MAURÍCIO VIEIRA DOS REIS

DESENVOLVIMENTO MULTIPLATAFORMA DE APLICATIVO MÓVEL UTILIZANDO O FRAMEWORK REACT NATIVE

LAVRAS – MG

2020

MAURÍCIO VIEIRA DOS REIS

DESENVOLVIMENTO MULTIPLATAFORMA DE APLICATIVO MÓVEL UTILIZANDO O FRAMEWORK REACT NATIVE

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Neumar Costa Malheiros
Orientador

LAVRAS – MG

2020

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Reis, Maurício Vieira dos

Desenvolvimento Multiplataforma de Aplicativo Móvel utilizando o framework React Native / Maurício Vieira dos Reis. 1^a ed. rev., atual. e ampl. – Lavras : UFLA, 2020.

41 p. : il.

Relatório de Estágio (graduação)–Universidade Federal de Lavras, 2020.

Orientador: Prof. Dr. Neumar Costa Malheiros .

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

MAURÍCIO VIEIRA DOS REIS

DESENVOLVIMENTO MULTIPLATAFORMA DE APLICATIVO MÓVEL UTILIZANDO O FRAMEWORK REACT NATIVE

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 08 de Setembro de 2020.

Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Danielle Lima Peixoto	ioasys

Prof. Dr. Neumar Costa Malheiros
Orientador

LAVRAS – MG
2020

Dedico este trabalho aos meus pais Ana e Afonso, que sempre estiveram ao meu lado, meu irmão Marcos, que sempre me motivou a continuar, meu irmão Fábio (in memoriam), que sempre esteve, está e estará comigo, e a todos os meus amigos.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me ajudado a enfrentar os obstáculos durante toda esta caminhada. Agradeço a minha família por me apoiarem e acreditarem em mim, sempre fazendo o possível e o impossível para que eu conseguisse concluir esta etapa. Agradeço a todos os meus amigos que, ao meu lado, tornaram essa jornada mais descontraída e principalmente por cada momento que passamos juntos, em especial, aos amigos das filiais I, II e III. Agradeço também a ioasys e Emakers Jr., pela oportunidade de trabalhar com pessoas incríveis que contribuíram para o meu crescimento pessoal e profissional. Por fim, agradeço ao meu orientador, Prof. Dr. Neumar Costa Malheiros, pela orientação, conhecimentos passados e por todo apoio e paciência ao longo da elaboração deste projeto. A todos vocês meu muito obrigado.

"O fracasso não é razão para você desistir, desde que continue acreditando."
(Naruto Uzumaki)

RESUMO

Com o crescimento incessante da internet, os aplicativos móveis estão ficando cada vez mais populares. Desenvolver aplicativos para dispositivos móveis pode ser uma tarefa bem complexa, pois existem diferentes sistemas operacionais e ambientes de desenvolvimentos, sem contar, os desafios presentes nesse tipo de desenvolvimento como, gerenciar os recursos limitados, heterogeneidade de dispositivos, experiência do usuário e entre outros. Com isso, torna-se necessário o desenvolvimento de aplicativos para diferentes plataformas e dispositivos móveis, afim de se alcançar um maior número de usuários. Existem abordagens e tecnologias de desenvolvimento que auxiliam o desenvolvedor na construção de aplicativos para diferentes plataformas. Um exemplo é o framework React Native que utiliza uma abordagem interpretada, na qual um mesmo código escrito em Javascript renderiza componentes nativos na tela em diferentes ambientes de execução. O objetivo deste trabalho foi o desenvolvimento de um aplicativo móvel, utilizando o framework React Native, para auxiliar os profissionais de contabilidade na prestação de seus serviços.

Palavras-chave: Desenvolvimento móvel. Abordagens multiplataformas. React Native.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis estão cada vez mais presentes no mundo inteiro. Conforme apresentado em (VEJA, 2019), cerca de 5,1 bilhão de pessoas no mundo inteiro possuem algum aparelho telefônico, o que equivale a 67% da população mundial. No Brasil, o número de *smartphones* quase se compara ao número de habitantes. De acordo com os dados apresentados em (INTERNET WORLD STATS, 2020), cerca de 62% da população possuem acesso à internet, isso mostra que o mundo está cada vez mais conectado e com isso, o uso de serviços disponíveis através da internet também cresce proporcionalmente.

Com esse crescente número de dispositivos móveis e o uso da internet, cresce também a procura por aplicativos que executem diferentes tarefas do dia a dia, desde jogos como passa tempo, até aplicativos que realizam movimentações financeiras. Disponibilizar serviços através de aplicativos para aparelhos de telefonia celular tende a ser um grande diferencial para o aumento do número de clientes, levando em consideração a agilidade e praticidade que o usuário ganha ao realizar o serviço pelo *smartphone*.

Para o aplicativo alcançar o maior número de usuários, ele precisa ser desenvolvido para diversos sistemas operacionais devido à heterogeneidade de dispositivos existentes. Existem diversas plataformas para dispositivos móveis, as empresas buscam alternativas eficientes para entregar aplicativos que possam funcionar nas diversas plataformas. Existem diversas abordagens no desenvolvimento multiplataforma, partindo de abordagens que precisam de códigos diferentes para produzir aplicativos em plataformas distintas, as abordagens que utilizam um mesmo código para gerar aplicativos para várias plataformas, e utilizam diferentes *frameworks* para auxiliar no desenvolvimento.

O objetivo principal deste trabalho foi o desenvolvimento de uma aplicação para dispositivos móveis para auxiliar os profissionais de contabilidade na prestação de serviços para seus clientes. Os objetivos específicos deste trabalho foram:

- Colocar em prática os conhecimentos adquiridos durante o período acadêmico;
- Entender os desafios comuns encontrados no desenvolvimento móvel;
- Expor as diferentes abordagens existentes de desenvolvimento multiplataforma;
- Implementar um aplicativo utilizando o framework React Native;

1.1 Ambiente de Trabalho

O estágio foi realizado na empresa ioasys fundada no ano de 2012 na cidade de Belo Horizonte, com filiais nas cidades de Lavras - MG, São Paulo - SP e Aracaju - SE. Atualmente, a empresa possui cerca de 150 colaboradores, divididos em todas as unidades da empresa, e tem como principal objetivo ajudar marcas a inovar através da transformação digital, princípios de metodologias ágeis e abordagem centrada no usuário.

Em 2017, a ioasys foi contemplada pela *Britsh Brazilian Awards* com um prêmio de reconhecimento a empresas que estão se destacando no mercado britânico. A empresa foi responsável pelo *Startup Games*, um jogo de investimento estratégico entre startups.

A ioasys atua em diversas áreas, entre as quais pode-se destacar: *design thinking*, *design sprint*, *UI/UX*, *mobile development*, *web development*, *machine learning*, entre outros. Desde o início da empresa, o foco principal foi em transformação digital, tecnologia e desenvolvimento. Atualmente, a empresa possui diversos projetos em conjunto com empresas como o Banco Inter, Burguer King, LATAM Airlines, Pfizer, Fleury, Localiza e outras.

1.2 Estrutura do documento

Este relatório está organizado como descrito a seguir. No Capítulo 2, serão explicados aspectos teóricos como: desafios presentes no desenvolvimento móvel, diferentes abordagens para o desenvolvimento multiplataforma, e o framework React Native, sua arquitetura e principais características. No Capítulo 3, serão abordadas as atividades desenvolvidas no projeto durante o período do estágio. Por fim, no Capítulo 4, serão apresentadas as considerações finais a respeito do projeto e do framework utilizado, bem como as contribuições desse estágio em termos de crescimento profissional.

2 REFERENCIAL TEÓRICO

Neste capítulo, estão descritos os desafios presentes no desenvolvimento de aplicações móveis, abordagens multiplataformas utilizadas para o desenvolvimento e uma visão geral sobre o framework React Native para desenvolvimento de aplicativos móveis.

2.1 Desafios em Desenvolvimento para Dispositivos Móveis

Com a crescente gama de dispositivos móveis que estão chegando ao mercado nos últimos anos, há uma grande variedade de tecnologias, plataformas e equipamentos. Apesar de muitos avanços, os desafios no desenvolvimento de aplicações móveis ainda estão presentes. Tanto na fase de projeto, quanto no decorrer do desenvolvimento, diversas questões devem ser levadas em conta para que os aplicativos alcancem o maior número possível de usuários.

Desenvolver aplicações para dispositivos móveis é uma missão bem complexa. O que torna as coisas ainda mais complicadas são as diferenças entre os kits de desenvolvimento de software (SDK) das plataformas como iOS e Android. Para cada plataforma, existem diferentes ferramentas, sistemas de compilação, interface de programação de aplicações (APIs) e dispositivos com diferentes recursos. De fato, a única coisa que os sistemas operacionais (SO's) dessas plataformas possuem em comum é que, em todos os dispositivos, é possível acessar o navegador web a partir do código nativo (CHARLAND; LEROUX, 2011).

Desde a escolha do dispositivo no qual o aplicativo será executado, até os recursos que serão utilizados, devem ser pensados e analisados, para que o usuário tenha navegabilidade simples e que não tenha os recursos disponíveis em seu aparelho totalmente consumidos.

De acordo com El-Kassas et al. (2017), dentre as particularidades e/ou restrições no desenvolvimento mobile, destacam-se os recursos limitados, heterogeneidade de plataforma e dispositivos, experiência do usuário e manutenibilidade, que serão explicadas a seguir:

2.1.1 Recursos Limitados

Por mais que os dispositivos móveis estão em crescente evolução, os recursos não são ilimitados. Existem várias limitações que podem comprometer o perfeito funcionamento da aplicação. O primeiro ponto importante sobre os recursos é a capacidade limitada de processamento ou poder computacional presente nos aparelhos. Mesmo se os dispositivos estão mais poderosos do que antes, eles ainda possuem limitações na memória, processamento, etc. Apesar

dos aplicativos não serem feitos com o principal propósito de processamento, essa questão pode ser um gargalo para diversas tarefas que precisam ser executadas pelo dispositivo.

Outro ponto bastante importante é o espaço de armazenamento disponível. Segundo apresentado em (EXAME, 2019), até o fim de 2019, a média de armazenamento dos aparelhos quase dobrou em relação a 2017. Porém, isso não significa que não seja mais necessário se preocupar, pois, com o aumento da memória, o usuário também pode querer instalar mais aplicações do que o habitual (UOL, 2017). Conforme apresentado em (GOOGLE, 2020), os aplicativos podem armazenar diversas informações como, preferências compartilhadas, dados privados ou públicos da aplicação, e banco de dados.

A conectividade também é um recurso importante em diversas aplicações, portanto, a disponibilidade da rede de acesso é essencial. Acesso intermitente pode comprometer o funcionamento correto e eficiente de alguns aplicativos. Desenvolver uma aplicação que precise de conectividade à internet 100% do tempo de uso é um grande desafio. Segundo apresentado em (GOTODATA, 2020), até final de 2021, serão cerca de 25 bilhões dispositivos conectados à internet em todo o mundo, o que ainda é um número pequeno considerado a população mundial.

2.1.2 Heterogeneidade de Plataformas e Equipamentos

Devido à existência de diversos sistemas operacionais, faz-se necessário um conhecimento maior sobre as ferramentas de desenvolvimento de aplicações. Sendo assim, o desenvolvedor precisará conhecer diferentes sistemas operacionais e SDK's, como aqueles exemplificados no Quadro 2.1. Além disso, pode-se utilizar diferentes linguagens de programação, caso seja necessário implementar uma aplicação que possa ser executada em diversas plataformas, a fim de alcançar um número maior de usuários. Isso pode ser tornar um problema, ao passo de ser necessário implementar novamente a aplicação, pois cada plataforma possui um *setup* diferente.

Com o grande número de dispositivos presentes no mercado, torna-se necessário diferentes versões da aplicação. Recursos necessários podem não estar disponíveis em alguns aparelhos, ora por versão do sistema operacional, ora por recursos suportados. Essas peculiaridades se tornam um grande desafio no desenvolvimento mobile.

O Quadro 2.2 apresenta uma previsão participação de mercado em relação aos sistemas operacionais no período de 2019 à 2022. Pode-se observar que os sistemas operacionais Android e iOS já dominam todo o mercado de plataformas para *smartphones*.

Quadro 2.1 – Diferenças entre plataformas mobiles de um ponto de vista do desenvolvimento

SO	Linguagem de Programação	Ambiente de Desenvolvimento	Loja de distribuição
Android	Java	Android Studio, Android SDK	Google Play
iOS	Objective-C/Swift	XCode	Apple Store
Windows Phone	Visual C#, C++	Visual Studio	Microsoft Store
RIM BlackBerry OS	Java	BlackBerry Plug-in para Eclipse	BlackBerry Apps World

Fonte: Adaptado de Latif et al. (2016)

Quadro 2.2 – Previsão de participação de mercado do sistema operacional para smartphones

Ano	2019	2020	2021	2022
Android	86,1%	85,4%	86,0%	86,2%
iOS	13,9%	14,6%	14,0%	13,8%
Outros	0,0%	0,0%	0,0%	0,0%
Total	100%	100%	100%	100%

Fonte: Adaptado de IDC Corporate USA (2020)

2.1.3 Experiência do Usuário

A experiência do usuário é um fator de grande importância. Com o desafio de fazer o melhor uso possível de recursos limitados como o espaço na tela, o design da interface do usuário assume maior importância do que nunca. Os usuários de dispositivos móveis, geralmente, procuram concluir rapidamente um tarefa simples. Devido à mobilidade, eles não podem tirar proveito de toda a gama de funcionalidade fornecida por um aplicativo Web tradicional como seria em um desktop (WASSERMAN, 2010).

Os passos de design e implementação são fases críticas no desenvolvimento da aplicação. Mesmo se um aplicativo deve desempenhar as mesmas funções em todas as plataformas, é impossível fazer o design uma vez para usar em qualquer plataforma. Podem existir limitações no tipo de dispositivo, cores no design, questões de acessibilidade e tamanho de tela, entre outras.

2.1.4 Manutenibilidade

Como dito anteriormente, existe uma variedade de sistemas operacionais e esses sistemas lançam atualizações quase que regularmente. Os aplicativos móveis, por sua vez, precisam se manter atualizados e alinhados nos recursos que foram descontinuados ou tiveram sua im-

plementação/utilização modificadas. De acordo com Perchat, Desertot e Lecomte (2013, p.1), os sistemas operacionais geralmente são atualizados a ponto de poder impactar os aplicativos, tornando-os inutilizáveis na versão atualizada do sistema.

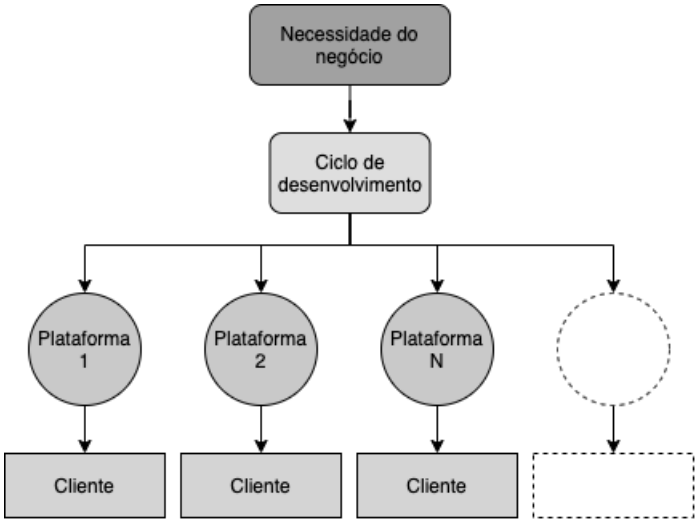
Levando em conta o número de plataformas nas quais o aplicativo está distribuído, as atualizações desse aplicativo também deverão ser distribuídas em todas essas plataformas. Um recurso disponível para o usuário com a aplicação atualizada, pode não estar disponível para outro em que o aplicativo está com uma versão mais antiga.

2.2 Desenvolvimento Multiplataforma

Atualmente, existem, no mercado, diversas abordagens para o desenvolvimento de aplicativos móveis. Dentre elas, podemos destacar: o desenvolvimento nativo, que utiliza linguagens e ferramentas específicas para uma determinada plataforma, e o desenvolvimento multiplataforma, que mescla todas as vantagens do desenvolvimento nativo, mas com a praticidade de escrever o código somente uma vez.

Dentro do desenvolvimento multiplataforma, há diversos tipos de abordagens que levam ao mesmo objetivo: utilizar um único ambiente de desenvolvimento para construir aplicações que executam em diversas plataformas partindo de um mesmo código. Esse princípio é conhecido como *develop once and run anywhere*. A Figura 2.1 apresenta o modelo de desenvolvimento multiplataforma.

Figura 2.1 – Modelo de desenvolvimento multiplataforma



Fonte: Adaptado de Corral, Janes e Remencius (2012)

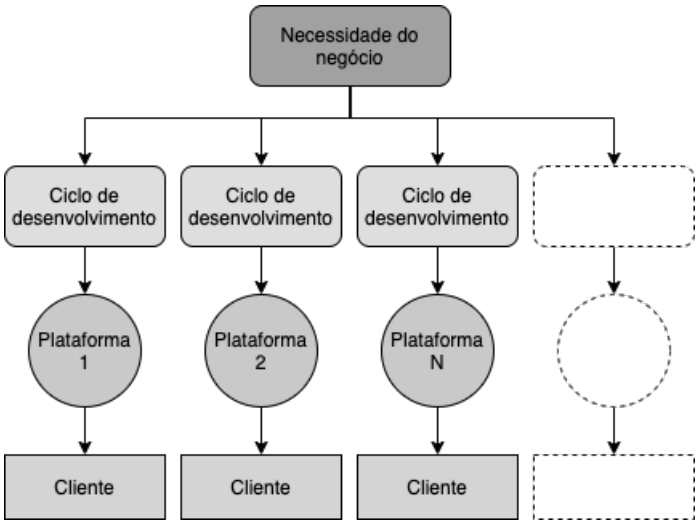
Neste trabalho, focaremos no estudo das seguintes abordagens de desenvolvimento: nativo, abordagem híbrida, interpretada e multi-compilada. Essas abordagens serão explicadas nas subseções seguintes.

2.2.1 Abordagem Nativa

As aplicações que utilizam a abordagem nativa exigem que o desenvolvedor tenha domínio sobre o SDK e as ferramentas necessárias no desenvolvimento para cada plataforma. Por questão de simplicidade, esse conjunto de ferramentas necessárias será denominado *ambiente de desenvolvimento* no decorrer deste relatório. Cada ambiente de desenvolvimento possui os seus próprios padrões de codificação, visando facilitar o desenvolvimento e unificar maneiras de se escrever os códigos. Os aplicativos produzidos assim, ficam então, limitados a somente um sistema operacional, aquele suportado pela plataforma escolhida. No Quadro 2.1, foram apresentados vários exemplos de plataformas e seus respectivos ambientes de desenvolvimento.

Quando o desenvolvedor precisa que sua aplicação possa ser executada em diversas outras plataformas, o código deverá ser reescrito no ambiente de desenvolvimento próprio da plataforma alvo, passando por todo o processo de desenvolvimento novamente. De acordo com Venteu e Pinto (2018), avaliando o público-alvo do aplicativo, pode-se escolher a abordagem nativa, caso seja um publico exigente, procurando uma aplicação com alta performance. A Figura 2.2 ilustra o modelo de desenvolvimento nativo, com ciclos de desenvolvimento específicos para cada plataforma.

Figura 2.2 – Modelo de desenvolvimento nativo



Fonte: Adaptado de Corral, Janes e Remencius (2012)

Os aplicativos nativos são também distribuídos em suas plataformas específicas como, por exemplo, aplicativos desenvolvidos para o sistema operacional iOS, são feitos utilizando Objective-C/Swift como linguagem de programação e disponibilizados unicamente na App Store[1], mantida pela Apple Inc. Já os aplicativos para Android, são desenvolvidos utilizando a linguagem Java/Kotlin e distribuídos pela Google Play Store, desenvolvida e operada pela Google.

2.2.2 Abordagem Híbrida

A abordagem híbrida, segundo Latif et al. (2016), é uma combinação das vantagens presentes nas funcionalidades nativas e de tecnologias web. Diferentemente das aplicações web, a abordagem híbrida distribui a aplicação na loja das plataformas, como se fosse um aplicativo nativo, mas com a vantagem de ter seu código escrito somente uma vez. um framework conhecido para construção de aplicações híbridas é o Ionic[2].

Como vantagens da abordagem híbrida, podemos destacar a interface do usuário que pode ser reusada nas diferentes plataformas e o acesso das funcionalidades do dispositivos, como desvantagens, o aplicativo hibrido possui menos performance que um aplicativo nativo e a interface do usuário não oferece o *look and feel* de aplicações nativas, para alcançá-la, seria necessário desenvolver o aplicativo tendo como foco, uma plataforma específica (EL-KASSAS et al., 2017).

2.2.3 Abordagem Multi-compilada

Segundo Hansson e Vidhall (2016), na abordagem multi-compilada, o código fonte escrito é convertido/compilado em um arquivo binário específico para a plataforma escolhida, então, com uma única implementação é possível obter binários nativos para diversas plataformas. Todo o desempenho do aplicativo gerado depende do quão eficiente é o compilador. Um exemplo dessa abordagem, é o *Xamarim*, uma plataforma de software livre para a criação de aplicativos que utiliza C# como linguagem de programação.

A principal vantagem dessa abordagem é que ela não necessita da camada de abstração presente nas outras abordagens, como por exemplo, na abordagem hibrida, os recursos do dispositivo são disponibilizados e acessados por meio de uma camada de abstração. O aplicativo

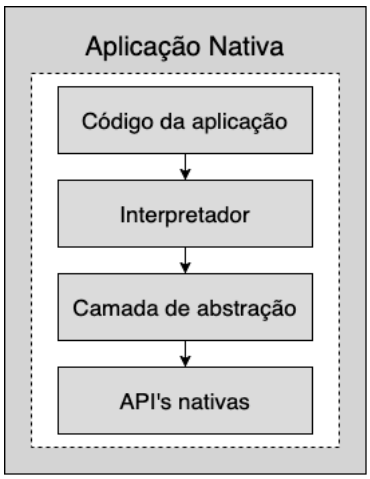
[1] <https://www.apple.com/ios/app-store/>
[2] <https://ionicframework.com/>.

produzido fornece todos os recursos que o aplicativo nativo forneceria, mas com a desvantagem de que, alguns recursos, como acesso a câmera e o serviço de localização, não podem ser "reusados", são específicos para cada plataforma, ou seja, o acesso ao recurso deve ser implementado tendo como foco cada plataforma em específico.

2.2.4 Abordagem Interpretada

A abordagem interpretada se destaca por sua similaridade com um aplicativo nativo. Aplicações interpretadas fornecem um *look and feel* (uma “sensação”) de aplicativo nativo e os componentes que são renderizados na tela são realmente nativos (Rahul Raj; Seshu Babu Tolety, 2012). O código escrito em uma linguagem comum, como *JavaScript*, se comunica com uma camada de abstração, para assim, acessar os componentes nativos, através de uma Interface de Programação de Aplicação (API), como mostra a Figura 2.3. Uma das tecnologias conhecidas para essa abordagem é o React Native[3], que ser á explicado com mais detalhes a seguir.

Figura 2.3 – Arquitetura de uma aplicação interpretada



Fonte: Adaptado de Rahul Raj e Seshu Babu Tolety (2012)

Cada abordagem tem suas próprias vantagens e desvantagens, fica a cargo da equipe de desenvolvimento analisar o propósito e os requisitos da aplicação e decidir a melhor abordagem a ser utilizada. Se o principal propósito for consumir e interagir com conteúdo online, a abordagem multi-plataforma pode ser uma boa escolha. Caso o aplicativo seja para uso que não utilize a internet, a abordagem nativa pode ser mais indicada (SMUTNý, 2012).

[3] <https://reactnative.dev/>

2.3 React Native

React Native é um framework para desenvolvimento de aplicativos móveis criado pelo Facebook Inc. Esse framework utiliza a abordagem interpretada para gerar aplicações multiplataformas. O desenvolvedor não precisaria gastar muito tempo desenvolvendo códigos diferentes para plataformas diferentes. O framework permite que um código escrito em JavaScript possa ser utilizado para criar aplicativos para diferentes plataformas, mesmo que algo especifico de plataforma seja exigido.

No início de 2015, o React Native suportava somente o desenvolvimento de aplicativos para a plataforma iOS. Com o passar do tempo, o framework foi se expandindo e, em setembro do mesmo ano, foi adicionado o suporte à plataforma Android (WEITERSHAUSEN; WITTE, 2015).

O React Native foi construído em cima dos conceitos do *ReactJS*, um framework JavaScript para desenvolvimento Web que se tornou open-source em 2013. Com essa mesma ideia, o React Native também é open-source e seu repositório no Github [4] está disponível para a comunidade sugerir correções e implementações de novas funcionalidades. A ultima versão lançada, a 0.63, já é um código bem mais estável do que as primeiras versões e já implementa ferramentas importantes que auxiliam no desenvolvimento, incluindo ferramentas de *debug*[5].

Uma dos grandes vantagens trazidas pelas últimas versões foi o suporte ao *fast refresh*, uma nova experiência de *reloading* dentro da aplicação. A cada alteração feita pelo o desenvolvedor, somente o arquivo alterado será recarregado para dentro da aplicação, conservando o estado atual do aplicativo e economizando tempo no desenvolvimento, visto que não será mais necessário recarregar toda a aplicação e percorrer todos os fluxos novamente até o local desejado (FACEBOOK, 2020a). Outra novidade, que está sendo cada vez mais aprimorada, é a *LogBox*, uma maneira mais amigável de apresentar para o desenvolvedor erros, avisos e reclamações na renderização dos componentes.

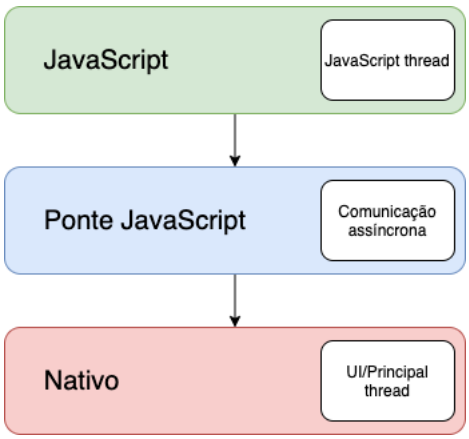
O React Native se comunica com a interface nativa através de uma ponte utilizando JavaScript e JavaScriptCore como interpretador (FACEBOOK, 2020c). A ponte JavaScript conecta o lado JavaScript e o lado nativo dos aplicativos (HANSSON; VIDHALL, 2016), invocando rotinas nativas em cada plataforma gerando o efeito de portabilidade. A Figura 2.4 exemplifica esse processo. A parte nativa, onde fica a *thread* principal, que também é utilizada

[4] https://github.com/facebook/react-native

[5] Processo de encontrar e reduzir defeitos em aplicativo de software ou até mesmo em hardware.

por qualquer aplicação nativa, é responsável por tratar as requisições referentes a renderização dos componentes na tela e também pelos gestos feitos pelo usuário. A *thread* do JavaScript, é onde todo código JavaScript é executado. A ponte JavaScript é basicamente uma camada de transporte, que transporta mensagens assíncronas do JavaScript para os módulos nativos (NOVICK, 2017).

Figura 2.4 – Arquitetura de comunicação React Native



Fonte: Adaptado de Hansson e Vidhall (2016)

O React Native é um framework para desenvolvimento móvel que utiliza a abordagem interpretada, ou seja, o seu código não é compilado para a linguagem nativa (java, objective-c, etc), ele é interpretado em tempo de execução e componentes realmente nativos são renderizados na tela de forma dinâmica. Nas subseções seguintes, serão explicados alguns dos conceitos principais nos quais o React Native se baseia: componentes, JSX, estado e props.

2.3.1 Componentes

Componentes podem ser definidos como conjuntos isolados de lógica (JavaScript), interface gráfica (JSX) e possivelmente estilização (CSS), tornado-os livre de outras dependências externas. Os componentes podem receber entradas e retornar elementos que serão renderizados na tela. Quando são renderizados, cria-se uma espécie de árvore de componentes com as relações de pai e filho.

No React Native, ao iniciar o aplicativo, o componente de entrada da aplicação é renderizado na tela e a partir daí, é criado uma árvore de componentes estabelecendo relações de componente pai e componentes filhos.

Existem 3 diferentes maneiras de se criar componentes dentro do React Native.

- A primeira maneira é criar classes que devem estender a super classe *Component* do *React*, e o único método, que obrigatoriamente deve ser implementado devido à herança, é o método *render()*, onde fica o código JSX dos elementos que serão renderizados na tela;

- A segunda maneira é criar simples funções que retornam códigos em *JSX*, como mostrado no exemplo da Figura 2.5;

- A terceira e última maneira é utilizar hooks, que são funções que permitem interligar métodos de ciclo de vida ao estado do componente por meio dos componentes funcionais (FACEBOOK, 2020b). Basicamente, somente por causa dos hooks, é possível deixar de utilizar classes na criação de componentes;

A Figura 2.5 mostra um trecho de código na linguagem *JavaScript* para criação do componente *Hello World*. Esse componente foi criado utilizando a estrutura de função, que, como dito anteriormente, deve retornar trechos de códigos em JSX, uma extensão de sintaxe JavaScript, que serão renderizados na tela.

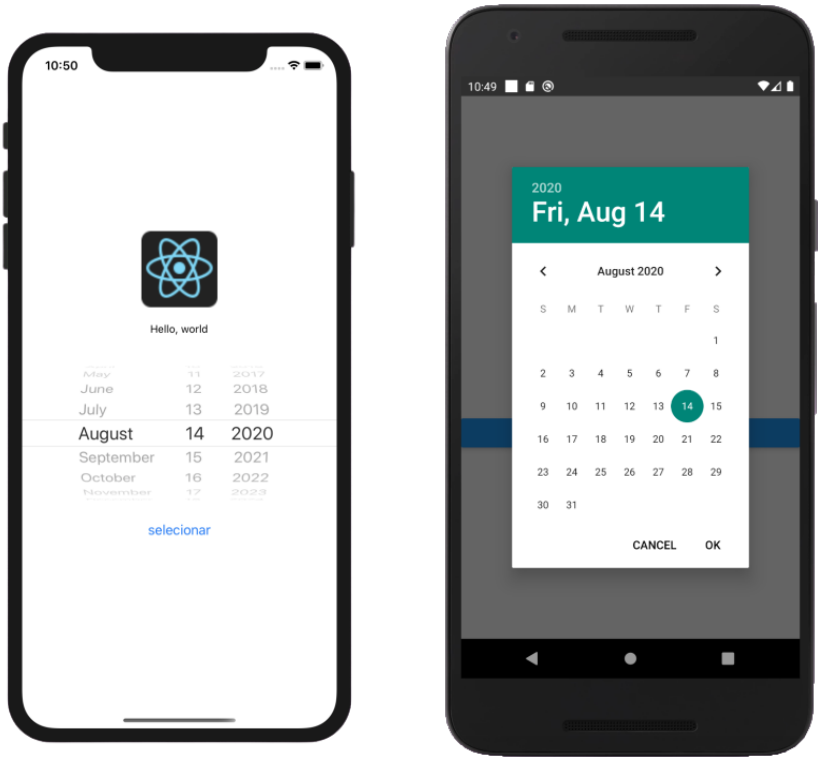
Figura 2.5 – Código que produz um componente de texto

```
1  import React from 'react';
2  import {View, Text, Button, Image} from 'react-native';
3  import DateTimePicker from '@react-native-community/datetimepicker';
4
5  function HelloWorld() {
6    return (
7      <View style={{flex: 1, justifyContent: 'center'}}>
8        <Image style={{
9          height: 100,
10         width: 100,
11         alignSelf: 'center',}}
12         source={{uri: 'https://reactnative.dev/img/tiny_logo.png'}}
13       />
14       <Text style={{alignSelf: 'center', padding: 20}}>Hello, world</Text>
15       <DateTimePicker value={new Date()} />
16       <Button title="selecionar" />
17     </View>
18   );
19 };
20
21 export default HelloWorld;
```

Fonte: Autor

A Figura 2.6 apresenta os componentes renderizados a partir do código da Figura 2.5 nas plataformas iOS 2.6(a) e Android 2.6(b). Analisando o código da Figura 2.5, nas linha 8 e 14, são criados um componente de imagem e um componente de texto, respectivamente, seguindo os estilos passados para o componente como propriedade. Esses dois elementos são renderizados da mesma forma em ambas as plataformas. Por outro lado, o elemento *DateTimePicker*, adicionado na linha 15, são renderizados de forma diferente pela API nativa de cada plataforma, como pode-se observar na figura.

Figura 2.6 – Componentes produzidos pelo código da Figura 2.5



(a) iOS (b) Android

Fonte: Autor

2.3.2 JSX

Criado pelo *React*, o JSX é uma linguagem de marcação com sintaxe semelhante ao XML. O objetivo principal do JSX é fornecer aos desenvolvedores uma maneira mais simples e intuitiva de produzir componentes (TREINAWEB, 2020). Essa sintaxe tende a ajudar o desenvolvedor nas etapas da construção e montagem dos componentes na tela.

Como o código escrito não é em JavaScript puro, deve então existir um transpilador que transforma esse código em JavaScript, para que ele possa ser interpretado. Um exemplo de transpilador que faz esse trabalho, é o Babel [6].

Ao utilizar a sintaxe JSX, existem algumas peculiaridades que precisam ser respeitadas. Na criação de componentes, por exemplo, precisa-se necessariamente começar com a primeira letra maiúscula, como mostrado na criação do componente de Imagem na linha 8 e do componente de texto na linha 14 da Figura 2.5. Além disso, as variáveis precisam estar entre colchetes na passagem de propriedades para o componente.

2.3.3 Estados e Props

Cada componente criado tem seu próprio estado dentro da árvore de componentes. Esse estado consiste em variáveis que são criadas dentro do componente para armazenar alguma informação necessária para a sua renderização na tela. Cada alteração nesse estado, gera uma re-renderização em todo o componente. Esse mecanismos confere o efeito reativo ao aplicativo. O estado serve como uma "memória" para o componente.

Props basicamente são propriedades que são passadas para o componente, a fim de auxiliar na logica própria e na renderização do componente. Fazendo uma comparação com funções, as props são como os parâmetros que são passados para funções. Na linha 8 da Figura 2.5, um objeto com os estilos que serão aplicados ao componente é passado como uma propriedade chamada *style*. Propriedades também poder ser funções que pertencem ao escopo de outro componente. Essa alternativa pode ser utilizada quando o estado de um componente precisa ser alterado por um componente filho.

[6] <https://babeljs.io/>

3 IMPLEMENTAÇÃO DO APLICATIVO

Neste capítulo, são descritas as atividades de todo o processo de desenvolvimento do aplicativo, realizadas no decorrer do estágio, inclusive os desafios encontrados ao longo desse estágio.

3.1 Descrição do Projeto

Este trabalho trata-se de um aplicativo para dispositivo móvel desenvolvido utilizando o framework *React Native*, para auxiliar os profissionais de contabilidade na comunicação e na prestação de serviços para com os seus clientes. O aplicativo é um mínimo produto viável (MVP) desenvolvido com as funcionalidades e serviços julgados essenciais para clientes da área de contabilidade.

A decisão por uma abordagem multiplataforma foi feita com o objetivo de simplificar e agilizar o processo de desenvolvimento do aplicativo, em um primeiro momento. Desenvolver com o framework que utiliza a abordagem interpretada foi devido à empresa não trabalhar com outra tecnologia de desenvolvimento multiplataforma, além do *React Native*.

3.2 Estrutura do Aplicativo

Conforme a aplicação foi evoluindo, ter uma estrutura bem definida se tornou essencial para evitar o desperdício de tempo na procura por algum recurso ou na decisão sobre onde colocar algum componente. O código fonte do aplicativo foi estruturado em 5 partes principais como descrito seguir:

- Componentes *core* - São os componentes criados para serem utilizados em diversos lugares do aplicativo, para garantir padronização dos componente, facilitar a manutenibilidade do componente e evitar a duplicação de código com a reutilização dos componentes;
- *Presentational* - Os componentes dentro deste diretório são os que realmente serão renderizados no aplicativo. Cada tela ou fluxo de navegação que existe dentro do aplicativo possui um diretório específico dentro do diretório *Presentational*, o que permite manter o isolamento entre as telas. A montagem dos componentes presentes nessa parte, foram feitas utilizando, na maioria das vezes, os componentes criados no diretorio *core*;

- *Store* - Essa pasta abriga toda a configuração do estado global do aplicativo, inclusive todos os hooks customizados desenvolvidos especificamente para o aplicativo e utilizados para acessar esse estado. Nessa parte, também são mantidas as funções que são utilizadas para fazer as requisições assíncronas para buscar os dados no backend;

- Configurações - Nessa parte, são organizados todos os arquivos que não precisam ser redefinidos a fim de manter as configurações separadas da lógica e tornar mais fácil o acesso em qualquer parte do código. Aqui são armazenados, por exemplo, os arquivos contendo as *strings* utilizadas em todo o aplicativo, cores que foram utilizadas na estilização dos componentes, arquivos de rotas e constantes utilizadas no projeto.

- *Assets* - Aqui são organizados todos os arquivos de imagens, ícones e fontes personalizadas, facilitando o uso em qualquer lugar do aplicativo.

- Testes - Os arquivos de configuração da estrutura de testes, os *mocks* utilizados e os testes dos componentes propriamente ditos, ficam em uma estrutura separada no aplicativo.

3.3 Processo de Desenvolvimento

O processo de desenvolvimento utilizado na empresa foi o *Scrum*, uma metodologia ágil para gestão e planejamento de projetos. No *Scrum*, os projetos são divididos em ciclos chamados de *Sprints* com uma duração bem definida. Neste projeto, a duração da sprint correspondia a duas semanas. As *sprints* definem um conjunto de atividades a serem realizadas durante o tempo estabelecido. Metodologias ágeis são iterativas, ou seja, o trabalho é feito aos poucos em ciclos, o projeto avança gradativamente a cada *Sprint*.

No início de cada *Sprint* é feita uma reunião com todo o time de desenvolvimento para discussão e definição das tarefas que serão inseridas na sprint que está para começar. As tarefas então, são pontuadas de acordo com sua complexidade e tempo estimado no desenvolvimento, essa estratégia é conhecida como *Scrum Poker*. Essa reunião liderada pelo *Product Owner* é chamada de *Sprint Planning*.

Ao final de cada *sprint*, é realizada uma reunião com o cliente, em que se apresentava tudo o que iria ser desenvolvido pelo time durante a sprint. Esse ritual é chamado de *Review Meeting*. Logo após essa reunião, era feita a *Sprint Retrospective*: uma reunião com o time para analisar o progresso do desenvolvimento durante a *sprint*, assim como os pontos fortes e os

pontos fracos encontrados. O objetivo desta retrospectiva é evitar aprimorar o processo e evitar que erros sejam repetidos na próxima *sprint*.

A cada dia da *sprint*, o time realizou uma breve reunião com o objetivo de esclarecer o que foi desenvolvido no dia anterior e o que seria desenvolvido no dia em questão, bem como os impedimentos encontrados e discussões sobre algo relevante para todo o time. Todo esse processo garante que o time siga alinhado no desenvolvimento e com certeza torna o desenvolvimento mais ágil.

A equipe de desenvolvimento, inicialmente era composta por 7 pessoas, subdivididas a seguir: 1 *Product Owner*, responsável por trazer os requisitos do cliente para a equipe de desenvolvimento; 1 *Scrum Master*, responsável por garantir que todos do time entendam e apliquem os princípios do *Scrum*; 1 designer, responsável por elaborar o layout da interface gráfica do aplicativo; 1 desenvolvedor frontend, responsável pela versão WEB do projeto; 2 desenvolvedores React-Native, responsáveis pelo desenvolvimento da versão *mobile* do projeto; e 1 desenvolvedor backend, responsável por toda a lógica e persistência dos dados na versão WEB e mobile. Posteriormente, foram adicionados mais 2 novos colaboradores ao projeto, observando eventuais necessidades, como férias e sobrecarga de desenvolvedores.

3.3.1 Apresentação do projeto

No início do projeto, foi feita uma reunião inicial com o cliente onde cada integrante do time fez uma breve apresentação individual para o cliente, a fim de que todos se conhecessem. Nessa reunião, foi apresentada para o time a principal ideia para o projeto, do que ele se tratava e quem o seria o publico alvo. Também foi apresentado, pelo *Product Owner*, todo o cronograma do projeto, bem como datas de lançamento das *releases* e estimativa de finalização do projeto.

Foi feita previamente uma reunião com o *Product Owner*, designer e cliente visando discutir questões relacionadas ao *mockup* a ser criado pelo designer junto a visão que o cliente espera do produto. Por fim, foi apresentada também, pelo designer, na reunião com todo o time, uma proposta inicial de *layout*, mostrando alguns dos principais fluxos presentes nas versões WEB e móvel, e que a equipe de desenvolvimento seguirá durante todo o desenvolvimento.

3.3.2 *Sprint* Inicial

Na primeira *sprint* do projeto, foi desenvolvido o fluxo de autenticação, recuperação de senha e cadastro de usuários no aplicativo.

Basicamente, todas as funcionalidades implementadas no aplicativo somente são liberadas mediante o cadastro e autenticação do usuário no aplicativo. A autenticação foi realizada através de requisição HTTP enviada ao backend, e então, é retornado um *token* no formato JWT (JSON Web Token) que contém informações do usuário autenticado. Esse token é utilizado no cabeçalho de todas as requisições HTTP feitas dentro do aplicativo.

O cadastro foi desenvolvido com uma abordagem um pouco diferente da usual. Geralmente, o fluxo de navegação de alguma funcionalidade, é realizado utilizando um gerenciador de histórico de navegação com a estrutura de dados no formato de pilha. Cada nova tela acessada é empilhada na pilha de telas e, quando o usuário pressiona o botão de voltar, a tela tela anterior é desempilhada e renderizada. De forma diferente, o aplicativo desenvolvido neste trabalho possui um tipo de barra de progresso que vai se movendo durante as etapas de cadastro. Essa barra de progresso funciona como um indicativo de qual etapa do cadastro o usuário se encontra. A abordagem utilizada foi transformar os componentes que possuem alguma interação com o usuário e transformá-los em pequenos componentes isolados, assim, uma única tela será renderizada e durante o preenchimento das informações, somente o conteúdo que o usuário possui interação é alterado.

A Figura 3.1, mostra duas etapas do fluxo de cadastro. A parte destacada em vermelho é onde os componentes para o preenchimento das informações são alterados. O restante permanece em todas as etapas.

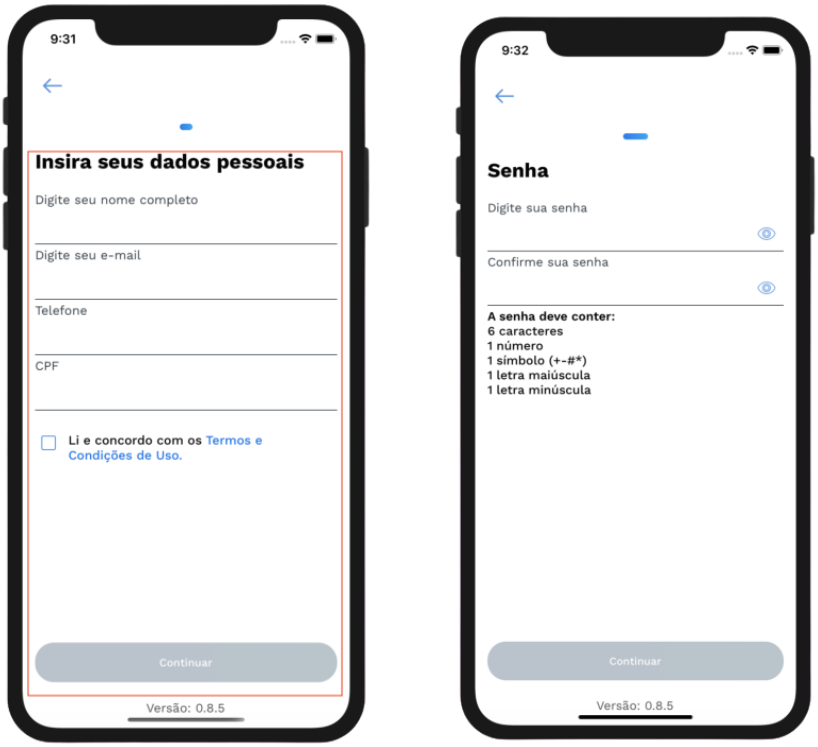
Essa abordagem foi utilizada exclusivamente para oferecer uma experiência fluida do movimento da barra de progresso durante todo o fluxo sem a troca de tela, que se parece com um aspecto de troca de folhas de um livro. Para a barra de progresso se mover durante o preenchimento das informações, foram utilizadas funções do *react-native-reanimated*, uma API de animações desenvolvida para evitar alguns problemas na API provida pelo *React-Native*.

Para uma melhor solidez na construção dos componentes, foi decidido utilizar a biblioteca *prop-types* para controlar as *props* que o componente receberá, tipo requerido, valores padrões e *props* obrigatórias. Essa é uma das práticas para forçar o desenvolvedor a definir tipos no *Javascript* e impor algumas limitações.

3.3.3 Estrutura do Estado Global

Nas etapa de cadastro para utilizar o aplicativo e cadastros dentro do aplicativo, foi utilizado o *Redux* para manter os dados armazenados enquanto o usuário estava preenchendo

Figura 3.1 – Telas iniciais do fluxo de cadastro



(a) Dados pessoais (b) Senha de acesso

Fonte: Autor

as informações. O Redux é uma biblioteca para gerenciar o estado global do aplicativo. Os componentes são dispostos no momento da renderização, utilizando o formato de árvore, como se fosse a DOM de uma aplicação web. Quando o desenvolvedor precisa de uma informação que está armazenada há alguns níveis acima nesta árvore, não é tão simples e nem trivial manipular a árvore de componentes para obter a informação desejada. O *redux* centraliza o estado e a lógica do aplicativo, permitindo alguns recursos avançados junto a uma ferramenta muito útil de *debug* (ABRAMOV, 2020).

A estrutura do *redux* foi construída utilizando um padrão chamado *Duck Pattern*, o qual evita a repetição de arquivos com o mesmo nome e as importações/exportações necessários na configuração padrão deseja do *redux*, organizando toda a estrutura de forma mais simples, amigável e legível. Para fazer essa estrutura ser escalável, foi utilizado o *Redux Sauce*, uma forma mais compacta de criar as funções necessárias para o funcionamento do *redux*. Para acessar o estado do aplicativo, foram criados *custom hooks* que auxiliam também na alteração desse estado.

Para armazenar o *token* e os dados do usuário que são recebidos na etapa de login, foi utilizado a biblioteca *redux-persistent* que provê uma espécie de estado que é persistido mesmo

após o encerramento do aplicativo. Quando o aplicativo inicia novamente, após o usuário já ter feito o login pela primeira vez, o estado persistido contendo o *token* de autenticação é carregado e pode utilizado no cabeçalho das requisições HTTP.

3.3.4 Paginação

Para algumas requisições feitas no aplicativo, especificamente as requisições que retornam lista de informações, foi feita uma paginação no backend para evitar que o tempo de espera e o tamanho da resposta da requisição vire um gargalo na utilização do aplicativo, comprometendo a experiência do usuário.

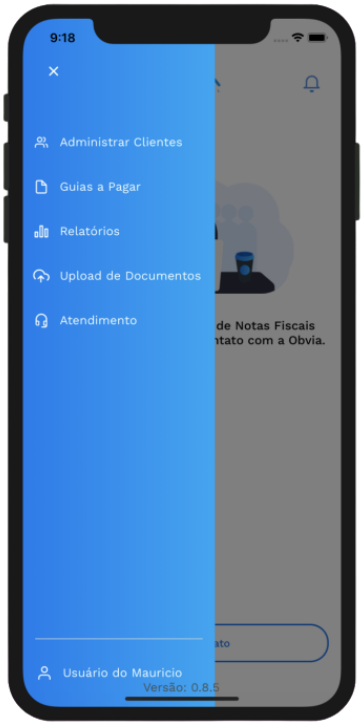
Foi preciso tratar a paginação no aplicativo para que as requisições de buscar a próxima página de dados fossem feitas no momento certo. Para isso, foi utilizado o componente *FlatList*, interface para renderização de listas que provê uma maneira simples de tratar a paginação. Dentro do componente, foi definido um *limite* de 85%, que é utilizado para delimitar quanto o usuário precisa chegar na renderização da lista para que uma função de *callback* seja executada. A função de *callback* foi utilizada para tratar a paginação, verificando a página atual e, em seguida, se necessário, feita a requisição para o backend buscando uma nova página de dados. Assim que a requisição é finalizada, a lista de itens é atualizada com os novos itens da página buscada e, consequentemente, incorporados à lista que o *FlatList* já estava renderizando sem nenhum atraso na visualização para o usuário. O componente *FlatList* é bastante utilizando no tratamento de listas, principalmente pela performance no reaproveitamento de componentes no momento da renderização.

3.3.5 Menu Lateral (*Drawer Navigation*)

Dentro do aplicativo, o usuário cadastrado possui diversas informações de status que ajudam na delimitação de funções disponíveis para a utilização pelo usuário. Esse status é recebido junto ao *token* de autenticação e armazenado no estado permanente. Um exemplo de status recebido é "Aguardando Certificado Digital", que é utilizado quando o usuário já concluiu o cadastro, efetuou o pagamento do valor de abertura e o próximo passo é fazer o envio do certificado. As funções que estão disponíveis para o usuário, de acordo com o seu status, foram dispostas em um menu lateral do qual adiciona no aplicativo uma novo gerenciador de navegação, o *Drawer Navigation*.

Para a construção do menu lateral, foi utilizado o recurso *drawer* disponibilizado pela biblioteca de navegação *react-navigation/drawer*. Esse componente de navegação inclui funcionalidades para tratar as ações de abertura e fechamento do menu, junto à sobreposição do menu em outras telas quando o menu está aberto. A Figura 3.2 apresenta um exemplo do menu para o usuário com o status de "Ativo".

Figura 3.2 – Menu lateral

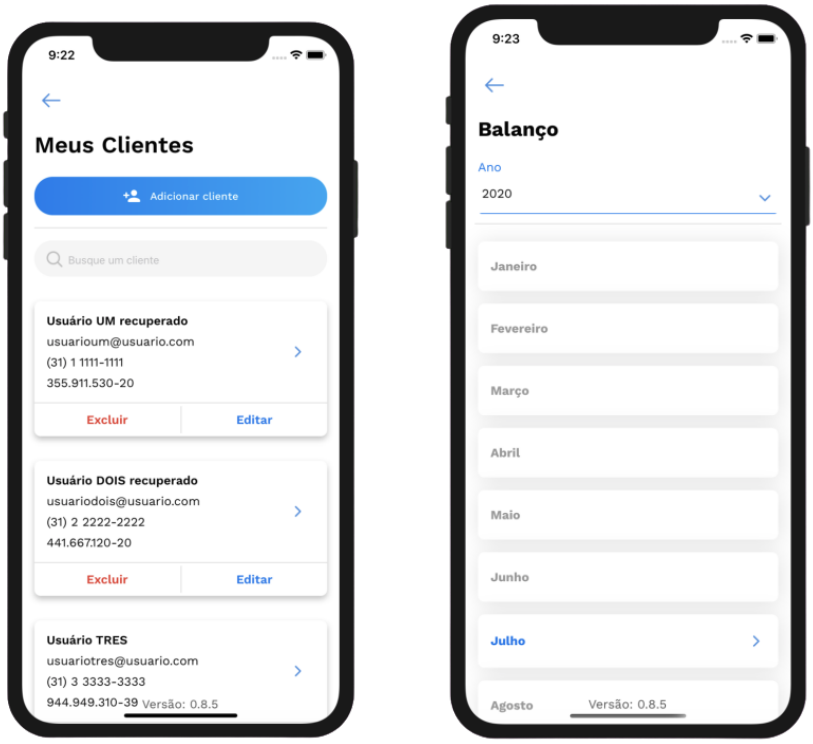


Fonte: Autor

3.3.6 Apresentação dos dados

Durante a utilização do aplicativo, diversos dados recebidos do *backend* precisam ser dispostos de alguma forma para o usuário consultar a qualquer momento. Na maioria das vezes, os dados são apresentados em forma de lista para tornar a visualização pelo usuário mais simples. Para cada tipo de dado renderizado no formato de lista, foi criado um componente no diretório *core* para representar cada item, a fim de garantir a padronização e o reaproveitamento do componente. A Figura 3.3 apresenta dois tipos de dados apresentados como lista, os componentes utilizados foram desenvolvidos sem o auxilio de componentes externos: a Figura 3.3(a) demonstra a lista de clientes cadastrados e a Figura 3.3(b) apresenta o Balanço dividido em meses, de forma que os meses que não possuem dados ficam com um aspecto de desabilitados.

Figura 3.3 – Exemplos de listagens de dados



(a) Listagem de clientes (b) Listagem de relatórios por mês

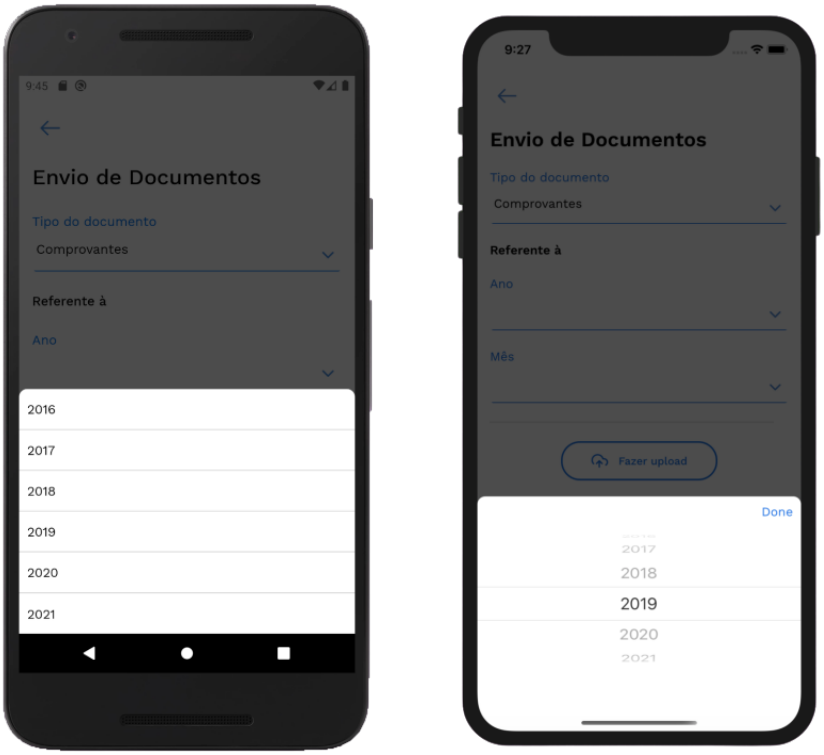
Fonte: Autor

Alguns componentes utilizados no aplicativo precisaram passar por alteração em relação ao componente base oferecido pelas bibliotecas utilizadas. O componente de *picker*, por exemplo, possui um comportamento diferente dentro do sistema operacional Android e iOS, portanto, foi preciso criar um novo componente baseado na biblioteca *react-native-picker*, para que pudéssemos ter um controle maior sobre o que seria renderizado em cada plataforma. A Figura 3.4 apresenta o comportamento do *picker* nas duas plataformas. Existem outros componentes que passaram pelo mesmo processo, mas também existem componentes que foram feitos sem utilizar nenhuma biblioteca base, como por exemplo, o *checkbox* mostrado na Figura 3.1(a) e o *radio-button* ilustrado na Figura 3.5.

3.3.7 Notificações

Umas das funcionalidades que foram implementadas no aplicativo é o serviço de notificações, que estabelece uma interação maior com os usuários, enviando mensagens personalizadas e alguns avisos sobre os serviços prestados, como por exemplo, a necessidade da realização de *upload* de documentos.

Figura 3.4 – Componente *picker* em diferentes plataformas



(a) Picker no Android (b) Picker no iOS

Fonte: Autor

Figura 3.5 – Radio button

Fonte: Autor

Uma maneira simples de implementar as notificações no *React-Native* foi utilizando o *Firebase*[1], plataforma de desenvolvimento gerenciada pela Google que auxilia e agiliza o desenvolvimento de aplicações. O *firebase* exige uma configuração adicional no projeto. É preciso definir como o aplicativo deve tratar uma notificação recebida. Em especial, para aparelhos com o sistema operacional *android*, foi preciso verificar se o dispositivo possuía as permissões necessárias para exibir/receber notificações. Para cada diferente dispositivo em que o aplicativo é executado, o *firebase* gera um *token* para o dispositivo. Esse *token* foi utilizado para enviar notificações individuais (por dispositivo) para os usuários.

Foram implementados dois tipos de notificações: *push notifications*, configuradas com o serviço de *cloud message* do *firebase*, e as notificações recuperadas por meio de requisições HTTP, que são renderizadas em tela específica no aplicativo. As notificações ficam armazenadas no backend e no momento em que são criadas, é enviado uma notificação pelo serviço de *push notifications* para o usuário e através dessa notificação, o aplicativo inicia exibindo as notificações recuperadas do backend. O componente responsável pela renderização da notificação foi desenvolvido utilizando o *Swipeable* da biblioteca *react-native-gesture-handler*. Produzindo um resultado de "arrastar"para limpar a notificação como mostra a Figura 3.6. O texto que aparece no começo da animação foi feito utilizando um valor animado na opacidade, para que o texto só começasse a aparecer de acordo com a quantidade em que o usuário fizesse o *swipe* para esquerda ou para a direita, isso tudo utilizando a biblioteca *react-native-reanimated*.

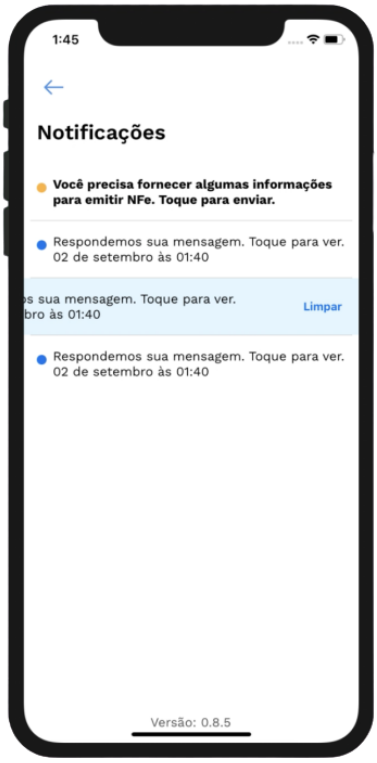
3.3.8 *Upload* de documentos

Uma importante funcionalidade desenvolvida foi a capacidade de permitir ao usuário realizar o envio de documentos para o contador, por exemplo, um comprovante de pagamento, seja por solicitação do contador ou por iniciativa do próprio usuário. O upload foi implementado utilizando a interface *FormData*, permitindo, assim, construir um objeto com pares chave e valor representando o formulário. Dessa forma, os dados podem ser enviados facilmente através de requisições HTTP.

Durante o desenvolvimento dessa funcionalidade, foi encontrada uma falha que impedia que a requisição que realizava o upload do documento fosse finalizada. Para fazer requisições HTTP, foi utilizada a biblioteca *axios*, que é um cliente HTTP baseado em *promisses* com uma implementação simples e que tem como principal vantagem, a criação de instância de um

[1] https://firebase.google.com/

Figura 3.6 – Tela de notificações



Fonte: Autor

objeto do axios com configurações customizadas. A implementação da requisição de *upload* não apresentou problema no sistema operacional Android. Porém, no sistema iOS, a requisição não era finalizada. Depois de muita pesquisa e algumas *sprints*, o erro foi encontrado. O projeto foi desenvolvido utilizando a versão mais atualizada do *React Native* (0.62), lançada pouco antes do início do desenvolvimento. Nessa versão, foi incluído suporte ao *Flipper*, uma poderosa ferramenta de debbuging para aplicativos móveis. A versão do flipper utilizada pelo React Native foi a responsável por não permitir as requisições de *upload* dos arquivos dentro do iOS. Como o flipper não estava sendo utilizado no projeto, ele foi removido e então, o upload começou a funcionar no iOS.

A Figura 3.7a apresenta um exemplo de requisição HTTP utilizando o axios, para atualização do token de acesso. Na linha 64, a requisição de atualização de recurso no backend é realizada para a rota *../auth/refresh-token* utilizando o método POST do HTTP. Com a ajuda do método *call* importado da biblioteca *redux-saga* para auxiliar a manipulação de tarefas assíncronas. Após o término da requisição e o recebimento da resposta do backend, o token contido no cabeçalho das requisições HTTP são atualizados com o novo token recebido. Como o axios foi utilizado pensando na customização das configurações, o token precisará ser atualizado so-

mente uma vez. A instância do axios armazena as informações recebidas para serem utilizadas nas requisições futuras.

Figura 3.7 – Requisição utilizando axios e redux saga

```
61
62  function* authRefreshToken({ tokens }) {
63    try {
64      const response = yield call(api.post, '/auth/refresh-token', tokens);
65
66      const { token, refreshToken } = response.data;
67
68      api.defaults.headers.common.Authorization = `Bearer ${token}`;
69
70      yield put({
71        type: Types.AUTH_REFRESH_TOKEN_SUCCESS,
72        token,
73        refreshToken
74      });
75    } catch (error) {
76      yield put({
77        type: Types.AUTH_REFRESH_TOKEN_FAILURE
78      });
79    }
80  }
81
```

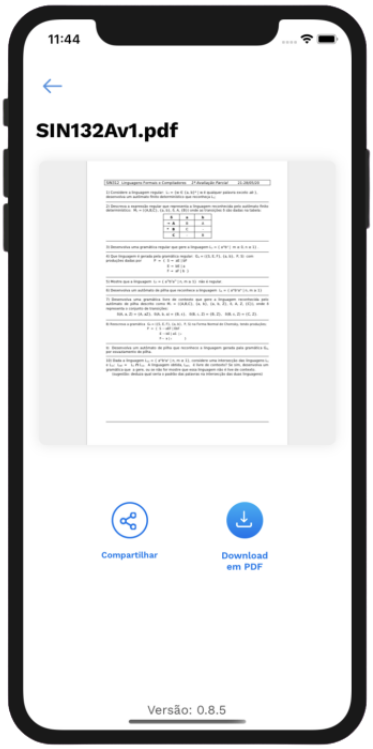
Fonte: Autor.

3.3.9 *Download* de documentos

O aplicativo também possui uma funcionalidade que permite ao usuário fazer o *download* de arquivos disponibilizados pelo contador, por exemplo, um relatório, como mostrado na Figura 3.8

No decorrer do desenvolvimento dessa funcionalidade, foi percebido que alguns sistemas operacionais possuem particularidades que precisam ser tratadas caso a caso para que o usuário não note discrepâncias na experiência de uso do aplicativo em plataformas diferentes. O sistema operacional Android possui um gerenciador de *downloads* nativo que facilitou o gerenciamento de algumas questões envolvendo o nome que será dado ao arquivo após o *download* e o local que será armazenado. Já no iOS, esse gerenciamento é um pouco mais complicado, pois a aplicação não pode acessar o sistema "externo"de diretórios, a não ser o diretório do próprio aplicativo. Isso dificulta que o arquivo seja armazenado em um local de fácil acesso para o usuário.

Figura 3.8 – Tela para download de um documento



Fonte: Autor

De frente com essa limitação nativa do sistema operacional, a solução mais simples encontrada foi executar o navegador padrão do usuário e, logo em seguida, abrir uma caixa de diálogo para que o usuário escolha o local de armazenamento, facilitando assim o acesso do arquivo fora do aplicativo. No momento em que o usuário clica no botão responsável por executar a ação de download, é verificado qual a plataforma que o aplicativo está sendo executado, e então, se o resultado for Android, o gerenciador nativo de downloads é iniciado, caso o resultado seja iOS, o navegador padrão é inicializado com as opções de download. Para inicializar o navegador padrão, foi utilizado o *Linking* do *React Native*, que oferece uma interface para interação com links, como abrir o aplicativo de e-mail ao clicar em um e-mail ou clicar em um número de telefone e, então, abrir o aplicativo de mensagens para enviar uma mensagem ao número clicado.

3.3.10 Testes Funcionais

Validar com o cliente o que foi desenvolvido durante a *sprint* é um importantíssimo passo para construir produtos de de qualidade e alinhados com os desejos do cliente. Outro passo de extrema importância é o teste das funcionalidades desenvolvidas e, consequentemente,

captar erros que não foram percebidos pelos desenvolvedores. A prática de executar testes nas aplicações, fora do ambiente e da equipe de desenvolvimento, contribui bastante para a *corretude* das funcionalidades, pois clientes reais podem criar cenários que são inesperados para o desenvolvedor.

Ao término de cada *sprint*, após a reunião de validação com o cliente, uma versão de testes era gerada e entregue ao cliente, para que as funcionalidades trabalhadas durante a sprint pudessem ser homologadas pelo donos do projeto. Essas versões foram publicadas utilizando o serviço interno da plataforma, Google Play Store para Android ou *TestFlight* para iOS.

3.3.11 Testes de Renderização

Em qualquer projeto que utiliza tecnologia baseada em componentização, pode surgir a necessidade de se fazer alterações nos componentes que já tinham sido desenvolvidos, seja para alterar as propriedades recebidas, rearranjar os dados exibidos ou, até mesmo, aplicar algum estilo especial. Foi criada utilizando a estrutura de função, que como dito anteriormente, precisa retornar trechos de códigos em JSX, uma extensão de sintaxe JavaScript, que serão renderizados na tela.

Para a criação dos testes, foi utilizado o framework *Jest*[2], desenvolvido e mantido pelo Facebook Inc.. Foram implementados testes de renderização que verificam se os componentes estão sendo renderizados na tela sem nenhum problema. Esses testes automatizam a tarefa de passar por todos os fluxos de navegação do aplicativo para verificar os componentes modificados, assim como os componentes que renderizam corretamente mas dispararam alertas (*warnings*) em relação à implementação. O código na Figura 3.9 apresenta um exemplo simples para o teste de renderização do componente de botão. Nas linhas 8 a 10, a renderização do componente é testada sem qualquer passagem de propriedades. Nas linhas 12 a 14, o botão é testado simulando o loading que é acionando quando o usuário pressiona o botão e o valor da propriedade é alterada. Por fim, nas linhas 16 a 18, a renderização do botão é testada simulando que o botão possui um comportamento de seleção de opção ao invés de executar alguma ação ao ser pressionado.

[2] https://jestjs.io/docs/pt-BR/tutorial-react-native

Figura 3.9 – Código de teste para o componente de botão

```
1  import 'react-native';
2  import React from 'react';
3  import { render } from 'react-native-testing-library';
4
5  import Button from '@/components/core/Button';
6
7  describe('Button', () => {
8    it('renders correctly', () => {
9      render(<Button />);
10   });
11
12   it('should render with activity indicator', () => {
13     render(<Button isLoading />);
14   });
15
16   it('should render with 8px border radius', () => {
17     render(<Button chooseOption />);
18   });
19 });
```

Fonte: Autor

4 CONCLUSÃO

O objetivo principal deste relatório foi a implementação de um aplicativo para assessoria em contabilidade utilizando o framework React Native. Os objetivos específicos foram apresentar os conceitos básicos envolvendo o desenvolvimento móvel, diferentes tipos de abordagens para desenvolvimento multiplataforma, inclusive os desafios mais comuns encontrados nesse tipo de aplicação.

Durante o estágio, foram adquiridos conhecimentos práticos sobre arquitetura das aplicações móveis, em especial, aplicações multiplataformas, comunicação entre o código e as funcionalidades nativas, padrões de projeto, estruturação da aplicação e utilização dos principais recursos dos dispositivos móveis. Foram implementadas funcionalidades que envolviam o consumo de dados vindos de fontes externas como API's, através de requisições HTTP, e sistemas de gerenciamento de banco de dados para dispositivos móveis. Além disso, foi utilizada a biblioteca *redux*, uma solução muito popular atualmente para gerenciar o estado dos aplicativos desenvolvidos utilizando o framework *React Native*.

O estágio realizado na ioasys proporcionou a aplicação dos conhecimentos adquiridos durante o período acadêmico em um contexto de mercado de trabalho. Foi possível compreender diversos desafios comuns no desenvolvimento de software, inclusive como lidar com situações inesperadas e a trabalhar utilizando a metodologias ágeis.

Foi possível também conhecer mais a fundo o framework open-source *React Native*, que permite escrever códigos em *javascript* que são interpretados como código nativo pelo dispositivo móvel. Assim, as aplicações podem ser executadas tanto no sistema operacional Android quanto no iOS. As atividades desenvolvidas proporcionaram o entendimento de todo o ecossistema da ferramenta, de como funciona a comunicação com a parte nativa dos dispositivos, e de como explorar o *fast refresh* durante o desenvolvimento.

A partir das experiências obtidas no decorrer do estágio, pôde-se perceber que a universidade teve um papel fundamental no processo de aprendizagem. Os conhecimentos adquiridos no curso de Ciência da Computação foram essenciais para o desenvolvimento das atividades e da capacidade de análise diante das tecnologias existentes no mercado. No entanto, alguns conceitos importantes no desenvolvimento de software poderiam ter mais ênfase na Universidade, como metodologias ágeis de desenvolvimento e alguns conceitos ligados à engenharia de software.

REFERÊNCIAS

ABRAMOV, D. Redux: predictable state container for javascript apps. 2020. Disponível em: <https://redux.js.org/>. Acesso em: 24 jul. 2020.

CHARLAND, A.; LEROUX, B. Mobile application development: Web vs. native. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 5, p. 49–53, maio 2011. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1941487.1941504>.

CORRAL, L.; JANES, A.; REMENCIUS, T. Potential advantages and disadvantages of multiplatform development frameworks–a vision on mobile environments. **Procedia Computer Science**, v. 10, p. 1202 – 1207, 2012. ISSN 1877-0509. ANT 2012 and MobiWIS 2012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050912005303>.

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

EXAME. **Memória média de celulares será de 80 GB até fim de 2019**. 2019. Disponível em: <https://exame.com/tecnologia/memoria-media-de-celulares-sera-de-80-gb-ate-fim-de-2019/>. Acesso em: 14 jun. 2020.

FACEBOOK. **Announcing React Native 0.61 with Fast Refresh**. 2020. Disponível em: <https://reactnative.dev/blog/2019/09/18/version-0.61>. Acesso em: 8 ago. 2020.

FACEBOOK. **Hooks de forma resumida**. 2020. Disponível em: <https://pt-br.reactjs.org/docs/hooks-overview.html>. Acesso em: 29 jun. 2020.

FACEBOOK. **JavaScript Environment**. 2020. Disponível em: <https://reactnative.dev/docs/javascript-environment.html#content>. Acesso em: 24 jun. 2020.

GOOGLE. **Visão geral do armazenamento de dados e arquivos**. 2020. Disponível em: <https://developer.android.com/training/data-storage>. Acesso em: 14 jun. 2020.

GOTODATA. **Avanço da Internet das coisas (IoT) exige investimento em hardware e sistemas mais seguros.** 2020. Disponível em: <http://gotodata.com.br/site/seguranca-digital/>. Acesso em: 22 jun. 2020.

HANSSON, N.; VIDHALL, T. **Effects on performance and usability for cross-platform application development using React Native**. Tese (Doutorado), 2016. Disponível em: <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-130022>.

IDC CORPORATE USA. **Smartphone Market Share**. 2020. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 12 set. 2020.

INTERNET WORLD STATS. **World internet usage and population statistics**. 2020. Disponível em: <https://www.internetworldstats.com/stats.htm>. Acesso em: 9 ago. 2020.

Latif, M. et al. Cross platform approach for mobile application development: A survey. In: **2016 International Conference on Information Technology for Organizations Development (IT4OD)**. [S.l.: s.n.], 2016. p. 1–5.

NOVICK, V. **React Native - Building Mobile Apps with JavaScript**. [S.l.]: Packt Publishing, 2017. ISBN 1787282538.

PERCHAT, J.; DESERTOT, M.; LECOMTE, S. Component based framework to create mobile cross-platform applications. **Procedia Computer Science**, v. 19, p. 1004 – 1011, 2013. ISSN 1877-0509. The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013). Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050913007485>.

Rahul Raj, C. P.; Seshu Babu Tolety. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: **2012 Annual IEEE India Conference (INDICON)**. [S.l.: s.n.], 2012. p. 625–629.

SMUTNý, P. Mobile development tools and cross-platform solutions. In: . [S.l.: s.n.], 2012.

TREINAWEB. **O que é JSX**. 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-jsx/>. Acesso em: 29 jun. 2020.

UOL. **Apps crescem em espaço ocupado, mas a memória do celular não; o que fazer?** 2017. Disponível em: <https://www.uol.com.br/tilt/noticias/redacao/2017/07/04/apps-crescem-em-espaco-ocupado-mas-a-memoria-do-celular-nao-o-que-fazer.htm>. Acesso em: 14 jun. 2020.

VEJA. **5,1 bilhão de pessoas têm celular no planeta, sendo 204 milhões no Brasil**. 2019. Disponível em: <https://veja.abril.com.br/economia/51-bilhao-de-pessoas-tem-celular-no-planeta-sendo-204-milhoes-no-brasil/>. Acesso em: 9 ago. 2020.

VENTEU, K. C.; PINTO, G. S. Desenvolvimento móvel híbrido. **Revista Interface Tecnológica**, v. 15, n. 1, p. 86–96, jun. 2018. Disponível em: <https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/337>.

WASSERMAN, A. I. Software engineering issues for mobile application development. In: **Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research**. New York, NY, USA: Association for Computing Machinery, 2010. (FoSER '10), p. 397–400. ISBN 9781450304276. Disponível em: <https://doi.org/10.1145/1882362.1882443>.

WEITERSHAUSEN, P. von; WITTE, D. **React native for android: How we built the first cross-platform react native app**. 2015. Disponível em: <https://engineering.fb.com/developer-tools/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/>. Acesso em: 21 jun. 2020.