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

Lucas M. Ribeiro[1], Windson Viana[2]

[1]Instituto Universidade Virtual (UFC Virtual)
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

lucas.monteiro58@gmail.com, 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 application development using the Cross-platform Ionic tool with native Android development by analysing accessibility aspects.*

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 as eles, como por exemplo, pessoas idosas ou com algum tipo de deficiência. Dentro deste contexto, esta pesquisa apresenta uma análise do suporte de acessibilidade em aplicações móveis. 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 aspectos de acessibilidade.*

1. Introdução

Os *smartphones* estão cada vez mais presentes no cotidiano das pessoas, seja para se comunicar, usar redes sociais ou qualquer outro aplicativo. Segundo uma pesquisa da Fundação Getúlio Vargas de São Paulo, em 2018 o Brasil atingiu a marca de 220 milhões de aparelhos ativos. Fazendo uma comparação com a quantidade de habitantes do país (207,7 milhões), são mais de um smartphone por pessoa. Esses números refletem na quantidade de aplicativos e no número de downloads nas lojas digitais. Segundo um levantamento da Statista, em 2018 existem cerca de 2,1 milhões[1] de aplicativos

[1] Statista: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

publicados na *Google Play*. Nesse mesmo ano, a quantidade estimada de *downloads* registradas para essa plataforma é cerca de 205,4 bilhões[2]. Esses dispositivos também tem um papel fundamental na inclusão digital de pessoas com deficiência. A forma mais comum presente nos smartphones são *softwares* de acessibilidade (e.g., *Talk back, Voice Over)* integrados ao sistema operacional que são capazes ler os textos, botões e as ações realizadas pelo usuário [Chantre 2015]. De acordo com o IBGE (2010), existem 45,6 milhões de pessoas com deficiência, sendo 6,5 milhões com algum tipo de deficiência visual, por exemplo. Os *smartphones* por meio de seus *softwares* de acessibilidade proporcionam o acesso a seus aplicativos e a *web mobile* a esse público.

Em relação aos modelos de *smartphones*, há dois sistemas operacionais que dominam cerca de 99% desses dispositivos[3]: o *Android* e o *iOS*. Ambos possuem linguagem, especificações e ambiente de desenvolvimento distintos. Isso demanda dos desenvolvedores de aplicativos que dominem as duas plataformas ou que as fábricas de *software* montem duas equipes de desenvolvedores, uma para cada plataforma. Dessa forma, há uma desperdício de tempo e dinheiro quando se trata de construção de aplicativos para ambos sistemas operacionais [Ferreira et al 2018][El-Kassas et al. 2015]. É nesse contexto que surge o conceito de ferramentas *cross-platform*, no qual o aplicativo é desenvolvido em uma linguagem e depois é compilado ou gerado para sistemas operacionais distintos, economizando tempo e dinheiro no seu desenvolvimento [El-Kassas et al. 2015]. Existem diversas abordagens *cross-plataform*, entre elas estão o *Ionic, React Native e o Flutter*.

Embora o desenvolvimento *cross-platform* possua vantagens no que diz respeito ao tempo de desenvolvimento, existem algumas limitações. Hansen et al. (2018) relata que, pelo o código de aplicativos multiplataforma ser desenvolvido em uma linguagem não nativa, o aplicativo pode não ter uma boa comunicação com o *hardware* e API (*Application Programming Interface*) do dispositivo. Isso faz com que a implementação dependa de plugins a fim de que haja uma conversa entre o código e o *hardware*. No entanto, esses *plugins* são bastante limitados e existem poucos ou nenhum suporte para outros recursos, como o de acessibilidade [Corral 2012].

Esse presente trabalho tem como objetivo principal analisar o atual suporte de ferramentas *cross-platform* no desenvolvimento de aplicativos acessíveis a deficientes visuais. A fim de alcançar o objetivo principal e identificar quais recursos podem ser implementados, foram estabelecidos os seguintes objetivos específicos: (i) realizar uma prova de conceito com o desenvolvimento de uma mesma aplicação acessível utilizando *Android* Nativo e um *framework* de desenvolvimento *cross-platform*; (ii) Identificar os pontos positivos e negativos durante o processo de desenvolvimento desses aplicativos

---

[2] Statista: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

[3] Market Share: https://www.netmarketshare.com/operating-system-market-share.aspx

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 desse trabalho deu-se da seguinte forma: As seções 2 e 3 apresentam a fundamentação teórica desta pesquisa. A seção 4 apresenta a metodologia utilizada. As seções 5 e 6 explicam a Prova de Conceito e seu desenvolvimento. Na seção 7 é feita a primeira avaliação, sem a implementação da acessibilidade. Na seção 8 é explicado como a acessibilidade foi implementada nos dois aplicativos. Na seção 9 e 10 é feita novas avaliações, dessa vez com a acessibilidade devidamente implementada. As seções 11 e 12 contam com a discussão dos resultados e a conclusão do trabalho.

2. Acessibilidade

O termo acessibilidade é usado para definir a possibilidade de qualquer pessoa, independente de suas capacidades físicos-motoras e perceptivas, culturais e sociais, desfrutar dos benefícios de uma vida em sociedade [Nicholl 2001]. Desse modo, a acessibilidade móvel trata-se do oferecimento efetivo de toda a informação para todos os utilizadores, independente da tecnologia ou plataforma utilizada e das capacidades sensoriais, motoras ou funcionais do utilizador [Chantre 2015].

As deficiências que necessitam dos recursos de acessibilidade são diversas. Dentre elas estão a pouca ou nenhuma visão, problemas de audição, mobilidade e cognição. Segundo o censo de 2010, 3,5%(6,5 milhões) da população do Brasil tem deficiência visual, 1.1% tem deficiência auditiva, 2.3% tem problemas de mobilidade e 1,4% tem problemas cognitivos [IBGE, 2010]. Por está mais presente, a deficiência visual é a mais requisitada no que diz respeito a produção de conteúdo acessível a ela.

Em 1999, foi estabelecido a primeira versão das Diretrizes para a Acessibilidade do Conteúdo da Web (WCAG), elaborada pelo grupo de trabalho do WAI (*Web Accessibility Initiative*) do comitê internacional W3C (*World Wide Web Consortium*), que regula os assuntos ligados à Internet. A WCGA está organizada em torno de quatro princípios principais que questionam se o produto possui elementos perceptíveis, operáveis, compreensíveis e robustos. Dentro das orientações do WCAG estão presentes as instruções do MWBP (*Mobile Web Best Practices*), que estão mais diretamentes ligadas às especificações gerais de construção de sistemas *web-mobile*. Essas diretrizes são referências até hoje na fomentação de recursos de acessibilidade em diversas plataformas, dentre elas o *Android* e *iOS* [Chantre 2015].

O guia para o desenvolvimento de aplicações móveis acessíveis, elaborado pelo SIDE (2015), cita alguns princípios que o aplicativo deve seguir a fim de garantir uma maior acessibilidade:

- Design Minimalista: Priorizar componentes relevantes e com função comprovada.
- Fluxo Natural: Ordenar as telas e os elementos da esquerda para direita, de cima para baixo.
- Coerência Externa: Posicionar os elementos de acordo com as convenções adotadas em outros *apps*.
- Coerência interna: Padrão interno de posicionamento dos elementos em todos os elementos da interface.
- Quantidade de informação Vertical: Evitar telas com muito conteúdo no sentido vertical, pois isso dificulta e torna cansativo a interação.
- Cores da interface: Utilizar cores que facilitem a identificação dos elementos na página, trabalhando com um bom contraste para usuários com baixa visão.

A acessibilidade para deficientes visuais nos smartphones é feita por um leitor de tela incorporado ao sistema operacional. No *Android*, o aplicativo para leitura de telas se chama *TalkBack*[4] e no *iOS,* se chama *Voice Over*[5]. Ambos são desenvolvidos pelas próprias fabricantes dos sistemas operacionais (*Google* e *Apple*). Mesmo sendo de sistemas operacionais distintos, os dois possuem funcionamento muito semelhante. Usam a emissão de som descritivo (*feedback* falado) quando algum aplicativo ou opção é clicada na tela. Desse modo, o usuário sabe o que está sendo selecionado no momento. Há também o reconhecimento de gestos de dois dedos ou três dedos na tela para realizar funções como rolagem do *scroll* ou acessar demais opções do aplicativo. A Tabela 1 sintetiza o uso dos leitores de tela.

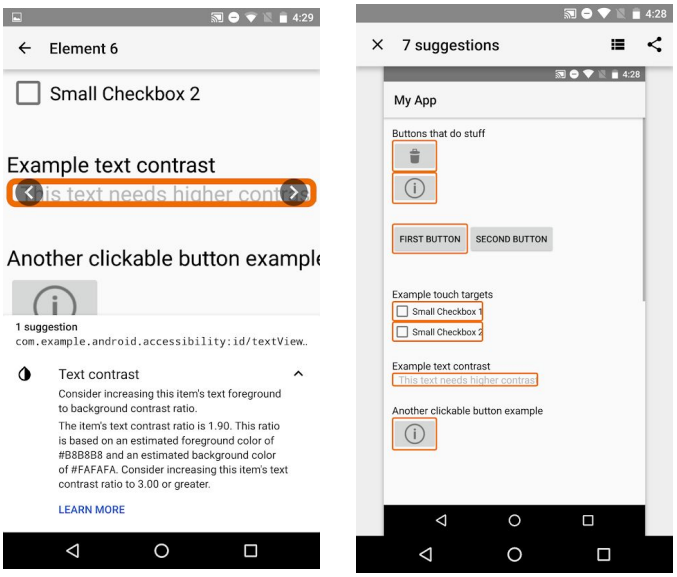


Figura 1 - Interface do Scanner de Acessibilidade

[4] TalkBack: https://support.google.com/accessibility/android/answer/6283677
[5] VoiceOver: https://www.apple.com/br/accessibility/iphone/vision/

Tabela 1 - Uso dos leitores de Tela

Tipo de navegação	Descrição	Perfil do usuário
Deslizando o dedo	O usuário toca com o dedo na tela e desliza sobre ela escutando os feedbacks do leitor de telas. O usuário pode realizar essa exploração contornando as extremidades do aparelho, de cima para baixo pelo centro da tela ou após encontrar um componente, deslizando o dedo no entorno dele para encontrar outros componentes próximos.	Usuários com perda total da visão.
Tocando aleatoriamente	O usuário toca com o dedo na tela aleatoriamente, podendo ser em um lugar de forte convecção de posicionamento ou através de uma cor de alto contraste para usuários de baixa visão.	Usuários com perda parcial ou total da visão.
Com o gesto TAB (*Swipe*)	O usuário desliza o dedo da esquerda para a direita (ou vice-versa) para passar os itens em elementos que podem receber foco.	Usuários com perda parcial ou total da visão.

Por sua vez, o *Accessibility Scanner*[6] (Scanner de acessibilidade) é um aplicativo desenvolvido pelo *Google* que ajuda a identificar oportunidades para melhorar a experiência do usuário no que diz respeito à acessibilidade. Ele analisa a tela do aplicativo, identifica os erros e sugere melhorias de acordo com as práticas recomendadas de acessibilidade para aplicações móveis, conforme mostra o exemplo da Figura 1. A análise do aplicativo verifica as marcações de conteúdo, implementação, tamanho da área do toque e situações de baixo contraste de elementos visuais. Desse modo, depois da verificação e da correção das falhas encontradas, a uma garantia maior de que o aplicativo poderá ser usado sem erros por pessoas com deficiência. [SIDE 2015].

[6] Accessibility Scanner: https://support.google.com/accessibility/android/answer/6376570

3. Abordagens Cross-Platform

Aplicações nativas são desenvolvidas com o uso de ferramentas e linguagens de programação particular para determinada plataforma, usando o SDK (*Software Development Kit*) e *frameworks* providos por ela (*Android* ou *iOS*). Os apps ficam vinculados a esse ambiente, executando apenas nos dispositivos da plataforma alvo. Para o desenvolvimento de uma aplicação nativa que possa ser executada em diferentes plataformas, faz-se necessária a criação de uma para cada plataforma, o que exige tempo, investimento e uma equipe com conhecimento nas variadas tecnologias usadas pelas plataformas alvo. Soluções *Cross-Platform* possibilitam a implementação de uma aplicação que pode ser executada em diferentes plataformas [El-Kassas et al. 2015].

Segundo a Taxonomia de Hansen et al. (2018), no desenvolvimento *Cross-Platform*, estão listadas as abordagens híbridas, interpretadas e cross-compilada. Todas essas trabalham com modelos de desenvolvimento diferentes. A abordagem híbrida permite o uso de tecnologias web regulares, tais como *HTML*, *CSS* e *JavaScript* combinadas com um componente de *WebView* nativo, que faz com que o código web seja exibido como um aplicativo. As ferramentas mais comuns para esse tipo de desenvolvimento são o *Phonegap*, *Ionic* e o *Sencha Touch*. A abordagem interpretada é semelhante a híbrida, mas difere pois não precisa do componente *WebView* para que o aplicativo possa ser executado. Isso porque os elementos dos aplicativos interpretados podem se tornar componentes de interface nativos, através de intérpretes *JavaScript* do próprio dispositivo. Dentre as ferramentas de abordagem interpretada destacam-se o *React Native*, *NativeScript* e *Adobe AIR*. Já no desenvolvimento *cross-compilado* a linguagem de criação dos aplicativos é compilada para código executável nativo. Ou seja, o aplicativo final se comporta como uma aplicação nativa, sem precisar de componentes *WebView* e de intérpretes *JavaScript* para sua execução. São exemplos de ferramentas *cross-compiladas* o *Flutter*, *Xamarin* e o *Corona* [Hansei et al. 2018].

Nos tópicos a seguir, estão contidas algumas informações sobre o desenvolvimento *Android* e a ferramenta *cross-platform* escolhida 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 um Sistema Operacional de código aberto desenvolvido e mantido pela *Google*. Ele possui uma arquitetura baseada em *Linux* para controlar os principais recursos relacionados a gerenciamento de memória, processos, automatização da rede e *drivers*. [Google Inc 2011]

Segundo Habsch (2012) a arquitetura do sistema operacional é composta por:

• *Applications*: composta pelos aplicativos nativos do Sistema Operacional *Android*.

• *Applications Frameworks:* composta pelo gerenciamento de *Activitys* e *views*, que podem ser manipuladas pelo desenvolvedores.

• *Runtime:* A camada responsável pela execução dos aplicativos.

• *Libraries:* Nesta camada encontramos diversas bibliotecas como a biblioteca C padrão, SQLite(Banco de Dados), OpenGL(Renderização 3D) etc.

• *Kernel:* Atua como uma camada de abstração entre hardware e as camadas superiores, permitindo acesso a recursos como áudio, vídeo e protocolos de rede.

A principal plataforma para o desenvolvimento dos aplicativos é o *Android Studio*, uma poderosa IDE(*Integrated Development Environment*) que auxilia na elaboração e construção dos apps. A linguagem de desenvolvimento é o Java ou Kotlin para a parte lógica do aplicativo, e XML para as demarcações de conteúdo de *layout.* A acessibilidade é tratada de maneira muito clara nessa plataforma. Existem guias e tutoriais a ser seguidos para tornar o aplicativo acessível [Google Inc 2011]. A Figura 2 apresenta a composição do sistema operacional *Android* como um todo.

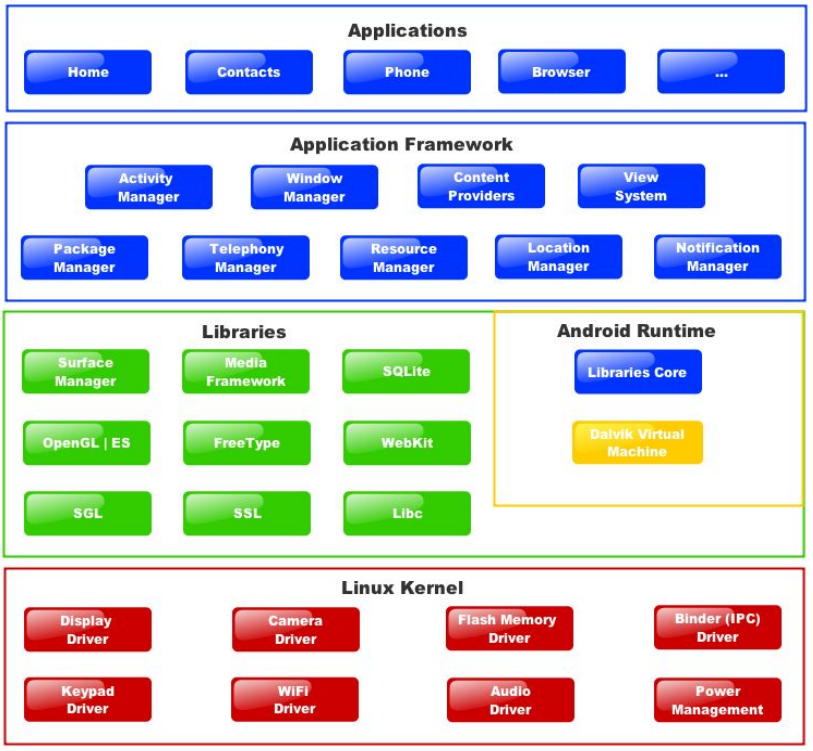


Figura 2 - Composição do Sistema Android

3.2. Ionic

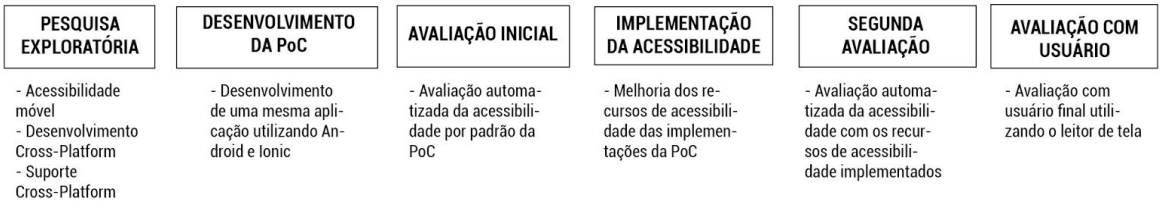
O *Ionic* é um *framework* de desenvolvimento de aplicações móveis que utiliza as linguagens Web (*HTML, CSS* e *JavaScript*). Ele utiliza ainda o *Angular*, que é uma biblioteca *JavaScript* desenvolvida pelo *Google*. Por utilizar tecnologias *web*, o aplicativo final se comporta como pequeno site que é executado em um componente *WebView* acoplado ao aplicativo. O acesso da camada nativa, como componentes de hardware e sensores é feito através de bibliotecas fornecidas pelo próprio *Ionic*. [Mendonça et al. 2011]

Segunda Lima (2019), uma das vantagens do *Ionic* é o fato do código ser reativo no momento do desenvolvimento. Isso possibilita acompanhar as mudanças em tempo real do aplicativo através de um navegador, por exemplo. No entanto, o aplicativo feito com *Ionic* apresenta um tempo de resposta ao clique maior comparado com o *Android* nativo. Isso inviabiliza sua utilização em apps de grande porte, que exijam uma maior performance. [Brito et al. 2018]

4. Metodologia

Na primeira parte deste trabalho, foi feita uma pesquisa exploratória, que segundo Prodanov e Freitas (2013), tem como finalidade proporcionar mais informações sobre o domínio investigado, possibilitando sua definição e seu delineamento. Desse modo, foram pesquisados conceitos sobre assuntos pertinentes à acessibilidade móvel e desenvolvimento *cross-platform*, bem como as ferramentas de desenvolvimento *Ionic*. A Figura 3 exibe as principais etapas dessa pesquisa.

Figura 3 - Resumo das etapas presentes na metodologia



Depois desse estudo exploratório, foi feito um estudo de caso baseado na Engenharia de Software Experimental. A Engenharia de Software Experimental é uma área de pesquisa cujo objetivo é evoluir o conhecimento em engenharia tradicional a partir da aplicação de abordagem científica (experimentação) na construção de novos métodos e técnicas para apoio ao desenvolvimento de software. Os estudos de caso, que fazem parte desse ramo da engenharia, visam observar um atributo específico e fazer uma comparação com atributos diferentes [Travassos; Gurov; Amaral 2002]. Neste trabalho, os atributos foram o desenvolvimento em Android nativo e *Cross-Platform* Ionic. Para a elaboração do estudo de caso, foi desenvolvida uma Prova de Conceito (PoC) de uma aplicação acessível. Para tal, as duas ferramentas escolhidas para estudo

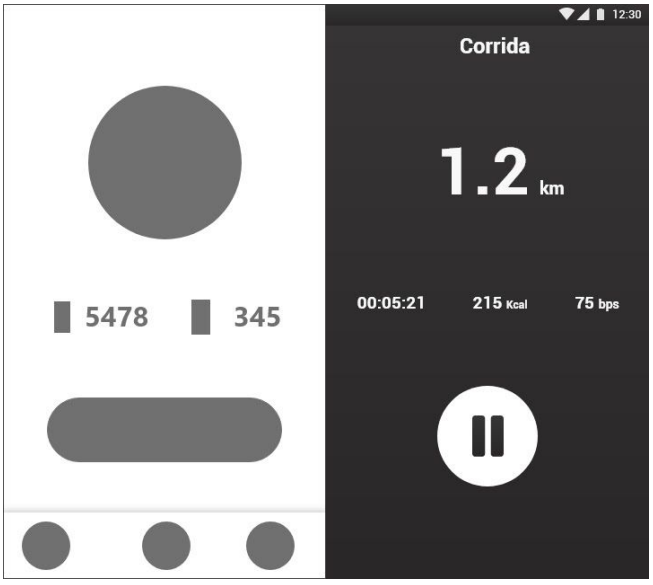
deste trabalho (*Android* e *Ionic*) foram utilizadas. Isto é, a PoC foi implementada em cada um das ferramentas.

A última etapa deste trabalho consistiu em avaliar a acessibilidade de cada uma das aplicações desenvolvidas. A primeira parte foi uma avaliação da acessibilidade antes de implementação dos recursos acessíveis. O objetivo foi observar como os aplicativos se comportam por padrão. Depois, foi feita uma nova avaliação com os recursos de acessibilidade já implementados, para poder analisar a acessibilidade dos recursos implementados.As ferramentas utilizadas nas avaliações 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.

5. Prova de Conceito

A aplicação trata-se de um aplicativo com a temática de fitness que 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.

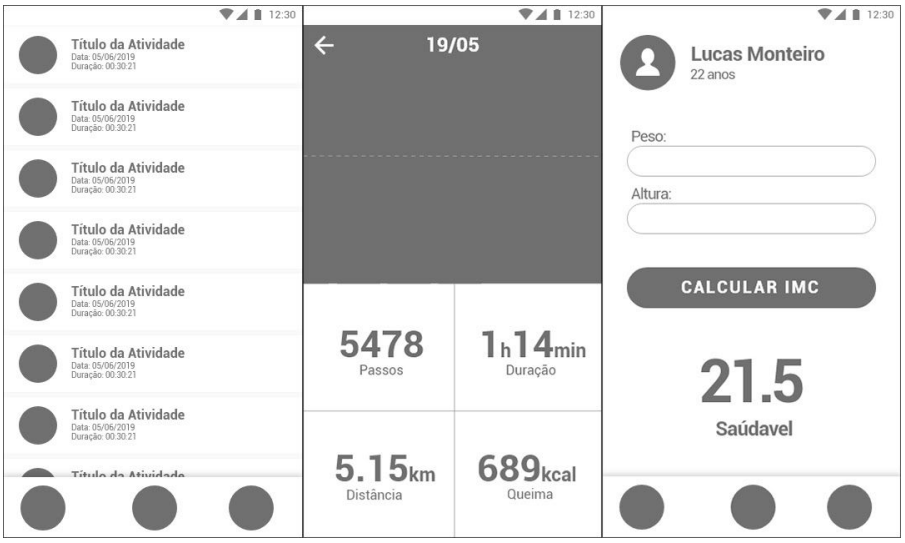
A Figura 4(a) ilustra a tela inicial do aplicativo, que contém informações do último treino. A navegação entre telas é feita pelo menu de três opções encontrado na parte inferior do aplicativo. Ao clicar no botão *começar,* localizado na tela inicial do aplicativo, será direcionado para a tela de exercício - Figura 4(b). Essa tela faz o monitoramento do treino em tempo real, permitindo acompanhar dados como tempo de exercício, gasto calórico e distância percorrida.



(a) T1- Tela Inicial (b) T2-Tela de Corrida

Figura 4 - Protótipo Aplicativo

A Figura 5(a) lista o histórico de exercícios realizados pelo usuário. Ao clicar em uma das opções da lista, o utilizador é direcionado para uma nova tela (Figura 5(b)) que contém os gráficos com as informações detalhadas sobre o exercício. Por último, a tela de perfil (Figura 5(c)) contém um formulário que permitirá ao usuário calcular o IMC (Índice de Massa Corporal).



(a) T3- Histórico de atividade (b) T4- Detalhes da atividade (c) T5- Perfil

Figura 5 - Protótipo do Aplicativo

6. Desenvolvimento das versões

6.1. Versão Android Nativa

A aplicação em *Android* nativo[7] foi desenvolvida utilizando o *Android Studio*, ferramenta padrão para o desenvolvimento de aplicativos do sistema operacional do Google. A linguagem de programação utilizada foi o Java com a API 28 do *Android*.

As três páginas 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,* que posiciona os elementos na tela com base no valor relativo ao *top* e *left* da tela ou dos elementos já posicionados. Os elementos visuais foram criados utilizando os *widgets* padrões do *Android*, como botões, *inputs* e label de texto. Depois de criados esses *widgets* foram personalizados de acordo com o visual que foi definido para o aplicativo.

[7] https://github.com/lucasmonteiro58/Android_TCC

O projeto final conta com 7 arquivos de layout XML e 3 classes *Java* principais (*MainActivity, CorridaActivity e GraficoActivity*). Existem outras classes secundárias que controlam as três páginas principais do aplicativo feitas com *Fragments*. Cada uma delas possui uma classe de *viewModel* que gerencia a visualização do aplicativo.

A Figura 6 representa o visual final da aplicação em *Android*, com todas features implementadas. Os testes de *layout* e programação das *features* do aplicativo foram feitos em um dispositivo android conectado ao computador via cabo USB.

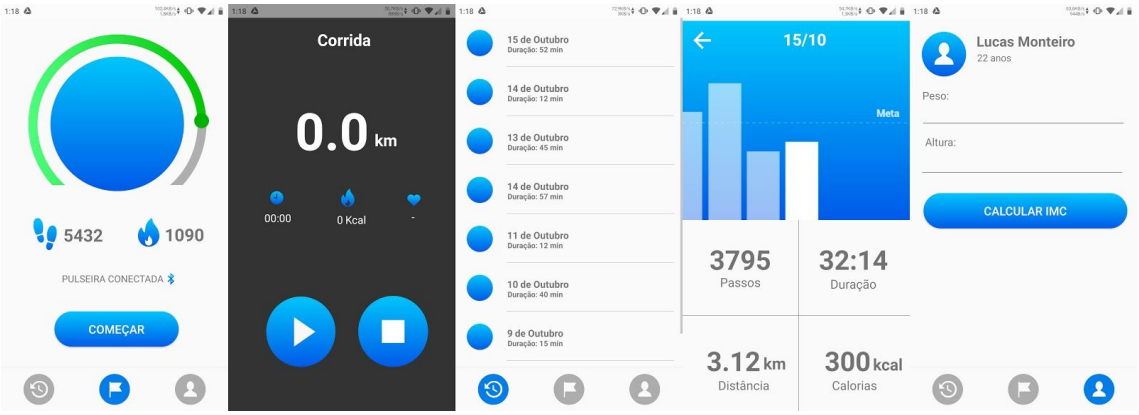


Figura 6 - Interface do aplicativo Android

6.2. Versão Ionic

A versão em *Ionic*[8] é desenvolvida baseada no *Angular*, um *framework* de desenvolvimento web. O projeto foi criado via comando CLI que o próprio *Ionic* disponibiliza. Esse comando gera todos os arquivos iniciais que o aplicativo precisa para funcionar. O desenvolvimento do código em si é baseado nas linguagens padrões da web (*HTML, CSS* e *javascript*), com as ferramentas que o angular oferece. A IDE (*Integrated Development Environment*) utilizada foi o *VScode*, pois esse possui plugins que facilitam a geração de códigos do próprio *Ionic* e *Angular*, bem como das linguagens padrões da web.

O comando inicial *ionic start FitApp tabs* gerou um template de navegação por *tabs* padrão do *Ionic* com o nome FitApp. A criação de novas página também foram feitas via comando CLI (*ionic generate page 'nome da página'),* que automatiza a geração de arquivos e configura as rotas de navegação. A Figura 7 mostra a estrutura que o projeto é organizado. Toda página contém os arquivos essenciais do projeto em ionic: um arquivo HTML, um arquivo CSS, e os arquivos *TypeScript* responsáveis pela lógica, modularização e rotas do *Angular*. A Figura 8 mostra o visual final da aplicação em *Ionic* com todas as features implementadas.

[8] https://github.com/lucasmonteiro58/Ionic_TCC

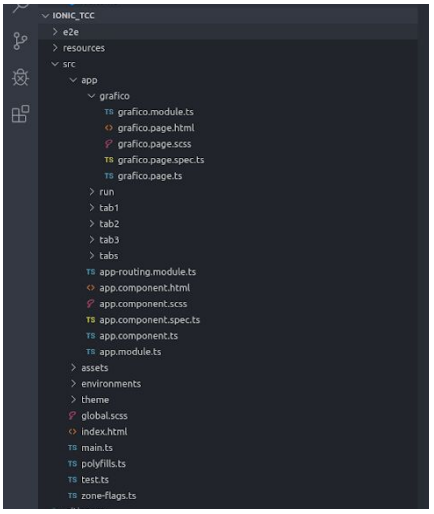


Figura 7 - Arquivos do projeto Ionic

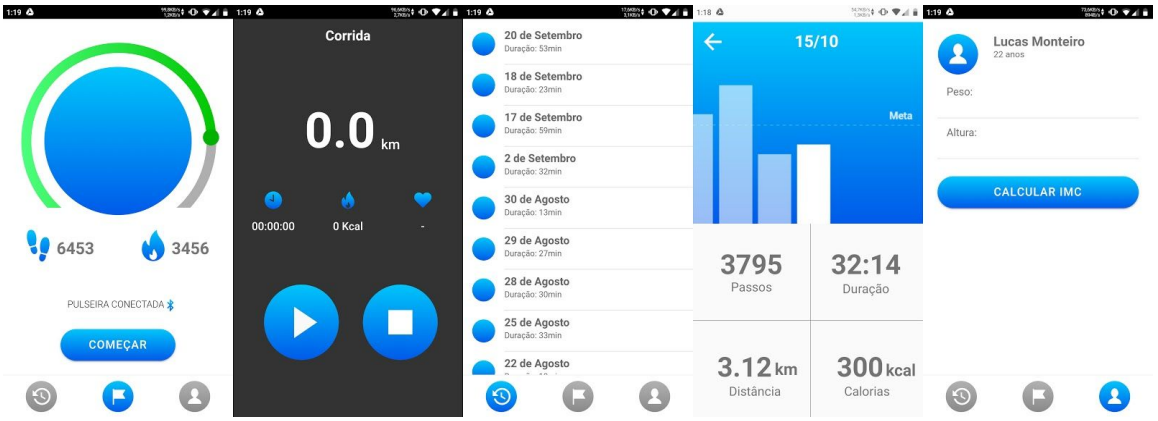


Figura 8 - Interface do aplicativo Ionic

O aplicativo pode ser testado a qualquer momento no navegador, através do comando *ionic serve,* muito útil na montagem do *layout*, não dependendo de um *smartphone* ou emulador para rodar o aplicativo. A compilação do aplicativo, com a geração do instalável para dispositivos *Android* foi feita através do comando *ionic cordova build*. A Tabela 2 lista algumas características do projeto de desenvolvimento do aplicativo na versão *Android* nativo e *Ionic*. O tamanho do aplicativo foi uma das maiores diferença nessa comparação.

Tabela 2 - Comparação entre as duas versões

	Quantidade de Arquivos	Tamanho do Projeto	Tamanho do APK	Tamanho após a instalação

Android	65	292 kb	2.62 MB	8.18 MB
Ionic	69	224 kb	10.06 MB	14,71 MB

7. Avaliação Inicial

Depois que os aplicativos foram desenvolvidos, foi feito um teste inicial utilizando as duas ferramentas já citadas (o *TalkBac*k e o *Scanner* de acessibilidade). O objetivo dessa avaliação era de verificar a acessibilidade das duas versões da PoC quando não há uma preocupação explícita do programador em implementá-la. Ou seja, a ideia era aferir como os componentes de interface disponíveis nas abordagens de desenvolvimento já tratam a acessibilidade e a integração com o leitor de tela.

7.1. Materiais e Métodos

Foram utilizadas como ferramentas avaliativas o leitor de tela *TalkBack* e o aplicativo *Scanner* de acessibilidade na versão 7.3 e 1.3, respectivamente. O aparelho usado para os testes foi um *smartphone* da fabricante Motorola, modelo XT1802 , com a versão 9 do *Android*.

7.2. Procedimento

Com o *TalkBack*, o proponente deste artigo 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. Já com o *Scanner* ne Acessibilidade, foi feita a verificação individual de cada tela no aplicativo. Cada checagem gerou uma lista de erros e aspectos que precisavam ser melhorados.

7.3. Resultados

7.3.1. Utilizando TalkBack

O aplicativo nativo fez a leitura dos textos corretamente, identificou os locais de toque bem como as caixas de edições de texto. Ele também saltou as imagens que não precisavam ser lidas. No entanto, alguns botões que eram representados só por 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.

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, já que todos os elementos ficam agrupados no principal. No entanto, é possível fazer a navegação linear (por *swipe*).

Outro erro foi que os elementos eram lidos em inglês, já que por padrão, a linguagem utilizada no *head* da página HTML era essa. O leitor de acessibilidade fez a leitura de todos os elementos na tela, não excluindo os elementos que tinham a intenção somente ilustrativas. As informações também eram lidas como se fosse uma página web, identificando qual tipo de elemento se tratava (gráfico, cabeçalho, por exemplo), o que não era desejado pois se tratava de um aplicativo. A Tabela 3 sintetiza a quantidade de erros em cada um das formas de desenvolvimento utilizando o leitor de tela *TalkBack*.

Tabela 3 - Quantidade de erros contabilizado 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
Android	8	0	5	12
Ionic	20	9	8	14

7.3.2. Utilizando o Scanner de acessibilidade

As Tabelas 4 e 5 mostram a quantidade de erros por tipo de cada tela após a verificação com o *Scanner* de acessibilidade. O Android apresentou 6 erros que estão concentrados basicamentes a alguns elementos visuais (botões imagens) não conter elementos textuais que possam ser lidos. Já o *Ionic* apresentou 8 erros que vão desde itens sem descrição à itens clicáveis ocupando o mesmo espaço na tela.

Tabela 4 - Erros 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	1 (imagem de progresso)	-	2 (campo de edição de texto)	2 (Botões Play e Stop)	1 (Botão de voltar)
Área de toque muito pequena	-	-	-	-	1 (Botão de voltar)

Tabela 5 - Erros identificados com o Scanner de Acessibilidade no Ionic

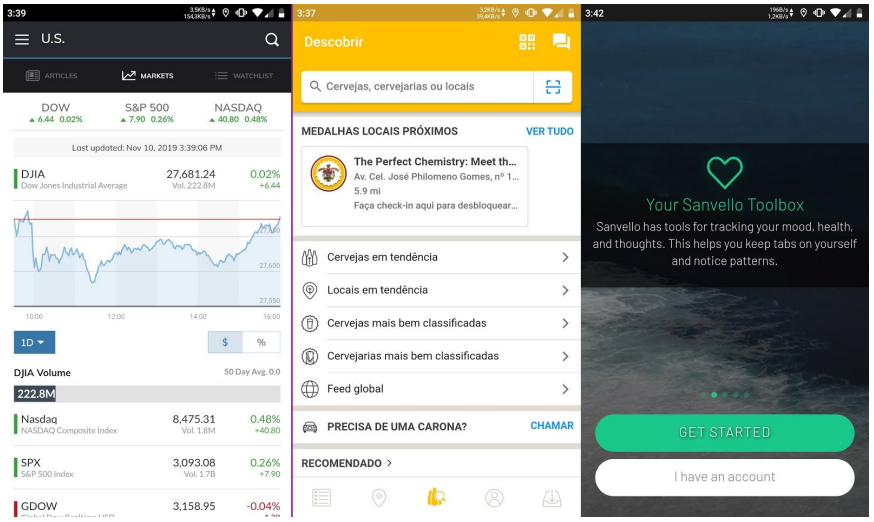
Erro	T1- Tela de Atividades	T2 - Tela de Histórico	T3 - Tela de Perfil	T4 - Tela de Corrida	T5 - Tela de Gráficos
Rótulo sem descrição	2 (Imagem de progresso e botões de navegação)	1 (Imagem da atividade)	2 (campo de edição de texto)	2 (Botões Play e Stop)	1 (Botão de voltar)
Área de toque muito pequena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

7.3. Testes com outros aplicativos Ionic

Um análise de três outros aplicativos desenvolvidos com o *Ionic* foi realizada com o intuito de 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[9] do *framework.* Os testes foram feitos usando o *Talkback* e *Scanner* de acessibilidade.

Os resultados foram semelhantes ao teste feito no aplicativo desenvolvido, apresentando o mesmo erro de agrupar todo o conteúdo em uma *webview*, não permitindo a navegação por toque. 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 aplicativos passaram por um processo de implementação da acessibilidade.

[9] Ionic ShowCase: http://showcase.ionicframework.com/apps/top



(a) Untappd (b) MarkerWatch (c) Pacifica

Figura 9 - Outros exemplos de aplicativos Ionic

8. Implementação das features de acessibilidade

8.1. Comportamento acessível esperado

A implementação da acessibilidade foi baseada nos requisitos do Guia de Acessibilidade Móvel, desenvolvido pelo SIDE (2015) . Em sua sessão de requisitos para o teste de acessibilidade, o guia lista uma série de parâmetros a ser adotados por designers e desenvolvedores. Para estudo deste artigo, foram selecionados os requisitos mandatórios das seções de interação e navegação listados na Tabela 6.

Tabela 6 - Requisitos mandatórios de acessibilidade

Requisito	Descrição
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.

8.2. Versão Android Nativa

O desenvolvimento da acessibilidade no *Android* utilizou as ferramentas que o sistema operacional oferece, que são explicadas no guia de acessibilidade disponível no site para desenvolvedores android para implementar os requisitos listados anteriormente.

O atributo *contentDescript* 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 (Figura 10).

```
<ImageView
    android:id="@+id/imageView6"
    android:layout_width="40dp"
    android:layout_height="50dp"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="0dp"
    android:layout_marginTop="0dp"
    app:srcCompat="@drawable/ic_fire"
    android:contentDescription="Calorias"/>
```

Figura 10 - Trecho do código do aplicativo Android

Elementos que precisavam ser ignorados no leitor de tela receberam o atributo *importantForAcessibiida='no'*. Desse modo, os elementos desnecessários não recebiam foco enquanto o leitor de tela estava ativado.

Alguns recursos precisaram ser implementados via código *Java*. Dentre eles está o feedback que resposta do cálculo de IMC do usuário, que precisa ser passado assim que clicado do botão de calcular. Para isso, foi utilizado o código

view.annouceForAcessibility('mensagem'). Por meio dessa configuração, é possível passar uma mensagem para o leitor de tela a qualquer momento da aplicação.

8.3. Versão Ionic

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 utilizando os recursos de acessibilidade do HTML. Para isso o aplicativo passou por uma remodelação nas *tags* HTML, organizando-as de acordo com o padrão de conteúdo recomendado na *W3school*[10].

Alguns atributos também foram usados para informar as demarcações de conteúdo. O ARIA (*Accessible Rich Internet Applications*) define definem as formas de tornar o conteúdo mais acessíveis às pessoas com deficiência. Por exemplo, a ARIA permite a marcação de regiões importantes na página (como uma caixa de busca, um cabeçalho, chamadas "pontos de referência"), para facilitar a navegação (agilizam a utilização de leitores de tela, por exemplo), *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 utilizado para ocultar determinado elemento do leitor de tela, equivalente ao *importantForAcessibilidade* do *Android.*

9. Segunda Avaliação

9.1. Materiais, Métodos e Procedimento

As ferramentas e os recursos utilizados foram o mesmo da primeira avaliação (*TalkBack, Scanner* de acessibilidade e dispositivo Motorola XT1802). Dessa vez, a avaliação foi feita 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 *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

[10] W3schools: https://www.w3schools.com/html/html_accessibility.asp

se comportava como um site, lendo os tipos de *tag* que o elemento estava encapsulado. No entanto, graças ao uso dos atributos do *ARIA*, os elementos continham a descrição necessária e foi possível ignorar os elementos desnecessários. A Tabela 7 mostra a quantidade de erros em relação aos requisitos listados na Tabela 6.

Tabela 7 - Quantidade de erros de acordo com os requisitos de acessibilidade

Requisito	Erros Android	Erros Ionic
R31 - Informar elementos visíveis	0	3
R32 - Informar o conteúdo assim que tocado	0	12
R33 - Feedback sonoro das ações	0	0
R34 - Feedback visual das ações	0	4
R35 - Link para tela principal	0	0
R36 - Botão de voltar	0	0
R39 - Navegação baseada em foco	0	12
R40 - Erros de interação	1	4

9.2.2 Utilizando o Scanner de acessibilidade

A versão em *Android* nativo apresentou resultado satisfatório de um aplicativo que atende os requisitos mínimos de acessibilidade. Por isso, o aplicativo *Scanner* de acessibilidade não identificou nenhum erro. O ionic apresentou o erro de vários itens clicáveis ocupando o mesmo espaço em todas as telas devido a *webview* se sobrepor aos elementos.

10. Avaliação com usuário final

O objetivo desta avaliação era coletar indícios da facilidade ou dificuldade de 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 da PoC.

10.1. Perfil do usuário

O usuário que participou da avaliação tinha 30 anos. Segundo ele, faz uso diário de celular e computador e tem certa familiaridade com tecnologia, já que é formado em Computação. Ele usa o celular com o auxílio de um leitor de tela, sendo *Whatsapp, Facebook* e Banco do Brasil os aplicativos mais utilizados.

Quanto ao grau da deficiência, ele 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 caminhada e dança.

10.2. Materiais e Métodos

O aparelho utilizado foi o mesmo dos testes anteriores (Motorola XT1802), com o leitor de tela *TalkBack*. O método aplicado para avaliar a satisfação, eficiência e a eficácia foi o SUS (*Scale Usability Score),* onde o usuário respondeu às seguintes perguntas sobre cada aplicativo com uma pontuação de 1 à 5, onde 1 ele discorda completamente e 5 concorda completamente [Brooke 2012]. Durante todo o teste foram feitas anotações por escrito 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 atrapalhado 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

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 ambientalizar na interface. A ordem de uso dos aplicativos foi

sorteada. Depois, foram passadas as seguintes tarefas para que o usuário pudesse realizar em cada versão do aplicativo.

Na tela Inicial:

- Identificar a quantidade de passos
- Ir para tela de corrida

Na Tela de corrida:

- Iniciar uma corrida
- Pausar uma corrida
- Parar uma corrida
- Ir para tela de histórico

Na Tela de histórico:

- Identificar uma atividade realizada
- Ir para tela de gráficos

Na Tela de gráfico:

- Identificar as informações do gŕafico
- Ir para tela de perfil

Na Tela de perfil:

- Calcular o IMC

Depois de realizar as tarefas, 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 pela implementação do suporte à acessibilidade.

10.4. Resultados

Para obter o resultado do questionário, foi feito o cálculo estabelecido pelo SUS. Nas perguntas ímpares é subtraído a pontuação que o usuário deu e as perguntas pares é diminuído a pontuação que o usuário atribuiu de 5. Depois o resultado é multiplicado por 2,5. O resultado máximo que pode ser obtido é 100. Por ser um método maduro, a comunidade de pesquisa conseguiu estabelecer índices de referência. Assim, a média do *System Usability Score* é 68 pontos. Para John Brooke (2012), se você fez menos pontos do que isso, você provavelmente está enfrentando problemas sérios de usabilidade em seu produto. A Tabela 9 contém as respostas do usuário para todas as perguntas do SUS em ambas as formas de desenvolvimento, bem como a pontuação final do SUS. O aplicativo *Android* conseguiu um total de 65 pontos, enquanto o Ionic fez 32,5.

Durante o teste, o usuário conseguiu realizar as tarefas com mais facilidade utilizando o aplicativo em *Android*, além de receber *feedbacks* das ações que realizava. Apesar de ter conseguido realizar todas as tarefas com o aplicativo *Ionic*, o usuário fez isso com muita dificuldade e depois de muitas tentativas. 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. Também é importante levar em consideração que o primeiro aplicativo testado foi o *Android*, então o usuário já tinha uma prévia de como o aplicativo era. Se *Ionic* tivesse sido o primeiro, provavelmente o usuário iria enfrentar bem mais dificuldades.

Tabela 9 - Respostas do usuário ao SUS

	Android	**Ionic**
Pergunta 1	4 (4 - 1 = 3)	2 (2 - 1 = 1)
Pergunta 2	3 (5 - 3 = 2)	4 (5 - 4 = 1)
Pergunta 3	3 (3 - 1 = 2)	3 (3 - 1 = 2)
Pergunta 4	1 (5 - 1 = 2)	2 (5 - 2 = 3)
Pergunta 5	4 (4 - 1 = 2)	1 (1 - 1 = 0)
Pergunta 6	2 (5 - 2 = 3)	5 (5 - 5 = 0)
Pergunta 7	4 (4 - 1 = 3)	2 (2 - 1 = 1)
Pergunta 8	3 (5 - 3 = 2)	5 (5 - 5 = 0)
Pergunta 9	4 (4 - 1 = 3)	3 (3 - 1 = 2)
Pergunta 10	1 (5 - 1 = 4)	2 (5 - 2 = 3)
TOTAL	26 x 2,5 = **65**	13 x 2,5 = **32,5**

11. Discussão

O resultado dos testes demonstram diferenças no suporte à acessibilidade nas duas formas de desenvolvimento. O sistema *Android,* por ser o padrão dos dispositivos apresentou uma melhor compatibilidade com a acessibilidade. Todas as features acessíveis puderam ser implementadas sem muito esforço, contando com diversos conteúdos na internet para auxiliar. Além disso, a própria fabricante do sistema operacional demonstra se importar com esse assunto, já que existe diversos guias para desenvolvedores deixarem o aplicativo mais acessível. A IDE de desenvolvimento (*Android Studio*), possui diversos atalhos para implementação do código acessível, sugerindo dicas e autocompletando códigos. A própria Google também desenvolveu o

Scanner de acessibilidade, ferramenta utilizada nos testes deste trabalho, que auxilia na correção do comportamento acessível do aplicativo.

Já o *Ionic* não demonstrou ser tão maduro quanto o desenvolvimento nativo em Android quando se trata da implementação de acessibilidade. Começando pelo conteúdo disponibilizado pela própria fabricante do *framework*, que não cita maneiras nem métodos que auxiliem a criar aplicativo acessível. No conteúdo da web, em geral, existem poucas páginas que tratam desse assunto, onde na maioria dos casos esse tema é abordado somente em discussões em fóruns de desenvolvimento.

A única forma encontrada para implementar o aplicativo acessível foi utilizando os recursos de acessibilidade padrões do HTML, já que o Ionic trabalha com essa linguagem. Com isso, foi possível adicionar a descrição que o leitor de tela poderia ler nos elementos. No entanto, os recursos implementados não se comportam como deveriam. Por utilizar os elementos padrões da web, o *TalkBack* leu o aplicativo como se fosse um site. Outro erro é que a *webview* utilizada para mostrar o aplicativo não é ignorada pelo leitor de tela e assume o papel de elementos principal do aplicativo, não permitindo que o aplicativo seja navegado por toque aleatório na tela (i.e., exploração da tela).

Nos testes antes da implementação da acessibilidade, o aplicativo Android se demonstrou mais coerente. O *Scanner* de acessibilidade identificou 51 erros de acessibilidade no aplicativo *Ionic,* enquanto no *Android* foram só 25. Depois da implementação da acessibilidade, o *Android* continuou se saindo melhor nos testes. O *Scanner* identificou 5 erros no aplicativo Ionic contra 0 do Android. Se baseando nos requisitos do Guia de Acessibilidade do SIDE (2015)**,** o aplicativo Ionic apresentou erros em todos os requisitos, com exceção ao R33. Já com o Android, foi identificado somente um erro no requisito R40, se tratando de um erro de interação que foi corrigido posteriormente. Ou seja, o aplicativo atendeu a todos os requisitos de acessibilidade estabelecidos aqui.

O teste dos aplicativos com o usuário também apontaram o desenvolvimento em *Android* como o mais satisfatório se tratando da sua usabilidade. O resultado do SUS apresentou uma pontuação de 65 para o *Android* contra 32,5 para o *Ionic*. O usuário ainda relatou que preferia utilizar o aplicativo em *Android* no dia a dia, pois com esse ele não teve dificuldade de utilização e também por ele descrever melhor os *feedbacks* de ações.

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 sistema operacional. Então, o suporte à acessibilidade no iOS não pode ser avaliado. As avaliações também levaram em consideração somente os itens obrigatórios do Guia de Acessibilidade do SIDE. Os

requisitos facultativos que ficaram de fora da avaliação poderiam trazer um resultado diferente para os testes. 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.

12. Conclusão

Esse trabalho abordou duas formas de desenvolvimento mobile distintas para o sistema operacional *Android.* O *cross-platform* tem se mostrado promissor nos últimos anos, no entanto ele ainda precisa de alguns ajustes O aspecto da acessibilidade aqui analisado apresentou resultado diferentes comparando as duas ferramentas (Nativo e *Ionic*). Foi implementado dois aplicativos, cada um utilizando uma forma de desenvolvimento. Durante o processo de implementação foi descrito os métodos para deixar o aplicativo acessível, abordando as dificuldades encontradas durante o processo.

Depois os aplicativos passaram por testes avaliativos para saber como o aplicativo se comportava antes e depois da implementação da acessibilidade. Em ambos os casos, o *Android* apresentou um número bem menor de erros que o Ionic. O teste com usuário também repetiu esse resultado, garantindo ao *Android* um comportamento acessível mais eficiente.

Como trabalhos futuros, pretende-se estudar outras formas de desenvolvimento *cross-platform* e verificar como a acessibilidade se comporta em cada uma. Também é desejável que os testes sejam feitos no *iOS*, outro sistema operacional que concentra um grande número de usuário. Nesse caso, a ferramenta a ser utilizada será o *VoiceOver*, leitor de tela padrão desse sistema operacional.

Referências

Bezerra, P. T. (2016). Desenvolvimento de aplicações mobile cross-platform utilizando phonegap. *Revista Observatorio de La Economía Latinoamericana*, 1(215):1– 30. Disponível em: `http://www.eumed.net/cursecon/ecolat/br/16/phonegap.html`. Acessado em: 18 set. 2018.

Brasil (2012). Secretaria de direitos humanos da presidência da república, secretaria nacional de promoção dos direitos da pessoa com deficiência. *Cartilha do censo 2010: pessoas com deficiência.*

Brooke, J. (2012). Sus - a quick and dirty usability scale. *Redhatch Consulting Ltd.*

Chantre, J. R. M. (2015). Testes automáticos de acessibilidade em aplicações móveis. *Curso de Engenharia de Informática, Universidade da Beira Interior, Covilhã*, 2015.

Disponível em: https://ubibliorum.ubi.pt/bitstream/10400.6/5775/1/4657_8856.pdf. Acesso em: 18 set. 2018.

Corral, L., Janes, A., and Remencius, T. (2012). Potential Advantages and Disadvantages of Multiplatform Development Frameworks: A Vision on Mobile Environments. *Procedia Computer Science: SciVerse ScienceDirec*t, Bolzano, v. 10, n. 1, p.736-743.

Mendonça, V. R. L., Bittar, T. J., and Souza Dias, M. (2011). Um estudo dos sistemas operacionais android e ios para o desenvolvimento de aplicativos.

El- Kassas, et al. (2017) Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*, [s.l.], v. 8, n. 2, p.163-190, jun. 2017.

Ferreira, C. M. S., peixoto, M. J. P., Duarte, P. A. S., Torres, A. B. B., Júnior, M. L. S, Rocha, L. S., and Viana, W. (2018). An Evaluation of Cross-Platform Frameworks for Multimedia Mobile Applications Development. *Revista Ieee América Latina,* v. 20, n. 4, abr. 2018.

Hansei, A. B., Groeli, T.-M., and Ghinea, G. (2018). A survey and taxonomy of core con- cepts and research challenges in cross-platform mobile development. *Acm Computing Surveys*, 1(1):1–31. Disponível em: https://bura.brunel.ac.uk/handle/2438/16569. Acessado em: 20 set. 2018.

Hubsch, E. (2012). Uma abordagem comparativa do desenvolvimento de aplicações para dispositivos móveis. *Faculdade de tecnologia de São Paulo.*

Nicholl, A. R. J. (2001). O ambiente que promove a inclusão: conceitos de acessibilidade e usabilidade. *Revista Assentamentos Humanos*, 3(2):49–60.

Pinheiro, J. M. S. (2010). Prova de conceito no projeto de redes de computadores. Disponível em: https://www.projetoderedes.com.br/artigos/artigo_prova_de_conceito_no_projeto_de_redes.php. Acessado em: 05 nov. 2018.

SIDE (2015). Guia para o desenvolvimento de aplicações móveis acessíveis. *Centro de Informática da Universidade Federal de Pernambuco*. Disponível em: http://http://www.sidi.org.br/guiadeacessibilidade/. Acessado em: 08 out. 2018.

Travassos, G. H., Gurov, D., and Amaral, E. A. G. (2002). Introdução à engenharia de software experimental. *Programa de Engenharia de Sistemas e Computação*. Disponível em:

http://cultura.ufpa.br/cdesouza/teaching/methods/6-ES-Experimental.pdf. Acessado em: 08 out. 2018.

Vargas, F. G. (2018). 29ª pesquisa anual do uso de ti. ed. São Paulo: *Fgv Eaesp,* 29. Disponível em: https://eaesp.fgv.br/sites/eaesp.fgv.br/files/pesti2018gvciappt.pdf. Acessado em: 15 set. 2018.