Desenvolvimento Nativo vs Ionic vs React Native: uma análise comparativa do suporte à acessibilidade em Android

Franklyn S. R. Bezerra[1], **Windson Viana**[2]

[1]Departamento de Computação - (UFC)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

[2]Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Mestrado e Doutorado em Ciências da Computação (MDCC)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

franklynseabra12@hotmail.com, windson@virtual.ufc.br

Abstract. *It is well known that smartphone apps play an essential role in our lives, whether for work, play, communication, among other tasks. However, these applications may not always be properly accessed by people who have some disability, such as visual impairment. In this context, this research is a continuation of the analysis of support for accessibility in mobile applications made by Ribeiro, Façanha e Viana (2020). The goal is to compare the application development on cross-platform tools, like React Native and Ionic, against the native development process in Android. We compared how the applications behave before and after the implementation of the accessibility requirements, and it was found, as a result, that the application in React Native got more errors than the application in native Android development, but fewer errors than the development in Ionic.*

Keywords: Accessibility, Mobile Development, React Native, Android, Ionic

Resumo. *É notório que os aplicativos para smartphones possuem um papel essencial em nossas vidas, seja para o trabalho, a diversão, a comunicação ou para a aprendizagem. Entretanto, nem sempre esses aplicativos podem ser adequadamente acessados por pessoas que têm algum tipo de deficiência, como a deficiência visual. Nesse contexto, esta pesquisa é uma continuação da análise do suporte à acessibilidade em ferramentas Cross-Platform feita por Ribeiro, Façanha e Viana (2020). O objetivo é comparar o desenvolvimento de aplicações multiplataforma, como React Native e Ionic, com o desenvolvimento de uma aplicação nativa em Android. Foram comparados como os aplicativos se comportam antes e após implementar os requisitos de acessibilidade e foi obtido como resultado que, em relação à acessibilidade, a aplicação em React Native obteve mais erros do que a aplicação em Android Nativo, porém menos erros do que o desenvolvimento em Ionic.*

Palavras-chave: Acessibilidade, Desenvolvimento móvel, React Native, Android, Ionic

1. Introdução

É enorme a variedade de operações que podemos fazer com nossos *smartphones* hoje em dia. Desde pagar contas, realizar tarefas do trabalho, se comunicar de diversas formas possíveis (e.g., por meio das redes sociais, de maneiras mais convencionais como ligar, SMS) e até mesmo usar o *smartphones* para diversão, como jogar e assistir filmes, séries e vídeos. Segundo um levantamento feito pelo Statista (2021), haviam cerca de 2,95 milhões de aplicativos publicados na *Google Playstore*. Já na *App store*, em julho de 2020, haviam sido registrados quase 4,37 milhões (STATISTA, 2020), sendo 957.390 jogos. Além disso, segundo uma pesquisa da Fundação Getúlio Vargas de São Paulo, na metade de 2020, o Brasil atingiu a marca de 234 milhões de aparelhos ativos (FGVCIA, 2020), um número maior do que a população Brasileira, de 211,8 milhões de habitantes no mesmo período (IBGE, 2020). Esses números mostram o crescimento do mercado de aplicativos móveis e a importância que eles têm na vida do século XXI.

Embora com todo o desenvolvimento das tecnologias de interação dos *smartphones*, muitos usuários ainda encontram dificuldades em usar tais dispositivos, especialmente, as pessoas com deficiência visual que necessitam de recursos e modais adicionais para perceber e interagir com as aplicações. Segundo dados do último senso do Instituto Brasileiro de Geografia e Estatística (IBGE), existem, no Brasil, 6,5 milhões de pessoas com alguma deficiência visual (OLIVEIRA et al., 2012).

Encontram-se, no entanto, softwares de acessibilidade como o *Talk Back* no *Android* e o *VoiceOver* no sistemas operacional da *Apple* que são capazes de ler textos, informações de botões e melhor explicar as ações realizadas pelo usuário ao interagir com os dispositivos. Esses serviços e softwares de acessibilidade proporcionam o acesso aos aplicativos e à Web para esse público. Contudo, pessoas com deficiência visual enfrentam desafios significativos ao usar esses dispositivos, e só o uso dos assistentes não é suficiente, é preciso que os apps tenham informações descritas de forma satisfatória, para permitir uma boa navegação acessível. E para isso, o suporte do ambiente de programação é fundamental e auxilia esse processo.

Sobre os Sistemas Operacionais (S.O.) que executam nos *smartphones*, há dois que dominam aproximadamente 99% desses dispositivos (STATCOUNTER, 2021): o *Android* e o *iOS*. Ambos possuem ambientes de desenvolvimento diferentes, fazendo com que seja demandado dois times de desenvolvimento ou desenvolvedores que dominem as duas plataformas quando se deseja atingir os dois S.O (Hui et al., 2013). Dessa forma há um desperdício de tempo e dinheiro para a construção de aplicações para ambos os sistemas operacionais (M. S. Ferreira et al., 2018). Nesse contexto, as ferramentas *cross-platform* surgem como uma alternativa, onde o desenvolvimento é feito em uma única linguagem e depois, o código é compilado ou gerado para sistemas operacionais distintos, gerando economia de tempo e dinheiro (EL-KASSAS et al., 2017). Entre as diversas abordagens *cross-platform* existentes, são abordadas nesse artigo o *Ionic* e o *React Native*.

Neste trabalho, tem-se como objetivo principal analisar o suporte do *React Native* no desenvolvimento de aplicações acessíveis à pessoas com deficiência visual e comparar com a análise do suporte do Ionic e do Android realizado por Ribeiro, Façanha e Viana (2020). Para alcançar tal meta e identificar os recursos a serem implementados, foram estabelecidos os seguintes objetivos específicos: (i) Realizar uma prova de conceito desenvolvendo uma aplicação acessível utilizando *React Native* para comparação com os aplicativos desenvolvidos em *Ionic* e *Android* Nativo feitos pelo artigo citado; (ii) Identificar os pontos positivos e negativos durante o processo de desenvolvimento, com foco na acessibilidade; (iii) Realizar testes utilizando uma ferramenta de verificação de acessibilidade para poder identificar possíveis falhas e trabalhar em

melhorias.

A organização do restante deste artigo ocorre da seguinte forma: As Seções 2 e 3 apresentam a fundamentação teórica para essa pesquisa. A secção 4 apresenta alguns trabalhos relacionados. A seção 5 apresenta a metodologia utilizada ao longo deste TCC. As Secções 6 explica a Prova de conceito e seu desenvolvimento. Na Seção 7, é feita a primeira avaliação, sem a implementação dos requisitos de acessibilidade. Na Seção 2, é explicado como a acessibilidade foi implementada no aplicativo desenvolvido. Nas Seções 9 e 10, foram feitas novas avaliações, dessa vez com os requisitos de acessibilidade implementados. As Seções 11 e 12 contam com a discussão dos resultados e a conclusão do trabalho.

2. Acessibilidade

A acessibilidade é o termo usado para indicar a possibilidade de que qualquer pessoa, independentemente de suas capacidades físico-motoras, possa usufruir de benefícios de uma vida em sociedade (NICHOLL; FILHO, 2001). A acessibilidade, no texto de software, pode ser definida pela ISO 9126 (ISO/IEC, 2001), onde ela é a prática inclusiva de fazer softwares que possam ser utilizados por todas as pessoas, independente de ter deficiência ou não. Desse modo, um dos desafios é entender e estruturar as estratégias necessárias para desenvolver e avaliar uma aplicação desenvolvida no cenário móvel (Guimarães; TAVARES, 2014).

Existem diversas pessoas que se beneficiam da acessibilidade, dentre elas, pessoas com pouca ou nenhuma visão, com deficiência visual, com dificuldades cognitivas e problemas de mobilidade. De acordo com o (CENSO, 2010), existem em torno de mais ou menos 6.5 milhões de pessoas com deficiência visual que possuem grandes dificuldades ou não conseguem enxergar de modo algum, quase 1.8 milhões com deficiência auditiva, 4.4 milhões com deficiência motora e 2.6 milhões com deficiência mental/intelectual. Por estar mais presente, a deficiência visual é a mais requisitada no quesito à produção de conteúdo e software acessível.

Através da Classificação Internacional da Funcionalidade, Incapacidade e Saúde (CIF), a OMS (Organização Mundial da saúde) detalha a deficiência visual em categorias que incluem desde a perda de visão leve até a ausência total de visão e baseia-se em uma perspectiva biopsicossocial para definir cegueira e baixa visão (WOHLIN, 2014). A cegueira ocorre quando o indivíduo apresenta acuidade visual menor que 0,1 (com a melhor correção no melhor olho) ou campo visual abaixo de 20 graus. É a completa falta de percepção visual de forma e luz (GARCÍA, 2017). Já a baixa visão (ambliopia, visão subnormal ou visão residual), é definida quando ocorre o comprometimento do funcionamento visual em amos os olhos mesmo após correção de erros de refração comuns com uso de óculos, lentes de contato ou cirurgias oftalmológicas (CAMPOS; SÁ; SILVA, 2007).

Em maio de 1999, foi criado pelo grupo de acessibilidade na web do W3C (*World Wide Web Consortium*) o WCAG (*Web Content Accessibility Guidelines*), um guia de acessibilidade de conteúdos na web. O WCAG está organizado em quatro blocos principais que dizem se o produto possui elementos perceptíveis, operáveis, compreensíveis e robustos. A acessibilidade móvel também está inclusa no guia da W3C, porém existe um outro guia especifico para aplicativos móveis, o MWBP (*Mobile Web Best Practices*) e os dois possuem o mesmo objetivo de melhorar a interação de usuários com deficiência.

O guia de desenvolvimento de aplicações móveis acessíveis elaborado por (SIDI, 2015) cita alguns princípios que os designers, desenvolvedores e testadores de aplicativos devem seguir para melhorar a acessibilidade. Alguns deles são:

- Design Minimalista: Privilegiar os componentes relevantes e com função comprovada, evitando o uso de informações desnecessárias ou irrelevantes;

- Fluxo natural: Ordenar as telas sequencialmente e os componentes em fluxo natural, na linguagem ocidental, da esquerda pra direita de cima pra baixo;
- Coerência externa (convenções): Privilegiar a localização dos componentes de acordo com convenções já conhecidas;
- Coerência interna: Garantir que o aplicativo tenha um padrão interno, cores, posicionamento e outros atributos, estabelecendo uma coerência em todas as telas da interface;
- Evitar muitas informações na tela vertical: Quanto mais informação na mesma tela, mais difícil e cansativa se torna a interação para as pessoas com deficiência visual, além de deixá-los perdidos;
- Contraste de cores na interface: Adotar cores que facilitem a identificação dos componentes e ajudem a perceber quando há alterações na tela.

Nos aplicativos móveis, a acessibilidade para pessoas com deficiência visual é feita por leitores de tela, que são ferramentas que oferecem *feedback* sonoro para cada operação realizada. No *Android*, tem-se o pacote de acessibilidade (GOOGLE, 2021) que inclui o *Talkback* e menus de acessibilidade. Já no kit de acessibilidade do (APPLE, 2021) *iOS* tem-se o *VoiceOver*, ambos desenvolvidos pelas próprias fabricantes dos sistemas operacionais. Mesmo sendo de providos por diferentes empresas e de sistemas diferentes, os dois possuem funcionamento semelhante. É usado o *feedback* falado descrevendo o que é selecionado na tela. Além disso, ambos possuem reconhecimentos de gestos com dois ou três dedos na tela para realizar funções, como dar uma rolagem do *scroll* na tela ou acessar demais opções do aplicativo.

Uma das aplicações mais comum para verificar a acessibilidade de um aplicativo é o *Scanner de acessibilidade*[1] do *Google*. Ele analisa a tela e identifica possíveis melhorias de acordo com as melhores práticas de acessibilidade, conforme mostra a Figura 1, na qual mostra erros de itens editáveis se sobrepondo e itens sem a descrição necessária. Porém existem diversas outras aplicações voltadas para fazer teste de acessibilidade, como o *WCAG Accessibility Checklist* para *iOS*[2]. No entanto como os testes realizados seriam no *Android*, optou-se por utilizar o *Scanner* feito pela Google.

3. Abordagens de desenvolvimento

Entre as abordagens de desenvolvimento tem-se a nativa e a usando *Cross-Platform*, que segundo a taxonomia de El-Kassas et al. (2015), pode ser classificada em: cross-compilada, de máquina virtual, baseada na web, em componentes, na nuvem e uma abordagem mista das anteriores. As aplicações nativas são desenvolvidas usando ferramentas e linguagens de programação particulares, usando SDK (*Software Development Kit*) e *frameworks* próprios. No *Android*, por exemplo, é usado *Java* ou *Kotlin*, sendo desenvolvido no *Android Studio*. No desenvolvimento nativo para iOS, usa-se as linguagens de programação *Objective-C* e *Swift* na IDE (*Integrated Development Environment*) *Xcode*. Porém, os apps ficam vinculados a cada ambiente, executando de forma exclusiva nos dispositivos da plataforma alvo. Para o desenvolvimento voltado para as duas plataformas, faz-se necessário criar dois aplicativos um para cada S.O. alvo. Isso pode ser feito por duas equipes diferentes, o que ocasiona em mais gastos, ou usando uma mesma equipe. No último caso, a equipe precisa ter conhecimento dos dois ambientes de desenvolvimento, o que pode também encarecer o processo por exigir profissionais mais bem qualificados. Além disso, demanda mais tempo de desenvolvimento já que haverá dois ciclos de implementação.

Soluções *Cross-Platform* possibilitam a implementação de uma aplicação que pode ser

[1]https://support.google.com/accessibility/android/answer/6376570

[2]https://apps.apple.com/us/app/wcag-accessibility-checklist/id1130086539

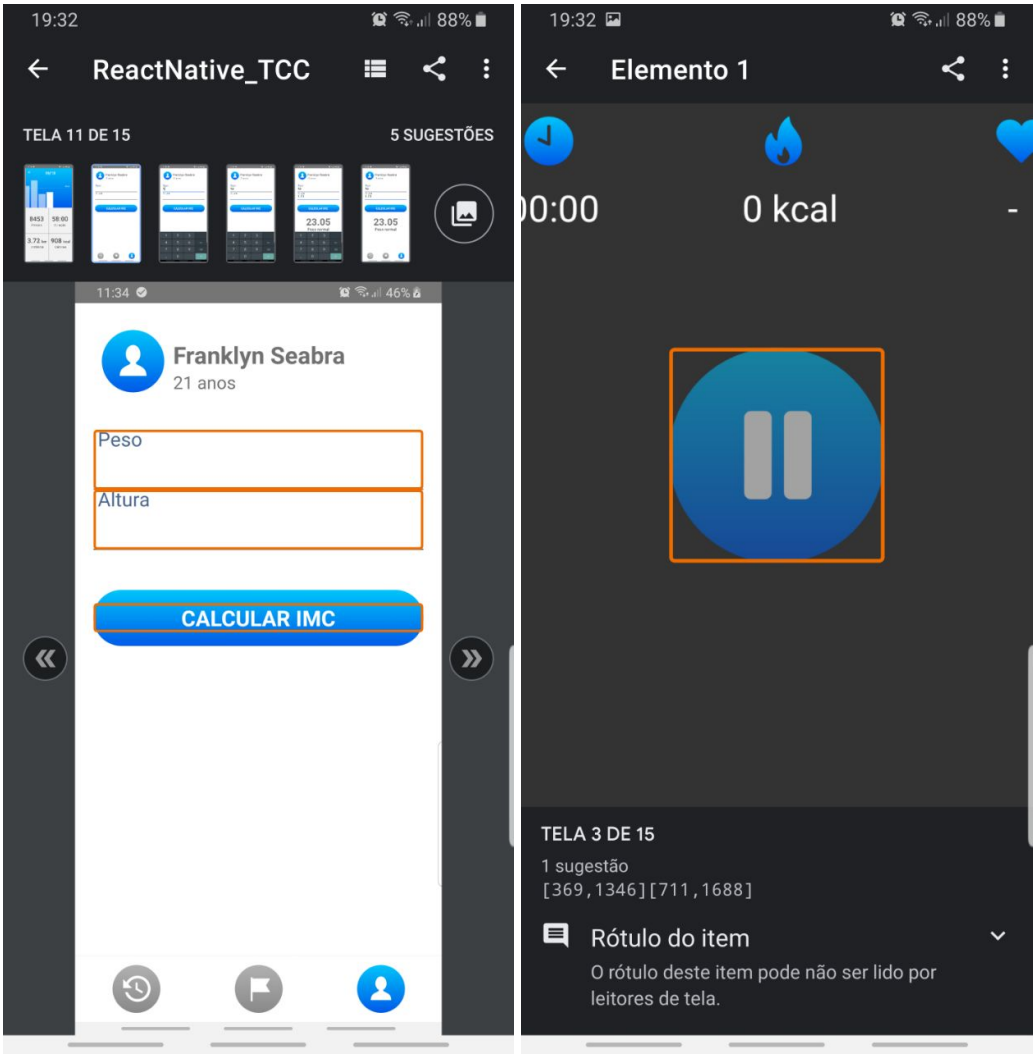


Figura 1. Scanner de acessibilidade

executada em diferentes plataformas (EL-KASSAS et al., 2017). Essa abordagem, embora ajude no tempo de desenvolvimento, ainda possui muitos desafios se comparado ao desenvolvimento nativo. Uma deles é a perda de desempenho, como abordado em (BIøRN-HANSEN et al., 2020). A implementação nessa abordagem também fica dependente de *plugins* para se conectar ao hardware do dispositivo e muitos desses são desenvolvidos por terceiros e podem possuir pouco ou nenhum suporte(BIøRN-HANSEN; GHINEA, 2018).

Segundo a taxonomia proposta em (BIØRN-HANSEN; GRØNLI; GHINEA, 2018), as abordagens de desenvolvimento *Cross-Platform* podem ser categorizadas em cinco grupos: híbridas, interpretadas, cross-compiladas, dirigidas por modelo (*Model-Driven*), e PWA (*Progressive Web Apps*). Todas trabalham com formas de desenvolvimento diferentes. A abordagem híbrida permite o uso de tecnologias web, como *HTML*, *CSS* e *Javascript* com a combinação de *WebView* nativo, que é simplesmente um *browser* embutido, permitindo assim a renderização das tecnologias web mencionadas. As ferramentas mais comuns para esse tipo de desenvolvimento são o *Cordova* (antigamente PhoneGap) (ADOBE, 2009–2021) e o *framework Ionic* (IONIC, 2013–2021). A abordagem interpretada é similar a abordagem híbrida, porém ela não depende de um *WebView* para renderizar seus componentes. Ao invés, apps interpretados podem renderizar componentes nativos devido a um interpretador *JavaScript* no dispositivo.

Dentre as ferramentas de abordagem interpretada destacam-se o *React Native* (FACEBOOK, 2015–2021) e *NativeScript* (PROGRESS, 2014–2021). No desenvolvimento *cross-compilado*, a linguagem de criação dos aplicativos é compilada para código executável nativo de cada plataforma desejada, ou seja, o aplicativo final se comporta como uma aplicação nativa, sem precisar de *WebView* ou interpretadores *JavaScript*. São exemplos de ferramentas *cross-compiladas*: o *Flutter* (GOOGLE, 2017–2021), *Xamarin* (MICROSOFT, 2011–2021) e o solar2D (antigamente Corona SDK) (LABS, 2008–2021). A abordagem de desenvolvimento dirigida por modelo vem do paradigma de programação de mesmo nome. Implementações com essa abordagem são raras entre os desenvolvedores e comunidades de aplicativos comercias. *Frameworks* dessa abordagem facilitam a geração de *interfaces* de usuário e lógica de negócio baseadas na construção de modelos e *templates* que são independentes de plataforma, e comumente é usada uma linguagem de domínio específico (e.g., *Mobl* (HEMEL; VISSER, 2011) e *Xmob* (GOAER; WALTHAM, 2013)).

Por último, tem-se a abordagem PWA (Progressive Web Applications). Um PWA é um *web app* com recursos aprimorados. Ele é hospedado em um servidor web para os usuários acessarem pela URL no navegador, um dos objetivos é dessa abordagem é permitir que os aplicativos web possam ter a aparência de um aplicativo nativo ou *cross-platform*. Os principais exemplos de ferramentas e *frameworks* que podem ser usadas para criar um PWA são: *Ionic*, *React.js*, *Vue.js* e *Angular*. No PWA, não é possível acessar todas as plataformas ou recursos do dispositivo, porém vem ganhando popularidade desde 2016 (BIØRN-HANSEN; GRØNLI; GHINEA, 2018).

Nos tópicos a seguir, estão contidas informações sobre o desenvolvimento nativo em *Android* e as ferramentas *cross-platform* escolhidas para um estudo mais aprofundado e que serviram como base de desenvolvimento do aplicativo mencionado nos objetivos deste trabalho.

3.1. Android

O Android é uma plataforma de software de código aberto para dispositivos móveis consistido de um sistema operacional, *middleware* e aplicações chave mantido pelo *Google*. Ele possui uma arquitetura baseada em *Linux* e assim consegue se aproveitar de vários recursos, como gerenciamento de memória, processos, automatização de rede e drivers (GANDHEWAR; SHEIKH, 2010). Como mostra a documentação [3], a arquitetura do *Android* pode ser subdividida em 6 camadas listadas a seguir.

- Aplicações: Diversos aplicativos que possibilitam a interação do usuário, como o navegador, a tela inicial, os contatos, etc;
- *Framework* de aplicações: Consiste de duas partes como bibliotecas escritas em C/C++. Eles serão chamadas através de *interfaces* Java pelos programadores, dentre elas tem-se o gerenciamento de *Activitys* e *views*;
- Runtime: É a camada responsável pela execução das aplicações;
- Bibliotecas Nativas: Nesta camadas encontramos as diversas bibliotecas nativas, como a biblioteca do C, *SQLite*(Banco de dados), *OpenGl*(Renderização 3D), etc;
- O *Kernel* e ferramentas de baixo nível: Permitem o acesso a recursos de hardware, como áudio, vídeo, recursos de rede, etc.

O principal ambiente de desenvolvimento para *Android* é o *Android Studio*[4], uma *IDE* que auxilia os desenvolvedores na elaboração e construções de apps. As linguagens usadas no

[3]https://developer.android.com/guide/platform

[4]https://developer.android.com/studio

desenvolvimento são Java e *Kotlin*[5] para a lógica da aplicação, e a linguagem de marcação *XML* para as demarcações e conteúdo de layout. A acessibilidade no *Android* nativo é tratada de uma maneira mais clara, existem guias e tutoriais que podem ser seguidos para tornar o aplicativo mais acessível[6]. Além disso, a própria IDE faz inspeção do código com verificações de *lint* que dão advertências (*warnings*) nos componentes que não atendem aos requisitos de acessibilidade. A Figura 2 apresenta a composição do sistema operacional *Android*.

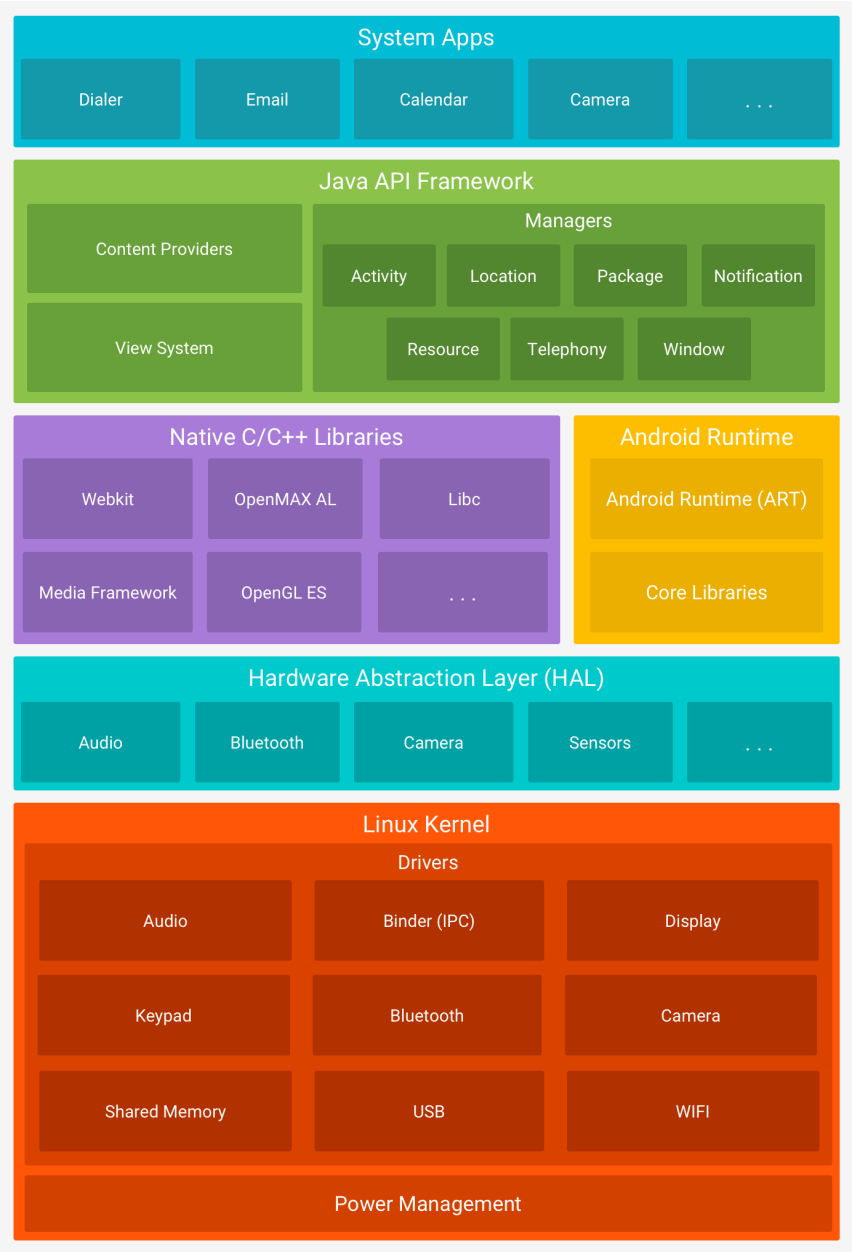


Figura 2. Pilha de softaware do Android

[5]https://kotlinlang.org/
[6]https://developer.android.com/guide/topics/ui/accessibility?hl=pt-br

3.2. Ionic

Ionic é um *framework* para a construções de aplicações móveis multiplataforma, ele possui código aberto e é construído nas tecnologias web (*HTML, CSS e Javascript*) e permite aos desenvolvedores reusar conhecimentos nessas tecnologias (SARMOTA; GHRERA, 2019). Ele utiliza *Angular*, que é uma *framework JavaScript* desenvolvido pela *Google*, porém possui suporte para React.js e Vue.js. Por usar tecnologias web, um aplicativo feito nessa tecnologia se comporta como um site executando no *WebView* acoplado ao código nativo. Esse *framework* fornece bibliotecas para acesso aos componentes nativos do dispositivo, como sensores e outros componentes de *hardware*. Ele também possui desenvolvimento reativo, ou seja, é possível acompanhar as mudanças em tempo real no aplicativo. No entanto, o aplicativo feito com esse *framework* apresenta tempo de resposta ao clique maior quando comparado com um aplicativo nativo em *Android* (BRITO et al., 2018). Isso pode inviabilizar a utilização do *Ionic* em alguns tipos de aplicações que exijam um maior desempenho.

3.3. React Native

React Native é um framework de código aberto para desenvolvimento de apps híbridos usando React.js, ele foi introduzido em 2015 pelo *Facebook*. Seu lema é "aprenda uma vez (React.js) e escreva em todo lugar (web, Android, iOS)". Assim como o Ionic, ele também permite aos desenvolvedores reutilizar conhecimentos das tecnologias web para o desenvolvimento de aplicativos móveis. Contudo, ele não se comporta como um site executando no *WebView*, ele interpreta os componentes primitivos escritos em *Javascript* e os renderiza como componentes nativos do sistema. O acesso à camada nativa é provido por bibliotecas do *React Native* ou de terceiros.

Esse *framework* fornece blocos de construção elementares, como Text, View, etc., que podem ser estilizados e usados para compor as aplicações. A grande vantagem é ter disponível a pilha de tecnologias da web e ainda assim construir aplicativos nativos com código em *Javascript*. Uma grande limitação é poder usar somente os recursos que são suportados pelo ambiente que o *React Native* ou que terceiros fornecem, além de mudanças constantes que podem tornar a manutenção mais custosa (SARMOTA; GHRERA, 2019).

4. Trabalhos Relacionados

Foram procurados, no *Google Acadêmico*, trabalhos relacionados com o tema de desenvolvimento acessível ou acessibilidade em *frameworks* multiplataforma. Durante a pesquisa foi notado que são poucos os trabalhos que levam em consideração a acessibilidade na comparação das formas de desenvolvimento, e os que foram achados são bem recentes, o que pode demonstrar uma conscientização ao desenvolver aplicativos mais acessíveis.

O trabalho de Rodrigues et al. (2020) compara como é o suporte à acessibilidade nos ambientes de programação em *Android* Nativo e em *Ionic*. Outra trabalho interessante é o do Braga (2019), onde ele compara o desenvolvimento nativo vs *React Native* através de um estudo de portabilidade do jogo *Híbrid Coup* acessível.

Na contribuição de Mascetti et al. (2020) é investigado como os *frameworks* de desenvolvimento *cross-platform* suportam a criação de aplicações móveis acessíveis para pessoas com deficiências visuais através de leitores de tela. Ele faz isso através de uma análise sistemática dos leitores de tela disponíveis para *iOS* e *Android*, além disso, eles analisaram e compararam dois populares *frameworks cross-platform*: *Xamarin* e o *React Native*.

5. Metodologia

Esse trabalho é fortemente influenciado pela pesquisa de Ribeiro, Façanha e Viana (2020). Em sua pesquisa, o suporte à acessibilidade do *Ionic* foi investigado por meio da criação de uma Prova de Conceito (PoC) de uma aplicação com temática *fitness*. Ao todo, quatro versões de um mesmo aplicativo foram criadas e comparadas. Duas versões (uma em *Android* Nativo e outra em *Ionic*) continham as interfaces das telas criadas sem se preocupar com requisitos de acessibilidade. O intuito era aferir o que, por padrão, as plataformas *Ionic* e o *Android* Nativo desempenham em termos de acessibilidade. Em seguida, essas duas versões foram refatoradas tentando se implementar um conjunto de requisitos de acessibilidade proposto pelo SIDI (2015). Ao final, essas versões foram submetidas a processos de checagem de acessibilidade, por meio de heurísticas, uso de checadores e avaliação com um usuário que possui baixa visão.

Nesta pesquisa, a metodologia segue processo semelhante, inclusive reusa a mesma PoC, com intuito de também comparar as versões produzidas em *React Native* com as versões anteriores feitas em (RODRIGUES et al., 2020).

A Figura 3 exibe as principais etapas desta pesquisa. Primeiramente, foi feita uma pesquisa exploratória na literatura acadêmica por meio da leitura de artigos científicos, que segundo (PRODANOV; FREITAS, 2013) tem como finalidade proporcionar mais informações sobre o assunto, possibilitando uma definição e delineamento do tema. Dessa maneira, foram pesquisados conceitos como acessibilidade móvel, desenvolvimento *cross-plataform*, bem como a ferramenta de desenvolvimento *React Native*.

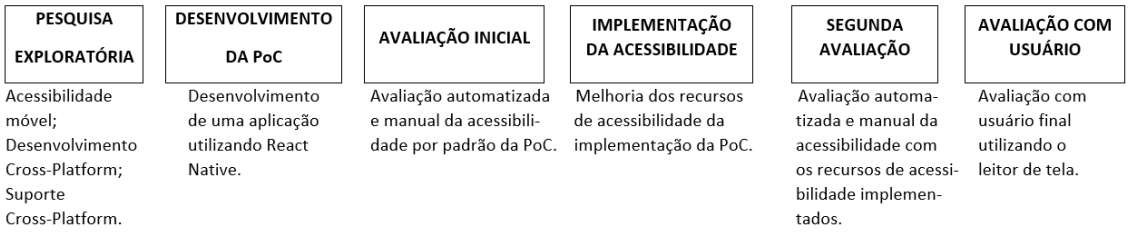


Figura 3. Metodologia de execução da pesquisa

Após esse estudo inicial, foi delineado um estudo baseado na Engenharia de Software Experimental semelhante ao realizado por Ribeiro et. al. (RIBEIRO; Façanha; VIANA, 2020). A Engenharia de Software Experimental é uma área de pesquisa cujo objetivo é melhorar a Engenharia de Software Tradicional, trazendo conceitos da abordagem científica (experimentação) na construção de métodos e técnicas para apoiar o desenvolvimento de software. Entre as três principais estratégias experimentais, como *Survey*, estudo de caso e experimento, optou-se por utilizar nessa pesquisa o estudo de caso. Este tipo de estudo visa observar um atributo específico e estabelecer um relacionamento, no caso fazer uma comparação, com atributos diferentes (TRAVASSOS; GUROV; AMARAL, 2002). Neste trabalho, os atributos foram o desenvolvimento *Cross-Platform* com React Native e as aplicações em Android e Ionic desenvolvidas em (RIBEIRO; Façanha; VIANA, 2020). Como mencionado, esse estudo de caso é uma continuação do trabalho de (RIBEIRO; Façanha; VIANA, 2020), no qual compara o desenvolvimento voltado para a acessibilidade em *Android* e *Ionic*. Nesse estudo iremos utilizar a mesma Prova de Conceito (PoC), porém introduziremos o *React Native* na comparação, ou seja a PoC foi também implementada nessa tecnologia.

A última etapa deste trabalho consistiu em avaliar a acessibilidade da nova aplicação desenvolvida e compara-lá com as demais já implementadas. A primeira parte foi uma avaliação

da acessibilidade antes de prover dos requisitos de acessibilidade, com o objetivo de observar como os aplicativos se comportam por padrão. Depois, foi feita uma nova avaliação, desta vez, com os requisitos acessíveis implementados, para então fazer uma nova análise dos novos recursos e do suporte aos mesmos em *React Native*. As ferramentas utilizadas foram o leitor de tela *TalkBack* e o aplicativo *Scanner* de acessibilidade. Por último, foi feita mais uma avaliação, dessa vez com um usuário final.

6. Prova de conceito

A prova de conceito trata-se de um aplicativo com temática *fitness* e que armazena informações das atividades físicas do usuário e gera relatórios de acompanhamento do progresso dos exercício feitos e também permite o monitoramento em tempo real do exercício.

Como mencionado anteriormente, as versões Nativa[7] e Ionic[8] já foram implementadas em (RIBEIRO; Façanha; VIANA, 2020) e foram usadas como referência para a implementação da versão em React Native.

6.1. Versão React Native

O framework *React Native*[9] é baseado no *React*, um *framework* de desenvolvimento web. O projeto foi criado via linha de comando que é provida pelo próprio *React Native*. Esse comando gera todos os arquivos iniciais que o aplicativo precisa para funcionar, assim como uma aplicação exemplo (*Hello World!*). O desenvolvimento é totalmente feito em *javascript* e se baseia fortemente no *framework* web já mencionado. A IDE utilizada foi o *VScode*, pois o mesmo oferece diversas extensões que facilitam o desenvolvimento e a organização e leitura do código.

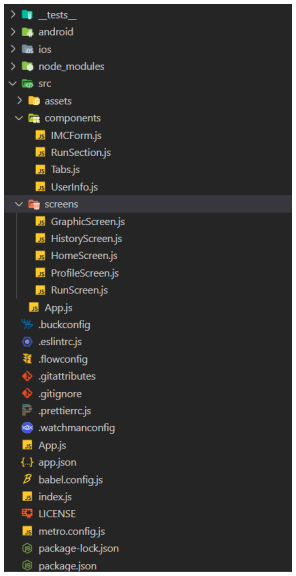


Figura 4. Arquivos do Aplicativo em React Native

O comando inicial *npx react-native init FitApp* gerou um código funcional *Hello World*, a criação e navegação de páginas é feito pela biblioteca *react-navigation*. A figura 4 mostra a estrutura do projeto e como ele está organizado. Assim como no *React*, no *React Native* é

[7]<https://github.com/lucasmonteiro58/Android_TCC>
[8]<https://github.com/lucasmonteiro58/Ionic_TCC>
[9]<https://github.com/Franklyn-S/ReactNative_TCC>

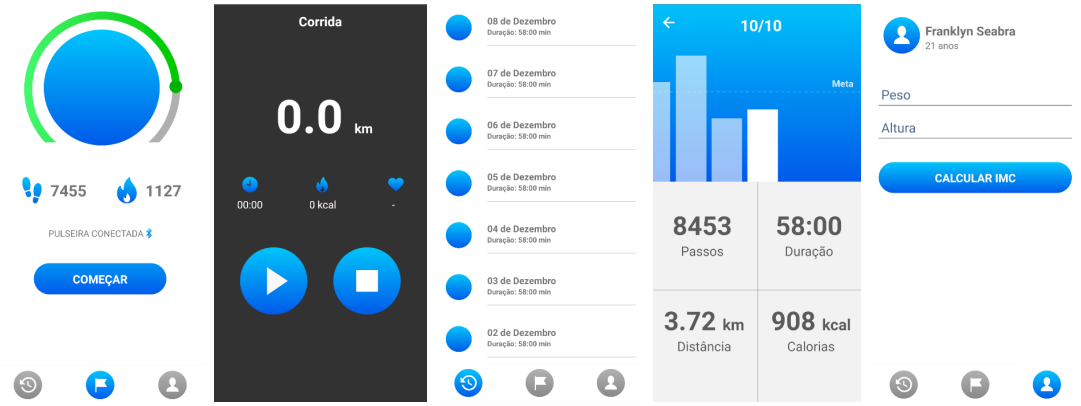


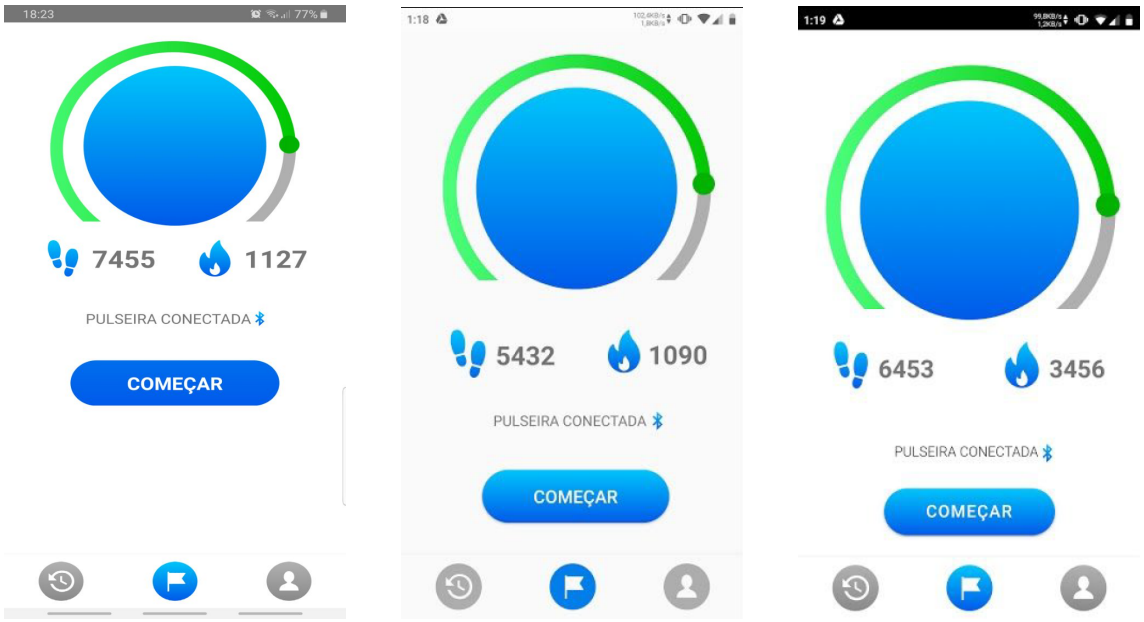
Figura 5. Interface do aplicativo *React Native*

possível criar componentes para facilitar o desenvolvimento e a futura manutenção do código, possibilitando o reuso dos componentes caso seja necessário. Na figura 5 é mostrado o visual da aplicação em *React Native* com todas as *features* implementadas.

O aplicativo pode ser testado em um celular físico ou em emulador através dos seguintes comandos na pasta do projeto: *react-native start* para iniciar o *bundle* e *react-native run-android*, no caso do *Android*, e *react-native run-ios*, no caso do *iOS*, para executar no emulador ou dispositivo físico conectado.

6.2. Comparação com as versões anteriores

Na Figura 6, é feita uma comparação da tela inicial feita nos 3 apps. Já a Tabela 1 mostra algumas características dos projetos que implementaram a PoC. O tamanho do aplicativo foi uma das maiores diferenças nessa comparação.



(a) Tela inicial React Native **(b) Tela inicial Android Nativo** **(c) Tela inicial Ionic**

Figura 6. Tela inicial implementada nas 3 versões

Tabela 1. Comparação inicial do projeto e APK entre as três versões

	Tamanho do Projeto	**Tamanho do APK**	**Tamanho do APK após a instalação**
Android	284 kb	2.61 MB	8.18 MB
Ionic	164 kb	10.06 MB	14.71 MB
React Native	120 kb	26.1 MB	51.61 MB

7. Avaliação Inicial

Após o desenvolvimento do aplicativo, foi feito um teste inicial utilizando as ferramentas *TalkBack* e *Scanner de acessibilidade*). O objetivo dessa avaliação é verificar a acessibilidade da versão *React Native* quando não há preocupação explícita do programador em implementar requisitos de acessibilidade e compará-la com os testes feitos, das versões *Android* e *Ionic* desenvolvidas anteriormente na mesma fase do processo. A ideia é aferir como os componentes de interface disponíveis já tratam a acessibilidade e a integração com o leitor de tela.

7.1. Materiais e Métodos

Foram utilizados como ferramentas avaliativas o leitor de tela *TalkBack* do pacote de Acessibilidade do *Android* na versão 9.1 e o aplicativo *Scanner* de acessibilidade na versão 2.2. O aparelho usado para os testes foi um *smarthphone* da fabricante Samsung, modelo N960U, com versão 9 do *Android*.

7.2. Como foi feito?

Com o *TalkBack*, foi feita uma navegação por todo o aplicativo observando os elementos que estavam em foco no leitor, bem como o *feedback* sonoro que cada um deles oferece ao usuário. Já com o *Scanner de Acessibilidade*, foi feita uma verificação em cada tela do aplicativo. Cada checagem gerou uma lista de erros e aspectos que precisavam ser melhorados.

7.3. Resultados

No *React Native*, o Talkback identificou corretamente os locais de toque, as caixas de edição de texto e as imagens de *background* não foram lidas. Esse comportamento também ocorreu na versão Nativa em Android. Porém, os botões representados por imagens também não foram lidos e os botões de navegação foram lidos com a descrição *Button*, dificultando o entendimento dos mesmos. O agrupamento de informações também não foi feito e as caixas de entrada de texto não foram lidas corretamente, pois não possuíam marcadores de identificação.

O aplicativo nativo em *Android* fez a leitura dos textos corretamente, identificou os locais de toque bem como as caixa de edição de texto. Ele também saltou imagens de *background* que não precisavam ser lidas. Contudo, alguns botões que eram representados só por imagens não foram lidos devido a não possuírem marcadores de identificação. O agrupamento de informações não foi feito, fazendo com que as informações fossem lidas de maneira separadas.

O aplicativo *Ionic* apresentou vários empecilhos para a navegação. O mais grave é que todo aplicativo é reconhecido como uma visualização da web, não permitindo a navegação por toque na tela. No entanto, é possível fazer a navegação linear (por *swipe*). Outro erro foi que os elementos eram lidos em inglês. O leitor de acessibilidade fez a leitura de todos os itens da tela, não excluindo os elementos que tinham intenção somente ilustrativa. As informações também

eram lidas como se fossem uma página web, identificando qual o tipo de elemento se tratava, como gráfico e cabeçalho, o que não era desejado pois se tratava de um aplicativo. Na tabela 2 tem-se o resumo dos erros identificados com o *Talkback*.

Tabela 2. Erros de acessibilidade identificados com o *Talkback*

Plataforma	Elementos sem marcadores ou marcadores errados	Elementos de background lidos	Erros de feedback pós ação	Erros de agrupamento de conteúdo
Android	8	0	5	12
Ionic	20	9	8	14
React Native	11	0	5	12

As tabelas 3, 4 e 5 mostram a quantidade de erros por tipo de cada tela após a verificação com o *Scanner de Acessibilidade*. O *Android* apresentou 7 erros, rótulos sem descrição e área de toque muito pequena. O *Ionic* apresentou 14 erros, entre eles, tem-se os mesmos apresentados no *Android*, além do erro de vários itens clicáveis ocupando o mesmo espaço devido a *webview* (RIBEIRO; Façanha; VIANA, 2020). Já no *React Native* foram identificados 9 erros, como o baixo contraste do texto com plano de fundo, vários itens clicáveis ocupando o mesmo espaço e rótulo sem descrição.

Tabela 3. Erros de acessibilidade identificados com o Scanner de Acessibilidade no Android

Erro	T1- Tela de Atividades	T2 - Tela de Histórico	T3 - Tela de Perfil	T4 - Tela de Corrida	T5 - Tela de Gráficos
Rótulo sem descrição	2 (Imagem de progresso)	-	2 (Campos de edição de texto)	3 (Botões de play, pause e stop)	1 (Botão de voltar)
Área de toque muito pequena	-	-	-	-	1 (Botão de voltar)

8. Implementação dos requisitos de acessibilidade

A implementação da acessibilidade foi baseada nos testes feitos com o *Talkback* e *Scanner* de acessibilidade, como também nos requisitos do Guia de Acessibilidade Móvel, desenvolvido pelo (SIDI, 2015). Em sua secção de requisitos para melhorar a acessibilidade, o guia lista uma série de parâmetros, mandatórios e desejáveis, a serem adotados por designers e desenvolvedores. Para

Tabela 4. Erros de acessibilidade identificados com o Scanner de Acessibilidade no Ionic

Erro	T1- Tela de Atividades	T2 - Tela de Histórico	T3 - Tela de Perfil	T4 - Tela de Corrida	T5 - Tela de Gráficos
Rótulo sem descrição	2 (Imagem de progresso e botões de navegação)	1 (Imagem da atividade)	2 (Campos de edição de texto)	3 (Botões de play, pause e stop)	1 (Botão de voltar)
Área de toque muito pequena	-	-	-	-	1 (Botão de voltar)
Vários itens clicáveis ocupando o mesmo espaço	1 (webview)	1 (webview)	1 (webview)	1 (webview)	1 (webview)

este artigo, foram selecionados os requisitos mandatórios das seções de interação e navegação listados na Tabela 6.

No aplicativo feito em *React Native* pelo proponente do artigo, foram utilizados requisitos de acessibilidade que a plataforma já oferece e que estão disponíveis na documentação [10] [11].

A tag *"accessible"* foi utilizada para agrupar as *Views* em um único bloco acessível. Já tag *accessibilityLabel* possui funcionamento semelhante ao *contentDescript* do *Android*, sendo usada para adicionar uma descrição nos elementos que precisarem. Também foi utilizado a tag *accessibilityHint* para dar uma dica do que vai acontecer caso o usuário aperte em algum determinado componente. Na Figura 7, é exibido um exemplo da utilização desses três atributos.

```
<TouchableOpacity
  onPress={() => navigation.navigate('Tabs')}
  style={backButton}
  accessible={true}
  accessibilityLabel="Voltar"
  accessibilityHint="Volta para a tela anterior">
  <LeftArrow height="48" width="48" />
</TouchableOpacity>
```

Figura 7. Trecho de código do aplicativo feito em React Native

[10] https://reactnative.dev/docs/accessibility
[11] https://reactnative.dev/docs/accessibilityinfo

Tabela 5. Erros de acessibilidade identificados com o Scanner de Acessibilidade no React Native

Erro	T1- Tela de Atividades	T2 - Tela de Histórico	T3 - Tela de Perfil	T4 - Tela de Corrida	T5 - Tela de Gráficos
Contraste do texto com plano de fundo pequeno	1 (Botão Começar)	-	1 (Botão Calcular IMC)	-	1 (data)
Vários itens clicáveis ocupando o mesmo espaço	-	-	2 (Inputs de texto)	-	-
Rótulo sem descrição	-	-	-	3 (Botões de play, pause e stop)	1 (Botão de voltar)

Para dar um *feedback* ao usuário após efetuar uma ação, por exemplo ao calcular o IMC, foi utilizado a função *announceForAccessibility* do *AccessibilityInfo*, que também é disponibilizado pelo pacote do *react-native*. Ele possui o mesmo funcionamento do *announceForAccessibility* do *Android*, permitindo passar uma mensagem para o leitor de tela a qualquer momento da aplicação.Também foi observado e corrigido a *responsividade* da aplicação, de modo que ao girar a tela, ainda seja possível operar o aplicativo.

9. Segunda avaliação

9.1. Materiais, Métodos e Procedimento

As ferramentas e os recursos utilizados foram os mesmos da primeira avaliação (*TalkBack*, *Scanner* de acessibilidade e dispositivo Samsung N960U). Porém, dessa vez o teste foi feito com os requisitos de acessibilidade devidamente implementados.

9.2. Resultados

9.2.1. Utilizando o TalkBack

A navegação com o *TalkBack* foi concluída com êxito no aplicativo em *React Native*, tendo um comportamento semelhante ao ocorrido no app em *Android*, onde não foram apresentados erros aparentes de *feedback e interação*, porém alguns componentes possuíam descrição a mais indesejada, como *"Button"* no componente de navegação por *tabs*. Já o *Ionic*, apresentou o mesmo erro da *webview* ser apresentada como primeiro elemento em foco. O aplicativo permaneceu se comportando como um site, lendo os tipos de *tag* que o elemento estava encapsulado. No entanto, pelo uso dos atributos *ARIA* os elementos continham a descrição necessária para o entendimento, e foi possível ignorar os elementos desnecessários.

Tabela 6. Requisitos mandatórios de acessibilidade escolhidos para implementar

Requisito	Descrição
R28	O teclado utilizado pela aplicação deve ser compatível com o contexto do campo.
R29	O teclado utilizado pela aplicação deve conter teclas que permitam a navegação entre os componentes da interface.
R31	O leitor de telas deve informar ao usuário todos os eventos visíveis.
R32	O leitor de telas deve informar o conteúdo de um componente assim que tocado, interrompendo qualquer leitura em andamento.
R33	A aplicação deve fornecer feedback sonoro sobre todas as ações executadas pelo usuário.
R34	A aplicação deve fornecer feedback visual sobre todas as ações executadas pelo usuário.
R35	A aplicação deve fornecer feedback visual sobre todas as ações executadas pelo usuário.
R36	As telas da aplicação devem disponibilizar o botão "Voltar"para a tela anteriormente acessada pelo usuário.
R38	A aplicação deve oferecer uma navegação sequencial entre as telas.
R39	A aplicação deve suportar a navegação baseada em foco.
R40	A aplicação deve informar possíveis erros de interação ao usuário.
R41	A aplicação deve guiar o usuário no primeiro uso.
R44	A aplicação deve fornecer instruções de preenchimento dos campos de entrada de dados.
R47	Componentes de interação em telas com grandes textos devem estar fixos e visíveis.

A Tabela 7 mostra a quantidade de erros em relação aos requisitos listados na Tabela 6.

9.2.2. Utilizando o Scanner de acessibilidade

Com a versão com em *React Native* foi possível fazer toda a navegação pelo aplicativo, porém no formulário do IMC foi apresentado o erro de que a *TextView* editável tem a propriedade *contentDescription* e com isso o leitor de tela poderia ler este atributo em vez do conteúdo editável, porém em testes feitos com o *Talkback* tal erro não ocorreu. Na versão em *Android* não foi identificado nenhum erro ao navegar pelo aplicativo utilizando o *Scanner* de acessibilidade. Por outro lado, o aplicativo em *Ionic* apresentou o erro de vários itens clicáveis ocupando o mesmo espaço em todas as telas devido ao *webview* se sobrepor aos elementos (RIBEIRO; Façanha; VIANA, 2020).

10. Avaliação com usuário final

O objetivo desta avaliação é coletar indícios da facilidade ou dificuldade de uso das versões por parte de usuários com deficiência visual de modo a mensurar diferenças na efetividade, eficiência e no grau de satisfação com o uso das versões da PoC.

Tabela 7. Lista de erros encontrados em cada versão de acordo com os requisitos de acessibilidade de SIDI (2015)

Requisito	Erros Android	Erros Ionic	Erros React Native
R28 – Teclado compatível com contexto	0	0	0
R29 – Teclado com navegação	0	2	0
R31 - Informar elementos visíveis	0	0	0
R32 - Informar o conteúdo assim que tocado	0	12	0
R33 - Feedback sonoro das ações	3	3	3
R34 - Feedback visual das ações	0	4	0
R35 - Link para tela principal	0	0	0
R36 - Botão de voltar	0	0	0
R38 - Navegação Sequencial	1	1	1
R39 - Navegação baseada em foco	0	12	0
R40 - Erros de interação	1	4	1
R41 – Guiar usuário no primeiro uso	1	1	1
R44 – Fornecer instruções de preenchimento de campo	0	0	0
R47 – Componentes com grandes textos devem ser fixos e visíveis	0	0	0

10.1. Perfil do usuário

O usuário que participou da avaliação tinha 31 anos, e foi o mesmo usuário que fez os testes no trabalhe de Ribeiro, Façanha e Viana (2020) em *Android* e *Ionic*. Segundo ele, faz uso diário de celular e computador e possui familiaridade com tecnologia, já que é formado em computação. Ele usa o celular com o auxílio de um leitor de tela, sendo o *Whatsapp*, *Youtube*, Banco do Brasil e *Discord* os aplicativos mais utilizados.

Quanto ao grau de deficiência, o voluntário se classificou como baixa visão, não consegue ler nem identificar textos. No entanto, consegue identificar luz e algumas tonalidades de cores. O usuário se demonstra apto a fazer atividades físicas, como academia que pratica frequentemente .

10.2. Materiais e Métodos

O aparelho utilizado foi de propriedade do próprio usuário do teste, um modelo XT1802 da Motorola, com o leitor de tela *TalkBack*. O método utilizado para avaliar a satisfação, eficiência e

a eficácia foi o SUS (*Scale Usability Score*), onde o usuário respondeu as seguintes perguntas para o aplicativo em *React Native* e *Android*, com uma pontuação de 1 à 5, onde 1 ele discorda completamente e 5 concorda completamente (BROOKE, 1996). Em seguida foi feita uma comparação com os resultados obtidos por (RIBEIRO; Façanha; VIANA, 2020). Vale ressaltar que os testes foram feitos com o mesmo usuário e durante o teste foram feitas anotações do comportamento do usuário.

1. Eu acho que gostaria de usar esse sistema com frequência.
2. Eu acho o sistema desnecessariamente complexo.
3. Eu achei o sistema fácil de usar.
4. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema.
5. Eu acho que as várias funções do sistema estão muito bem integradas.
6. Eu acho que o sistema apresenta muita inconsistência.
7. Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente.
8. Eu achei o sistema confuso de usar.
9. Eu me senti confiante ao usar o sistema.
10. Eu precisei aprender várias coisas novas antes de conseguir usar o sistema.

10.3. Procedimento

Primeiramente foi informado ao usuário sobre o aplicativo, explicando seu intuito e funcionalidades. Após foi feito uma explanação inicial no aplicativo para ambientar o usuário na interface. A ordem de uso dos aplicativos foi sorteada. Depois, foram passadas as seguintes tarefas para o usuário realizar na versão em *React Native* e *Android* respectivamente.

1. Na tela inicial:
    - Identificar a quantidade de passo
    - Ir para tela de corrida
2. Na tela de corrida:
    - Iniciar uma corrida
    - Pausar uma corrida
    - Parar uma corrida
    - Ir para a tela de histórico
3. Tela de histórico:
    - Identificar uma atividade realizada
    - Ir para a tela de gráficos
4. Na tela de gráficos:
    - Identificar as informações da tela de gráfico
    - Ir para a tela de perfil

5. Na tela de perfil:
    - Calcular o IMC

Depois de realizar as tarefas, o usuário foi solicitado a responder o questionário do SUS sobre o aplicativo. As duas versões possuindo versões semelhantes, as diferenças percebidas em seus usos se deram apenas pela implementação do suporte à acessibilidade.

Tabela 8. Respostas do usuário ao SUS

	Android	React Native
Pergunta 1	5 (5 - 1 = 4)	5 (5 - 1 = 4)
Pergunta 2	1 (5 – 1 = 4)	2 (5 – 2 = 3)
Pergunta 3	5 (5 - 1 = 4)	5 (5 - 1 = 4)
Pergunta 4	1 (5 – 1 = 4)	1 (5 – 1 = 4)
Pergunta 5	5 (5 - 1 = 4)	5 (5 - 1 = 4)
Pergunta 6	1 (5 – 1 = 4)	1 (5 – 1 = 4)
Pergunta 7	5 (5 - 1 = 4)	5 (5 - 1 = 4)
Pergunta 8	1 (5 – 1 = 4)	1 (5 – 1 = 4)
Pergunta 9	5 (5 - 1 = 4)	5 (5 - 1 = 4)
Pergunta 10	1 (5 – 1 = 4)	1 (5 – 1 = 4)
TOTAL	40 x 2,5 = 100	39 x 2,5 = 97,5

10.4. Resultados

Para obter os resultados é preciso fazer o cálculo estabelecido pelo SUS, no qual estabelece que nas perguntas ímpares é subtraído 1 da pontuação que o usuário deu e nas perguntas pares subtraísse de 5 a pontuação que o usuário atribuiu, ao final o resultado é multiplicado por 2,5. O resultado máximo que pode ser obtido seguindo essa metodologia é 100. Por ser um método já consolidado pela comunidade cientifica, ele possui índices de referência. Desse modo, a média do *System Usability Score* é 68 pontos. Para (BROOKE, 1996), o aplicativo que fizer menos do que a média possui problemas sérios de usabilidade. Na tabela 8 é possível ver os resultados do usuário para as perguntas e suas respectivas pontuações no SUS. O aplicativo *Android* fez 100 pontos, já o app em *React Native* fez 97,5, o que não demonstra uma diferença significativa entre os dois. O resultado com uma pontuação mais elevada se deve ao fato de o usuário já ter utilizado a aplicação nos testes anteriores

11. Discussão

O resultado dos testes mostram diferenças quanto ao suporte à acessibilidade nas três formas de desenvolvimento. No desenvolvimento nativo em *Android* foi notado um maior suporte à acessibilidade, com maior riqueza na documentação. Inclusive podemos notar que a própria

fabricante do sistema demonstra se importar com esse assunto, oferecendo diversos guias para desenvolvedores implementarem o suporte a acessibilidade. A própria IDE (*Android Studio*) sugere dicas e disponibiliza atalhos para implementar um aplicativo mais acessível. Podemos destacar também que a própria Google desenvolveu o *Scanner* de acessibilidade, ferramente essa que foi de extrema importância para os testes deste trabalho, auxiliando na correção do comportamento acessível do aplicativo.

O *Ionic* não demonstrou ser maduro o suficiente em comparação com o desenvolvimento nativo e o desenvolvimento em *React Native*. Apesar de possuir *plugins* para a acessibilidade, não foram suficientes para implementar a acessibilidade de fato, então foi recorrido aos recursos de acessibilidade da web. No entanto os recursos implementados não se comportam como deveriam, o *TalkBack* leu o aplicativo como se fosse um site. Outro erro é que o *webview*, que é utilizada para mostrar o aplicativo, não foi ignorada pelo leitor de tela e ficou no lugar de elementos principais do aplicativo, não permitindo que o aplicativo seja navegado por toque aleatório na tela (i.e., exploração da tela)

Já no *React Native* o resultado é simular ao do *Android*, com um excelente suporte a acessibilidade, com uma página e *APIs* dedicadas à acessibilidade na documentação. Além disso, podemos escrever código nativo em *Android* e *iOS*, então se alguma API provida pelo Facebook e comunidade não suportar alguma função especifica, poderíamos implementar de forma nativa. A IDE utilizada (*Visual Studio Code*) também oferece algumas funções que ajudam no desenvolvimento acessível, por exemplo, autocompletando as *tags* de acessibilidade.

Nos testes antes da implementação da acessibilidade, o aplicativo desenvolvimento em *Android* nativo se demonstrou mais coerente, porém o app em *React Native* não ficou muito atrás. O *Scanner* de acessibilidade identificou 51 erros no *Ionic*, 36 erros no *React Native* e 25 erros no *Android*. Após a implementação o *Android* continuou saindo melhor nos testes, com 0 erros, enquanto o *React Native* teve 2 erros e o *Ionic* 5. Porém se baseando nos requisitos do Guia de Acessibilidade do (SIDI, 2015), o aplicativo *Ionic* apresentou diversos erros, principalmente no R32 - Informar o conteúdo assim que tocado e R39 - Navegação baseada em foco, pois *webview* atrapalhava a navegação e também não dificultava na informação do conteúdo. Já o *Android* e *React Native* tiveram pontuação similar.

O resultado do SUS para o aplicativo *Android* foi de 100, já para o aplicativo em *React Native*, foi de 97,5. Assim, os testes dos aplicativos com o usuário indicam que a diferença é pouco, mas ainda perceptível entre os dois.

Tem-se como **limitações** desta pesquisa o fato de o estudo ter sido feito somente em aparelhos *Android*, sendo que o desenvolvimento *cross-platform* é um conceito para facilitar o desenvolvimento multiplataforma. Devido a falta de recursos, não foi possível fazer testes no *iOS*. As avaliações baseadas no Guia de Acessibilidade do SIDI só levaram em consideração requisitos mandatórios de interação e navegação. Os requisitos facultativos e de interface, além de outros citados pelo SIDI, não foram levados em consideração.

A avaliação foi feita com um indivíduo, o que não demonstra um resultado concreto para este tipo de teste, embora já aponte algumas diferenças no suporte à acessibilidade entre as diferentes versões.

12. Conclusão

Este trabalho abordou três formas de desenvolvimento para dispositivos móveis distintas no contexto do sistema operacional *Android*. O uso de ferramentas *cross-plataform* tem se mostrado promissor e vem cada vez mais melhorando a experiência de desenvolvimento e dos usuários. O

aspecto da acessibilidade aqui analisado apresentou divergências entre as ferramentas *cross-plataform* e o desenvolvimento nativo, porém foi constatado, através da implementação de um aplicativo acessível, que o *React Native* possui mais recursos para dar suporte à acessibilidade que o *Ionic*, porém não possui tantos como o *Android*. No entanto, como já mencionado, é possível utilizar do desenvolvimento nativo em um aplicativo feito em *React Native*, pois ele é interpretado e assim é gerado um código em *Android* e outro nativo em *iOS*, de forma que esses códigos podem ser modificados. Durante a implementação foi descrito nesse artigo os métodos utilizados para deixar o aplicativo acessível, abordando as dificuldades encontradas durante o processo.

Depois, o aplicativo passou por testes avaliativos para saber como ele se comportava antes e depois da implementação dos requisitos de acessibilidade. Os resultados foram comparados com os testes feitos em (RIBEIRO; Façanha; VIANA, 2020) em um aplicativo similar porém desenvolvido em outras tecnologias (*Ionic* e *Android* nativo). Como resultado da comparação, o *React Native* apresentou alguns erros a mais em relação ao *Android*, porém, muitos erros a menos em relação ao *Ionic*. Possuindo um comportamento acessível similar ao desenvolvimento em *Android* nativo.

Como trabalhos futuros, recomenda-se estudar outras formas de desenvolvimento *cross-platform*, como o *Flutter* que vem ganhando bastante popularidade na comunidade, e verificar como a acessibilidade se comporta nessas plataformas. Também é desejável que os testes sejam feitos em *iOS*, outro sistema operacional que possui grande uma quantidade de usuários. Para isso a ferramenta que deve ser utilizada é o *VoiceOver*, leitor padrão desse sistema operacional. Além disso um teste com um número maior de participantes é recomendado.

Referências

ADOBE. *Apache Cordova*. 2009–2021. <https://cordova.apache.org/>. Acessado: 2021-04-14.

APPLE. *Pacote de Acessibilidade do IOS*. 2021. <https://www.apple.com/br/accessibility/vision/>. Acessado: 2021-04-14.

BIØRN-HANSEN, A.; GRØNLI, T.-M.; GHINEA, G. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 5, nov. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3241739>.

BIøRN-HANSEN, A.; GHINEA, G. Bridging the gap: Investigating device-feature exposure in cross-platform development. In: . [S.l.: s.n.], 2018. p. 1–8.

BIøRN-HANSEN, A. et al. An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*, 07 2020.

BRAGA, R. C. Desenvolvimento nativo vs react native: um estudo de portabilidade do jogo híbrido coup acessível. 2019.

BRITO, H. et al. Javascript in mobile applications: React native vs ionic vs nativescript vs native development. In: IEEE. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2018. p. 1–6.

BROOKE, J. Sus: a "quick and dirty'usability. *Usability evaluation in industry*, v. 189, 1996.

CAMPOS, I.; SÁ, E.; SILVA, M. Formação continuada a distância de professores para o atendimento educacional especializado. *Governo Federal. Brasília*, 2007.

CENSO, I. Disponível em:< http://www. censo2010. ibge. gov. br/>. *Acesso em*, v. 23, 2010.

EL-KASSAS, W. et al. Taxonomy of cross-platform mobile applications development approaches, ain shams eng. *J*, v. 8, n. 2, p. 163–190, 2015.

EL-KASSAS, W. S. et al. Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, v. 8, n. 2, p. 163–190, 2017. ISSN 2090-4479. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2090447915001276>.

FACEBOOK. *React Native*. 2015–2021. <https://reactnative.dev/>. Acessado: 2021-04-14.

FGVCIA. *Brasil tem 424 milhões de dispositivos digitais em uso*. 2020. <https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia>. Acessado: 2021-04-09.

GANDHEWAR, N.; SHEIKH, R. Google android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering*, Citeseer, v. 1, n. 1, p. 12–17, 2010.

GARCÍA, J. Livro branco da tecnologia assistiva no brasil. *São Paulo: Instituto de Tecnologia Social–ITS BRASIL*, 2017.

GOAER, O. L.; WALTHAM, S. Yet another dsl for cross-platforms mobile development. In: *Proceedings of the First Workshop on the Globalization of Domain Specific Languages*. New York, NY, USA: Association for Computing Machinery, 2013. (GlobalDSL '13), p. 28–33. ISBN 9781450320436. Disponível em: <https://doi.org/10.1145/2489812.2489819>.

GOOGLE. *Flutter*. 2017–2021. <https://flutter.dev/>. Acessado: 2021-04-14.

GOOGLE. *Pacote de Acessibilidade do Android*. 2021. <https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=pt_BR&gl=US>. Acessado: 2021-04-14.

Guimarães, A. P. N.; TAVARES, T. A. Avaliação de interfaces de usuário voltada à acessibilidade em dispositivos móveis: Boas práticas para experiência de usuário. In: *Anais Estendidos do XX Simpósio Brasileiro de Sistemas Multimídia e Web*. Porto Alegre, RS, Brasil: SBC, 2014. p. 22–29. ISSN 2596-1683. Disponível em: <https://sol.sbc.org.br/index.php/webmedia_estendido/article/view/4923>.

HEMEL, Z.; VISSER, E. Declaratively programming the mobile web with mobl. *SIGPLAN Not.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 10, p. 695–712, out. 2011. ISSN 0362-1340. Disponível em: <https://doi.org/10.1145/2076021.2048121>.

Hui, N. M. et al. Cross-platform mobile applications for android and ios. In: *6th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*. [S.l.: s.n.], 2013. p. 1–4.

IBGE. *Estimativa da população dos municípios para 2020*. 2020. <https://portal.fgv.br/noticias/brasil-tem-424-milhoes-dispositivos-digitais-uso-revela-31a-pesquisa-anual-fgvcia>. Acessado: 2021-04-09.

IONIC. *Ionic Framework*. 2013–2021. <https://ionicframework.com/>. Acessado: 2021-04-14.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. [S.l.]: ISO/IEC, 2001.

LABS, C. *Solar 2D*. 2008–2021. <https://solar2d.com/>. Acessado: 2021-04-14.

M. S. Ferreira, C. et al. An evaluation of cross-platform frameworks for multimedia mobile applications development. *IEEE Latin America Transactions*, v. 16, n. 4, p. 1206–1212, April 2018. ISSN 1548-0992.

MASCETTI, S. et al. Developing accessible mobile applications with cross-platform development frameworks. *arXiv preprint arXiv:2005.06875*, 2020.

MICROSOFT. *Xamarin*. 2011–2021. <https://dotnet.microsoft.com/apps/xamarin>. Acessado: 2021-04-14.

NICHOLL, A. R. J.; FILHO, J. J. B. O ambiente que promove a inclusão: conceitos de acessibilidade e usabilidade. *Revista Assentamentos Humanos*, v. 3, n. 2, p. 49–60, 2001.

OLIVEIRA, L. M. B. et al. Cartilha do censo 2010: pessoas com deficiência. Secretaria de Direitos Humanos da Presidência da República (SDH-PR)., 2012.

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

PROGRESS. *Native Script*. 2014–2021. <https://nativescript.org/>. Acessado: 2021-04-14.

RIBEIRO, L.; Façanha, A.; VIANA, W. Desenvolvimento nativo vs ionic: uma análise comparativa do suporte à acessibilidade em android. In: *Anais do XII Simpósio Brasileiro de Computação Ubíqua e Pervasiva*. Porto Alegre, RS, Brasil: SBC, 2020. p. 1–27. ISSN 2595-6183. Disponível em: <https://sol.sbc.org.br/index.php/sbcup/article/view/11218>.

RODRIGUES, A. et al. Open challenges of blind people using smartphones. *International Journal of Human–Computer Interaction*, Taylor & Francis, v. 36, n. 17, p. 1–24, 2020. Disponível em: <https://doi.org/10.1080/10447318.2020.1768672>.

SARMOTA, S. S.; GHRERA, S. P. Cross platform mobile application development. Jaypee University of Information Technology; Solan; HP, 2019.

SIDI. *Guia para o desenvolvimento de aplicações móveis acessíveis, Centro de Informática da Universidade Federal de Pernambuco*. 2015. Disponível em: <https://www.sidi.org.br/guiadeacessibilidade/>, Acessado em: 2021-02-02.

STATCOUNTER. *Mobile Operating System Market Share Worldwide*. 2021. <https://gs.statcounter.com/os-market-share/mobile/worldwide>. Acessado: 2021-04-09.

STATISTA. *Número de aplicações disponíveis na App Store 2008 à 2020*. 2020. <https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008>. Acessado: 2021-04-09.

STATISTA. *Número de aplicações disponíveis na Play Store de dezembro de 2009 à dezembro de 2020*. 2021. <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acessado: 2021-04-09.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. Introdução à engenharia de software experimental. UFRJ, 2002.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. [S.l.: s.n.], 2014. p. 1–10.

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)

B469d Bezerra, Franklyn.
Desenvolvimento Nativo vs Ionic vs React Native: uma análise comparativa do suporte à acessibilidade em Android / Franklyn Bezerra. – 2021.
25 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2021.
Orientação: Prof. Dr. Windson Viana de Carvalho.

1. acessibilidade. 2. Desenvolvimento móvel. 3. React Native. 4. Android. 5. Ionic. I. Título.

CDD 005
