UMA ANÁLISE COMPARATIVA ENTRE FLUTTER E REACT NATIVE COMO FRAMEWORKS PARA DESENVOLVIMENTO HÍBRIDO DE APLICATIVOS MOBILE: ESTUDO DE CASO VISANDO PRODUTIVIDADE

Jonathan Neves, Vilmar Mendes Junior
Curso de Ciência da Computação
Universidade do Sul de Santa Catarina (UNISUL) – Tubarão, SC – Brazil
nevesjon98@gmail.com; vilmar.mendes@unisul.br;

Resumo: Com a popularização das aplicações móveis e com os *smartphones* cada vez mais acessíveis para as pessoas, é natural o surgimento de uma grande demanda por desenvolvimento de aplicativos para resolver diversos tipos de problemas. Para uma empresa, que visa ter o menor custo, ou para desenvolvedores que desejam ter a maior produtividade durante o processo de desenvolvimento, a escolha de uma boa ferramenta é fundamental para a criação e manutenção de uma aplicação. Este trabalho visa realizar uma análise comparativa entre os dois *frameworks* mais populares atualmente do mercado voltados para o desenvolvimento híbrido de aplicativos *mobile*, focando pontualmente na produtividade durante o processo de desenvolvimento.
Palavras-Chave: Produtividade, Desenvolvimento Híbrido, Flutter, React Native.

Abstract: As mobile applications and *smartphones* become increasingly popular and accessible to the population, it is normal to have a large demand in the app development field for various sorts of problems. The choice of a good development *framework* is essential for the creation and maintenance of an application, mostly for a company whose goal is having the lowest cost possible or for developers who wish to have good productivity while developing a project. This article aims to carry out a comparative analysis between the two most popular hybrid mobile development *frameworks* in the market, focusing on productivity during the development process.
Keywords: Productivity, Hybrid Development, Flutter, React Native.

1 INTRODUÇÃO

As aplicações móveis ajudam o usuário em diversos momentos do cotidiano por conta de sua praticidade de uso e facilidade na comunicação. Devido a isso, há um crescimento alto na demanda de aplicações *mobile,* entretanto:

> O custo para a criação de cada aplicação é alto, contando com o custo dos recursos, tempo, manutenção e evolução, portanto, manter dois ou mais aplicativos é uma tarefa cara. Para amenizar este problema, foram criados frameworks para desenvolvimento de aplicações móveis, ou seja, é utilizado uma plataforma base para implementar uma única aplicação. (FORTUNATO DE LIMA, 2019, p. 13).

Por isso, o *framework* é uma aplicação que oferece um conjunto de ferramentas que torna o desenvolvimento de aplicações mais produtivo e flexível.

Entretanto, uma dúvida recorrente que acaba surgindo entre desenvolvedores e equipes de desenvolvimento no início de um projeto, ou aqueles que desejam aprender uma nova tecnologia, é qual *framework* escolher, principalmente pelo número de vantagens e desvantagens que cada um apresenta. Tanto *Flutter* quanto *React Native* são os *frameworks open source* de desenvolvimento híbrido para aplicativos *mobile* mais populares atualmente. De acordo com a plataforma de hospedagem de código fonte GITHUB [2020], o *React Native* conta com mais de 90 mil *stars*, e o *Flutter* com mais de 105 mil *stars*. Com base no site de estatísticas STATISTA [2019], *React Native* é o terceiro *framework* mais procurado entre os desenvolvedores, e o *Flutter* é o sexto, considerando *frameworks* de várias categorias. Os números de aplicações móveis só tendem a aumentar e os *frameworks* auxiliam na produtividade e qualidade dos serviços desenvolvidos.

Portanto, este é um trabalho acadêmico realizado por estudantes, o qual visa realizar uma análise comparativa entre os dois *frameworks*, *Flutter* e *React Native,* a partir de um aplicativo desenvolvido do zero, utilizando ambos os *frameworks* sem conhecimento prévio com desenvolvimento mobile, para que ambos os autores possam começar no mesmo nível de conhecimento. Foram registradas as dificuldades encontradas e o tempo gasto de desenvolvimento, apontando as comparações gerais relacionadas à produtividade, como curva de aprendizado e adaptação. O intuito deste artigo é auxiliar de forma positiva na escolha de *frameworks* para desenvolvedores e equipes de desenvolvimento, baseado em suas principais necessidades de projeto para desenvolvimento *mobile*.

2 CONTEXTUALIZAÇÃO

2.1 MOTIVAÇÃO

A produtividade, conforme define o site XERPA [2019, p. 1], "É a relação entre a quantidade do que é produzido com relação aos insumos utilizados ou quantidade de tempo investido no processo". Com a tecnologia e toda a competição de mercado, existe a necessidade por parte das empresas de agilizar o processo de desenvolvimento ao mesmo tempo que desejam manter a qualidade do produto final.

Durante a escolha dos critérios e parâmetros de avaliação entre os dois *frameworks*, *Flutter* e *React Native,* percebeu-se uma grande escala de artigos semelhantes comparando diversos *frameworks,* muitas vezes, chegando em conclusões semelhantes. Artigos esses que utilizam critérios como desempenho, performance, arquitetura, interface, além de opiniões gerais e prós e contras.

O critério escolhido para a comparação é a produtividade, sendo o principal diferencial e englobando parâmetros como curva de aprendizado, dificuldades encontradas, tempo de desenvolvimento, código, documentação e conteúdo online. O intuito é analisar o rendimento de desenvolvimento, para um desenvolvedor com zero conhecimento da ferramenta ou sem experiência com desenvolvimento de *apps*, facilidade de aprendizado e adaptação,

possivelmente servindo como métrica para desenvolvedores que possam escolher o *framework* ideal para determinado cenário de projeto.

2.2 DESENVOLVIMENTO HÍBRIDO

O termo *Cross-Platform Development* (Desenvolvimento Multi Plataformas), também conhecido como Desenvolvimento Híbrido, é a prática de desenvolver produtos de software para múltiplas plataformas ou ambientes de software. Engenheiros e desenvolvedores utilizam vários métodos para acomodar diferentes sistemas operacionais ou ambientes para uma única aplicação ou produto (CROSS-PLATFORM DEVELOPMENT, 2020).

Os *frameworks* de desenvolvimento híbrido são populares por permitirem o desenvolvimento multi plataformas, possibilitando um único código ser compilado para várias plataformas sem a necessidade de reescrevê-lo, tornando estas aplicações mais práticas e econômicas para o mercado.

Tanto o *Flutter* quanto o *React Native* são *frameworks* para desenvolvimento híbrido, tendo como principal vantagem o código único para várias plataformas, como *Android* e *iOS* e até para *Web,* garantindo maior facilidade no desenvolvimento, custos mais baixos e tornando a manutenção mais simples para todas as plataformas.

2.3 FLUTTER

Como cita no livro *Flutter* in Action publicado em dezembro de 2019 por WINDMILL, Eric:

> *Flutter* é um *sdk mobile* (Kit de Desenvolvimento de Software Móvel) e *open-source* desenvolvido pelo *Google,* a sua essência é sobre capacitar todos a criar belos dispositivos móveis. Seja você do mundo do desenvolvimento web ou do desenvolvimento móvel nativo, o *Flutter* facilita mais do que nunca a criação de aplicativos móveis de uma maneira familiar e simplificada. (2019, Cap. 1, p. 1).

Complementa em seguida o autor Windmill de forma breve sobre o *framework Flutter*: "É uma plataforma que fornece tudo o que você precisa para criar aplicativos: mecanismo de renderização, componentes da *UI*, estruturas de testes, ferramentas, um host e muitos outros recursos necessários para criar um aplicativo" (2019, Cap. 1, p. 1). Os aplicativos feitos em *Flutter* são implementados para *Android* e *iOS* a partir do mesmo código. Atualmente é possível também criar aplicativos *Web* ou *Desktop* para os principais sistemas operacionais.

Os aplicativos *Flutter* são escritos na linguagem de programação chamada *Dart, que* também pertence e é mantida pelo *Google*. Não é uma das linguagens mais utilizadas em produção pelas empresas, porém vem se destacando pelos seus recursos e componentes que facilitam a construção de *UI* e facilidade de aprendizado.

2.4 REACT NATIVE

De acordo com o livro *Learning React Native* (EISENMAN, 2018, p. 17) "*React Native* é um *framework JavaScript* utilizado para escrever aplicações reais e de renderização nativa para *iOS* e *Android"*. Possui grandes semelhanças com o *ReactJS*, seu *framework* equivalente para escrever aplicações *web.* Ambos os *frameworks* são *open source*, e foram criados pela empresa *Facebook*.

O fato de utilizar-se a mesma linguagem e estrutura tanto para o *React* quanto para o *React Native* é um dos principais fatores que leva os desenvolvedores e equipes a escolher esse ecossistema. Com uma única linguagem e sintaxe, é possível escrever aplicações tanto para *web* quanto para *mobile*, o que torna possível a reusabilidade de código entre os dois *frameworks.*

3 TRABALHOS CORRELATOS

3.1 AVALIAÇÃO DE FRAMEWORKS PARA O DESENVOLVIMENTO DE APLICAÇÕES HÍBRIDAS

Neste trabalho são comparados pelo autor Fortunato de Lima [2019] os *frameworks Flutter, Ionic, NativeScript, Vue* e *React Native*. Para realizar o estudo foram desenvolvidas quatro mini aplicativos para analisar as funcionalidades nativas do celular, além de um levantamento de questionário para coletar opiniões de desenvolvedores estudantes a respeito dos *frameworks*. Entre as mini aplicações desenvolvidas temos:

- Teste de Câmera: a possibilidade de fotografar e filmar utilizando este recurso via *framework*;
- Teste de *GPS*: identificar a posição atual do usuário utilizando este recurso via *framework*;
- Teste de Armazenamento: avaliar gravação, alteração e remoção de dados salvos em memória interna;
- Teste de *Bluetooth*: conectar o dispositivo ao *Bluetooth*.

Após os testes de comparação entre todas as aplicações entre os frameworks, é mensurado a média do tempo de execução, do consumo de memória e do uso de processador para cada aplicação.

Focando na conclusão dos dois *frameworks* principais, *Flutter* e *React Native*, o autor conclui, "O *Flutter* apresentou um bom compromisso entre os critérios analisados, obtendo o menor consumo médio de memória, o segundo menor tempo médio de execução e o terceiro menor uso de processador" (FORTUNATO DE LIMA, 2019, p.75), vale ressaltar que teve desempenho superior principalmente em iOS.

Continuando a conclusão do autor: "O React Native obteve o menor consumo de memória e uso de processador, sendo o terceiro em tempo de execução [...] Dentre todos os frameworks, o React Native apresentou o tempo mais longo para geração de arquivo de execução." (FORTUNATO DE LIMA, 2019, p.75).

3.2 REACT NATIVE VS FLUTTER, FRAMEWORK MULTIPLATAFORMA PARA APLICAÇÕES MÓVEIS

Nesta dissertação é utilizada uma abordagem bastante interessante para realizar a análise comparativa entre *React Native* e *Flutter*. Primeiramente, foi desenvolvido pelo autor Wu [2018] uma aplicação de cinema com quatro telas utilizando alguns recursos do próprio

framework. Em seguida são analisados alguns critérios como:

- *Routing* (Roteamento) a forma na qual cada um projeta os caminhos de rota para navegar entre tela;
- *Views* (Interface) compara a modularidade dos componentes na tela e a "estilo" que é aplicada na *View*;
- Performance comparando o *FPS* (quadros por segundo) entre eles. Primeiro é realizado um teste de Scroll Vertical com 1000 itens na tela e mensurado o FPS. Segundo é avaliado a velocidade de leitura e escrita de arquivos internos, para isso é aberto pelo aplicativo um arquivo de texto, escrito uma frase e no final o arquivo é gravado, calculando a média de tempo para realizar essa função.

Com base nas análises do autor, ele conclui:

> React Native possui conceitos inovadores, como o fluxo de dados unidirecional e JSX, são bem adotados e digeridos no React Native. Com sua forte comunidade, é uma ótima opção para iniciar um aplicativo multiplataforma do zero. *Flutter* apesar de novo, tem boas expectativas para o futuro. O design é bastante sofisticado em relação ao React Native. A consistência e a organização da sintaxe e do nível do SDK são um dos pontos positivos para os desenvolvedores. A renderização de widgets por meio de um mecanismo dedicado aumenta o desempenho do mesmo. (WU, 2018, p.25)

4 METODOLOGIA

Para ser possível realizar a análise comparativa entre os *frameworks*, foi desenvolvido um aplicativo que se trata de um agregador de jogos e notícias relacionadas, onde o usuário pode ficar informado sobre as últimas notícias do mundo dos *games* e também obter informações gerais sobre eles. A criação do projeto foi dividida nas seguintes etapas:

- Prototipagem (1): Ideia do projeto, pesquisa de *APIs* para fornecer os dados e Protótipo da aplicação;
- Desenvolvimento (2): Divisão de tarefas de desenvolvimento, desenvolvimento da aplicação em etapas e registros da experiência de desenvolvimento por tarefa;
- Levantamento (3): Análise do desenvolvimento, seleção dos critérios de avaliação e conclusão dos resultados.

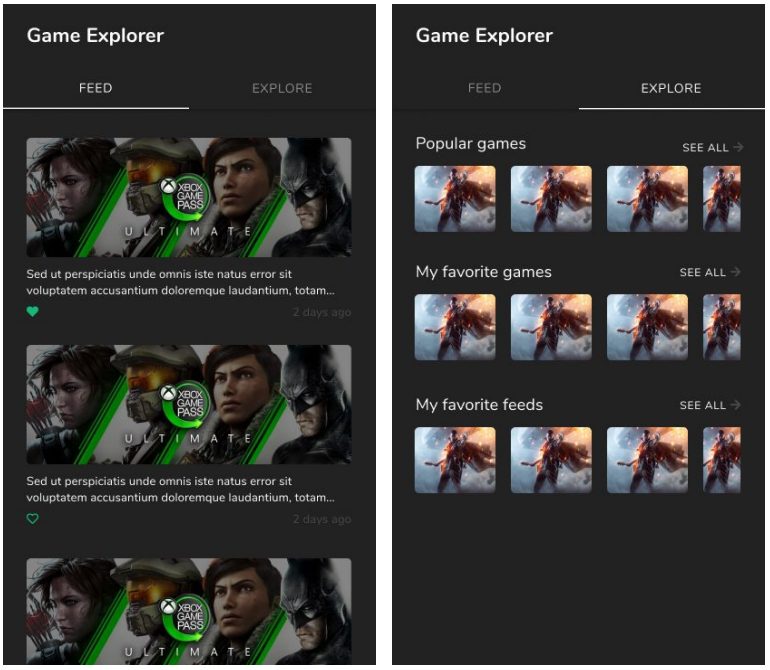
4.1 PROTOTIPAGEM

Para o início do projeto, tornou-se necessário uma aplicação base com uma boa diversidade de recursos para servir como testes para a comparação, com uma interface agradável e intuitiva. O protótipo do aplicativo pode ser visualizado nas Figuras 1 a 6.

Todo o aplicativo foi feito com dados reais consumidos pela *API* da *IGDB,* uma plataforma de jogos que disponibiliza um *web service* contendo diversas informações sobre o mundo dos jogos. Para consumir a *API,* foi feito um *backend* intermediário utilizando *NodeJS* com o *framework Express* e banco de dados não relacional *MongoDB,* que consome esse serviço

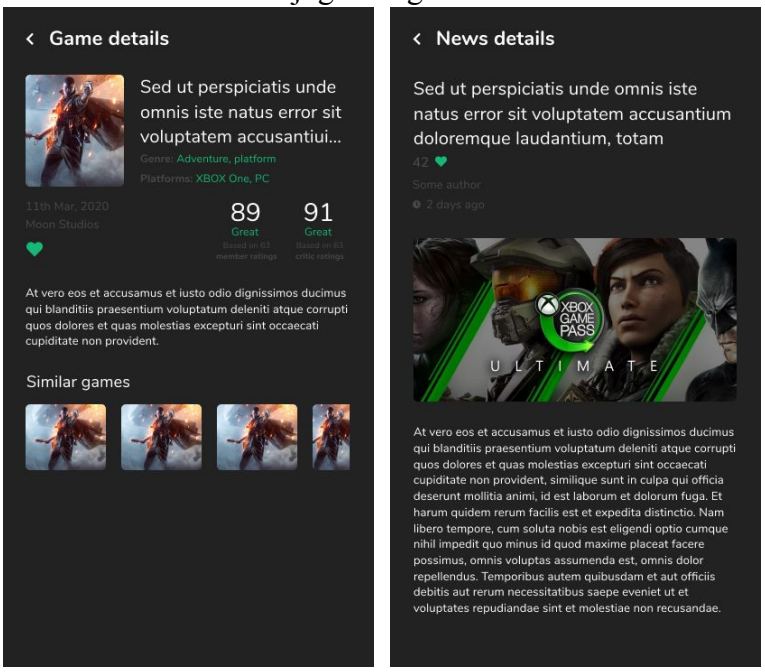
e disponibiliza os dados padronizados e formatados para ambos os aplicativos. O *backend* foi hospedado na plataforma de hospedagem *Heroku.*

Figura 1 - Feed de notícias e Figura 2 - Carrosséis com jogos e notícias



Fonte: Autores, 2020

Figura 3 - Detalhamento do jogo e Figura 4 - Detalhamento da notícia



Fonte: Autores, 2020

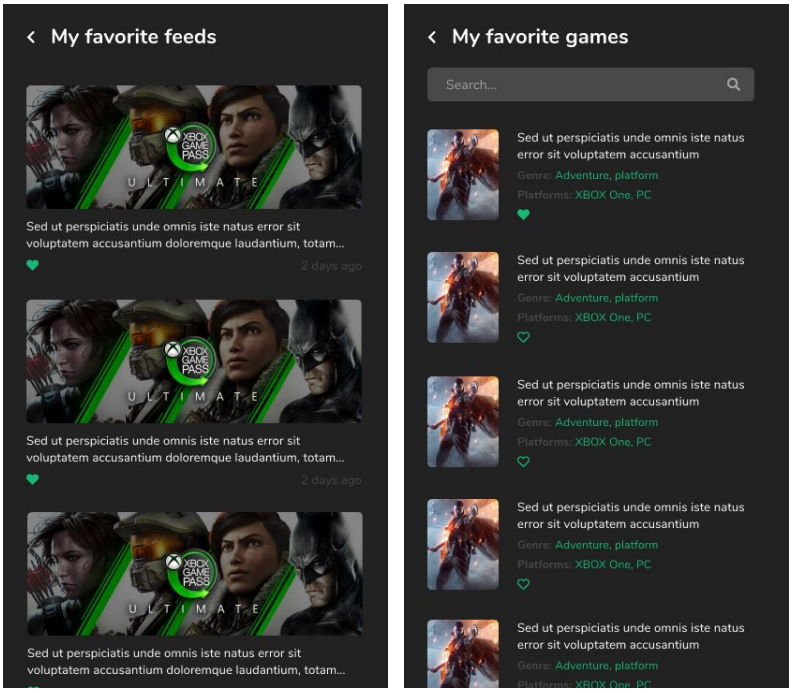
4.2 DESENVOLVIMENTO

Para a análise de produtividade, cada autor deste projeto desenvolveu o aplicativo utilizando um dos *frameworks*, sem conhecimento prático sobre eles ou experiência com desenvolvimento de aplicação móvel, para que a comparação possa ser analisada de forma imparcial. Para isso,

foram necessários alguns primeiros passos para conhecer o básico sobre os frameworks: para o autor responsável pela aplicação em *Flutter,* foi desenvolvido o app de exemplo da documentação do *Flutter,* na seção “Write your first app” (Escreva seu primeiro aplicativo). Já para o autor responsável pelo *React Native,* foi realizado primeiramente um curso de *JavaScript* e em seguida um curso de fundamentos de *React Native*, totalizando aproximadamente 10 horas de curso online. Ao longo do desenvolvimento do aplicativo, ambos os autores utilizaram as próprias documentações dos *frameworks*, além de materiais disponíveis na internet.

Antes de iniciar o desenvolvimento, foram criadas tarefas para cada etapa, compostas por funcionalidades e telas do aplicativo, para melhor organização no desenvolvimento e também para haver um controle do processo de levantamento de registros para a comparação de produtividade entre os *frameworks*.

Figura 5 - Listagem das notícias e Figura 6 - Listagem dos jogos



Fonte: Autores, 2020

4.3 LEVANTAMENTO

Durante o desenvolvimento, foram registradas, por cada autor, informações a respeito do desenvolvimento de cada etapa em questão, levantando pontos como: análise da etapa, tempo gasto de desenvolvimento, conteúdo utilizado para buscar a solução do problema, principais dificuldades encontradas e imagens da tarefa.

Baseado nas etapas realizadas, foram levantados alguns critérios julgados importantes para a comparação de produtividade entre *Flutter* e *React Native*: Instalação do *framework*, Emulação da Aplicação, Consumo de *API*, Estilização de Interface, Documentação, Conteúdos Online, Dependências Externas, Estabilidade de Atualizações, Build do Aplicativo, Pontos Positivos e Negativos.

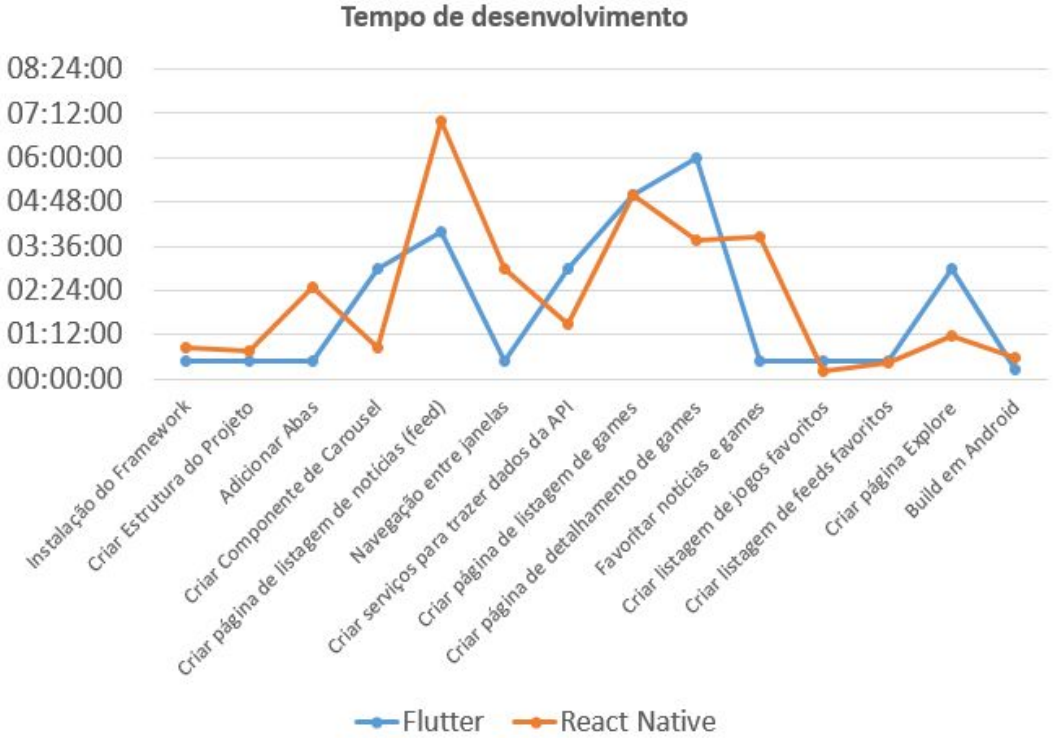
No final do estudo foram coletados todos os registros de desenvolvimento por ambos os autores analisando pontos fundamentais de cada etapa, como principais dificuldades encontradas

por ambos, diferenças de tempo gasto no desenvolvimento, facilidade de adaptação, curva de aprendizado e facilidade de encontrar soluções através da *internet*, agrupando todos estes pontos dentro dos critérios citados acima.

5 RESULTADOS DA COMPARAÇÃO

Como se trata de um estudo de caso avaliando a produtividade com o uso dos *frameworks*, foram definidos critérios importantes para esta comparação e, toda a análise é baseada nas experiências, durante o desenvolvimento, dos autores em seus respectivos *frameworks*. Foram utilizados os registros das análises de desenvolvimento acerca de cada tarefa e também como base em algumas análises, o tempo gasto de desenvolvimento por etapa, que podem ser analisados no Gráfico 1.

Gráfico 1. Tempo de Desenvolvimento entre *Flutter* e *React Native*



Fonte: Elaborado pelos autores (2020).

Os critérios de avaliação foram utilizados durante as etapas de desenvolvimento levando em consideração pontos que são cruciais para a produtividade. Diante disso, cada autor escreveu uma análise para cada critério, baseado em suas experiências com o respectivo *framework.* Foram apontadas as dificuldades encontradas durante o processo, as facilidades que os recursos do *framework* podem oferecer e situações que influenciam diretamente na produtividade de desenvolvimento. Após isso, foi realizada uma revisão para tornar uma análise justa e imparcial. No final, aconteceu um debate entre ambos os autores e levantados os resultados das análises nos respectivos critérios, levando em consideração as vantagens e desvantagens de cada *framework*, juntamente com o tempo gasto de desenvolvimento em cada tarefa, sempre focando na produtividade como parâmetro de avaliação.

5.1 INSTALAÇÃO DO *FRAMEWORK*

5.1.1 React Native

O *React Native* apresenta duas formas de instalação diferentes, através do *Expo* ou utilizando emuladores *Android* e *iOS*. A instalação através do *Expo* é muito simples: com o *NodeJS* instalado, basta duas linhas de comando para conseguir instalar e criar o projeto na máquina. Já instalando através de um emulador convencional pode ser um processo mais demorado e requer atenção, principalmente a instalação do *Android Studio* que pode gerar erros dependendo de como foi configurado.

5.1.2 Flutter

Para a instalação do *Flutter*, é necessário fazer o *download* do SDK e adicionar ao *Path* do sistema. Feito isso, já é possível rodar comandos do *Flutter* no terminal, e a documentação do *framework* recomenda a execução de um comando específico para verificar o status da instalação do *Flutter* e se há dependências que precisam ser instaladas. Se falta a *SDK* do *Android,* por exemplo, o link para o download já é disponibilizado. A instalação do *SDK* já inclui a versão compatível da linguagem *Dart.*

5.1.3 Resultados

Ambos apresentam documentações muito bem detalhadas de como instalar o *framework* em diferentes ambientes sem muitas dificuldades. O *React Native* apresenta um adicional que é o *Expo* que não precisa ter instalado na máquina o emulador *Android/iOS*.

5.2 EMULANDO A APLICAÇÃO

5.2.1 React Native

O *React Native* oferece duas opções para a emulação: o *Expo* e o emulador *Android/iOS.* O *Expo* é relativamente simples de ser utilizado, através de um *QR Code* e o aplicativo instalado no celular é possível emular direto no dispositivo, porém o *Expo* pode apresentar suas limitações, o que foi um problema durante a fase de desenvolvimento.

Primeiramente, o *Expo* não carrega as fontes como é feito no emulador *Android*, onde é necessário colocar os *assets* na pasta *build* e referenciar no código. No *Expo* é necessário armazenar e carregar as fontes de forma assíncrona antes de compilar o projeto, já que ele não armazena de forma prática. No início é trabalhoso achar uma solução para a fonte nova funcionar no projeto, e conforme utiliza outros tipos de *assets* foi necessário realizar o mesmo carregamento por código. No projeto foram utilizados *webviews,* e durante essa etapa o conteúdo não carregava na tela, e após várias pesquisas não foi encontrada uma solução.

No final foi necessário migrar o projeto para emular no *Android*, com isso todos estes problemas foram solucionados de uma forma muita mais simples e organizada, a *WebView* funcionou corretamente e não foi necessário carregar os *assets* pelo código antes da inicialização. Apesar do *Expo* ser prático e simples, pode apresentar limitações no decorrer do

projeto que requerem soluções específicas.

5.2.2 Flutter

No *Flutter,* a emulação da aplicação é muito simples de ser feita. É possível rodar pela linha de comando ou, como explicado na documentação do *framework*, utilizando o próprio *Visual Studio Code* com a extensão do *Flutter* instalada. Nesse caso, basta utilizar a própria interface da *IDE* para iniciar o *debug* da aplicação, juntamente com um emulador, e nesse caso basta selecionar qual dispositivo virtual irá ser utilizado durante a execução da aplicação. Também é possível utilizar o próprio celular conectado por *USB* para isso.

5.2.3 Resultados

Com exceção do emulador virtual de *Android* e *iOS* na qual ambos funcionam de forma convencional ou uso do celular conectado por *USB* para emular a aplicação, o *React Native* apresenta a opção adicional do *Expo* devido a sua praticidade de instalação e de utilização sem complicações, além da possibilidade de uso de recursos nativos do celular, como câmera, microfone e *GPS*. O Flutter, por sua vez, embora não ofereça uma ferramenta como o *Expo*, apresentou grande produtividade por possuir um *build* rápido e *debug* fácil de ser feito.

5.3 CONSUMINDO *APIs JSON*

5.3.1 React Native

O processo de consumir *APIs* no formato *JSON,* no *React Native* é relativamente simples, para esta tarefa foi utilizada a biblioteca *Axios* para consumir e exibir na tela os dados retornados. Neste quesito a linguagem *JavaScript* acaba sendo uma grande vantagem neste tipo de recurso, conhecendo bem o processo de requisição no protocolo *HTTP* e de como manipular uma lista de dados, é possível apresentar os dados na tela sem muitas dificuldades e de forma prática.

5.3.2 Flutter

Para consumir dados externos por *JSON* na linguagem *Dart* em si, não é um processo tão fácil quando comparado ao *JavaScript*. O desenvolvedor precisa mapear os *DTOs* (Objeto de Transferência de Dados) com todos os atributos, e também criar um método específico para converter os dados vindos do *JSON* para o *DTO.* Dessa forma, o consumo de *APIs* exige alguns passos a mais devido às particularidades da linguagem, diferente de como é feito na linguagem *JavaScript.*

Os métodos para fazer requisição *GET* no protocolo *HTTP* também exigem que faça um *decode* para então ser convertido para o *DTO* mapeado, o que não é necessário no *JavaScript* - principalmente utilizando bibliotecas como o *Axios* - onde as coisas são mais simples de serem feitas nesse sentido.

5.3.3 Resultados

O fato de o *React Native* utilizar *JavaScript* se tornou uma grande vantagem ao *framework,* pois

tal linguagem apresenta uma praticidade muito grande quando se trata de manipulação de dados. Com auxílio da biblioteca *Axios,* é possível requisitar dados dentro de uma variável qualquer e tratar ela da forma que achar necessária sem dificuldades.

Conforme pode ser visto no Gráfico 1, nesta etapa o *React Native* na questão de produtividade em desenvolvimento se destacou com uma diferença de 1 hora e 30 minutos a menos do que o *Flutter.*

5.4 ESTILIZAÇÃO DE LAYOUT

5.4.1 React Native

A estilização no *React Native* é muito simples e intuitiva, com o estudo realizado antes de inicializar o projeto, foi possível moldar as interfaces conforme o protótipo sem dificuldade. A sintaxe *JSX* permite estilizar o projeto de diferentes formas, por exemplo, utilizando um arquivo global, é possível chamá-lo e definir por padrão o estilo em todas as telas. Também é possível criar um trecho de código em cada componente para estilizar cada tela e *tag* da forma desejada e individual, tornando a manutenção mais fácil. Após algum tempo de prática e entendendo principalmente o alinhamento de itens, o desenvolvedor se acostuma com a sintaxe e se torna algo relativamente simples.

5.4.2 Flutter

A estilização no *Flutter* é baseada em *widgets.* Tudo o que envolve a estilização do componente é feito dessa forma. No começo a experiência é um pouco desafiadora, pois tudo é feito com os *widgets* do próprio *framework e* o desenvolvedor precisa saber quais *widgets* servem para determinado propósito no *layout*, e isso tem uma certa curva de aprendizado. Porém, com o tempo e prática, as coisas ficam relativamente mais fáceis, pois o desenvolvedor vai se acostumando com os recursos do *framework*, e apesar de tudo, a documentação é completa e oferece detalhes sobre cada *widget* disponibilizado.

Um ponto a ser analisado também é a complexidade que essa forma de estilização pode acabar oferecendo: com o tempo, se não houver cuidado com boas práticas, o código pode acabar ficando complexo e difícil de entender. Também é recomendado utilizar componentes separados quando o código fica muito complexo.

5.4.3 Resultados

O *React Native* utiliza uma estilização similar a como é feito na *web,* com *HTML* e *CSS,* já o *Flutter* utiliza a linguagem *Dart* e apresenta um processo de estilização único, porém requer ao desenvolvedor um estudo a respeito da customização de *layout*, aumentando um pouco a curva de aprendizado. Conhecendo as propriedades de *layout,* e tendo conhecimento básico de *CSS* e *HTML,* a criação de *UI* no *React Native* torna-se relativamente fácil, não exigindo muito do conhecimento do desenvolvedor.

É possível notar também através do Gráfico 1, que as etapas de estilização de interface o *React Native* se mostrou muito mais produtivo, como nas etapas Criar página de detalhamento

de games, Criar componente de *carousel* e Criar página *Explore* onde teve tempos muito menores de desenvolvimento comparado ao *Flutter*, com exceção da etapa de criar página de listagem de notícia, onde foi encontrado dificuldades para implementar a *WebView*. A princípio a estilização é muito mais rápida e prática no *React Native*, encurtando o tempo gasto no desenvolvimento.

5.5 DOCUMENTAÇÃO

5.5.1 React Native

A documentação do *React Native* peca na organização e muitas vezes acaba não sendo de grande ajuda, pois muitas vezes não é possível encontrar todas as respostas ou o desenvolvedor acaba se deparando com exemplos de versões diferentes que podem gerar erros em versões mais recentes. É comum se deparar com erros apresentados no *GitHub* por usuários sem nenhuma solução, pois ainda não foi corrigido. A documentação pode ser um problema para desenvolvedores novos, muitas vezes não é possível solucionar um problema direto pela documentação precisando recorrer por outros meios.

Um caso encontrado foi durante a etapa de desenvolvimento da navegação de janelas: seguindo a documentação, não foi possível solucionar o problema pois o exemplo estava em uma versão mais antiga, gerando assim erros de importação no projeto. Ou o caso do build do aplicativo onde ele ensina como gerar o arquivo de publicação *AAB* mas não explica como gerar a aplicação através de um arquivo *APK*.

5.5.2 Flutter

A documentação do *Flutter* é algo positivo do *framework*. Ao entrar no site pela primeira vez, os passos são claros e intuitivos: a documentação explica desde como fazer o *setup* do *framework* em todas as plataformas e após a instalação, há um passo-a-passo para criar o primeiro aplicativo utilizando a ferramenta.

Há também toda a base que o desenvolvedor precisa ter para poder utilizar os *widgets*, plugins, testes e até *deploy,* com detalhes e inclusive vídeos explicativos com animações. Durante a etapa de definição do tema, a documentação não foi o suficiente para suprir todas as necessidades do autor. Entretanto, fora isso, a documentação do *framework* em questão é algo muito positivo.

5.5.3 Resultados

O *React Native* não possui uma documentação muito organizada e, em alguns casos, desatualizada a respeito de alguns componentes. O *Flutter* apresenta uma documentação organizada e detalhada de cada *widget*, podendo ser encontrado sem dificuldade.

5.6 CONTEÚDOS *ONLINE*

5.6.1 React Native

A comunidade do *React Native* é inegavelmente muito grande e existe um número riquíssimo de cursos, tutoriais, artigos, projetos e soluções de dúvidas *online* criadas pela própria comunidade. Porém, o *React Native* apresentou certa dificuldade de se trabalhar no quesito de versões, pois constantemente ele recebe novas atualizações, onde soluções e tutoriais que eram apresentados a um ou dois anos atrás podem não rodar mais em sua última versão, se tornando obsoleto. Para isso, foi necessário na hora de buscar, filtrar por resultados mais recentes para evitar possíveis problemas de versões de soluções mais antigas. Mas no final, há uma comunidade unida sempre trabalhando e acolhendo novos desenvolvedores, sempre disposta a ajudar e encontrando novas soluções para problemas recentes.

5.6.2 Flutter

A comunidade do *Flutter* é vasta, há muito conteúdo online, vídeos, tutoriais, e diversos clones de interfaces de aplicativos populares no *YouTube.* Entretanto, no *React Native* é evidente que a comunidade é mais madura e existe mais conteúdo, sem falar em todo o material disponível para *ReactJS,* que grande parte do conhecimento é transferível para o *React Native.*

5.6.3 Resultados

Ambos *frameworks* possuem uma comunidade ativa, sendo que o *React Native* possui mais conteúdo disponível e dificilmente o desenvolvedor terá dificuldade de encontrar soluções na *internet* para algum tipo de funcionalidade ou componentes. A falta de conteúdo não chega a ser um problema no *Flutter,* considerando-se que é um *framework* mais recente e que seu número de conteúdo *online* vem crescendo a cada dia.

5.7 DEPENDÊNCIAS EXTERNAS

5.7.1 React Native

A instalação de dependências no *React Native,* assim como vários *frameworks front-end*, são feitas utilizando o *runtime NodeJS*. Para instalar uma dependência nova é relativamente simples, com uma única linha de comando referenciando a biblioteca desejada, é possível instalar a dependência direto no projeto, através do arquivo *package.json* é possível visualizar todas as dependências e suas respectivas versões instaladas.

O *React Nativ*e apresenta bibliotecas externas vastas, muitas vezes criadas pela própria comunidade. Um caso comum é conflito de versões entre dependências, normalmente é necessário reduzir a versão da dependência para uma versão compatível e instalar novamente. Em alguns casos quando ocorrem erros e avisos durante o *debug* é necessário deletar todas as bibliotecas instaladas e com único comando, reinstalar todas dependências, automaticamente irá atualizar para as versões mais recentes, o que custa um pouco de tempo durante o desenvolvimento, dependendo do número de dependências, a instalação neste processo pode demorar, sendo um caso que ocorre com frequência.

5.7.2 Flutter

O *Flutter* também lida bem com dependências externas, elas são gerenciadas em um arquivo no formato *yaml* e basta especificar as dependências nesse arquivo e salvar o arquivo, e elas serão instaladas automaticamente sem a necessidade de rodar um comando adicional. As dependências mais comuns são encontradas no site *pub.dev,* e após escolher o pacote basta copiar para o arquivo com a versão desejada, salvar e o projeto já estará pronto para utilizá-la. Porém, é notável a diferença na quantidade de bibliotecas externas quando comparado ao *React Native:* a quantidade de recursos externos que seu concorrente possui é maior.

5.7.3 Resultados

Ambos os frameworks se comportam bem quando o assunto é manipulação de dependências externas e funcionam de forma parecida. O *React Native* pode oferecer certa vantagem de acordo com o contexto, pois a quantidade de pacotes e recursos externos é maior se comparado ao *Flutter*, porém para alguns pode ser um problema diante de tanta variedade, dificultando ainda mais a localização da melhor solução para um problema específico.

5.8 ESTABILIDADE DE VERSÕES

5.8.1 React Native

As mudanças frequentes que o *React Native* recebe, mostraram-se um dos seus pontos negativos quando se trata de produtividade. Ele recebe constantes atualizações tornando o processo de encontrar soluções um problema, a documentação sofre um pouco com isso, pois precisa constantemente ser atualizada sempre que lança uma versão nova, e muitas vezes não se encontra a solução para sua versão atual. O mesmo se dá para o conteúdo *online,* conforme comentado anteriormente, soluções passadas podem gerar problemas ou serem obsoletas em versões mais novas.

Durante o desenvolvimento houve dores de cabeça em relação a importação de biblioteca onde gerava conflitos com outras dependências devido a versão, sendo necessário muitas vezes fazer um *downgrade*, reduzindo a versão da biblioteca.

5.8.2 Flutter

Durante o desenvolvimento do projeto, nesse ponto o *Flutter* não oferece problemas, todo o material obtido online foi aplicável no projeto, no máximo foram necessários alguns ajustes. Não foi notado nenhuma incompatibilidade de versão que atrapalhasse o desenvolvimento ou problemas com dependências.

5.8.3 Resultados

Neste critério, o *Flutter* não apresentou conflitos de versões durante o processo de desenvolvimento. Pode-se dizer que, no ecossistema do *JavaScript*, é muito mais fácil encontrar bibliotecas/pacotes e soluções já prontas para diversos problemas, portanto no *React Native* acaba sendo mais comum encontrar soluções obsoletas ou desatualizadas.

5.9 BUILD DA APLICAÇÃO

5.9.1 React Native

A documentação não ensina como gerar o arquivo *APK* e apenas mostra como gerar o arquivo de *AAB* para publicação na *PlayStore*, mostrando mais uma vez sua carência de informação. Foi preciso percorrer outros conteúdos como artigos e *Stack Overflow* para encontrar uma solução, e seguindo corretamente os passos é possível gerar o *APK* sem dificuldade. É preciso primeiramente gerar a chave, mover o arquivo da chave para o diretório *android/app*, editar um arquivo para adicionar uma configuração de assinatura e com mais três comandos é possível gerar a aplicação pronta para ser utilizada no dispositivo *Android*.

5.9.2 Flutter

Para gerar a *APK* do aplicativo utilizando o *Flutter* foi uma tarefa muito simples de ser feita. Foi necessário apenas algumas consultas na documentação, conectar o celular ao computador via *USB* e rodar alguns comandos no terminal, e o aplicativo já estava disponível para rodar no *smartphone* em seu build pronto para produção. Há também o processo de publicação nas lojas de aplicativos, e nesse caso são necessários alguns passos a mais, porém no experimento prático feito neste trabalho foi feito até a geração da *APK* da versão de produção do *app.*

5.9.3 Resultados

Pela falta de informação que a documentação do *React Native* apresentou, foi necessário percorrer outros artigos na qual explicava como criar o arquivo *.apk* custando tempo, contudo a *build* do *React Native* exige etapas a mais como gerar uma chave de assinatura, configurar um arquivo e executar três comandos nos terminais, a *build* demora cerca de 5 minutos dependendo, no final é necessário passar o arquivo gerado no computador para o celular.

Já o *Flutter* é muito mais prático e rápido, com poucas etapas e com o celular conectado ao computador via *USB* a aplicação já está disponível para instalação. A configuração do *React Native* não é difícil, porém se mostrou mais trabalhosa e demorada que o *Flutter.* Conforme pode ser analisado no Gráfico 1 o *Flutter* foi mais produtivo com uma diferença de 20 minutos em relação ao *React Native*.

5.10 PONTOS POSITIVOS

5.10.1 React Native

Baseado na experiência de desenvolvimento em *React Native,* a utilização do JSX para criação de *UI* é muito agradável e intuitiva, e entender a sintaxe se torna um processo simples e rápido para desenvolver telas para sua aplicação. A possibilidade de criar componentes separados e chamá-los em forma de *tag* facilita muito a leitura e a manutenção do código. Juntando a dinamicidade que o *JavaScript* entrega, o *React Native* pode ser uma ótima escolha para

desenvolvedores com conhecimento mais avançado em programação *WEB* e que já conhecem a linguagem *JavaScript,* sendo um ponto positivo para a produtividade.

Outro ponto para ser elogiado é a crescente comunidade de desenvolvedores, com inúmeros conteúdos disponíveis gratuitamente na internet para dar suporte a novos desenvolvedores.

Por último, emular o projeto utilizando *Expo* tem suas vantagens, em poucos minutos você consegue criar seu projeto e compilar sem dificuldade, incluindo *Hot Reload* (Atualização em tempo real), modo *Debug*, além de ter acesso a recursos nativos de celular, como câmera, microfone, GPS de forma simples.

5.10.2 Flutter

Um ponto positivo ao desenvolver utilizando o *framework Flutter*, é a rapidez e facilidade de fazer as coisas. Basta alguns passos de instalação e o desenvolvedor já consegue rodar a aplicação no celular ou no emulador sem dificuldade alguma, e sem a incidência de erros. Basta alguns comandos para gerar a *APK* do aplicativo e rodar o aplicativo em sua versão de *build* no *smartphone.*

Outro ponto a ser considerado é a sintaxe da linguagem *Dart*, que é uma linguagem moderna e lembra muito o *JavaScript* e o *Java,* e é adequada para ser utilizada em um *framework* como o *Flutter*.

Podemos falar também do conteúdo da documentação, que explica tudo de forma clara e detalhada, inclusive apresenta vídeos para apresentar o funcionamento dos *widgets.* Contém instruções compreensíveis de praticamente tudo o que o desenvolvedor precisa saber para o dia-a-dia.

5.11 PONTOS NEGATIVOS

5.11.1 React Native

O primeiro ponto é em relação à documentação que deixa a desejar, pois não é muito organizada e nem sempre é possível encontrar soluções, sendo necessário percorrer por outros meios, como se algumas informações em relação a componentes fossem omitidas na documentação.

Outro problema recorrente é em relação a instabilidade de versão: o *React Native* está em constante atualização, sendo assim muitas soluções passadas acabam se tornando obsoletas em versões mais atuais. Além dos *bugs* corrigidos durante a atualização, muitas vezes é alterada a sintaxe de diversas dependências, tornando-se uma dor de cabeça encontrar soluções de problemas. Muitas vezes se depara com erros e avisos durante o processo de compilação envolvendo bibliotecas, sendo necessário às vezes reduzir para uma versão compatível, ou deletar a pasta com todas bibliotecas e instalá-la novamente, ou tentar resolver o problema direto dos arquivos da biblioteca em questão.

Por último, baseado na experiência de desenvolvimento, é a falta de um componente de navegação simples de entender e fácil de implementar. Durante o processo foi necessário estudar a fundo diversos exemplos, sendo que muitos exemplos não são a maneira mais

eficiente, no final foi preciso refazer toda a estrutura do projeto para se adequar a biblioteca de navegação.

5.11.2 Flutter

Podemos considerar a linguagem *Dart* como um dos pontos negativos do *Flutter*: apesar de ser uma boa linguagem e ter uma boa sintaxe, atualmente não é muito comum encontrar desenvolvedores especializados nessa linguagem, o que faz com que a contratação de novas pessoas fique mais difícil se comparado com o *frameworks Javascript.*

Outro ponto a ser considerado como um aspecto negativo do *framework,* é sua imaturidade. *Flutter* é um *framework* que não está no mercado há muito tempo, e por isso não é inteiramente estável. Assim como o próprio *framework,* sua linguagem também não é a mais madura do mercado, principalmente quando comparada às linguagens utilizadas no desenvolvimento nativo, como *Swift* e *Kotlin.*

É importante destacar também a limitação de suas bibliotecas. Como uma arquitetura relativamente nova de desenvolvimento, pode ser que o *Flutter* não ofereça todas as funcionalidades necessárias. Para isso, as bibliotecas externas vêm para resolver esses problemas específicos e, embora muitas vezes exista um pacote externo para resolver tais problemas, a quantidade encontrada não é tão grande quando comparada a de seu concorrente.

6 CONCLUSÃO

A escolha de um *framework* para determinado projeto, na maioria dos casos, irá depender de diversos fatores, entre eles a experiência dos desenvolvedores com a linguagem e/ou com o *framework* a ser escolhido, e não somente questões técnicas. Entretanto, o objetivo deste artigo foi realizar uma abordagem prática focando principalmente no processo de desenvolvimento e levantando análises de comparações relacionada a produtividade, para que, ao escolher o *framework,* se possa saber quais facilidades e dificuldades o(a) desenvolvedor(a) eventualmente irá encontrar ao utilizar a ferramenta escolhida em seu projeto.

Ambos os *frameworks* estudados neste trabalho possuem seus pontos positivos e negativos. Porém, ambos, de forma geral, se saíram muito bem nos testes, afinal são grandes ferramentas utilizadas amplamente no mercado. Os *frameworks* apresentaram resultados satisfatórios conforme a experiência de desenvolvimento de cada autor, o *React Native* com sua dinamicidade utilizando *JSX* que lembra muito o desenvolvimento *Web,* e o poder que o *JavaScript* oferece durante o desenvolvimento, irá entregar muita flexibilidade e produtividade ao projeto. Já o *Flutter,* por sua vez, que apesar de utilizar *Dart*, uma linguagem relativamente nova*,* a curva de aprendizado não se acentuou tanto, e para um desenvolvedor familiarizado com programação orientada a objetos pode ser uma boa escolha. Se mostrou um *framework* de experiência agradável de aprendizado, com uma linguagem que apresenta boa sintaxe e uma grande variedade de *widgets* prontos para serem utilizados.

De acordo com as análises feitas neste trabalho, pode se dizer que, para desenvolvedores que já conhecem a linguagem *JavaScript* e desenvolvimento *Web* em geral, é possível que a

produtividade no *React Native* seja maior comparado com *Flutter*, pois aliando o poder da linguagem e os recursos que o *framework* oferece, o desenvolvedor poderá tirar o máximo de aproveitamento. Por outro lado, o *Flutter* se mostrou um *framework* de fácil adaptação e não deixou nada a desejar: acompanhado de uma ótima documentação, uma vasta quantidade de *widgets* e uma comunidade que cresce cada vez mais, o desenvolvedor irá sem dúvidas ter uma boa experiência de desenvolvimento.

Para projetos futuros, pretendemos realizar comparações mais técnicas entre ambos os *frameworks,* adicionando novos critérios de avaliação, além de possivelmente incluir outras tecnologias populares no mercado. Comparações como, consumo de memória, uso de processador, quadros por segundos, consumo de bateria entre dispositivos *Android* e *iOS,* sintaxe de código, além de testar também funcionalidades nativas do dispositivo, como uso de Câmera, GPS, microfone, notificação, acesso ao diretório do celular, entre outros. Visando sempre analisar as principais dificuldades e facilidade destas implementações nos *frameworks*.

7 REFERÊNCIAS

EISENMAN, Bonnie. **Learning React Native. Building Native Mobile Apps with JavaScript.** Printed in the United States of America. Published by O'Reilly Media, Inc, Outubro de 2017. Acesso em: 15 de abril de 2020.

IGDB. **API IGDB Documentation**. Disponível em: <https://api-docs.igdb.com/#about> Acesso em: 14 de abril de 2020.

FORTUNATO DE LIMA, Fernando. **Avaliação de Frameworks para o desenvolvimento de aplicações híbridas**. 2019. Disponível em: <http://dspace.unipampa.edu.br:8080/jspui/bitstream/riu/4224/1/Fernando%20Fortunato%20de%20Lima%20-%202019.pdf>. Acesso em: 07 de setembro de 2020.

GITHUB. FACEBOOK: **React Native**. Disponível em: <https://github.com/facebook/react-native> Acesso em: 22 de março de 2020.

GITHUB. FLUTTER: **Flutter.** Disponível em: <https://github.com/flutter/flutter> Acesso em: 22 de março de 2020.

LIU, Shanhong. STATISTA. **Most wanted library, framework, or tool among developers worldwide, as of early 2019**. Disponível em: <https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/>. Acesso em: 20 de março de 2020.

TAVARES, Henrique Leal. **Introdução a Desenvolvimento de Aplicações Híbridas.** (Mar. 2018) Disponível em: <http://revista.fatecgarca.edu.br/index.php/efatec/article/view/113/109> Acesso em: 26 de maio de 2020.

TECHOPEDIA: Educating IT Professionals To Make Smarter Decisions. **CROSS-PLATFORM DEVELOPMENT**. Disponível em: <https://www.techopedia.com/definition/30026/cross-platform-development> Acesso em: 05 de setembro de 2020.

WINDMILL, Eric. **Flutter in Action.** Shelter Island, New York. Manning Publications, Dezembro de 2019**.** Disponível em: <https://livebook.manning.com/book/flutter-in-action> Acesso em: 15 de abril de 2020.

WU, Wenhao. **React Native vs Flutter, cross-platform mobile application frameworks.** (Mar. 2018). Disponível em <https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1> Acesso em: 07 de setembro de 2020.

XERPA. **Tudo sobre produtividade: conceitos, dicas e vantagens.** Disponível em: <https://www.xerpa.com.br/blog/tudo-sobre-produtividade/#:~:text=Produtividade%20%C3%A9%20a%20rela%C3%A7%C3%A3o%20entre,de%20tempo%20investido%20no%20processo.> Acesso em: 06 de setembro de 2020.