



UNIVERSIDADE FEDERAL DO CEARÁ
INSTITUTO UFC VIRTUAL
CURSO DE GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS

RONDINELLY CASTELO BRAGA

**DESENVOLVIMENTO NATIVO VS REACT NATIVE: UM ESTUDO DE
PORTABILIDADE DO JOGO HÍBRIDO COUP ACESSÍVEL**

FORTALEZA

2019

RONDINELLY CASTELO BRAGA

DESENVOLVIMENTO NATIVO VS REACT NATIVE: UM ESTUDO DE PORTABILIDADE
DO JOGO HÍBRIDO COUP ACESSÍVEL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto UFC Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Orientador: Prof. Dr. Windson Viana de Carvalho

FORTALEZA

2019

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)

B795d Braga, Rondinely Castelo.
Desenvolvimento nativo vs React Native : um estudo de portabilidade do jogo híbrido Coup Acessível /
Rondinely Castelo Braga. – 2019.
39 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual,
Curso de Sistemas e Mídias Digitais, Fortaleza, 2019.
Orientação: Prof. Dr. Windson Viana de Carvalho.

1. Acessibilidade. 2. React Native. 3. Cross-platform. I. Título.

CDD 302.23

À minha mãe, que sempre acreditou em mim e nos meus sonhos, me dando todo o apoio necessário para que eu chegasse até aqui.

AGRADECIMENTOS

À minha família que me deu todo o apoio necessário durante essa longa jornada na graduação.

Ao Prof. Dr. Windson Viana de Carvalho por me orientar durante este projeto, e por me apresentar a este e a outros projetos de acessibilidade, nos quais ele vem trabalhando com muita dedicação.

A Universidade Federal do Ceará, não só por toda a educação que me foi passada nas aulas e bolsas, mas também pelos amigos que fiz durante esse tempo.

RESUMO

Existem aproximadamente 528 mil pessoas incapazes de enxergar e 6 milhões de pessoas que possuem dificuldade permanente de enxergar no Brasil. A Internet e os sistemas de informação em geral são populadas por recursos com forte apelo visual. Acessibilidade digital na Web é uma tentativa de incluir essas pessoas e se define como uma série de recursos que possibilita a navegação, a compreensão e a interação de qualquer pessoa na web independentemente de suas dificuldades, sem ajuda de ninguém. Como uma decorrência desse movimento de prover a acessibilidade digital, surgiram também iniciativas pela acessibilização de jogos digitais. Essas iniciativas também incluem a criação de jogos específicos para esse público como uma forma de também inseri-los no universo fascinante dos jogos. Estes, exercem papéis que vão além da promoção da ludicidade com impactos na educação, relações sociais e reabilitação física. O desenvolvimento de jogos acessíveis, exige recursos (e.g., integração com leitor de tela, *Text-to-Speech*) que fazem com que os desenvolvedores optem pelo desenvolvimento nativo, o que acaba por restringir o alcance do público ao aplicativo. O objetivo deste trabalho é comparar o jogo híbrido “Coup Acessível” desenvolvido em Android Nativo com uma nova versão feita em *React Native*, mostrando assim a possibilidade do desenvolvimento de jogos acessíveis em ferramentas *Cross-Platform*, sem que ocorra perda significativa de performance ou aumento excessivo nos esforços de desenvolvimento. Para isso, foram realizadas e monitoradas algumas atividades nas duas versões do aplicativo. Embora o aplicativo em *React Native* tenha demonstrado uma menor performance se comparado à versão nativa, seu desenvolvimento é descomplicado e pode ser visto como uma boa alternativa de desenvolvimento *Cross-Platform* para aplicações acessíveis.

Palavras-chave: Acessibilidade. React Native. Cross-platform.

ABSTRACT

There are approximately 528,000 blind people in Brazil and about 6 million people with permanent visual impairment. The Internet and information systems are, in general, permeated by resources with strong visual appeal. Digital accessibility on the web is an attempt to include visually impaired people that is defined as a series of features that make anyone capable of browsing, understanding and interacting on the web (regardless of their difficulties) without anyone's help. As a result of this effort to provide digital accessibility, initiatives for making accessible digital games have emerged. These initiatives include creating games that target this audience as a way of inserting them into the fascinating universe of games. Games, today play roles that go beyond promoting playfulness, with impacts on education, social relationships and physical rehabilitation. Accessible game development currently requires features (e.g., screen reader integration, TTS) that makes developers opt for native development, which ends up restricting the access to the application. This paper aims to compare the "Coup Accessible" hybrid game developed on Android Native with a new version made in React Native, thus showing the possibility of developing accessible games on Cross-Platform tools, without significant loss of performance or excessive increase in development efforts. Therefore, some activities were performed and monitored in both versions of the application. Although the React Native application has shown lower performance compared to the native version, its development is uncomplicated and can be seen as a good cross-platform development alternative for accessible applications.

Keywords: Accessibility, React Native, *Cross-Platform*

LISTA DE FIGURAS

Figura 1 – Cartas e moedas do jogo coup	14
Figura 2 – Cartas do Jogo Híbrido	15
Figura 3 – Site de Documentação do React Native	17
Figura 4 – Criando novo projeto	19
Figura 5 – Instalando aplicativo	20
Figura 6 – Iniciando aplicativo	20
Figura 7 – Instalação do componente de TTS	21
Figura 8 – Utilização do componente de TTS	22
Figura 9 – Definindo idioma do TTS	22
Figura 10 – Utilização do componente de NFC	23
Figura 11 – Fluxo de telas	23
Figura 12 – Tela de Menu Principal	24
Figura 13 – Tela de Ler Informação das Cartas	24
Figura 14 – Tela de Ajuda	25
Figura 15 – App.js	26
Figura 16 – Diretórios do Android e do IOS	26
Figura 17 – TelaPrincipal.js	27
Figura 18 – TelaJogar.js	28
Figura 19 – TelaAjuda.js	29
Figura 20 – Cartas.js	29
Figura 21 – Variáveis utilizadas para diminuir tamanho do Apk	30
Figura 22 – Sugestões de melhoria de acessibilidade	31
Figura 23 – Resultados da Atividade 1	33
Figura 24 – Resultados da Atividade 2	35
Figura 25 – Resultados da Atividade 3	35
Figura 26 – Utilização máxima de memória por atividade	36
Figura 27 – Utilização máxima de CPU por atividade	37
Figura 28 – Utilização máxima de energia por atividade	37

LISTA DE ABREVIATURAS E SIGLAS

NFC *Near-field communication*

TTS *Text-to-Speech*

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Objetivos	12
<i>1.1.1</i>	<i>Objetivo Geral</i>	<i>12</i>
<i>1.1.2</i>	<i>Objetivos Específicos</i>	<i>12</i>
1.2	Organização do documento	12
2	FUNDAMENTAÇÃO TEÓRICA	13
2.1	Acessibilidade em Jogos	13
<i>2.1.1</i>	<i>Características necessárias</i>	<i>13</i>
<i>2.1.1.1</i>	<i>Terremoto de Áudio (Audio Quake)</i>	<i>13</i>
<i>2.1.1.2</i>	<i>Serialização</i>	<i>14</i>
<i>2.1.1.3</i>	<i>Ícones de áudio e sinais de áudio</i>	<i>14</i>
2.2	Jogo Híbrido Coup Acessível	14
<i>2.2.1</i>	<i>Coup</i>	<i>15</i>
<i>2.2.2</i>	<i>Funcionamento do jogo híbrido</i>	<i>15</i>
<i>2.2.3</i>	<i>Desenvolvimento Nativo vs Cross-Platform</i>	<i>16</i>
3	METODOLOGIA	18
3.1	Portando o Jogo para React Native	18
3.2	Análise comparativa da performance e qualidade do código	18
4	DESENVOLVIMENTO - COUP ACESSÍVEL EM <i>CROSS-PLATFORM</i>	19
4.1	Configurando o ambiente	19
<i>4.1.1</i>	<i>Criando um novo projeto</i>	<i>19</i>
<i>4.1.2</i>	<i>Executando o Projeto</i>	<i>20</i>
4.2	Primeiros passos no React Native	20
4.3	Bibliotecas	21
<i>4.3.1</i>	<i>Text To Speech</i>	<i>21</i>
<i>4.3.2</i>	<i>NFC Manager</i>	<i>22</i>
4.4	Interface	23
4.5	Código	25
<i>4.5.1</i>	<i>App.js</i>	<i>25</i>
<i>4.5.2</i>	<i>Tela de menu principal</i>	<i>27</i>

4.5.3	<i>Tela de ler informações das cartas</i>	28
4.5.4	<i>Tela de Ajuda</i>	28
4.5.5	<i>Cartas.js</i>	29
4.5.6	<i>Gerando o aplicativo</i>	30
5	TESTES	31
5.1	Análise comparativa do código e da Acessibilidade	31
5.2	Avaliação de Desempenho	32
5.2.1	<i>Materiais e Métodos</i>	32
5.2.2	<i>Atividades Realizadas</i>	32
5.2.3	<i>Procedimento</i>	32
5.2.4	<i>Resultados</i>	33
5.2.4.1	<i>Atividade 1 - “Iniciar o aplicativo”</i>	33
5.2.4.2	<i>Atividade 2 - “Ir para página de ler informações da carta, ler carta, clicar no botão nome e escutar nome da carta”</i>	34
5.2.4.3	<i>Atividade 3 - “Ir para página de ajuda, escutar informação de dois personagens e voltar para tela principal”</i>	34
5.2.4.4	<i>Organização dos Dados Obtidos</i>	36
5.3	Discussão	37
6	CONCLUSÃO	39
	REFERÊNCIAS	40

1 INTRODUÇÃO

Segundo o Ministério da Saúde (2017), existem aproximadamente 528 mil pessoas incapazes de enxergar e 6 milhões de pessoas que possuem dificuldade permanente de enxergar no Brasil.

Em um artigo do site Movimento Web para Todos (2019), acessibilidade digital na Web é definida como uma série de recursos que possibilita a navegação, a compreensão e a interação de qualquer pessoa na web (independentemente de suas dificuldades), sem ajuda de ninguém. Em outras palavras: uma internet acessível para todo mundo!

A acessibilidade digital também inclui o segmento de entretenimento. De fato, jogos digitais, se tornaram uma parte importante na cultura de crianças e jovens no mundo, em especial, nos países desenvolvidos e em desenvolvimento (ARCHAMBAULT *et al.*, 2007). E segundo Cheiran (2013), pessoas com deficiências visuais são reconhecidamente o público mais excluído da participação em jogos digitais, devido ao fato de que a maioria dos jogos fornece *feedback* por meio de recursos visuais. A busca pela acessibilização de jogos e a criação de jogos específicos para esse público é uma forma de também incluí-lo no universo fascinante dos jogos. Estes, hoje, exercem papéis que vão além da promoção da ludicidade com impactos na educação, relações sociais e reabilitação física.

Entretanto, o desenvolvimento de jogos acessíveis ainda é permeado de vários desafios, como por exemplo, o desenvolvimento de aplicações e jogos acessíveis em sua maioria, está limitado a plataformas nativas, devido ao fácil acesso a APIs dos sensores dos dispositivos, o que dificulta bastante o trabalho de quem quer desenvolver para mais de uma plataforma (e.g., Android e iOS). Outra barreira a ser superada no desenvolvimento de jogos acessíveis é que a maioria dos *feedbacks*, como citado anteriormente, são fornecidos por meio de recursos visuais. Para Silva e Santos (2014), a principal desvantagem de um aplicativo nativo está no fato de ser executado apenas na plataforma para a qual foi desenvolvido, o que faz com que o tempo, o custo e o esforço sejam bem maiores para disponibilizar um mesmo aplicativo para mais de uma plataforma.

Levando em conta a importância de jogos acessíveis e as limitações que os desenvolvedores encontram no desenvolvimento desses jogos, a questão de pesquisa deste TCC versa sobre a questão: como ampliar o número de pessoas com deficiência visual com acesso a jogos acessíveis sem que a aplicação tenha uma perda em desempenho (e.g., tempo de execução, memória ocupada, consumo de energia) e facilidade de seu uso?

1.1 Objetivos

1.1.1 Objetivo Geral

O principal objetivo deste trabalho é verificar se o desenvolvimento *Cross-Platform* pode contribuir para um maior alcance de usuários com deficiência visual aos jogos acessíveis sem que ocorra perda de qualidade e desempenho de hardware nos jogos.

1.1.2 Objetivos Específicos

Para isso, um jogo híbrido implementado em Android será portado para uma ferramenta *Cross-Platform* e uma comparação do seu desempenho será realizada.

1.2 Organização do documento

Este trabalho está organizado em 6 capítulos, incluindo este. No Capítulo 2, é apresentado o jogo objeto de estudo deste trabalho e é explicada a diferença entre aplicações nativas e *Cross-Platform*. No Capítulo 3, a metodologia utilizada no desenvolvimento deste projeto é discutida. O Capítulo 4 conta como ocorreu o desenvolvimento do projeto, desde a configuração do ambiente ao aplicativo concluído. No Capítulo 5, é demonstrado as fases de testes junto aos resultados obtidos. E conclui-se no Capítulo 6, com um breve resumo de todo o conteúdo apresentado, assim como trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo destina-se a apresentar e explicar os conceitos principais deste trabalho, além de apontar os livros e artigos que dão base a esta pesquisa. Neste Capítulo, é explicada a importância da acessibilidade nos jogos e as características necessárias para que um jogo seja acessível. Ainda neste estudo é apresentado o jogo híbrido Coup Acessível que será o objeto de estudo deste trabalho e serão explicadas as diferenças entre aplicações nativas e *Cross-Platform*.

2.1 Acessibilidade em Jogos

A necessidade por jogos acessíveis fica ainda mais nítida pelo fato de que os jogos auxiliam no desenvolvimento dos jovens. Segundo Archambault *et al.* (2007), o fato de pessoas não usarem os jogos por suas deficiências é lamentável por dois motivos. O primeiro motivo é que pessoas com deficiência são as que mais se beneficiam da tecnologia, utilizando-a para facilitar sua vida em diversas situações. O segundo motivo é que crianças com deficiência podem se beneficiar de jogos digitais, melhorando seu desenvolvimento cognitivo e psicomotor. Na pesquisa de Hellkvist (2017), por exemplo, é apontado que os sistemas operacionais Android e IOs são as melhores plataformas para pessoas com deficiências visuais e deixa claro seus benefícios para aprender o Braille por meio de jogos.

2.1.1 Características necessárias

Segundo Araújo *et al.* (2017), os feedbacks de áudio dos jogos, frequentemente não são o bastante para indicar toda a informação necessária para entender uma cena ou os possíveis caminhos de navegação. A principal característica de jogos acessíveis é o tratamento de rotinas e narrativa com recursos sonoros. Araújo *et al.* (2017), apresenta quais são as três técnicas mais utilizadas para este tratamento de rotinas e narrativas em jogos acessíveis

2.1.1.1 Terremoto de Áudio (*Audio Quake*)

Simula um radar e usa metáforas de som para indicar a posição de objetos móveis e fixos. Por exemplo, sons são emitidos da posição do inimigo, e ganham intensidade de acordo com a aproximação do inimigo.

2.1.1.2 Serialização

Essa técnica trata os sons com diferentes tipos de informação por níveis de prioridade em relação ao tempo de sonorização (e.g., som do inimigo, obstáculos).

2.1.1.3 Ícones de áudio e sinais de áudio

Propõe adicionar efeitos sonoros ou pistas sonoras para promover a identificação do objeto, ou para descrever ações no jogo. Por exemplo, é executada uma colisão e, alguns segundos é executado outro áudio indicando que há uma parede na frente do avatar do jogador.

2.2 Jogo Híbrido Coup Acessível

Figura 1 – Cartas e moedas do jogo coup



Fonte: Elaborado pelo autor (2019).

Albuquerque (2018), no projeto "Cartas, Acessibilidade e Diversão", desenvolveu o jogo móvel(híbrido) Coup para cegos¹, no qual usou tecnologias como *Near-field communication* (NFC), *Text-to-Speech* (TTS) e Interface Multimodal. Esse jogo é o objeto de estudo neste projeto, que foi portado do Android nativo para uma abordagem *Cross-Platform*.

¹ Premiado no III PRÊMIO LF DE COMPUTAÇÃO.

2.2.1 *Coup*

O jogo “Coup para cegos” é uma adaptação digital e acessível do jogo “Coup”. Albuquerque (2018) explica que Coup é um jogo de cartas para dois a seis jogadores de blefe e dedução social, com tempo de jogo de aproximadamente vinte minutos, originalmente lançado em 2012. O jogo foi escolhido o melhor jogo de cartas pelo prêmio Golden Geek em 2013, considerado ainda um jogo de baixa complexidade com jogabilidade focada na interação social entre os participantes, usando apenas quinze cartas e algumas moedas. Na Figura 1 são mostradas as cartas e moedas utilizadas no jogo.

2.2.2 *Funcionamento do jogo híbrido*

Figura 2 – Cartas do Jogo Híbrido



Fonte: Albuquerque (2018)

A infraestrutura computacional do jogo, explica Albuquerque (2018), é formada pelo projeto Tardigrade, um framework que possibilita o desenvolvimento de jogos de cartas ubíquas que apresentam interação do real com o virtual. A leitura das cartas é feita por meio de *tags* NFC etiquetadas em cada carta do jogo, e para os feedbacks em áudio é utilizada a ferramenta TalkBack do Android. A Figura 2 mostra as cartas criadas para o jogo híbrido coup acessível, plastificadas e já com uma tag NFC no interior de cada uma.

Cada carta possui um código registrado na tag NFC, que quando aproximada do celular realiza alguma ação. Se for a carta de ajuda, a tela de ajuda aparecerá. Se for carta de personagem, a tela ler informações da carta aparecerá, nessas duas telas o Jogador pode clicar

nos botões e receber as informações que deseja por áudio.

2.2.3 *Desenvolvimento Nativo vs Cross-Platform*

Silva e Santos (2014) em “Os Paradigmas de Desenvolvimento de Aplicativos para Aparelhos Celulares” apresentam as dificuldades encontradas no desenvolvimento de aplicativos móveis, e as opções que um desenvolvedor tem, acompanhadas de suas vantagens e desvantagens.

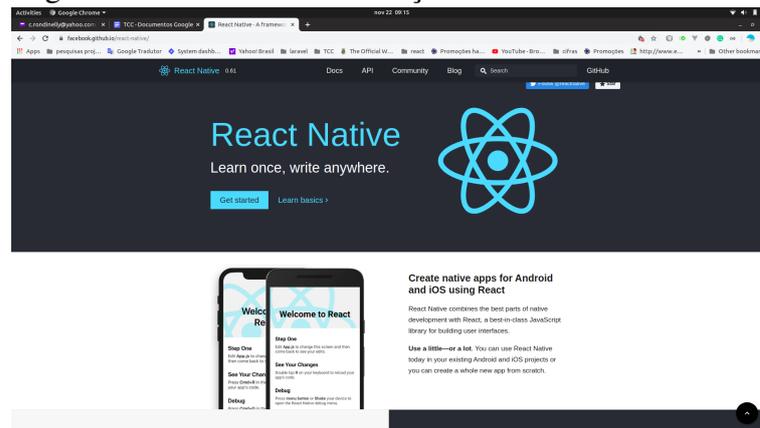
Majchrzak e Grønli (2017) explicam que um aplicativo, para ser executado no Android, no IOS, e possivelmente em outras plataformas como Windows Phone, pode ser desenvolvido de três formas, nativo (e.g., Android, IOs, Windows Phone, etc). No desenvolvimento Nativo o desenvolvedor tem mais facilidade no acesso a APIs de hardware do aparelho, porém para que o aplicativo seja executado em mais de uma plataforma, ele terá que ser desenvolvido duas vezes, duplicando assim o trabalho. No desenvolvimento em Web puro é utilizado HTML, CSS e Javascript, e pode ser executado em qualquer dispositivo com um navegador. No desenvolvimento *Cross-Platform* são utilizados frameworks para que o aplicativo seja desenvolvido uma vez e implantado nas plataformas desejadas. Um aplicativo *Cross-Platform* poderá ser executado em várias plataformas, porém encontra-se uma certa dificuldade em acessar as APIs do aparelho, fazendo com que desenvolvedores de jogos escolham o desenvolvimento nativo quando se trata de aplicações que necessitam de um melhor desempenho de hardware.

Quando se trata de jogos acessíveis para pessoas com deficiência visual, as dificuldades no desenvolvimento aumentam ainda mais, devido às necessidades especiais que esses usuários têm para interagir com o dispositivo móvel, o que faz necessário o uso de uma ferramenta que não só permita um só desenvolvimento para mais de uma plataforma, mas que também não deixe a desejar em recursos de acessibilidade.

Existem vários *frameworks* de desenvolvimento *Cross-Platform*. Optou-se neste estudo pelo React Native. Esta escolha foi pautada na pesquisa de Brito *et al.* (2018). Em sua pesquisa, os autores relatam que o React Native teve o melhor desempenho dentre as ferramentas estudadas (i.e., React Native, Ionic, NativeScript, Android, IOS) . O React Native teve nota superior em vários aspectos em relação às outras plataformas, incluindo “Aprendizagem e qualidade da documentação” e “Tempo de resposta e velocidade”, algumas vezes, mostrando nota maior até que o próprio desenvolvimento nativo.

Em outro trabalho, publicado por Calderaio (2017) no site Medium, o React Native também se sobressaiu. Comparando o desenvolvimento em React Native e o desenvolvimento em

Figura 3 – Site de Documentação do React Native



Fonte: Foto tirada do site de documentação do React Native (2019).

IOs nativo de uma aplicação simples, a abordagem *Cross-Platform* se sobressaiu nas categorias de utilização de GPU e de utilização de memória, apesar de o React Native ter tido desempenho inferior ao IOs nativo na categoria de utilização da CPU. O autor do estudo concluiu que “Agora estou mais convencido do que nunca de que o React Native é o framework do futuro - tem tantas vantagens, e tão poucas desvantagens” (CALDERAIO, 2017, tradução nossa).

O React Native é um *framework* de código aberto desenvolvido pelo Facebook, que diferente de outros *frameworks*, não utiliza HTML, mas sim gera o código nativo da(s) plataforma(s). Esse framework disponibiliza de um site, apresentado na Figura 3, com toda sua documentação e tutoriais de como configurar o ambiente e como utilizar suas APIs, além de variados exemplos.

3 METODOLOGIA

O presente estudo, segundo Gil (2002), é classificado como uma pesquisa exploratória quantitativa já que busca o aprimoramento de ideias para a resolução de um problema. Devido ao curto tempo para a finalização deste trabalho, foi utilizada o método ágil de desenvolvimento *SCRUM*, onde foram definidos cinco *sprints* para a concluir o projeto a tempo, configuração do ambiente, busca e teste de componentes de TTS, busca e teste de componentes de NFC, desenvolvimento da interface, e a unificação da interface com as bibliotecas.

3.1 Portando o Jogo para React Native

Como o objetivo deste estudo é fazer uma comparação entre os jogos acessíveis desenvolvidos em plataforma nativa e *Cross-Platform*, para identificar uma possibilidade de desenvolvimento para mais de uma plataforma com um menor esforço e aumentando assim um maior alcance de usuários, foi utilizado como objeto de estudo o jogo móvel “Coup para cegos”, que foi desenvolvido por Albuquerque (2018) somente na plataforma Android. Desta forma, foi necessário o desenvolvimento do mesmo jogo em React Native.

3.2 Análise comparativa da performance e qualidade do código

Com o jogo desenvolvido em React Native foi possível realizar uma comparação dos dados de desempenho de hardware e performance dos dois jogos no sistema operacional Android. A comparação se limita ao Sistema Operacional Android já que o jogo “Coup para cegos” foi desenvolvido somente em Android Nativo.

A comparação de dados de hardware é executada com a observação de alguns parâmetros, como utilização de memória, utilização de CPU, na execução de determinadas ações no jogo.

4 DESENVOLVIMENTO - COUP ACESSÍVEL EM *CROSS-PLATFORM*

Este capítulo destina-se a apresentar todas as fases de desenvolvimento do jogo híbrido Coup Acessível em React Native, assim como, todas as dificuldades encontradas.

4.1 Configurando o ambiente

Na documentação do React Native, são apresentadas duas formas de se utilizar este framework: usando o Expo CLI e o React Native CLI. O mais indicado para quem está iniciando no React Native é o Expo CLI pela facilidade de acesso a APIs desenvolvidas pelo expo.com e por não ser necessário a instalação de qualquer dependência nem configuração de código nativo, além de ainda permitir a implantação para ambas as plataformas sem que seja necessário utilizar o Xcode ou Android Studio.

Inicialmente o Expo CLI foi o escolhido para este projeto, porém, durante a fase de busca por bibliotecas que tornassem possível o desenvolvimento do Coup Acessível em React Native, constatou-se que o Expo ainda não dá suporte a tecnologia NFC. Com isso, o React Native CLI foi o utilizado neste trabalho.

4.1.1 Criando um novo projeto

Para iniciar o desenvolvimento em React Native, é necessário instalar o Node.js, o React Native CLI, o JDK e o Android Studio. Considerando que tudo já estava instalado e configurado corretamente seguindo as documentações do React Native, a única coisa necessária para criar o projeto é digitar o comando da Figura 4 no terminal, onde “coup-acessivel” representa o nome do projeto. Ao digitar esse comando, é criado um novo diretório onde ficam todos os arquivos do projeto.

Figura 4 – Criando novo projeto

```
react-native init coup-acessivel
```

Fonte: Elaborado pelo autor (2019).

4.1.2 Executando o Projeto

Para rodar o projeto é necessário navegar para dentro do diretório criado, conectar um celular em modo de depuração ao computador e digitar o comando apresentado na Figura 5 no terminal.

Figura 5 – Instalando aplicativo

```
'react-native run-android' - para Android  
'react-native run-ios' - para IOS
```

Fonte: Elaborado pelo autor (2019).

O comando da Figura 5 irá instalar o aplicativo no dispositivo. Porém, para que o aplicativo funcione, resta ainda executar o comando apresentado na Figura 6 para que o projeto seja executado no celular e qualquer mudança feita no código seja refletida no aplicativo em execução no dispositivo.

Figura 6 – Iniciando aplicativo

```
react-native start
```

Fonte: Elaborado pelo autor (2019).

4.2 Primeiros passos no React Native

A documentação do React Native é bastante completa e têm vários exemplos de como executar determinadas atividades (e.g., mostrar um texto na tela), além de explicar como o React Native funciona por trás do código. Porém, como o tempo para o desenvolvimento do projeto era curto, foi decidido estudar a tecnologia por meio de vídeo aulas, com o intuito de diminuir a curva de aprendizagem.

O curso escolhido para estudo foi o “The Complete React Native + Hooks Course [2019 Edition]” encontrado na Udemy.com e ministrado por Stephen Grider. A didática do curso

é muito boa e não demorou para que fossem desenvolvido exemplos simples, como navegação entre telas e estilização de elementos, isso ajudou no ganho de conforto ao trabalhar com o *framework*.

Não foi necessário completar o curso para dar início ao desenvolvimento já que as funcionalidades mais importantes, necessárias para este trabalho, teriam que ser encontradas em componentes externos, então foi dado início a fase de busca pelas bibliotecas que a aplicação necessitava.

4.3 Bibliotecas

As principais dificuldades enfrentadas no desenvolvimento deste trabalho foi encontrar bibliotecas que dessem suporte às tecnologias necessárias para o Coup Acessível em React Native. O processo de busca por essas bibliotecas foi realizado em duas etapas relativamente simples, pesquisa e teste.

Basicamente, o projeto necessita de duas tecnologias para que a replicação do aplicativo nativo seja feita com sucesso, o *TTS*, *Text-to-Speech* (texto para fala), para que o usuário receba todos os feedbacks por meio de áudio e a leitura de tags NFC. É importante dizer que todos os componentes utilizados são compatíveis com Android e IOS.

4.3.1 Text To Speech

A primeira biblioteca encontrada e testada para o projeto foi a ‘react-native-tts’ que adiciona a um projeto React Native a tecnologia *Text-to-Speech*, esse biblioteca se mostrou bastante simples tanto na instalação quanto em seu uso.

Figura 7 – Instalação do componente de TTS

```
npm install --save react-native-tts
react-native link react-native-tts
```

Fonte: Elaborado pelo autor (2019).

Para utilizar um componente externo no React Native, é necessário além da instalação do componente, vincular tal componente ao projeto, para isso foram utilizados os seguintes

comandos da Figura 7.

O uso desse componente, como citado anteriormente, é bem simples, basta importar o componente e utilizar com o método 'speak()', como é mostrado na Figura 8.

Figura 8 – Utilização do componente de TTS

```
import Tts from 'react-native-tts';  
Tts.speak('Texto a ser falado.');
```

Fonte: Elaborado pelo autor (2019).

Além do método 'speak()', deste componente também foi usado o 'setDefaultLanguage()', método que torna possível escolher a linguagem de fala. Para escolher o idioma português, foi utilizado o seguinte código apresentado na Figura 9.

Figura 9 – Definindo idioma do TTS

```
Tts.setDefaultLanguage('pt-BR');
```

Fonte: Elaborado pelo autor (2019).

4.3.2 *NFC Manager*

A tecnologia principal para o desenvolvimento deste trabalho é o NFC. E pode-se dizer que a maior dificuldade enfrentada neste projeto foi encontrar um componente com essa tecnologia que realmente funcionasse no React Native. Como citado anteriormente, foi devido a essa biblioteca que não foi possível que o projeto fosse desenvolvido com o Expo CLI, devido à falta de suporte a esse tipo de tecnologia pela expo.com.

O componente escolhido foi o 'react-native-nfc-manager'. Após instalação e importação desta biblioteca no projeto, sua utilização se dá com o código mostrado na Figura 10.

Na primeira linha da Figura 10, o NFC Manager é iniciado e faz com que a leitura do NFC seja esperada pelo aplicativo, e na segunda é definida a função que será chamada toda

Figura 10 – Utilização do componente de NFC

```

NfcManager.registerTagEvent();

NfcManager.setEventListener(
    NfcEvents.DiscoverTag, tag => {
        console.log('tag', tag);
    }
);

```

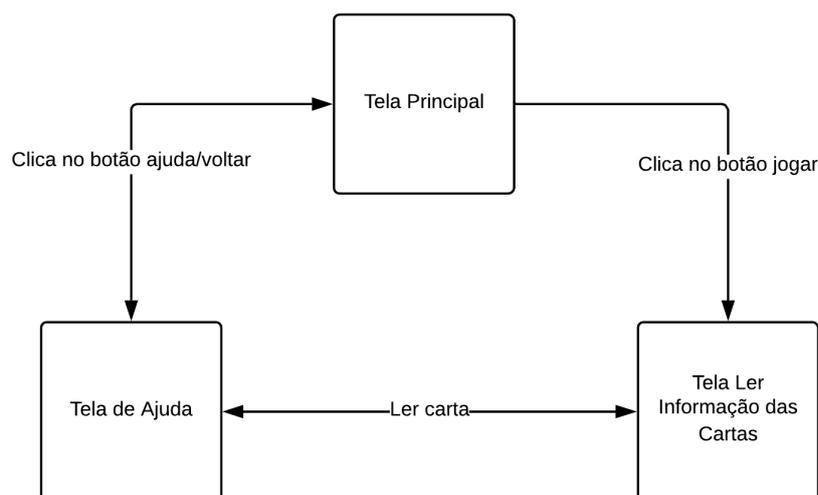
Fonte: Elaborado pelo autor (2019).

vez que uma tag NFC for lida. Na Figura 10, sempre que uma tag for lida, o conteúdo desta tag será impresso no terminal.

4.4 Interface

Assim como no Coup Acessível em Android Nativo, esse projeto também têm somente três telas, a tela de menu principal, a tela de ler informações das cartas e a tela de ajuda. Há uma navegação simples entre as telas do aplicativo, como é mostrado na Figura 11, por meio dos botões do aplicativo, botão de voltar do próprio dispositivo, ou mesmo ao ler uma carta NFC, que pode levar o usuário para a tela correta.

Figura 11 – Fluxo de telas



Fonte: Elaborado pelo autor (2019).

A primeira tela, *Tela de Menu Principal*, apresentada na Figura 12, se difere um pouco da tela do projeto original, dado que na primeira versão do aplicativo ele possui um botão

apagar dados, e na versão em React Native foi decidido que não seria necessário, pois isso não faz parte do jogo em si, mas sim da habilidade de leitura e gravação de dados de uma tag NFC.

Figura 12 – Tela de Menu Principal



Fonte: Foto tirada da nova versão do aplicativo (2019).

A *Tela Ler Informação das Cartas* é exatamente igual a da versão original do aplicativo. Nesta tela, como é mostrado na Figura 13, há três botões, o botão “Nome” que ao ser clicado, retorna o nome da carta selecionada por áudio, o botão “Descrição” que retorna uma breve descrição da carta, e o botão “Detalhes” que retorna outros detalhes da carta. Se pressionado qualquer um dos botões sem que nenhuma carta tenha sido lida, o áudio retornado será equivalente ao texto “Leia uma carta antes”.

Figura 13 – Tela de Ler Informação das Cartas



Fonte: Foto tirada da nova versão do aplicativo (2019).

A *Tela de Ajuda*, como mostrado na Figura 14, também está visualmente igual a sua versão original, possui seis botões, onde cinco deles estão com os nomes dos personagens, e um é o botão de voltar, todos do mesmo tamanho e com cores diferentes. Os botões dos personagens, ao serem clicados, retornam um áudio explicando sobre o personagem.

Figura 14 – Tela de Ajuda



Fonte: Foto tirada da nova versão do aplicativo (2019).

Um vídeo exibindo a execução do aplicativo está disponível no youtube.

<https://www.youtube.com/watch?v=XZqEGY93Lpg&feature=youtu.be>.

4.5 Código

Quando um novo projeto React Native é criado, é gerado um diretório raiz que contém, entre outros arquivos, o App.js e as pastas do Android e do IOS, onde ficam o projeto nativo de cada sistema operacional. O App.js recém criado contém elementos com algumas instruções do React Native, e são essas instruções que aparecem na tela quando se executa o projeto.

4.5.1 App.js

No App.js deste Projeto, foi configurado apenas o stackNavigator, que é o componente de navegação do próprio React Native, no qual foi necessário importar todas as telas do aplicativo, definir uma tela inicial e um nome de rota e título para cada uma, além de configurar o estilo do cabeçalho das páginas para ficar igual ao aplicativo original, com o texto branco e o fundo azul. A Figura 15 apresenta a estrutura deste arquivo.

Nas pastas “ios” e “android”, ficam os projetos nativos gerados de cada sistema, assim como mostra a Figura 16.

Figura 15 – App.js

```

1  import ...
6
7  const navigator = createStackNavigator(
8    routeConfigMap: {
9      Principal: {
10       screen: TelaPrincipal,
11       navigationOptions: {
12         title: 'Tela de Menu Principal',
13       },
14     },
15     Ajuda: {
16       screen: TelaAjuda,
17       navigationOptions: {
18         title: 'Tela de Ajuda',
19         headerLeft: null,
20       },
21     },
22     Jogar: {
23       screen: TelaJogar,
24       navigationOptions: {
25         title: 'Tela Ler Informação das cartas',
26         headerLeft: null,
27       },
28     },
29   },
30   stackConfig: {
31     initialRouteName: 'Principal',
32     defaultNavigationOptions: {
33       title: 'App',
34       headerStyle: {
35         backgroundColor: '#1957e0',
36       },
37       headerTintColor: 'white',
38     },
39   },
40 );
41
42 export default createAppContainer(navigator);
43

```

Fonte: Foto tirada do projeto React Native (2019).

Figura 16 – Diretórios do Android e do IOS

- ▼ android
 - ▶ .gradle
 - ▶ app
 - ▶ gradle
 - build.gradle
 - gradle.properties
 - ▶ gradlew
 - gradlew.bat
 - settings.gradle
- ▼ ios
 - ▶ coup
 - ▶ coup-tvOS
 - ▶ coup-tvOSTests
 - ▶ coup.xcodeproj
 - ▶ coupTests
 - Podfile

Fonte: Foto tirada do projeto React Native (2019).

4.5.2 Tela de menu principal

Como em todos os outros arquivos, é necessário importar tudo que se vai usar, nessa tela foi importado, além do React que é padrão, os elementos Text (tag de texto), StyleSheet (utilizado para adicionar estilo à interface), TouchableOpacity (tag utilizada como um botão, ao ser clicada, ocorre a ação pré-definida) e View (utilizado para que seja possível o uso de estilos e posicionamento de elementos usando flexbox). Além disso foi importado o componente de TTS e o método ‘escutarLeituraCarta’ do arquivo Carta.js que será explicado um pouco mais a frente. Veja como ficou o código desta tela na Figura 17.

Figura 17 – TelaPrincipal.js

```

1 import React from 'react';
2 import {Text, StyleSheet, TouchableOpacity, View} from 'react-native';
3 import Tts from 'react-native-tts';
4 import {escutarLeituraNfc} from '../helpers/Auxiliar';
5
6 class TelaPrincipal extends React.Component {
7   constructor(props) {
8     super(props);
9     Tts.setDefaultLanguage('pt-br');
10    Tts.speak('Tela de menu Principal');
11    escutarLeituraNfc(this.props.navigation);
12  }
13
14  render() {
15    return (
16      <View>
17        <TouchableOpacity
18          autoFocus={true}
19          style={styles.jogarBtn, styles.btnStyle}
20          onPress={() => this.props.navigation.navigate('Jogar')}>
21          <Text style={styles.textButton}>Jogar</Text>
22        </TouchableOpacity>
23        <TouchableOpacity
24          style={styles.ajudaBtn, styles.btnStyle}
25          onPress={() => this.props.navigation.navigate('Ajuda')}>
26          <Text style={styles.textButton}>Ajuda </Text>
27        </TouchableOpacity>
28      </View>
29    );
30  }
31 }
32
33 const styles = StyleSheet.create({...});
34
35 export default TelaPrincipal;

```

Fonte: Foto tirada do projeto React Native (2019).

Com o stackNavigator já definido no App.js, a navegação nesta e nas outras telas se dá por meio das propriedades que chegam para cada tela. Sempre que uma tela é aberta, há informações que chegam por meio do objeto props, este objeto contém o objeto navigation que é acessado como na linha 7 da Figura 17 e com esse objeto, podemos utilizar o método navigate que é o utilizado para navegar de uma tela a outra, basta utilizar este método como nas linhas 20 e 25 da Figura 17.

4.5.3 Tela de ler informações das cartas

Nesta tela, referenciada no código como TelaJogar, como se pode ver na Figura 18, além do que já foi importado na tela de Menu Principal, também foram importados, o NfcManager, que possui os métodos necessários para ler uma tag nfc e o Ndef-lib, necessário para converter o conteúdo da tag em string. Além disso, também foram importados mais alguns outros métodos do arquivo Carta.js.

Figura 18 – TelaJogar.js

```

1 import ...
14 class TelaJogar extends React.Component {
15   _willBlurSubscription;
16   carta = '';
17   constructor(props) {
18     super(props);
19     Tts.setDefaultLanguage('pt-br');
20     Tts.speak('Tela ler informações das Cartas ');
21
22     if (props.navigation.getParam('cod', 'NO-CODE') !== 'NO-CODE') {
23       this.carta = props.navigation.getParam('cod', 'NO-CODE');
24     } else {
25       Tts.speak('APROXIME O CELULAR DA CARTA QUE DESEJA LER');
26     }
27     NfcManager.registerTagEvent();
28     NfcManager.setEventListener(NfcEvents.DiscoverTag, tag => {
29       let message = Ndef.stringify(tag.ndefMessage);
30       message = message.slice(13, 16);
31       dizerCartaNome(message);
32       if (message !== 'AJ1') {
33         this.carta = message;
34       } else {
35         Tts.speak('Carta de ajuda, leia outra carta. ');
36       }
37     });
38   }
39   componentDidMount() {...}
40   render() {
41     return (
42       <View accessible={true}>
43         <TouchableOpacity
44           style={styles.nomeBtn, styles.btnStyle}
45           onPress={() => dizerCartaNome(this.carta)}>
46           <Text style={styles.textoBtn}>Nome</Text>
47         </TouchableOpacity>
48         <TouchableOpacity/>
49         <TouchableOpacity/>
50       </View>
51     );
52   }
53 }
54
55 const styles = StyleSheet.create({...});
56 export default TelaJogar;

```

Fonte: Foto tirada do projeto React Native (2019).

Nesta tela, usou-se uma nova função do objeto navigation, o “getParam()”, este método pega o parâmetro passado por meio do navigate e foi utilizado para caso o usuário esteja em outra tela e leia uma carta de personagem, o código da carta é enviado como parâmetro para esta página, sem a necessidade de ler uma carta novamente para ouvir suas informações.

4.5.4 Tela de Ajuda

Na Figura 19 é apresentado o código da Tela de Ajuda

Figura 19 – TelaAjuda.js

```

1  import ...
12
13  class TelaAjuda extends React.Component {
14    _willBlurSubscription;
15    constructor(props) {
16      super(props);
17      Tts.setDefaultLanguage('pt-br');
18      Tts.speak('Entenda a ação de cada carta');
19      Tts.speak('Tela de ajuda');
20      Tts.speak('Aproxime o celular da carta que deseja ler');
21      escutarLeituraNfc(this.props.navigation);
22    }
23    componentDidMount() {...}
24    render() {
25      return (
26        <View accessible={true}>
27          <TouchableOpacity
28            style={styles.btnStyle, styles.assassinoBtn}
29            onPress={() => dizerDetalhes( cod: 'As1')}>
30            <Text style={styles.textButton}>Assassino</Text>
31          </TouchableOpacity>
32          <TouchableOpacity
33            style={styles.btnStyle, styles.capitaoBtn}
34            onPress={() => dizerDetalhes( cod: 'Cp1')}>
35            <Text style={styles.textButton}>Capitão</Text>
36          </TouchableOpacity>
37          <TouchableOpacity/>
38          <TouchableOpacity/>
39          <TouchableOpacity/>
40          <TouchableOpacity/>
41        </View>
42      );
43    }
44  }
45
46  const styles = StyleSheet.create({...});
105
106  export default TelaAjuda;
107

```

Fonte: Foto tirada do projeto React Native (2019).

4.5.5 Cartas.js

Para evitar códigos duplicados nas páginas, e também por questão de organização, foi criado o arquivo ‘Carta.js’, mostrado na Figura 20. neste arquivo ficam todas as funções relacionadas as cartas do jogo, como “dizerNomeCarta()”, “dizerDescricaoCarta()”.

Figura 20 – Cartas.js

```

1  import Tts from 'react-native-tts';
2  import NfcManager, {NfcEvents} from 'react-native-nfc-manager';
3  import Ndef from '../..ndef-lib';
4
5  const CARTA_ASSASSINO = 'As1';
6  const CARTA_CONDESSA = 'Cd1';
7  const CARTA_CAPITAO = 'Cp1';
8  const CARTA_DUQUE = 'Dq1';
9  const CARTA_EMBAIXADOR = 'Em1';
10 const CARTA_AJUDA = 'Aj1';
11
12 export function dizerCartaNome(codigo) {...}
38 export function dizerDescricao(cod) {...}
65 export function dizerDetalhes(cod) {...}
96 export function escutarLeituraCarta(navigation) {...}
09

```

Fonte: Foto tirada do projeto React Native (2019).

4.5.6 Gerando o aplicativo

Um ponto que o React Native deixou muito a desejar foi o tamanho do apk gerado, enquanto o aplicativo do Android nativo tinha o tamanho de 2,5 MB, o desenvolvido em React Native na primeira vez em que foi gerado ficou com o tamanho de 23 MB. Após algumas pesquisas na tentativa de diminuir o tamanho do apk, foi encontrado o artigo “Shrink your React Native application size dramatically!”(KUMAR, 2018). Nele, os autores explicam como reduzir este tamanho ao máximo ao gerar o aplicativo para a plataforma Android, que é o foco deste trabalho, já que o projeto original está desenvolvido somente em Android.

Para reduzir o tamanho do aplicativo é necessário fazer algumas alterações no arquivo build.gradle que fica no no diretório ‘/Android/app’. Neste arquivo se encontram as duas linhas mostradas na Figura 21.

Figura 21 – Variáveis utilizadas para diminuir tamanho do Apk

```
def enableProguardInReleaseBuilds = false
def enableSeparateBuildPerCPUArchitecture = false
```

Fonte: Elaborado pelo autor (2019).

Tudo que se precisa fazer é mudar essas duas variáveis para true. Kumar (2018), explica que essa primeira variável faz com que o bytecode do java que é gerado durante a construção do aplicativo seja comprimido, além de tentar fazer com que todos os componentes utilizados no aplicativo também sejam reduzidos. Já essa segunda variável faz com que sejam gerados dois apks, referentes ao dois principais tipos de arquitetura de sistema que o Android suporta, o armeabi e o x86, o que não é nenhum incômodo pois os dois apks podem ser subidos para a playstore, que se encarrega de distribuir o apk certo para cada dispositivo.

Após essas alterações, foram gerados dois apks, cada um com menos de 8 MB, o que ainda é mais que o triplo do tamanho do apk do Android nativo, porém já não é um arquivo tão grande para baixar e nem ocuparia tanto espaço como o primeiro apk gerado de 23 MB.

5 TESTES

Neste Capítulo, são explicadas todas as formas de comparação e testes realizados entre as duas versões do aplicativo, assim como as ferramentas e métodos utilizados para obter os resultados apresentados.

5.1 Análise comparativa do código e da Acessibilidade

Utilizando o plugin Statistics para comparar o código das duas versões do aplicativo, notou-se uma grande diferença em relação ao tamanho do projeto, tanto em espaço de armazenamento como em linhas de código. Enquanto o tamanho projeto nativo é 51,3 MB com 274.925 linhas de código, o tamanho do projeto React Native é de 140,9 MB com um total de 2.257.603 linhas de código. Na versão do React Native, efetivamente foram escritas pelo proponente deste TCC 303 linhas.

Figura 22 – Sugestões de melhoria de acessibilidade



Fonte: Foto tirada do aplicativo Scanner de Acessibilidade ao scanear aplicativo coup acessível em React Native (2019).

A interface gráfica do Coup Acessível em React Native tenta ser o mais próximo possível do original desenvolvido em Android. E ao scanear os dois aplicativos com o scanner de acessibilidade da Google disponível na Playstore, as telas obtiveram o mesmo resultado. Como as cores dos elementos eram iguais, as melhorias que o Scanner de Acessibilidade sugeriu foi

somente em relação ao contraste do texto com a cor do botão, exatamente as mesmas sugestões que o aplicativo original teve ao ser escaneado. A Figura 22 mostra as sugestões de melhorias para as telas *Tela Principal*, *Tela Ler Informações das Cartas* e *Tela de Ajuda* nessa ordem.

5.2 Avaliação de Desempenho

Rieger e Majchrzak (2019) no artigo “Towards the definitive evaluation framework for Cross-Platform app development approaches”, explicam a dificuldade que os desenvolvedores encontram para avaliar aplicativos *Cross-Platform*, e propõem uma lista de critérios de avaliação. Neste trabalho avaliamos o aplicativo levando em conta os seguintes critérios: UI Design (Utilizando o aplicativo Scanner de Acessibilidade e verificando que as duas versões do aplicativo tiveram mesmos resultados), Performance (Utilização de memória, CPU e Energia em determinadas atividades) e Padrões de uso (Garantindo que os mesmos recursos de acessibilidade estivessem disponíveis na segunda versão do aplicativo).

5.2.1 Materiais e Métodos

Para os testes de performance foram utilizados: um notebook Dell Inspiron 15 7000 com Sistema Operacional Ubuntu, 16GB de RAM, processador i7 de 8^a geração e SSD de 128GB; um Moto Z3 Play com processador 1.8 GHz 8 Core e 4GB de memória RAM; e um cabo USB Tipo C. A ferramenta Android Profiler do Android Studio, também foi utilizada para fazer o monitoramento dos dados de utilização de memória, CPU e energia.

5.2.2 Atividades Realizadas

As atividades definidas para teste foram: “Iniciar o aplicativo”, “Ir para página de ler informações da carta, ler carta, clicar no botão nome e escutar nome da carta” e “Ir para página de ajuda, escutar informação de dois personagens e voltar para tela principal”. O monitoramento de todas as atividades ocorreu por 10 segundos, para que pudesse ser verificado também, como o aplicativo se comporta depois da atividade.

5.2.3 Procedimento

Para realizar o teste, assim como o projeto original do aplicativo, o projeto Android gerado pelo React Native foi aberto no Android studio afim de ser monitorado pelo Android

Profiler. Dessa forma, foi utilizada a mesma ferramenta para comparar de forma justa as duas aplicações.

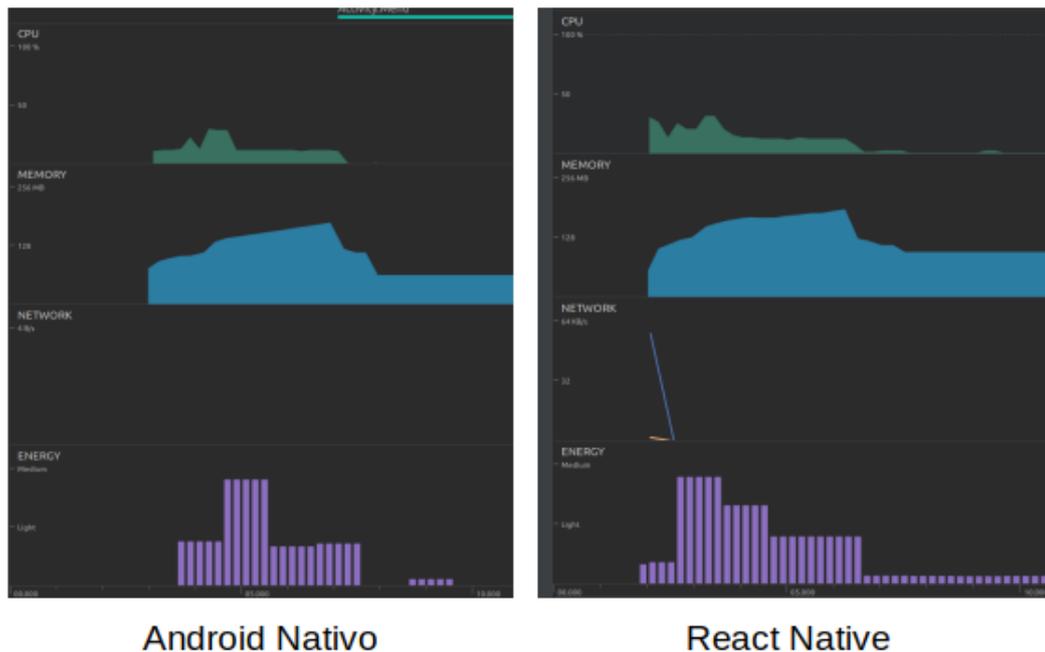
Com o dispositivo mobile configurado e conectado em modo de Depuração ao computador por meio de um cabo USB, o projeto foi iniciado pelo Android Studio (Similar ao comando “react-native run-android” do react native), assim tornando possível o monitoramento das informações de performance por meio da ferramenta Android Profiler.¹

5.2.4 Resultados

A seguir, são apresentados os resultados obtidos para cada atividade nos dois aplicativos, nativo e React Native.

5.2.4.1 Atividade 1 - “Iniciar o aplicativo”

Figura 23 – Resultados da Atividade 1



Fonte: Foto tirada da ferramenta Android Profiler durante a Atividade 1 (2019).

Ao Iniciar o aplicativo nativo, houve um pico de utilização de memória de 188,7 MB, após alguns segundo se estabilizou em 68,6 MB, já a versão do React Native iniciou com um pico de 200 MB e estabilizou em 82,8 MB. O aplicativo nativo teve utilização de 30,9% de CPU e voltou para 0%, enquanto a nova versão iniciou com 31% e depois ficou com uma alternância

¹ <https://developer.android.com/studio/profile/android-profiler>

entre 0 e 2% de utilização de CPU. A utilização de energia é medida no Android Profiler com 4 parâmetros, *None* (Nenhuma), *Light* (Leve), *Medium* (Médio) e *Heavy* (Pesado). As duas versões do aplicativo iniciam com uma utilização *Medium* de energia, porém enquanto o aplicativo nativo estabiliza-se em *None*, a nova versão fica alternando frequentemente entre *None* e *Light*. A saída dos testes do aplicativo Android e do aplicativo React Native encontra-se na Figura 23.

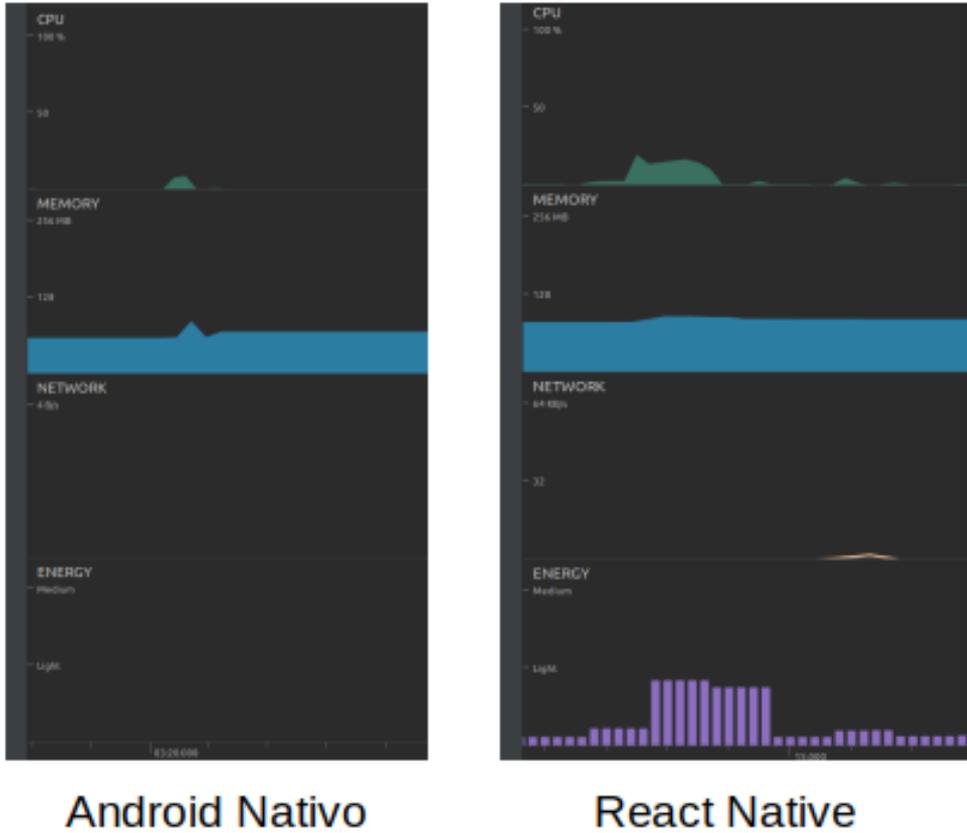
5.2.4.2 Atividade 2 - "Ir para página de ler informações da carta, ler carta, clicar no botão nome e escutar nome da carta"

Logo após o primeiro teste, ainda com as aplicações abertas, foi realizada a sequência de passos definida para a Atividade 2, onde houve na versão nativa, como mostra a Figura 24, um pico de utilização de memória de 90,7 MB e estabilizou-se em 72 MB, e na versão do React ocorreu um pico de 97 MB e estabilizou em 94 MB. Quanto a utilização de CPU, o aplicativo nativo teve 11% no início voltando para 0% logo depois, enquanto a nova versão iniciou com uma variação entre 17% e 15% e estabilizou-se em uma alternância constante entre 0% e 3% de utilização de CPU. A utilização de energia da versão em React Native se manteve em *Light*, enquanto a utilização da versão nativa varia um pouco em *Light* e volta para *None*.

5.2.4.3 Atividade 3 - "Ir para página de ajuda, escutar informação de dois personagens e voltar para tela principal"

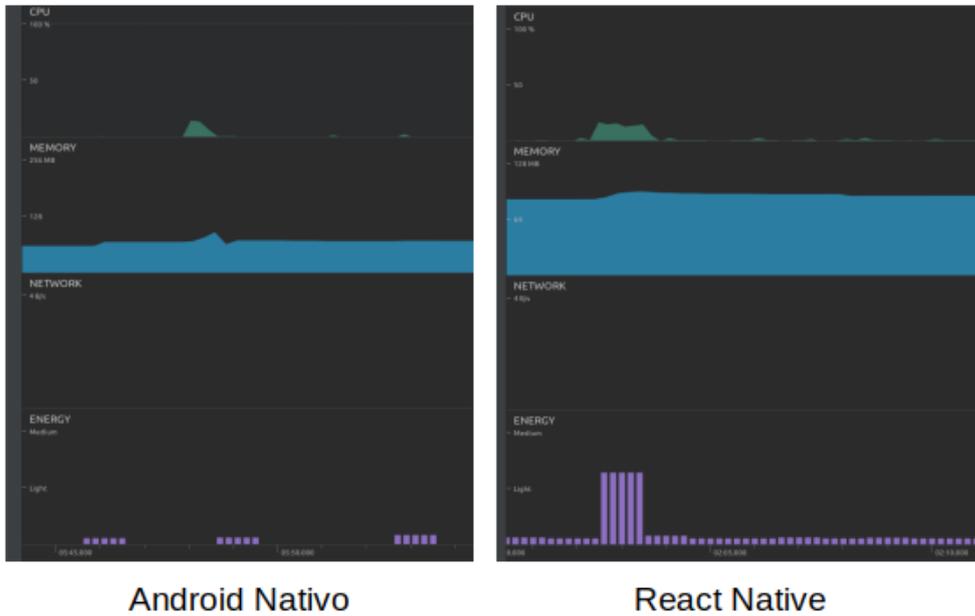
Logo após Atividade 2, sem fechar os aplicativos, foi iniciada a sequência de passos da Atividade 3. Assim como mostra a Figura 25, a utilização de memória do aplicativo em React Native subiu para 101 MB e logo desceu para 97 MB, enquanto a versão nativa do aplicativo, subiu para 99 MB e após alguns segundos se estabilizou em 79,8 MB. O aplicativo nativo teve utilização de 14% de CPU no início e se estabilizou em 0%, enquanto a nova versão iniciou com variação entre 17% e 15% e estabilizou em uma alternância entre 0% e 3% de utilização de CPU. A versão nova do aplicativo aumenta um pouco, mas ainda se mantendo *Light* e volta ao estado anterior em poucos segundos, possui leves variações entre *Light* e *None* sempre que o celular vibra com o toque, devido ao TalkBack. A versão nativa se mantém em *None* durante todos teste.

Figura 24 – Resultados da Atividade 2



Fonte: Foto tirada da ferramenta Android Profiler durante a Atividade 2 (2019).

Figura 25 – Resultados da Atividade 3

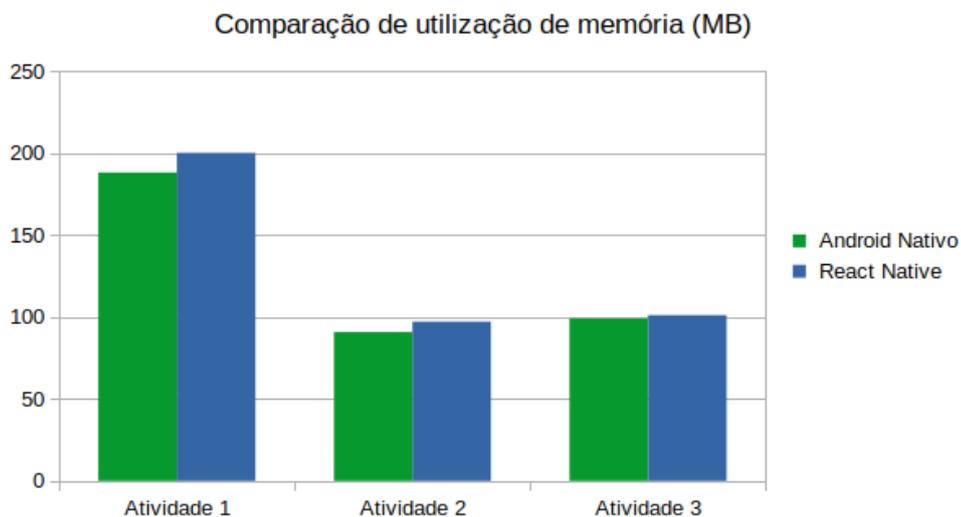


Fonte: Foto tirada da ferramenta Android Profiler durante a atividade 3 (2019).

5.2.4.4 Organização dos Dados Obtidos

Com o objetivo de facilitar o entendimento do resultado dos testes, foram gerados gráficos comparando a utilização máxima de cada recurso de performance monitorado em cada atividade. Na Figura 26 podemos ver o gráfico de comparação de memória utilizada. A Atividade 1 foi a que teve maior diferença quanto a utilização de memória, na segunda versão do aplicativo houve um aumento de 5,98%.

Figura 26 – Utilização máxima de memória por atividade

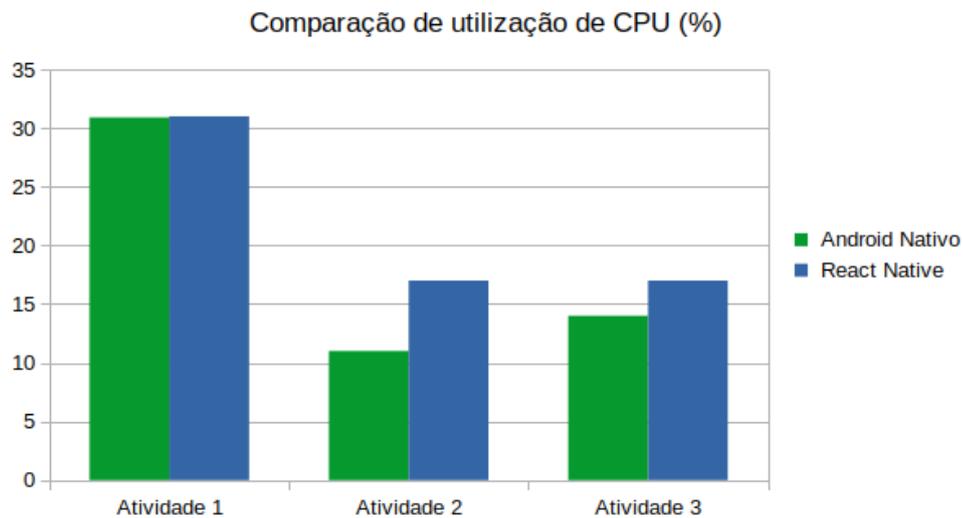


Fonte: Elaborado pelo autor (2019).

A comparação de utilização de CPU é apresentada na Figura 27, e assim como a utilização de memória, apresenta uma leve diferença entre as duas versões do aplicativo nas Atividades 1 e 3, já a Atividade 2 foi a que teve maior diferença entre as versões do aplicativo, mostrando que o aplicativo desenvolvido em React Native teve uma utilização de CPU 54,54% maior que o desenvolvido em Android Nativo.

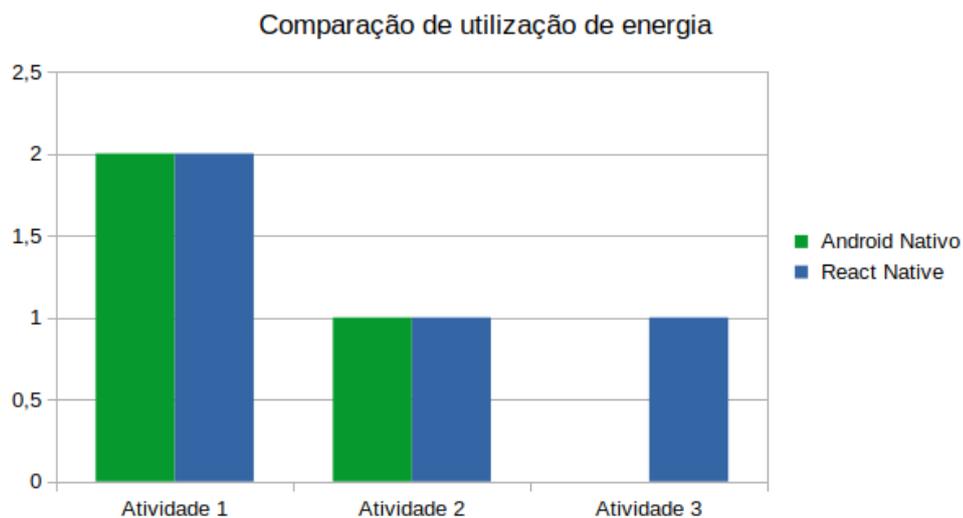
Na Figura 28, os parâmetros de monitoramento de energia do Android Profiler foram traduzidos em números, onde *None* é 0, *Light* é 1, *Medium* é 2 e *Heavy* é 4.

Figura 27 – Utilização máxima de CPU por atividade



Fonte: Elaborado pelo autor (2019).

Figura 28 – Utilização máxima de energia por atividade



Fonte: Elaborado pelo autor (2019).

5.3 Discussão

Percebe-se que há uma diferença de performance entre as duas versões do aplicativo. O aplicativo nativo tem melhores ou semelhantes resultados em todos os testes feitos. É importante salientar que o aplicativo React Native está tão acessível quanto a versão original do aplicativo. E também que, uma vez que, os componentes necessários foram encontrados e testados, o desenvolvimento foi bastante simples.

Vale ainda lembrar que, embora nos testes de performance, o aplicativo em React Native tenha tido desempenho inferior em praticamente todos os critérios avaliados, a diferença não é tão significativa e não chega a refletir na usabilidade do aplicativo. O que deixa ao

desenvolvedor a tomar a decisão, se vale a pena desenvolver somente uma vez com uma certa perda de performance ou desenvolver duas vezes ou mais dependendo do número de plataformas desejadas, mas com uma melhor performance.

6 CONCLUSÃO

Foi feita a portabilidade do Jogo Coup Acessível de Android para React Native, tornando possível a comparação de performance entre as duas plataformas. Os resultados obtidos embora em sua maioria tenham sido negativos para a versão em React Native, o aplicativo desta plataforma foi desenvolvido em poucos meses, sem muita complexidade e cumpriu com todos os recursos de acessibilidade utilizados no aplicativo original.

Mostrando assim que o React Native pode ser uma boa alternativa para o desenvolvimento de aplicações acessíveis. E como a comunidade deste *Framework* está cada vez maior e com mais recursos, é possível que futuramente o React Native possa ser usado até mesmo para o desenvolvimento de aplicações mais complexas.

Como trabalhos futuros, objetiva-se a implantação do aplicativo React Native na plataforma IOs e o monitoramento de sua performance. E para investigar mais a fundo os recursos de acessibilidade da plataforma React Native, pretende-se fazer o mesmo com um aplicativo mais complexo.

Link do projeto no BitBucket:

<https://bitbucket.org/rondinellycastelo/coup-react/src/master/>

REFERÊNCIAS

- ALBUQUERQUE, D. "**Cartas, Acessibilidade e Diversão**". 2018. Disponível em: <https://amauroboliveira.files.wordpress.com/2018/10/projeto-_cartas-acessibilidade-e-diversc3a3o_.pdf>.
- ARAÚJO, M.; FAÇANHA, A.; DARIN, T.; SÁNCHEZ, J.; ANDRADE, R.; VIANA, W. Mobile audio games accessibility evaluation for users who are blind. In: . [S.l.: s.n.], 2017. p. 242–259. ISBN 978-3-319-58702-8.
- ARCHAMBAULT, D.; OSSMANN, R.; GAUDY, T.; MIESENBERGER, K. Computer games and visually impaired people. **Upgrade**, 01 2007.
- Brito, H.; Gomes, A.; Bernardino, A. S. e. J. Javascript in mobile applications: React native vs ionic vs nativescript vs native development. In: **2018 13th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.: s.n.], 2018. p. 1–6. ISSN null.
- CALDERAIO, J. **Comparing the Performance between Native iOS (Swift) and React-Native**. 2017. Disponível em: <<https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2>>.
- CHEIRAN, J. F. P. **Jogos inclusivos : diretrizes de acessibilidade para jogos digitais**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, <https://lume.ufrgs.br/handle/10183/77230>, 3 2013.
- GIL, A. **Como Elaborar Projetos de Pesquisa**. [S.l.]: EDITORA ATLAS S.A., 2002. v. 4.
- HELLKVIST, M. **Braille Hero: Feedback modalities and their effectiveness on alphabetic braille learning**. Dissertação (Mestrado) — University of Skövde, School of Informatics., <http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Ahis%3Adiva-13864>, 2017.
- KUMAR, R. **Shrink your React Native application size dramatically!** 2018. Disponível em: <<https://medium.com/@rishii.kumar.chawda/reduce-your-react-native-app-size-dramatically-5430d773c92f>>.
- MAJCHRZAK, T. A.; GRØNLI, T.-M. Comprehensive analysis of innovative cross-platform app development frameworks. In: . [S.l.: s.n.], 2017.
- Ministério da Saúde. **Dia do Cego**. 2017. Disponível em: <<http://bvsmis.saude.gov.br/ultimas-noticias/2572-13-12-dia-do-cego>>.
- Movimento Web para Todos. **O que é acessibilidade digital?** 2019. Disponível em: <<https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2>>.
- RIEGER, C.; MAJCHRZAK, T. A. Towards the definitive evaluation framework for cross-platform app development approaches. **Journal of Systems and Software**, v. 153, p. 175 – 199, 2019. ISSN 0164-1212. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0164121219300743>>.
- SILVA, M. M. da; SANTOS, M. T. P. Os paradigmas de desenvolvimento de aplicativos para aparelhos celulares. **T.I.S.**, Fortaleza, v. 3, n. 2, p. 162–170, 2014.