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

Lucas M. Ribeiro[1], **Agebson R. Façanha**[2,3], **Windson Viana**[1,3]

[1]Instituto Universidade Virtual (UFC Virtual) – Universidade Federal do Ceará (UFC)
CEP 60.320-275 – Fortaleza, CE – Brasil

[2]Núcleo de Tecnologia Assistiva (NTA)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
CEP 60.410-426 – Fortaleza, CE – Brasil

[3]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

lucas.monteiro58@gmail.com, agebson@ifce.edu.br, windson@virtual.ufc.br

Abstract. *Smartphone apps are part of the daily lives of the world's population and influence how these people have fun, work, and learn. The importance of these applications further increases the need to include an audience that may have difficulty accessing them. For example, older people or people with disabilities may be excluded. In this context, this research presents an analysis of accessibility support in mobile applications. The focus of the study is to compare the application development process using the Ionic Cross-platform tool with the native Android development concerning these aspects of accessibility. The initial results obtained from the creation of a proof-of-concept application on both platforms and an investigation with users showed deficiencies in Ionic regarding this support.*

Resumo. *Aplicativos para smartphones fazem parte do cotidiano de um larga parte da população mundial e influenciam a forma como estas pessoas se relacionam, trabalham e aprendem. A importância desses aplicativos aumenta ainda mais a necessidade de incluir um público que pode ter dificuldades de acesso a eles, como por exemplo, pessoas idosas ou com deficiência visual. Dentro deste contexto, esta pesquisa apresenta uma análise do suporte de acessibilidade em plataformas de desenvolvimento de aplicações móveis, em especial, para pessoas com deficiência visual. O foco da pesquisa é comparar o desenvolvimento de aplicativos usando a ferramenta Cross-platform Ionic com o desenvolvimento nativo em Android com relação a esses aspectos de acessibilidade. Os resultados iniciais obtidos a partir da criação de uma aplicação prova de conceito nas duas plataformas e de uma investigação com usuários mostraram deficiências no Ionic quanto a esse suporte.*

1. Introdução

Os *smartphones* estão cada vez mais presentes no cotidiano das pessoas, seja para se comunicar, usar redes sociais ou o pedir *delivery*. Esses dispositivos também têm um

papel fundamental na inclusão digital de pessoas com deficiência. De acordo com o IBGE, em 2010, já existiam 45,6 milhões de pessoas com deficiência no Brasil, sendo 6,5 milhões com algum tipo de deficiência visual [IBGE 2010]. A forma mais comum de inclusão nesses dispositivos vem do uso de serviços de acessibilidade (e.g., *Talk Back* e *Voice Over*) integrados ao sistema operacional que são capazes de ler os textos, botões e as ações realizadas pelo usuário ou mudar configurações como contraste e tamanho de fontes do sistema [Chantre 2015]. Apesar do avanço desses serviços de acessibilidade, muitos desenvolvedores desconhecem como melhor definir suas aplicações para seu correto funcionamento. Os aplicativos necessitam cumprir alguns requisitos como conter informações textuais alternativas aos componentes visuais e integrar essa definição com o serviço de acessibilidade do sistema operacional. Vendome et al. 2019, por exemplo, observaram que somente 50,08% de um conjunto de 13 mil aplicativos têm conteúdo de assistência para todos os seus componentes da interface gráfica. Por outro lado, 46,25% desse universo tinha pelo menos metade dos elementos com descrições de conteúdo vazias, impedindo sua leitura por meio do serviço de acessibilidade [Vendome et al. 2019].

O problema da acessibilidade deficitária pode estar tanto associado ao desconhecimento dos desenvolvedores quanto à plataforma de programação móvel utilizada. Hoje, há dois sistemas operacionais (SO) que dominam cerca de 99% dos dispositivos móveis[1]: o Android e o iOS. Ambos possuem linguagem, especificações e ambiente de desenvolvimento distintos assim como funções para suporte de acessibilidade. No plano de desenvolvimento, muitas empresas têm adotado o conceito de ferramentas *cross-platform*, no qual um aplicativo é desenvolvido em uma linguagem e depois é compilado ou gerado para SOs distintos, economizando tempo e dinheiro no seu desenvolvimento [El-Kassas et al. 2017] [Rieger and Majchrzak 2019].

Dentro desse contexto, o intuito da presente pesquisa é analisar o nível da integração de uma dessas ferramentas *cross-platform* com os serviços de acessibilidade: o Ionic. Ou seja, **a questão de pesquisa central** do artigo é "a plataforma Ionic oferece suporte de acessibilidade tão efetivo quanto a plataforma nativa de programação do Android?" Além disso, o objetivo foi também analisar que contramedidas podem ser efetuadas para melhorar a acessibilidade dos componentes visuais de um aplicativo desenvolvido com o *framework* Ionic.

A literatura científica possui diversos artigos focados na comparação entre o desenvolvimento *cross-platform* e o nativo [Ferreira et al. 2018, Biørn-Hansen et al. 2018, Biørn-Hansen et al. 2020, Rieger and Majchrzak 2019, Krainz et al. 2018]. Como exemplo, Biørn-Hansen et al. 2018, relatam que o acesso a informações de alguns sensores ou recursos multimídia do dispositivo podem não estar disponível nas abordagens *cross-platform*. Isso faz com que um aplicativo dependa de *plugins* de terceiros ou camadas de código extra para integrar o código do aplicativos ao *hardware* do dispositivo. Essa integração tem impactos tanto no consumo de memória como no desempenho dos aplicativos *cross-platform*. Embora muitos desses estudos comparativos existam, o aspecto de acessibilidade não é analisado nestas pesquisas. Apenas Krainz et al. 2018, propõe uma abordagem *cross-platform* para geração de aplicações acessíveis baseada em MDD (Model Driven Development), porém não analisa as plataformas *cross-platform* existentes.

[1] `https://www.netmarketshare.com/operating-system-market-share.aspx`

Similarmente a essas pesquisas comparativas, criou-se um aplicativo Prova de Conceito (PoC). Este foi desenvolvido utilizando o Android Nativo e o Ionic de forma a identificar os pontos positivos e negativos durante o processo de desenvolvimento desses aplicativos com foco na acessibilidade. Os resultados das avaliações das versões com ferramentas automatizadas e com a avaliação de um usuário baixa visão indicam que o suporte à acessibilidade do Ionic é menor e mais dificultoso para os desenvolvedores o implementarem se comparado com o ambiente de desenvolvimento nativo do Android.

O restante deste artigo é estruturado da seguinte forma: a Seção 2 apresenta os requisitos de acessibilidade utilizados na pesquisa. Na Seção 3, a metodologia do estudo é apresentada juntamente como *design* da Prova de Conceito (PoC). Já a seção 4 detalha o desenvolvimento das duas versões (nativa e *cross-platform*) da PoC e discorre sobre suas avaliações iniciais. Após tal análise, na seção 5 detalha-se as implementações das *features* voltadas especificamente para os requisitos de acessibilidade. Por fim, foi realizada uma avaliação com um usuário final com deficiência visual, descrita na Seção 6. E na Seção 7, são apresentadas as considerações finais e propostas de trabalhos futuros.

2. Requisitos de acessibilidade

O conhecimento sobre as dificuldades vividas pelos usuários com deficiência visual é crucial para desenvolver uma boa navegação em aplicativos móveis e implementar os recursos de acessibilidade de forma adequada. Entretanto, essa vivência nem sempre é possível por grande parte de desenvolvedores. Foi pensando nisso que um grupo de pesquisadores da UFPE estabeleceu um conjunto de requisitos de acessibilidade para nortear o desenvolvimentos de aplicativos móveis. O Guia de Acessibilidade Móvel[2] está disponível online. Em sua seção de requisitos para o teste de acessibilidade, o guia lista uma série de parâmetros a ser adotados por designers e desenvolvedores. Neste artigo, foram selecionados os requisitos mandatórios das seções de interação e navegação. São eles:

- 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 - As telas da aplicação, exceto popups (pequenas janelas que se abrem por cima da tela sendo visualizada), devem disponibilizar link para a tela principal do aplicativo.
- R36 - As telas da aplicação devem disponibilizar o botão “Voltar” para a tela anteriormente acessada pelo usuário.
- R39 - A aplicação deve suportar a navegação baseada em foco.
- R40 - A aplicação deve informar possíveis erros de interação ao usuário.

3. Metodologia

3.1. Etapas da Pesquisa

Esta pesquisa pode ser categorizada como exploratória em busca de estabelecer indícios iniciais do suporte deficiente à acessibilidade em ferramentas *cross-platform*. O elemento

[2] `http://www.sidi.org.br/guiadeacessibilidade`

central da pesquisa é um aplicativo Prova de Conceito (PoC) que foi implementado em duas versões, uma utilizando o Android Nativo e outra o Ionic. A escolha do Ionic se deu por sua popularidade. Junto com o React Native e Flutter, são os *frameworks cross-platform* mais utilizados nos últimos dois anos [Rieger and Majchrzak 2019, Biørn-Hansen et al. 2020]. O Ionic é um *framework* de desenvolvimento de aplicações móveis que utiliza as linguagens Web (HTML, CSS e JavaScript) com forte integração com o *framework* de desenvolvimento Web Angular[3]. A Figura 1 exibe as principais etapas desta pesquisa.

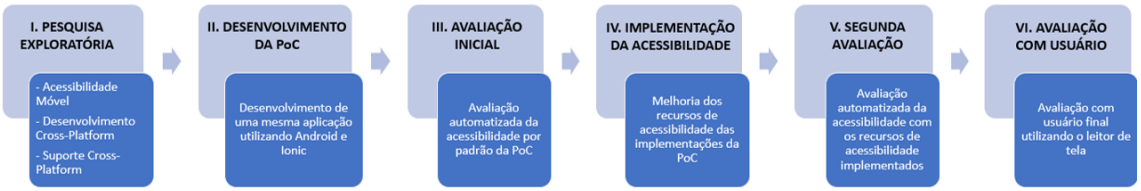

Figura 1. Resumo das etapas presentes na metodologia

Em um primeiro momento de desenvolvimento, utilizou-se os recursos padrão para criação das interfaces gráficas da PoC em cada abordagem (etapa II). Em seguida, foram realizados testes de acessibilidade com uma ferramenta de verificação de acessibilidade para identificar possíveis falhas e trabalhar em melhorias nas duas versões (etapa III). A quarta etapa da pesquisa consistiu no redesign dessas aplicações seguindo os requisitos do Guia de Acessibilidade Móvel por meio do uso dos recursos disponíveis em cada abordagem (e.g., uso de descritores). Depois, foi feita uma nova avaliação com os recursos de acessibilidade já implementados, para poder analisar a acessibilidade das novas interfaces. Nos dois casos, as mesmas ferramentas forma utilizadas (etapa V). Por fim, uma avaliação com um usuário baixa visão foi realizada com as duas versões da PoC.

Nas avaliações manuais, o aplicativo PoC era usado com o assistente de acessibilidade do Android (TalkBack) habilitado. Além disso, as avaliações automatizadas foram feitas com o aplicativo Accessibility Scanner[4] da Google. Esse app da Google verifica automaticamente as telas do aplicativo e fornece sugestões para melhorar a acessibilidade delas baseado, por exemplo, em buscas por etiquetas de conteúdo e em avaliações do tamanho das regiões de toque.

3.2. Prova de Conceito - PoC

A PoC desenvolvida nesta pesquisa é denominada FitApp e tem a temática de *fitness*. Ela armazena informação das atividades físicas do usuário e gera relatórios para que se possa acompanhar o progresso dos exercícios. Além disso, é possível fazer o monitoramento em tempo real do exercício, por meio de uma *smartband* conectada ao dispositivo. Na Figura 2, estão presentes as telas do design desejado da PoC em alta fidelidade. A tela inicial do aplicativo contém informações do último treino. A navegação entre telas é feita pelo menu de três opções encontrado na parte inferior do aplicativo. Ao clicar no botão **começar**, localizado na tela inicial do aplicativo, será direcionado para a tela de exercício. Essa tela monitora o treino, permitindo acompanhar dados como tempo de exercício, gasto calórico

[3]https://angular.io/

[4]https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=pt_BR

e distância percorrida. A tela de histórico lista os exercícios realizados pelo usuário. Ao clicar em uma das opções da lista, o utilizador é direcionado para uma nova tela que contém os gráficos com as informações detalhadas sobre o exercício. Por último, a tela de perfil contém um formulário que permitirá ao usuário calcular o IMC (Índice de Massa Corporal).

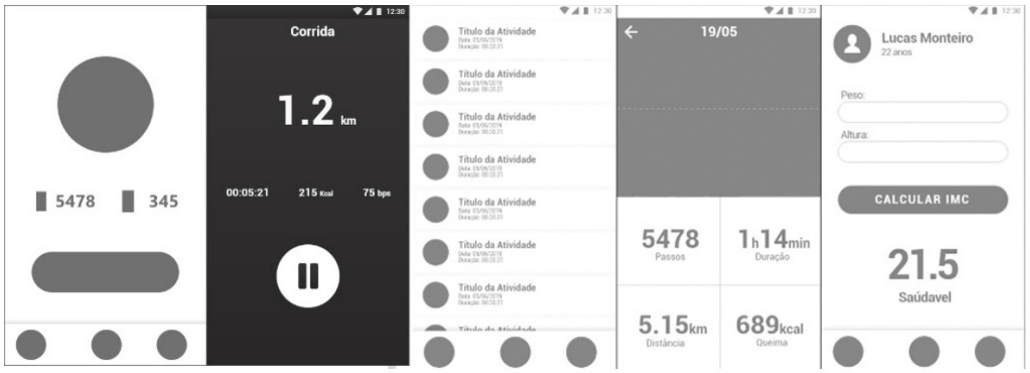

Figura 2. Telas da PoC em alta fidelidade

4. Desenvolvimento das versões e avaliação inicial

4.1. Versão Android Nativa

A aplicação em Android Nativo[5] foi desenvolvida utilizando a linguagem de programação Java. As três telas principais (histórico, atividade e perfil) são *Fragments* controladas por um *BottomNavigatorView*, utilizado para fazer a alternâncias entre as três telas principais. Já a tela de corrida e gráficos foram criadas utilizando uma nova *Activity*. O padrão de *layout* utilizado foi o *Relative Layout*, já os elementos elementos visuais foram criados utilizando os *widgets* padrões do Android, como botões, *inputs* e *label* de texto. Os componentes foram personalizados de acordo com o visual definido para a PoC. O aplicativo nativo ao final tinha 65 arquivos com um apk de tamanho 2,62 MB e 8,18 MB após a instalação.

4.2. Versão Ionic

A versão em Ionic[6] foi desenvolvida baseada no Angular. O comando inicial *ionic start FitApp tabs* gerou um *template* de navegação por *tabs* padrão do Ionic com o nome FitApp. Toda página contém os arquivos essenciais do projeto: um arquivo HTML, um arquivo CSS, e os arquivos *TypeScript* responsáveis pela lógica, modularização e rotas do Angular. A versão compilada contém uma WebView, que no sistema Android é responsável por proporcionar a visualização do conteúdo web (HTML, Css e JavaScript) do aplicativo. O aplicativo pode ser testado no navegador, não dependendo de um *smartphone* ou emulador para rodá-lo. A compilação do aplicativo, com a geração do instalável para dispositivos Android foi feita com o comando *ionic cordova build*. O aplicativo desenvolvido com o Ionic ao final tinha 69 arquivos com um apk de tamanho 10,06 MB (quase 4 vezes maior do que a versão nativa) e 14,71 MB após a instalação. A Figura 3 apresenta três das cinco telas da versão Android e da versão Ionic.

[5]`https://github.com/lucasmonteiro58/Android_TCC`
[6]`https://github.com/lucasmonteiro58/Ionic_TCC`

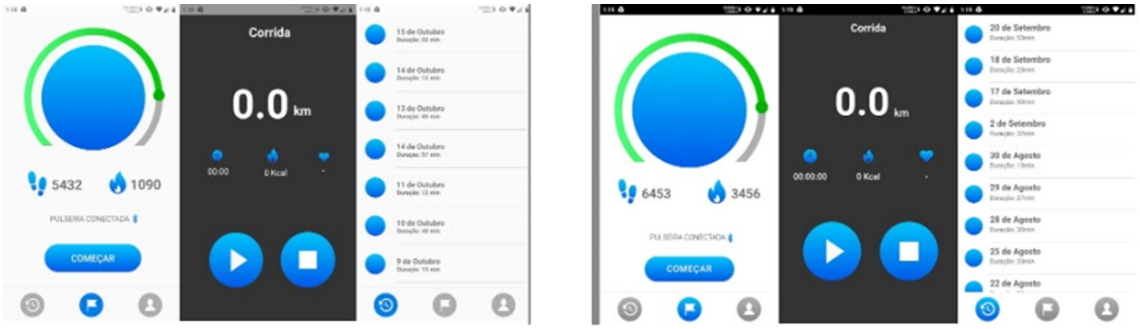

Figura 3. Telas do FitApp em Android Nativo (à direita) e em Ionic (à esquerda)

4.3. Avaliação inicial das PoC

Depois que os aplicativos foram prototipados, realizou-se avaliações automatizadas utilizando o aplicativo Accessibility Scanner e avaliações manuais com o auxílio do leitor de tela TalkBack. Como exposto na metodologia, os testes iniciais foram executados nas duas versões da PoC. Versões estas em que não ocorreram preocupações explícitas do programador em implementar a acessibilidade, ou seja, a construção das interfaces utilizou os componentes configurados por padrão de cada plataforma. O aparelho usado para os testes foi um *smartphone*, modelo XT1802, com a versão 9 do Android.

Cada tela das duas versões da PoC foi validada com o Accessibility Scanner que gerou uma lista de erros e aspectos a serem melhorados. Por exemplo, a versão nativa apresentou sete erros referentes aos elementos visuais (botões imagens) que não continham elementos textuais associados. Já a versão com o Ionic apresentou quatorze erros que vão desde itens sem descrição à itens clicáveis ocupando o mesmo espaço na tela. No caso do Ionic, era indicada um erro em cada tela sinalizando que vários itens clicáveis estavam ocupando o mesmo espaço da tela. Este erro é referente ao *Webview* e os componentes que o sobrepõem. Com o TalkBack, um dos autores, seguindo as boas práticas do Android, fez a 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.

Na interação com o aplicativo nativo, o leitor de tela conseguiu realizar a leitura dos textos corretamente, identificou de forma aceitável os elementos virtuais, em especial, as caixas de edições de texto. Ele também saltou as imagens que não precisavam ser lidas. No entanto, alguns botões virtuais representados somente por uma imagem ficaram sem marcadores de identificação, não permitindo, desse modo, saber sobre o que se tratavam. Os agrupamentos de informações não foram feitos, fazendo com que as informações fossem lidas de maneira separadas.

Já na versão desenvolvida pelo aplicativo de *cross-plataform* Ionic, o leitor de tela encontrou vários empecilhos para a navegação. O mais grave é que todos os elementos da tela do aplicativo ficam agrupados num único componente principal (um *Webview*), afetando diretamente a exploração manual da tela. Neste tipo de exploração, o usuário passa o dedo sobre a tela em busca dos componentes existentes e escuta pelo assistente de acessibilidade a descrição e tipo de ação associada. No entanto, é possível fazer a navegação linear (por *swipe*) para acessar e ouvir os elementos individuais da tela. Na navegação linear, a leitura da tela irá funcionar como “ab” no computador que muda o foco de qual elemento da tela está sendo lido. Usando esse modo de navegação com a

versão do Ionic, o leitor de tela fez a leitura dos elementos virtuais como se fossem uma página *web*, identificando qual tipo de elemento se trata (gráfico, cabeçalho, por exemplo). O que não é desejável pois se trata de um aplicativo. Outro erro foi que os elementos eram lidos em inglês, já que por padrão, é a língua utilizada no *head* da página HTML.

Tabela 1. Quantidade de erros contabilizados com o TalkBack

Plataforma	Qtd. de elementos sem marcadores	Qtd. de elementos não ignorados	Qtd. de erros de feedback pós ação	Qtd. de erros agrupamento de conteúdo	Totais
Android	8	0	5	12	25
Ionic	20	9	8	14	47

A Tabela 1 sintetiza a quantidade de erros em cada uma das abordagens utilizando o leitor de tela TalkBack. Tais disparidades inicialmente causaram estranheza aos pesquisadores que resolveram realizar uma nova análise em três outros aplicativos desenvolvidos com o Ionic para averiguar se essa quantidade de erros encontrados era uma particularidade da PoC ou se tinha maior relação com a plataforma de desenvolvimento. O objetivo era comparar os resultados obtidos nos testes do aplicativo desenvolvido com outros três testados. Foram escolhidos os aplicativos *Untappd*, *MarkerWatch* e *Pacifica* desenvolvidos em Ionic e disponíveis no site[7] do *framework*. Os resultados foram semelhantes ao teste feito com a PoC, destacando-se o mesmo erro de agrupar todo o conteúdo em uma *Webview*, não permitindo a navegação por exploração da tela conforme descrito.

Os aplicativos também se comportaram como se fossem um *site web*, fazendo a leitura do tipo de conteúdo de cada elemento focado. Esse teste forneceu um forte indicativo de problemas graves no suporte à acessibilidade na plataforma. No entanto, não se pode confirmar se os desenvolvedores desses aplicativos tiveram cuidado em respeitar recomendações de acessibilidade.

5. Implementação das *features* de acessibilidade e reavaliação

5.1. Utilizando o suporte de acessibilidade

A implementação dos requisitos de acessibilidade na versão nativa para Android utilizou as ferramentas que o próprio SO oferece. O atributo *contentDescription* é utilizado nos elementos que precisam de uma descrição, para que essa seja reconhecida no leitor de tela e assim o usuário possa ter o *feedback* do elemento que está em foco. Elementos que precisavam ser ignorados no leitor de tela receberam o atributo *importantForAcessibility='no'*. Alguns recursos precisaram ser implementados via código Java. Dentre eles, está o *feedback* que é a resposta do cálculo de IMC do usuário, que precisa ser dito assim que o botão de calcular é acionado. Para isso, foi usado o código *view.annouceForAcessibility('mensagem')*. Com ele, é possível passar uma mensagem para o leitor de tela a qualquer momento da aplicação.

Em contraponto, o Ionic não oferece nenhuma *feature* ou componente específico para a implementação da acessibilidade. No entanto, por se tratar de uma tecnologia *web*, é possível deixar o aplicativo acessível seguindo as recomendações do WCAG[8] e,

[7] *Ionic ShowCase*: `http://showcase.ionicframework.com/apps/top`

[8] `https://www.w3.org/WAI/standards-guidelines/wcag/`

assim, utilizar e remodelar as *tags* HTML, organizando-as de acordo com o padrão de conteúdo recomendado na W3school[9]. Por exemplo, a marcação de regiões importantes na página (como uma caixa de busca, um cabeçalho, chamadas "pontos de referência"), atributos para informar as demarcações de conteúdo para facilitar a navegação (agilizam a utilização de leitores de tela), *JavaScript* para *widgets*, sugestões de preenchimento de formulário e mensagens de erro, atualizações de conteúdo em tempo real e muito mais. No código do aplicativo, o atributo *aria-label* foi usado para adicionar uma descrição de um elemento da página que será lido pelo leitor de tela, semelhante ao *contentDescription* no Android. Já o atributo *aria-hidden*, é usado para ocultar alguns elementos do leitor de tela, equivalente ao *importantForAcessibility* do Android.

5.2. Avaliação do *redesign* das PoC

As ferramentas e os recursos utilizados foram os mesmos da primeira avaliação (TalkBack, Accessibility Scanner e dispositivo Motorola XT1802). Dessa vez, as avaliações foram feitas com os requisitos de acessibilidade devidamente implementados. A navegação com o TalkBack foi concluída com êxito no aplicativo Android. Não foram apresentados erros aparentes de *feedback* e interação. Já o Ionic, apresentou o mesmo erro da *Webview* ser apresentada como primeiro elemento em foco. O aplicativo ainda se comportava como um site, lendo os tipos de *tag* que o elemento estava encapsulado. No entanto, com o uso dos atributos de acessibilidade para HTML, os elementos passaram a fornecer a descrição necessária e foi possível ignorar os elementos desnecessários. No total, a versão Ionic apresentou 34 erros (17 a menos que na avaliação anterior). Já a versão nativa apresentou resultado satisfatório atendendo os requisitos mínimos de acessibilidade. Assim, o aplicativo Accessibility Scanner não identificou violações. Já com a versão, o app indicou novamente o erro de vários itens clicáveis ocupando o mesmo espaço em todas as telas devido a *Webview* se sobrepor aos elementos (5 erros ao total). Nos links na nota de rodapé pode se ver o uso da versão Ionic[10] e Nativa[11] por meio de navegação em *swipe*.

6. Estudo de caso com um usuário final

O objetivo desta análise era coletar indícios sobre o uso das duas versões por parte de usuários com deficiência visual de forma a mensurar diferenças na efetividade, eficiência e no grau de satisfação com o uso das versões.

O usuário que participou da avaliação é autodeclarado com baixa visão (não identificando letras em um texto, mas identifica presença de luz e algumas tonalidades de cores), de sexo masculino, com 30 anos e bacharel em Computação. Faz uso diário de *smartphone* e computador com o auxílio de um leitor de tela, sendo *Whatsapp*, *Facebook* e Banco do Brasil os aplicativos mais utilizados. O usuário pratica atividades físicas, como caminhada e musculação. O aparelho utilizado foi o mesmo dos testes anteriores (XT1802), com o leitor de tela TalkBack.

O instrumento adotado para obter de forma eficiente dados sobre as percepções, expectativas e interação dos participantes com o aplicativo foi o SUS (*Scale Usability Score*) [Brooke 2013]. O SUS apresenta excelente rendimento e consistência de resultados para testes com tamanhos de amostra relativamente pequenos

[9] `https://www.w3schools.com/html/html_accessibility.asp`
[10] `https://youtu.be/lB1nKhvczT4`
[11] `https://youtu.be/15v8fRKA7Kc`

[Tullis and Stetson 2004] e consiste em dez declarações, no formato de escala Likert, através das quais os participantes podem avaliar seu nível de concordância com a aplicação. As pontuações dadas pelos usuários são posteriormente convertidas em uma escala de 0 a 100, usando cálculos com penalidades ponderadas, o que indica o grau de aprovação do aplicativo avaliado.

6.1. Procedimentos

O primeiro passo do teste foi falar para o usuário sobre o aplicativo, explicando seu intuito e funcionalidades. Logo em seguida, foi feita uma explanação inicial de cada aplicativo para poder se ambientar na interface. A ordem de uso dos aplicativos foi sorteada. Depois de realizar as tarefas (e.g., realizar corrida, consultar histórico), o usuário respondeu ao questionário do SUS sobre cada aplicativo. Como as duas versões da aplicação proporcionam o mesmo conjunto de tarefas e são visualmente semelhantes, as diferenças percebidas em seus usos se deram apenas pelo suporte à acessibilidade.

6.2. Resultados e Discussão

Após o sorteio, o usuário primeiro usou a versão Android e depois a versão em Ionic. Ele conseguiu realizar as tarefas com mais facilidade utilizando a versão nativa, além de receber *feedbacks* das ações que realizava. Apesar de também ter conseguido realizar todas as tarefas com a versão do aplicativo desenvolvido em Ionic, o usuário fez isso com muita dificuldade e depois de muitas tentativas. Ressalta-se que a primeira versão testada foi a nativa Android, então o usuário já tinha uma prévia de como o aplicativo era ao utilizar a outra versão. Caso isso não tivesse ocorrido, provavelmente o usuário enfrentaria ainda mais dificuldades. Além disso, o fato do leitor de tela não ignorar a *Webview* do aplicativo fez com que o usuário se atrapalhasse na navegação entre os elementos, pois não podia realizar as ações por toque. A falta de *feedback* em algumas ações também causaram dúvidas no usuário, como por exemplo, saber se a corrida tinha iniciado ou saber o resultado do cálculo do IMC. Tais situações refletiram diretamente nos baixos índices obtidos nas análises de usabilidade com o SUS. A versão nativa conseguiu um total de 65 pontos e a outra versão obteve 32,5.

A avaliação com o usuário mostrou que os erros apontados nos testes anteriores também se repetiram em seu uso real e que afetam diretamente o julgamento do usuário quanto a usabilidade dos aplicativos, sendo a versão nativa que obteve o melhor resultado. Com 65 pontos, classifica-se com uma usabilidade satisfatória, conforme Brooke 2013. Vale ressaltar, entretanto, que o valor de 32,5 obtido no SUS pela versão desenvolvida com Ionic revela um índice de usabilidade totalmente insatisfatório.

7. Considerações Finais

Esse trabalho analisou o suporte de acessibilidade em duas abordagens de programação para a plataforma Android. A abordagem *cross-platform* tem se mostrado promissora, embora ainda necessite de melhorias em diversos pontos como desempenho e mantenabilidade. O aspecto da acessibilidade analisado nesta pesquisa também apresentou resultado díspares comparando as duas abordagens (Nativo e Ionic). Durante o desenvolvimento, foi usado um processo em duas etapas para implementar requisitos de acessibilidade em duas versões de um mesmo aplicativo. Depois de cada etapa, as versões passaram por testes avaliativos para saber como o aplicativo se comportava antes e depois

da implementação da acessibilidade. Em ambos os casos, a programação nativa apresentou um número menor de erros que o Ionic. E a análise dos vídeos de utilização mostram bem a diferença de usabilidade entre as versões.

Como **limitações** desta pesquisa, destaca-se o fato de o estudo ter sido feito somente em aparelhos Android, sendo que o desenvolvimento *cross-platform* é baseado no princípio de desenvolver aplicativos para mais de um SO. As avaliações também levaram em consideração somente os itens obrigatórios do Guia de Acessibilidade (SIDI). A avaliação com usuários foi feita somente com um indivíduo, o que pode não trazer um resultado tão concreto para esse tipo de teste, embora já sinalize grandes diferenças no suporte à acessibilidade das versões criadas.

Como trabalhos futuros, pretende-se estudar outras ferramentas *cross-platform* e verificar como a acessibilidade é suportada em cada uma delas. Também é desejável que os testes sejam feitos no iOS e com um número de usuários mais significativo.

Referências

Biørn-Hansen, A., Grønli, T.-M., and Ghinea, G. (2018). A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Computing Surveys (CSUR)*, 51(5):1–34.

Biørn-Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A., and Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*.

Brooke, J. (2013). Sus: A retrospective. *J. Usability Studies*, 8(2):29–40.

Chantre, J. R. M. (2015). *Testes automáticos de acessibilidade em aplicações móveis*. PhD thesis, Universidade da Beira Interior, Covilhã, Portugal.

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., and Wahba, A. M. (2017). Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, 8(2):163–190.

Ferreira, C. M., Peixoto, M. J., Duarte, P. A., Torres, A. B., Junior, M. L. S., Rocha, L. S., and Viana, W. (2018). An evaluation of cross-platform frameworks for multimedia mobile applications development. *IEEE Latin America Transactions*, 16(4):1206–1212.

IBGE, I. (2010). Censo demográfico 2010. *IBGE: Insituto Brasileiro de Geografia e*.

Krainz, E., Miesenberger, K., and Feiner, J. (2018). Can we improve app accessibility with advanced development methods? In *International Conference on Computers Helping People with Special Needs*, pages 64–70. Springer.

Rieger, C. and Majchrzak, T. A. (2019). Towards the definitive evaluation framework for cross-platform app development approaches. *Journal of Systems and Software*, 153:175–199.

Tullis, T. S. and Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Usability professional association conference*, pages 1–12.

Vendome, C., Solano, D., Liñán, S., and Linares-Vásquez, M. (2019). Can everyone use my app? an empirical study on accessibility in android apps. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 41–52. IEEE.