes das plataformas nativas (REACT NATIVE, 2021). Também é possível a construção de seus próprios componentes para Android e iOS quando é necessário atender a uma necessidade exclusiva de sua aplicação - muitos desses componentes personalizados são compartilhados nas comunidades oficiais da própria ferramenta.

Por ser derivado do React, é importante entender alguns conceitos para compreender o seu funcionamento, sendo o JSX um dos principais. O JSX é uma linguagem de marcação declarativa, com sintaxe semelhante ao HTML, que é escrita em conjunto com JavaScript para definir o layout de um componente. Apesar de não ser obrigatório, seu uso é recomendado por tornar o processo de codificação menos verboso e complicado (MASIELLO e FRIEDMANN, 2017).

Segundo Masiello e Friedmann (2017), o objetivo do React Native é permitir que os desenvolvedores escrevam aplicativos nativos de alta qualidade para iOS e Android usando tecnologias familiares da web. Embora o layout seja escrito usando JSX, toda a

regra de negócios é escrita em JavaScript. Além disso, a composição do aplicativo adota a estrutura de árvore de componentes aninhados da mesma forma que o React.

2.2. Flutter: Belíssimos aplicativos nativos em tempo recorde

O Flutter foi lançado em 2017, mas sua primeira versão foi disponibilizada no final de 2018. Apesar de mais recente, este framework tornou-se rapidamente tão popular quanto seu principal concorrente. As aplicações desenvolvidas com Flutter são escritas na linguagem de programação Dart e utilizam um mecanismo simples de comunicação entre o código Dart e o código nativo chamado *platform channels* (ANJOS, 2020).

Em seu website oficial, a ferramenta é descrita como o kit de ferramentas de interface do Google para desenvolver aplicativos bonitos e nativamente compilados em dispositivos móveis, web, desktop e incorporados a partir de uma única base de código (FLUTTER, 2021). Também são destacados alguns recursos como a funcionalidade *hot reload*, que recarrega as atualizações realizadas no código-fonte em uma máquina virtual em execução em menos de um segundo, permitindo assim testes em tempo real, e sua extensa biblioteca de *widgets* - componentes gráficos - que implementam o design padrão dos dispositivos Android, Material Design, e iOS, Cupertino, proporcionando desempenho completamente nativo (DAHL, 2019).

A linguagem de programação Dart, também desenvolvida pelo Google, é uma linguagem orientada a objetos e fortemente tipada. Ela é a base para o desenvolvimento de aplicativos usando o Flutter. A sua escolha pode ser justificada, dentre outras razões, por permitir tanto a compilação *just-in-time* (JIT), responsável por habilitar o *hot reload* durante a fase de desenvolvimento, quanto a compilação *ahead-of-time* (AOT), que habilita a compilação do código-fonte em código de máquina nativo quando ele está pronto para ser implantado em produção (FOREWORD, 2020).

É possível desenvolver aplicativos com Flutter usando qualquer editor de texto em conjunto com uma ferramenta de linha de comando. Entretanto, a documentação oficial recomenda o uso dos editores Android Studio, IntelliJ, VS Code ou Emacs, por possuírem plugins que auxiliam no desenvolvimento do código.

3. Materiais e Métodos

3.1. React Native

Para utilizar o React Native é importante compreender os fundamentos do JavaScript e a linguagem de marcação JSX, além de conhecer alguns conceitos de React, como componentes, propriedade e estado. Os componentes (*components*) são os principais blocos na construção de um aplicativo. Eles são como funções que recebem entradas e retornam elementos que descrevem como a UI deve ser exibida. A propriedade (*props*) é utilizada para passar dados que não serão modificados para um componente quando ele é renderizado e o estado (*state*) é útil ao lidar com dados que sofrem alterações com o tempo ou conforme ocorre uma interação do usuário. Toda vez que um estado muda, a UI é renderizada novamente para refletir as alterações, mas o React Native preserva o estado anterior entre as novas renderizações mantendo assim um histórico de cada modificação.(LIM, 2020)

Ao escolher o React Native para desenvolver uma aplicação móvel, será necessário preparar o ambiente instalando algumas ferramentas auxiliares. A primeira delas é o NodeJS, pois utiliza-se o *npm* (*Node Package Manager*) para gerenciar as dependências dos aplicativos e instalar todos os demais pacotes via linha de comando -

há também a opção de utilizar o gerenciador de pacotes *yarn*. Para iniciantes no desenvolvimento móvel, é mais indicado configurar o ambiente com a Expo CLI, que é um conjunto de ferramentas e recursos desenvolvidos para facilitar o uso de React Native. Já os desenvolvedores que já estão familiarizados com esse tipo de desenvolvimento podem optar por utilizar o React Native CLI, que requer a instalação prévia do Xcode ou do Android Studio. A documentação disponível no website oficial do React Native explica passo a passo como realizar essas configurações e recomenda o uso do editor de texto VS Code, que possui plugins muito úteis para auxiliar no desenvolvimento.(REACT NATIVE, 2021)

Em sua arquitetura atual a comunicação entre o thread JS e o nativo ocorre através de uma ponte (*bridge*), conforme ilustra a Figura 2. Toda vez que a comunicação é necessária, os dados são serializados como JSON, enviados a uma fila e decodificados após a chegada para então serem processados, sendo todo esse processo assíncrono. Esta arquitetura apresenta algumas desvantagens, por exemplo, não se tem referência direta ao elemento nativo, não permite controle das operações uma vez que o processo é assíncrono e em casos mais complexos pode ocorrer lentidões na performance de renderização. (FELDMAND, 2019)

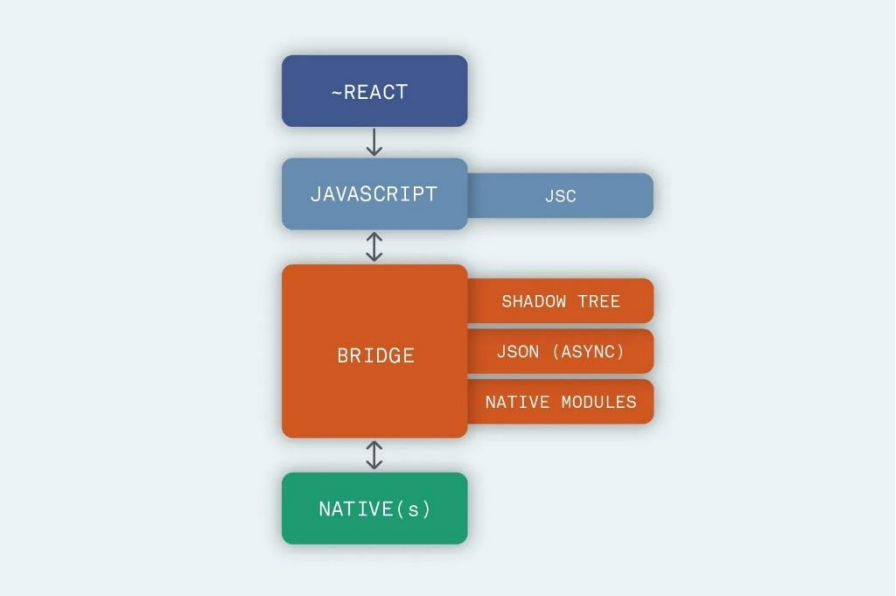

Figura 2. Arquitetura antiga do React Native
Fonte: (Sciandra, 2019)

Em busca de melhorar o desempenho e tornar o framework ainda mais atrativo, o time do React Native decidiu investir fortemente em mudanças na sua arquitetura para resolver os problemas apresentados e, inclusive, preparar o ambiente para expandir a reutilização de código em diferentes plataformas além do Android e do iOS, como Windows, MacOS e VR. Na nova arquitetura, apresentada na Figura 3, a principal mudança ocorre com o novo renderizador Fabric que deixa de utilizar a ponte para se comunicar com o lado nativo e passa a utilizar o JSI (*JavaScript Interface*), que permite a comunicação síncrona entre o JavaScript e a interface nativa. Esta nova arquitetura já vem sendo testada em produção em larga escala, conforme divulgação recente do time do Facebook, porém ainda não há uma data de liberação oficial dos novos recursos. (GROSS, 2021)

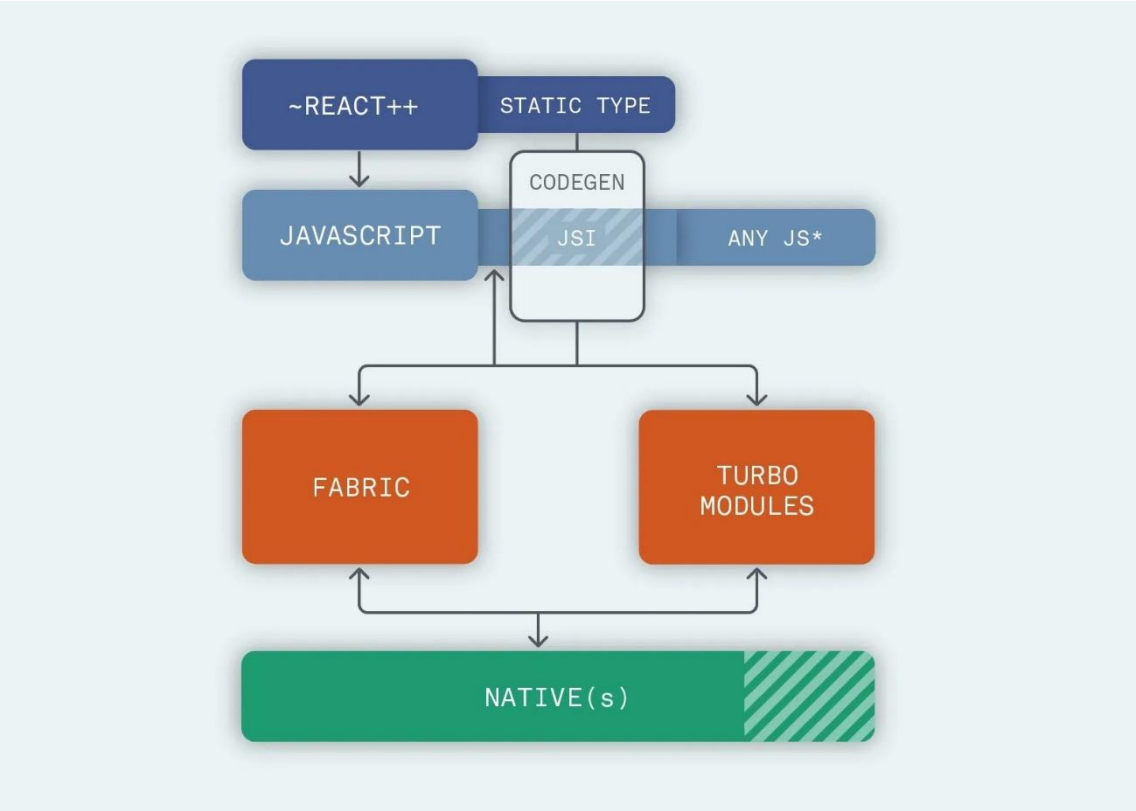

Figura 3. Nova arquitetura do React Native
Fonte: (Sciandra, 2019)

3.2. Flutter

O Flutter possui uma estrutura em camadas, e possui um pacote de componentes embutidos (*in-built*) que possui todas as tecnologias necessárias para o desenvolvimento de aplicativos móveis. Ele utiliza a estrutura Dart com um mecanismo Skia C++ e não precisa de uma ponte (*bridge*) ou meio de interação com os módulos nativos. A estrutura em camadas do Flutter está ilustrada na figura 4 (ARQUITETURA FLUTTER, 2021; LIMA, 2021).

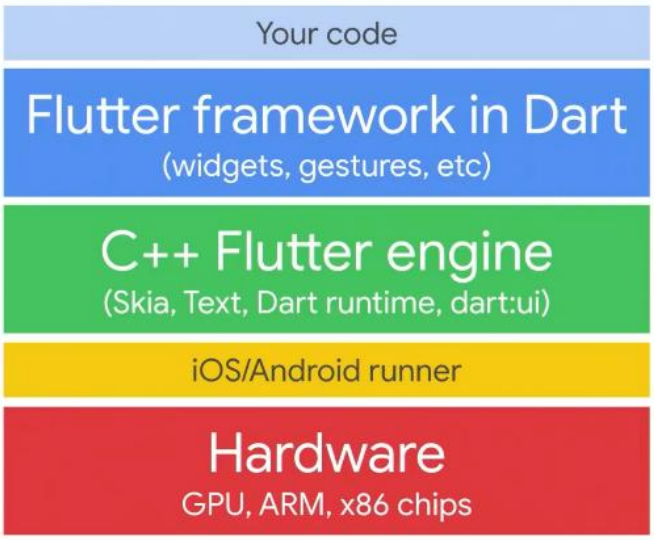

Figura 4: Camadas do flutter
Fonte: (Arquitetura Flutter, 2021)

A primeira camada representa o código Dart do aplicativo, códigos específicos de cada plataforma (caso haja), *packages* e *plugins*, imagens, fontes e qualquer outro ativo do projeto.

O Dart framework também é uma estrutura em camadas, ilustrada na figura 5. Os elementos chave nesta seção da estrutura são as classes básicas, *widget* e camadas de renderização, serviços como animação, pintura, gestos e bibliotecas (Cupertino e Material). A camada *widget* ajuda na definição das classes reutilizáveis e a renderização ajuda a lidar com as operações de layout (LIMA, 2021).

O Flutter *engine* é o responsável por unir o código do usuário e o do framework e prepará-los para que sejam executados pelo hardware. Permite o suporte da renderização para todas as aplicações. A linguagem usada neste mecanismo é C++ e é quem implementa as bibliotecas do Flutter, API, animações, gráficos, arquitetura de *plug-in* etc. Esta camada também se concentra em renderização, agendamento de quadros e *pipelining*, gerenciamento de tempo de execução, layouts de texto e resolução de ativos (OGLIARI, 2021).

Depois dessa etapa, o *runner* se encarrega de delegar e controlar as instruções binárias ao hardware.

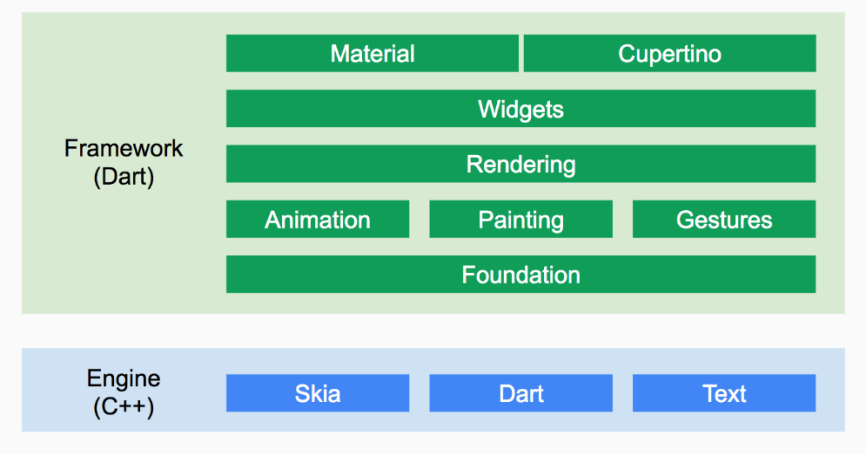

Figura 5: Camadas do Framework dart e Engine (C++)
Fonte: (Arquitetura Flutter, 2021)

Para configurar o ambiente, primeiramente é necessário escolher o sistema operacional que o Flutter irá trabalhar, podendo ser Windows, MacOS, Linux ou ChromeOS com Linux. O sistema escolhido deve atender a alguns requisitos mínimos de sistema, conforme presentado na tabela 1. Além disso, é necessário obter e instalar o Flutter SDK, onde se encontram os pacotes e ferramentas de linha de comando necessários para desenvolver aplicativos entre plataformas (FLUTTER DEV - SYSTEM REQUIREMENTS, 2021).

Sistema Operacional	Espaço em disco	Ferramentas
Windows 7 SP1 ou posterior (64 bits), baseado em x86-64.	1,64 GB (não inclui espaço em disco para IDE / ferramentas).	Windows PowerShell 5.0 ou mais recente (pré-instalado com o Windows 10) Git para Windows 2.x, com a opção **Usar Git no prompt de comando do Windows** .
macOS	2,8 GB (não inclui espaço em disco para IDE / ferramentas).	Flutter usa gitpara instalação e atualização. Recomenda-se instalar o Xcode , que inclui git, mas você também pode instalar gitseparadamente .
Linux (64 bits)	600 MB (não inclui espaço em disco para IDE / ferramentas).	**Ferramentas** de linha de comando disponíveis em seu ambiente: bash; curl; file; git 2.x; mkdir; rm; unzip; which; xz-utils; zip. **Bibliotecas compartilhadas** : o testcomando Flutter depende desta biblioteca estar disponível em seu ambiente: libGLU.so.1- fornecido por pacotes mesa, como libglu1-mesano Ubuntu / Debian e mesa-libGLUno Fedora.
Chrome OS (64 bits) com Linux (Beta) ativado	600 MB (não inclui espaço em disco para IDE / ferramentas).	**Ferramentas** de linha de comando disponíveis em seu ambiente: bash; curl; git 2.x; mkdir; rm; unzip; which; xz-utils. **Bibliotecas compartilhadas** : o testcomando Flutter depende desta biblioteca estar disponível em seu ambiente: libGLU.so.1- fornecido por pacotes mesa, como libglu1-mesano Ubuntu / Debian

Tabela 1. Requisitos de sistema para execução do Flutter.
Fonte: (Autoria própria, 2021)

O Flutter SDK contém: Dart SDK; Mecanismo de renderização 2D otimizado para dispositivos móveis com suporte para texto; Estrutura moderna de estilo de reação; Conjunto rico de *widgets* que implementam materiais design e estilos iOS; APIs para testes de unidade e integração; APIs de interoperabilidade e plug-in para conectar ao sistema e SDKs de terceiros; *Headless test runner* para executar testes no Windows, Linux e Mac; Dart DevTools para testar, depurar e criar perfis de seu aplicativo (FLUTTER DOCUMENTATION, 2021).

No Flutter, tudo é desenvolvido a partir de *widgets*, que são os componentes de aplicação que permitem ao desenvolvedor, interagir com várias funcionalidades. Eles descrevem a aparência de sua visualização de acordo com sua configuração e estado atuais e os principais utilizados são: *Container; Image; Text; Icon; RaisedButton; Scaffold; Appbar; PlaceHolder; Row; Column; ListView*. Está disponível no site da ferramenta um catálogo com suas respectivas funcionalidades (FLUTTER DOCUMENTATION, 2021).

Com o uso dos *Widgets*, os componentes criados são altamente personalizáveis, oferecendo ao desenvolvedor, maior flexibilidade na construção de UIs mais amigáveis ao usuário final. Um ponto importante, é que isso pode resultar em aplicativos com classes muito grandes, modos de nomeação inconsistentes e arquiteturas incompatíveis com a necessidade, podem impactar em problemas, por isso, aplicações de média e grande complexidade vêm tomando a *Clean Architecture* como uma das melhores soluções, de modo a criar uma arquitetura que seja facilmente escalável, testável e manutenível (MICHIURA, 2020).

4. Resultados e Discussões

Foram selecionados alguns estudos comparativos entre os frameworks Flutter e React Native, listados na Tabela 2. Para avaliar as ferramentas, alguns autores optaram por realizar testes de desempenho enquanto outros analisaram características de infraestrutura além de critérios relacionados ao processo de desenvolvimento. Nos trabalhos analisados, fica claro que são pequenas características que influenciam na escolha entre essas tecnologias e que esta escolha varia de acordo com os critérios considerados mais relevantes para cada aplicação.

O Flutter foi o framework mais escolhido, considerando principalmente critérios como desempenho em aplicativos com listas muito grandes ou cálculos de alta carga, experiência no desenvolvimento, facilidade de automação de testes e menor propensão a mudanças significativas. O React Native destacou-se com relação ao apoio da comunidade, a maturidade da ferramenta e o menor custo inicial para desenvolvimento de novas aplicações.

Referência	Ano	Autor(es)	Metodologia de comparação
React Native vs Flutter, cross-platform mobile application frameworks	2018	Wenhao Wu	Estudo de caso de Desempenho (FPS), Roteamento (navegador) e visualizações (modularidade e estilo).
Performance comparison between React Native and Flutter	2019	Jakub Jagiełło	Testes de desempenho para verificar como efetivamente as estruturas estão sustentando os FPS (frames per second) em cenários diferentes.
Mobile Architecture Task Force - Why we think Flutter will help us scale mobile development at Nubank	2019	Alexandre Freire, André Moreira e outros	Experiência de desenvolvimento (testes de usuário, depurador, estabilidade, relatório de falhas), viabilidade de longo prazo (comunidade), proximidade da plataforma nativa, facilidade ao incrementar extensões, documentação, histórico de mudanças significativas, restrições em app stores, limitações, custo inicial.
Flutter VS React Native	2020	Sathish Kumar	Linguagem de programação, estabilidade, apoio da comunidade, desempenho (componentes embutidos, arquitetura), documentação, componentes de UI (APIs, bibliotecas), automação da construção e lançamento de aplicações.
Flutter vs Native vs React-Native: Examining Performance	2020	Ihor Demedyuk & Nazar Tsybulskyi	Teste com uso intensivo de memória (algoritmo Gauss–Legendre) e CPU (algoritmo Borwein) em iOS e em Android.

Tabela 2. Estudos comparativos envolvendo os frameworks Flutter e React Native
Fonte: (Autoria própria, 2021)

Em termos de empregabilidade, foram realizadas pesquisas nos sites da plataforma de empregos indeed.com de 25 países para analisar a oferta de empregos para cada framework. A primeira pesquisa foi realizada procurando por vagas que citavam o nome de cada ferramenta em qualquer parte de seu anúncio e a segunda pesquisa considerou apenas as vagas cujo nome de cada ferramenta apareciam no título da vaga ofertada. Os resultados da pesquisa podem ser consultados na tabela 3.

País	Pesquisa 1				Pesquisa 2			
	Flutter		"React Native"		title:Flutter		title:"React Native"	
	Nº vagas	%	Nº vagas	%	Nº vagas	%	Nº vagas	%
Brasil	**560**	**46,7%**	**640**	**53,3%**	**111**	**47,4%**	**123**	**52,6%**
EUA	885	15,3%	4.900	84,7%	109	9,8%	1.000	90,2%
Índia	1.327	36,5%	2.308	63,5%	589	40,6%	861	59,4%
Japão	2.354	84,3%	437	15,7%	244	94,6%	14	5,4%
França	1.840	74,1%	643	25,9%	49	30,8%	110	69,2%
Alemanha	574	38,1%	934	61,9%	168	62,7%	100	37,3%
Reino Unido	314	29,8%	740	70,2%	23	14,9%	131	85,1%
Canadá	200	24,3%	623	75,7%	16	21,1%	60	78,9%
México	149	29,5%	356	70,5%	50	37,9%	82	62,1%
Holanda	103	21,1%	385	78,9%	19	23,2%	63	76,8%
Polônia	131	28,1%	335	71,9%	44	33,6%	87	66,4%
Itália	153	40,4%	226	59,6%	17	47,2%	19	52,8%
Suécia	81	23,1%	269	76,9%	4	10,0%	36	90,0%
Indonésia	136	46,3%	158	53,7%	22	38,6%	35	61,4%
Austrália	47	17,3%	225	82,7%	8	15,1%	45	84,9%
Espanha	70	26,0%	199	74,0%	8	24,2%	25	75,8%
Argentina	74	28,5%	186	71,5%	32	46,4%	37	53,6%
China	144	61,3%	91	38,7%	11	73,3%	4	26,7%
Rússia	64	31,7%	138	68,3%	20	39,2%	31	60,8%
Portugal	45	27,1%	121	72,9%	6	17,1%	29	82,9%
África do Sul	59	37,1%	100	62,9%	6	30,0%	14	70,0%
Turquia	47	40,2%	70	59,8%	12	30,0%	28	70,0%
Coreia do Sul	36	35,3%	66	64,7%	5	23,8%	16	76,2%
Suíça	24	29,6%	57	70,4%	2	22,2%	7	77,8%
Arábia Saudita	6	42,9%	8	57,1%	5	55,6%	4	44,4%

Tabela 3. Ofertas de emprego no site indeed.com: Flutter vs. React Native
Fonte: (Autoria própria, 2021)

Observa-se que, com exceção do Japão, França, Alemanha, China e Arábia Saudita, a oferta de empregos para React Native foi maior nos dois tipos de pesquisas realizadas. É importante considerar que este framework foi lançado alguns anos antes do do Flutter e que realizar a migração entre estas ferramentas nem sempre é um processo vantajoso e viável.

Com relação ao Brasil, é interessante destacar que a oferta de empregos está equilibrada em ambas as pesquisas. Ao analisar a distribuição das vagas encontradas na pesquisa 2 por região, verifica-se que as ofertas estão concentradas nas regiões sudeste e sul e que as vagas para se trabalhar com Flutter já são maioria na região sul, conforme ilustrado na Figura 6. Dentre essas vagas, em quase 63% existe a possibilidade de trabalho remoto.

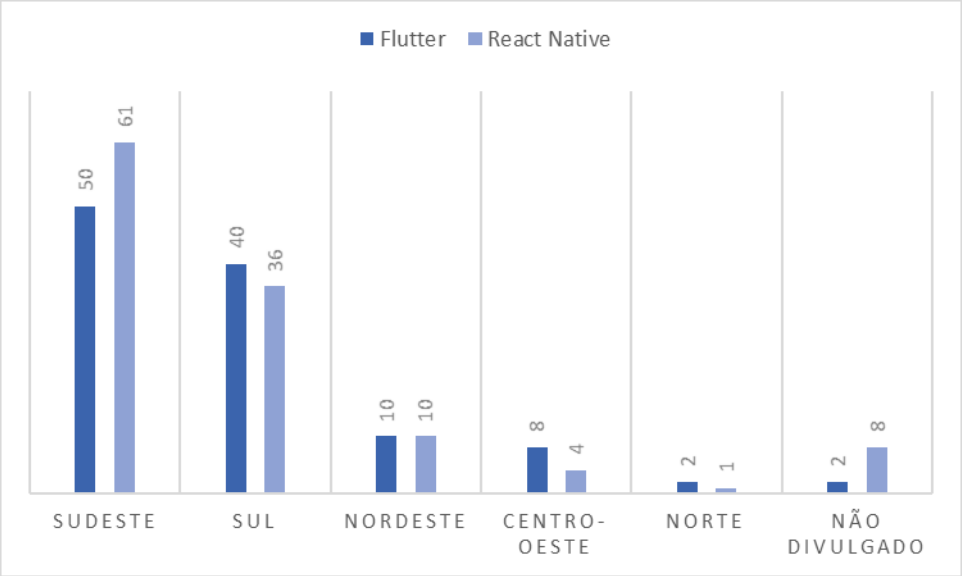

Figura 6: Vagas no Brasil de cada ferramenta por região
Fonte: (Autoria própria, 2021)

Cabe destacar também os aplicativos que possuem telas construídas com estes frameworks. Aplicativos como Facebook, Instagram, Pinterest, Shopify e Uber Eats optaram por adotar o React Native como opção multiplataforma. Já empresas como a Google, BMW, Nubank, Alibaba e Ebay fazem uso do Flutter em suas aplicações móveis. No website oficial de cada ferramenta é possível consultar informações mais detalhadas de cada uma das empresas mencionadas, além de conhecer outros clientes que as utilizam.

Outro ponto que deve ser levado em consideração na escolha de uma das ferramentas está relacionado ao conhecimento técnico prévio e à curva de aprendizado. Para utilizar o React Native será necessário aprender duas tecnologias e não apenas uma, como é o caso do Flutter, o que deve ser ponderado especialmente por quem está começando na área de desenvolvimento mobile.

5. Conclusão

O sucesso de um projeto de desenvolvimento de software depende de vários fatores sendo um dos seus principais a escolha da tecnologia que será utilizada. Em se tratando do desenvolvimento de aplicações móveis, há inúmeras possibilidades que vão desde aplicações nativas, web apps, híbridas às multiplataformas.

Nesta pesquisa foram analisadas as ferramentas de desenvolvimento multiplataforma Flutter e React Native por manterem o desempenho nativo e serem as soluções de código aberto mais utilizadas na construção de aplicativos móveis para os sistemas operacionais Android e iOS.

O framework Flutter se mostrou mais atrativo, especialmente no que diz respeito ao seu funcionamento simplificado, melhor desempenho, menor curva de aprendizagem e menor propensão a mudanças significativas. Além disso, nota-se que as oferta de emprego vem crescendo constantemente, estando equilibradas quando se analisa o mercado de trabalho brasileiro. Entretanto, é importante considerar que muitas vezes a escolha por um dos frameworks não cabe apenas ao desenvolvedor, mas sim à empresa para qual ele presta serviços.

Referências o ano de publicação deve ser após autor

ANJOS, L. H. dos. Android e Kotlin – Flutter vs React Native. In: **Anais**...The Developer's Conference, 2020.

Arquitetura FLUTTER. Disponível em: <https://www.flutterparainiciantes.com.br/arquitetura>. Acesso em: 25 set. 2021.

DAHL, O. Exploring End User's Perception of Flutter Mobile Apps. **MALMO University**, 2019. Disponível em: <https://www.diva-portal.org/smash/record.jsf?pid=diva2:1480395>. Acesso em: 25 set. 2021.

DEMEDYUK, I.; TSYBULSKYI, N. **Flutter vs Native vs React-Native: Examining Performance**. (2020). Disponível em: < https://inveritasoft.com/blog/flutter-vs-native-vs-react-native-examining-performance>. Acesso em: 18 ago. 2021.

FELDMAND, C. **React Native — A Bridge To Project Fabric — Part 1**. (2019). Disponível em: <https://medium.com/swlh/react-native-a-bridge-to-project-fabric-part-1-5af6a53c0d83>. Acesso em: 15 set. 2021.

Flutter Dev - System requirements. Disponível em: <https://flutter.dev/docs/get-started/install/chromeos>. Acesso em: 20 set. 2021.

Flutter Documentation. Disponível em: <https://flutter.dev/docs>. Acesso em: 12 ago. 2021.

FREIRE, A. et al. Mobile Architecture Task Force - Why we think Flutter will help us scale mobile development at Nubank. **Nubank**, 2019.

GROSS, J. The New React Native: Bringing the Fabric renderer to the "Facebook" app. In: **Anais**... React Native EU 2021 - Virtual Edition, 2021.

Indeed. Disponível em: <https://br.indeed.com/>. Acesso em: 26 out. 2021.

JAGIELLO, J. **Performance comparison between React Native and Flutter.** Umea University, p. 26, 2019.

KUMAR, S. Flutter vs React Native. A comparison study on Hybrid technologies for secure mobility solution. **OptiSol Business Solutions**, 2020.

LIM, G. **Beginning React with Hooks**. [s.l: s.n.], 2020.

LIMA, R. **Flutter em 2021 (parte 1) : Arquitetura**. (2021).Disponível em: <https://digitalinnovation.one/artigos/flutter-em-2021-parte-1-arquitetura>. Acesso em: 24 set. 2021.

LIU, S. **Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021**. Disponível em: <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>. Acesso em: 10 ago. 2021.

MASIELLO, E.; FRIEDMANN, J. **Mastering React Native**. Birmingham: Packt Publishing, 2017.

MICHIURA, F. **A importância da Clean Architecture no Flutter**. (2020). Disponível em: <https://www.objective.com.br/insights/a-importancia-da-clean-architecture-no-flutter/>. Acesso em: 18 ago. 2021.

OGLIARI, R. S. **Uma Idéia para Arquitetura de Aplicativos Flutter**. (2021). Disponível em: <https://dev.to/ricardoogliari/uma-ideia-para-arquitetura-de-

aplicativos-flutter-32cn>. Acesso em: 10 set. 2021.

React Native. Disponível em: <https://reactnative.dev/docs/getting-started>. Acesso em: 14 ago. 2021.

SCIANDRA, L. **The New React Native Architecture Explained: Part Four**. (2019). Disponível em: <https://formidable.com/blog/2019/lean-core-part-4/>. Acesso em: 3 set. 2021.

STATCOUNTER. **Desktop vs Mobile vs Tablet Market Share Worldwide.** Disponível em: <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#yearly-2016-2021>. Acesso em: 30 out. 2021.

STATISTA. **Number of mobile app downloads worldwide from 2016 to 2020.** Disponível em: <https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads>. Acesso em: 26 out. 2021.

WINDMILL, E. **Flutter in Action**. New York: Manning Publications Company, 2020.

WU, W. React Native vs Flutter, cross-platform mobile application frameworks. **Metropolia University**, n. March, p. 28, 2018.