Werliton Carlos Sousa da Silva

Aplicações Móveis Nativas com React Native e Firebase: Um Estudo de Caso

São Luis

2018

Werliton Carlos Sousa da Silva

Aplicações Móveis Nativas com React Native e Firebase: Um Estudo de Caso

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal do Maranhão - UFMA

Departamento de Informática

Curso de Ciência da Computação

Orientador: Prof. Dr. Tiago Bonini Borchartt

São Luis

2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Silva, Werliton Carlos Sousa da.
Aplicações Móveis Nativas com React Native e Firebase: Um Estudo de Caso / Werliton Carlos Sousa da Silva. - 2018.
72 p.

Orientador(a): Prof. Dr. Tiago Bonini Borchartt.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luis, 2018.

1. Firebase. 2. Native Mobile Application. 3. React Native. 4. Social. I. Borchartt, Prof. Dr. Tiago Bonini. II. Título.

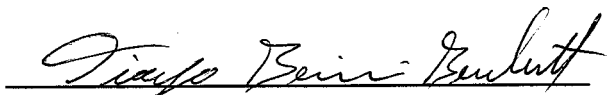
Werliton Carlos Sousa da Silva

Aplicações Móveis Nativas com React Native e Firebase: Um Estudo de Caso

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

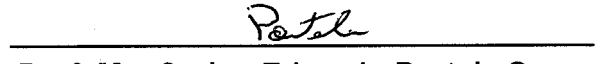
Aprovado em 11 de julho de 2018

BANCA EXAMINADORA



Prof. Dr. Tiago Bonini Borchartt
(Orientador)
Universidade Federal do Maranhão

Prof. Dr. Geraldo Braz Junior
Universidade Federal do Maranhão



Prof. Me. Carlos Eduardo Portela Serra de Castro
Universidade Federal do Maranhão

Dedico este trabalho à minha mãe, Maria da Conceição Lima Sousa e à minha irmã Karliane Sousa da Silva, pois compartilham comigo esta vitória. Que este trabalho seja inspiração para os apaixonados pelo desenvolvimento mobile sendo, também objeto de estudo de futuros acadêmicos interessados por essa área.

Agradecimentos

Quero agradecer primeiro a Deus, por ser meu maior amigo em todos os momentos. Agradecer imensamente à minha mãe, Maria da Conceição Lima Sousa, mulher guerreira, cumprindo papel de pai e mãe até aqui, que sempre foi e será minha maior inspiração para atingir meus objetivos. Agradecer ainda a ela por ter me dado toda a educação base e por todo o esforço para eu ter uma educação de qualidade, apesar de toda dificuldade. Agradecer à minha irmã, Karliane Sousa da Silva, pela sua doação de irmandade, que sempre esteve nos momentos mais difíceis e que conseguimos vencer. Agradecer à minha vizinha Rita de Cássia, minha segunda mãe no ensino médio, compartilhando até o pouco que tinha. Agradecer à meu tio Anestor Lima, peça importante na conclusão do meu ensino médio. Agradecer à meu tio Edinaldo Lima, meu irmão e pai, pelos momentos de parceria e conselhos. Agradecer ao Padre Roney, meu amigo e pastor, por me guiar enquanto cristão católico. Agradecer à minha noiva, Dayanne Silva, pelo companheirismo e paciência. Agradecer ao meu pai biológico, João Carlos Rocha da Silva, por ter me colocado no mundo. E a todos que ajudaram inderetamente para a realização deste sonho.

“Seja a mudança que você quer ver no mundo.”
(Mahatma Gandhi)

Resumo

O trabalho presente demonstra a capacidade da linguagem JavaScript para criação de aplicações móveis. Apresentando os principais conceitos que circundam o React Native para o desenvolvimento de aplicativos móveis nativos e multiplataforma, utilizando apenas JavaScript. Assim como, o Firebase, uma plataforma da Google, que oferece vários serviços de *backend* para aplicações móveis e web, tais como: Authentication, Firebase Database e Firebase Storage. Para demonstrar esses conceitos a nível prático foi desenvolvido o aplicativo Vitrine de Projetos Sociais para Android, como estudo de caso. Cujo objetivo principal é ser elo de comunicação entre pessoas e ONG's ou entre ONG's e ONG's. Fazendo com que as ações e projetos de ONG's, que às vezes são desapercebidos, sejam vistos pelo público podendo engajar doadores e voluntários.

Palavras-chave: Aplicação Móvel Nativa. React Native. Firebase. Social.

Abstract

The present work demonstrates the ability of the JavaScript language to create mobile applications. Introducing the core concepts behind React Native for creating cross-platform native mobile applications using just JavaScript. Like Firebase, a Google platform, it offers various backend services for mobile and web applications such as Authentication, Firebase Database and Firebase Storage. To demonstrate these concepts at a practical level the Vitrine Social Projects for Android application was developed as a case study. Whose main objective is to be a communication link between people and NGOs or between NGOs and NGOs. Making the actions and projects of NGO's, which are sometimes unnoticed, are seen by the public and can engage donors and volunteers.

Keywords: Native Mobile Application. React Native. Firebase. Social.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

APAR	Associação de Pais e Amigos Reviver
API	Interface de Programação de Aplicativos
CNPJ	Cadastro Nacional da Pessoa Jurídica
CSS	Cascading Style Sheets
DOM	Modelo de Objeto de Documento
ES6	ECMAScript 2015
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
JSON	Notação de Objetos JavaScript
LESS	Leaner Style Sheets
MDN	Mozilla Developer Network
NPM	Gerenciador de Pacotes Node
POST	Power On Self Test
REST	Transferência de Estado Representacional
RN	React Native
RBDMS	Relational Database Management System
SASS	Syntactically Awesome Style Sheets
SVG	Scalable Vector Graphics
SDK	Kit de Desenvolvimento de Software
SMS	Serviço de Messagem Curta
XML	eXtensible Markup Language
UI	Interface do Usuário
URL	Uniform Resource Locator
UX	Experiência de Usuário

Sumário

1 INTRODUÇÃO

O uso de *smartphone* e aplicativos móveis estão cada vez mais presentes entre as pessoas ao redor do mundo. De acordo com os dados de uma pesquisa realizada pelo site Mobile Time[1] em parceria com a empresa Opinion Box[2], é possível analisar que 87% dos brasileiros utilizam o sistema operacional Android em seus aparelhos. O iOS aparece com apenas 8% e Windows Phone com 4%. Destes usuários, 94% afirmam que já baixaram e instalaram algum aplicativo em seu *smartphone*. E 47,9% afirmaram que desinstalaram algum app de seus *smartphones* há mais de 24 horas e menos de 6 meses (SOLUçõES, 2017). Diante disso, é possível concluir que o Brasil é um mercado em pleno crescimento para uso de aplicativos. No entanto, a maior parte dos aparelhos é composta por modelo Android de baixo custo, ou seja, possuem poucos recursos, seja de memória ou processamento. Por isso, é muito importante pensar em quais plataformas ter como alvo ao construir aplicativos *mobile*.

Segundo Lopes (2016), as plataformas nativamente oferecem a possibilidade de criar aplicativos. Usando o Android SDK e a linguagem Java, pode-se desenvolver para o sistema do Google. A Apple oferece ferramentas para iOS e permite usar Objective-C ou Swift. No Windows Phone, usa-se C# e toda suite de desenvolvimento Microsoft. Cada plataforma tem sua combinação de linguagem e, principalmente, APIs específicas. Mesmo usando uma linguagem comum, é muito difícil escrever aplicações nativas multiplataforma.

Tentando solucionar esse problema, tem-se as aplicações híbridas. De acordo com Lopes (2016), a solução mais comum atualmente para construção de aplicativos multiplataforma é o Cordova, que é basicamente uma mistura do desenvolvimento Web com recursos do desenvolvimento nativo. Ele usa o ponto forte da Web de ter linguagens padronizadas e um ambiente de execução, o navegador, para construir aplicativos. São Apps instaláveis que podem ser publicadas nas lojas e podem usar recursos nativos da plataforma, mas são escritas em HTML, CSS e JavaScript. Paralelo a isso, tem-se *frameworks* que prometem auxiliar na criação de aplicativos nativos usando uma única linguagem. Um deles será o foco principal deste trabalho, o React Native (OPENSOURCE, 2018b).

[1] Site dedicado ao mercado de conteúdos, aplicações, plataformas e soluções para celulares, smartphones e tablets

[2] Empresa focada em pesquisa de mercado criando soluções digitais que ajudam empresas e pessoas a tomar decisões com segurança, sempre baseadas em informação

1.1 Definição do Tema

O tema aqui apresentado é oriundo da necessidade de demonstrar a capacidade de criar aplicativos móveis multiplataforma com uma única linguagem e nativamente, usando recursos de *backend* que o Google Firebase (FIREBASE, 2018) disponibiliza. Serão apresentados os principais conceitos da ferramenta React Native(FACEBOOKNATIVE, 2018a) juntamente com a API do Google, chamada Firebase no desenvolvimento do aplicativo utilizado como estudo de caso, o Vitrine de Projetos Sociais.

1.2 Justificativa

Nos dias atuais a indústria de aplicativos é dominada por dois gigantes: Android e iOS. Tendo isso como base, para empresas e desenvolvedores, é primordial entregar seu produto para a maioria dos usuários, adaptando-se a ambas as plataformas. As plataformas tem sua própria maneira de desenvolver aplicativos e apenas se assemelham uns aos outros. Um problema que a indústria enfrenta aqui, é ter que contratar pessoal com conhecimento em iOS ou Android, ou ambos, para criar dois aplicativos separados que, consequentemente, exigem manutenção paralela.

Frameworks Multiplataformas que poderiam resolver esse problema, não vêm conseguindo criar aplicativos com o mesmo padrão visual ou funcional que os *frameworks* nativos oferecem, por exemplo, o Ionic. O Facebook anunciou um novo *framework* chamado React Native que promete oferecer uma experiência totalmente nativa com o uso de apenas uma base de código (AXELSSON; CARLSTRÖM, 2016). Esse *framework* será utilizado neste trabalho para desenvolvimento do aplicativo Vitrine de Projetos Sociais.

Muitas aplicações precisam de serviços de *backend* para funcionar com o UI do *frontend*. Isso significa que deveria haver código e servidores **backend** para trabalhar com apps móveis. Para isso, Firebase da Google, mostra-se como uma solução para esse demanda disponibilizando vários recursos, sendo alguns utilizados neste trabalho.

No que diz respeito à situação das Organizações Não Governamentais (ONG's), a revista Época descreve em uma das suas publicações sobre elas o seguinte (BARBOZA, 2017): "A sala com dezenas de funcionários concentrados em computadores e divididos por áreas, com luz fria, ar-condicionado e grandes mesas sinuosas decoradas com brindes diversos, como calendários e chaveiros, poderia ser de qualquer empresa não fosse por um detalhe. Num canto, cinco pessoas se revezam entre pegar um cupom de Nota Fiscal Paulista, programa do governo de São Paulo que devolve

aos consumidores até 20% do imposto cobrado sobre serviços e mercadorias, e digitar alguns números na tela. Assim, manualmente, elas contabilizam os cupons deixados por clientes de 1.500 lojas, supermercados e restaurantes em prol da ONG Ação Comunitária do Brasil, ou simplesmente Vocação. De trocado em trocado, recolheram R$ 2,9 milhões no ano passado e ajudaram a mudar a vida de ao menos 12.448 jovens que foram beneficiados pelos programas de formação socioeducativa da ONG. A Vocação repassa dinheiro a entidades menores que focam em mostrar às crianças e aos adolescentes carentes da periferia de São Paulo que eles podem ser protagonistas de suas próprias vidas. Isso acontece via formação de educadores, distribuição de material didático, desenvolvimento de novas metodologias de aprendizagem e cidadania, capacitação de lideranças comunitárias e inserção de jovens no mercado de trabalho de maneira digna".

A realidade de muitas entidades no Brasil é que elas têm um trabalho próximo a comunidade, precisam de apoio e ajuda para manterem suas ações e não sabem como divulgá-las e, também, não sabem como conseguir ajuda. É possível citar a APAR [3]. Entidade participante do Terceiro Setor que de acordo Falconer (1999), Terceiro setor, entre todas as expressões em uso, pode-se citar: organização não-governamental, sociedade civil, sem fins lucrativos, filantrópicas, sociais, solidárias, independentes, caridosas, de base, associativas, etc, é o termo que vem encontrando maior aceitação para designar o conjunto de iniciativas provenientes da sociedade, voltadas para à produção de bens públicos, como, por exemplo, a conscientização para os direitos da cidadania, a prevenção de doenças transmissíveis ou a organização de ligas esportivas.

Diante da realidade desta e de tantas outras entidades, o objetivo do aplicativo, que será desenvolvido neste trabalho, é que seja literalmente uma vitrine, analogamente à uma vitrine de loja, onde os responsáveis pela loja expõem seus produtos e possíveis clientes os observam, compram ou indicam interesse por algum. Desta mesma forma, a função principal do aplicativo é expor as ações, projetos e atividades de entidades comunitárias fazendo com que possíveis doadores[4] visualizem essas atividades. Além disso, o aplicativo poderá permitir que pessoas marquem uma entidade como favorita, para receber notificações de atividades daquela entidade. Permitirá, também, que pessoas possam se inscrever em cursos ou projetos divulgados pela entidade e, também se candidatar a trabalhos voluntários ou oferecer ajuda/apoio de acordo com os solicitados pela entidade.

[3] Associação de Pais e Amigos Reviver

[4] Usuários que queiram ajudar uma entidade, quer seja com trabalho voluntário, quer seja com apoio financeiro

1.3 Objetivos

1.3.1 Objetivo Geral

Desenvolver uma solução para atender às necessidades que ONG's têm em divulgar eventos e conseguir ajuda de voluntários, através de uma aplicação móvel colaborativa, utilizando os recursos que o React Native oferece para desenvolvimento mobile, descrevendo a linguagem de programação utilizada, processos de concepção, módulos, apoiado com API Google Firebase para construção do *backend*.

1.3.2 Objetivos Específicos

- Demonstrar os recursos do React Native e da API do Google Firebase.
- Apresentar o aplicativo Vitrine de Projetos Sociais, que será desenvolvido como experimento de aplicação nativa.
- Apresentar detalhes de funcionalidades do aplicativo.

1.4 Organização do trabalho

No Capítulo 2 são apresentados os conceitos teóricos necessários que embasam o presente trabalho de forma completa. São descritos conceitos relacionados à Computação Móvel, à linguagem JavaScript, às ferramentas React, base para React Native e à plataforma Google Firebase. O Capítulo 3 apresenta o processo de concepção e desenvolvimento do estudo de caso, a arquitetura do sistema da aplicação e as tecnologias de desenvolvimento. No Capítulo 4 são apresentados os resultados alcançados, sendo encerrado com o Capítulo 5, onde são expostas as conclusões e considerações finais acerca do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados conceitos importantes que embasam este trabalho, como Computação Móvel e suas caracteríticas. Será apresentado, também, conceito de Desenvolvimento Multiplataforma, logo após será demonstrada a linguagem de programação predominante juntamente com seus recursos, versões e padrões, posteriormente uma rápida distinção entre framework e biblioteca para enfim explicar detalhadamente a biblioteca React. Em seguida, a ferramenta motivadora deste trabalho, React Native, utilizado para atingir o objetivo de desenvolver um aplicativo nativo e consequentemente serão demonstradas as caracteríticas e recursos do Google Firebase.

2.1 Computação Móvel

Muitas das tarefas que no nosso cotidiano, que outrora seriam possíveis ser realizadas com maior perfomance usando um notebook, hoje podem ser feitas usando um dispositivo móvel. Processar um texto, realizar pesquisas e compras na internet e outras tarefas de computação podem ser feitas de qualquer lugar usando um tablet ou smartphones. Segundo Djurup (2013), isso tem conduzido para um novo caminho de trabalho, que frequentemente é chamado de Computação Móvel.

Computação Móvel refere a tarefas computacionais realizadas por um usuário móvel com o *smartphone* dele. Os aparelhos de *smartphone* normalmente têm limitação de processamento e armazenamento. Estes aparelhos por si só não têm a capacidade de realizar cálculo computacionais pesados envolvidos em muitas aplicações do dia a dia e frequentemente servem somente como camada de frente para invocar aplicações remotas. Neste caso, a aplicação remota pode estar executando em um servidor poderoso em um lugar distante. Computação móvel, portanto, envolve invocação contínua de aplicações rodando em servidores remotos através da comunicação sem fio. Assim sendo, o estudo de Computação Móvel envolve o estudo dos mecanismos de invocação no cliente final, a tecnologia fundamental de comunicação sem fio e as tecnologias do lado do servidor (PATTNAIK; MALL, 2015).

2.2 Desenvolvimento Multiplataforma

Desenvolvimento Multiplataforma é a prática de desenvolver produtos de software ou serviços para múltiplas plataformas ou ambientes de software. A ideia é desenvolver uma aplicação ou produto que funcione bem em mais de um ambiente es-

pecífico. Alguns dos fundamentos para o desenvolvimento multiplataforma inclui compilar diferentes versões do mesmo programa para plataformas diferentes. Geralmente, aplicações multiplataformas podem ser pouco eficientes. Pois elas podem requerer processos redundantes ou armazenamento de arquivos em pastas de vários sistemas que elas dão suporte (TECHNOPEDIA, 2017) .

2.3 JavaScript

JavaScript é a linguagem de programação da Web. A esmagadora maioria dos sites modernos usa JavaScript e todos os navegadores modernos - em desktops, consoles de jogos, *tablets* e *smartphones* - incluem tradutores de JavaScript, tornando JavaScript a linguagem de programação mais onipresente do histórico (FLANAGAN, 2011). Ela roda nos mais diferentes navegadores, como Google Chrome, Firefox, Safari, Microsoft Edge e Internet Explorer. Diferentes navegadores têm diferentes interpretadores que executam código JavaScript (ANTHONY MURRAY NATHANIEL, 2017).

O nome ”JavaScript” é realmente um pouco enganador. Exceto por uma semelhança sintática superficial, o JavaScript é completamente diferente da linguagem de programação Java. E o JavaScript há muito tempo superou suas raízes de linguagem de script para se tornar uma linguagem de propósito geral robusta e eficiente. A versão mais recente da linguagem, ECMAScript6, define novos recursos para o desenvolvimento sério de software em grande escala (FLANAGAN, 2011).

Segundo Flanagan (2011), o JavaScript foi criado na Netscape [1] nos primeiros dias da Web e, tecnicamente, ”JavaScript’ é uma marca comercial licenciada da Sun Microsystems (agora Oracle) usada para descrever Implementação da língua do Netscape (agora Mozilla). Netscape enviou a linguagem para padronização para a ECMA - a Associação Europeia de Fabricantes de Computadores e devido a problemas de marca registrada, a versão padronizada da linguagem estava presa com o nome estranho ”ECMAScript”. Para os mesmos motivos da marca registrada, a versão da linguagem da Microsoft é formalmente conhecida como ”JScript”. Na prática, todo mundo chama a linguagem de JavaScript.

Chinnathambi (2016), apresenta uma pequena lista de recursos que o JavaScript oferece para adicionar interatividade em uma página web, como pode ser visto a seguir:

- Ouvir eventos como um *click* do *mouse* e fazer algo;
- Modificar o HTML e CSS de uma página antes que a mesma seja carregada;

[1] Netscape Communications Corporation é uma empresa de serviços de computadores nos EUA

- Criar jogos impressionantes, como ”Cut the Rope”[2] que funciona no navegador;
- Comunicar dados entre o servidor e o navegador;
- Permitir a interação com uma *webcam*, microfone e outros dispositivos.

De acordo com Zakas (2016), os recursos principais da linguagem JavaScript são definidos no padrão ECMA-262. A linguagem é definida no padrão ECMAScript. JavaScript nos navegadores e em Node.js é atualmente um super conjunto de ECMAScript. Navegadores e Node.js adicionam mais funcionalidades através de objetos e métodos adicionais. O contínuo desenvolvimento do ECMA-262 é vital para o sucesso de JavaScript como um todo.

Ainda segundo Zakas (2016), ECMAScript 6 ou ES6, atingiu um *status* completo em 2015 e foi formalmente apelidado ”ECMAScript 2015”. Porém o nome mais popular conhecido pelo desenvolvedores é ECMAScript 6. Os recursos variam largamente de objetos e padrões completamente novos para mudanças de sintaxe e novos métodos em objetos existentes. A mais recente versão, ES7, contém somente dois novos recursos e foi ratificada em junho de 2016 (ANTHONY MURRAY NATHANIEL, 2017).

2.4 Framework versus Biblioteca

Para Johnson & Foote (1988), citado por Ferreira (2005), um *framework* é um conjunto de classes que incorpora um projeto abstrato para soluções de uma família de problemas associados.

De acordo com Johnson (1991), citado por Torezani (2015), um *framework* é um conjunto de objetos que colaboram com o objetivo de atender a um conjunto de responsabilidades para uma aplicação específica ou um domínio de aplicação.

Enquanto que uma biblioteca contém classes separadas que podem ser usadas independentemente umas das outras: o usuário instancia as classes e chama os métodos dela. O uso dessas bibliotecas são muito semelhantes ao uso de módulos e bibliotecas de subrotina . Bibliotecas de classe são principalmente focadas em reuso de código, raramente em análise e projeto (FERREIRA et al., 2005).

Segundo Viljamaa (2001), citado por Ferreira (2005), quando se usa bibliotecas, o código da aplicação é responsável pelo controle de fluxo. Já nos *frameworks*, o método principal está contido dentro dele e ele chama o código da aplicação e vice-versa.

[2] É um jogo eletrônico de quebra-cabeça desenvolvido pela ZeptoLab e publicado pela Chillingo

2.5 React

Segundo Hudson (2016), atualmente é dificil trabalhar na web sem ter ouvido falar algo sobre React. Desenvolvido pelo Facebook, Airbnb está usando, assim como Netfix, Uber e outras empresas .

Conforme Robbestad (2016), React não um *framework*. React representa o V no padrão de design MVC (Modelo-Visão-Controlador). Ele é uma biblioteca de Javascript para construção de interfaces de usuários que pode ser combinada com *frameworks* como AngularJS, Ember e Meteor ou em conjunto com outras bibliotecas Javascript, por exemplo o Knockout.

React é baseado em componentes que podem ser encapsulados gerenciando seu próprio estado. Uma vez que a lógica dos componentes está escrita em Javascript em vez de modelos, pode-se facilmente passar dados através do aplicativo e manter o estado fora do DOM.

2.5.1 DOM Real e Virtual DOM

Segundo Vipul (2016), React é fundamentado na ideia que a manipulação do DOM (Modelo de Objeto de Documento) é uma operação custosa e que deve ser minimizada. Eisenman (2015), diz que para renderizar interfaces interativas de usuário em um navegador, desenvolvedores devem editar o DOM. Isto é um passo que tem um significante impacto na performance.

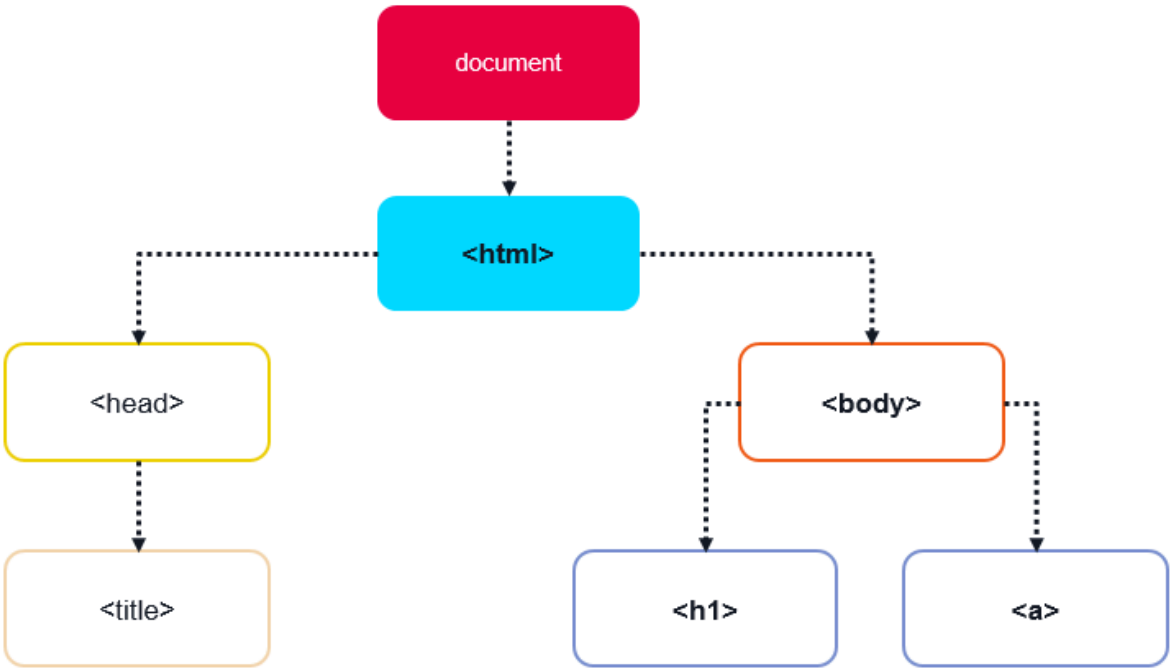
Ainda de acordo com Vipul (2016), React também é declarativo. Quando há alguma mudança de dados, percebe conceitualmente o botão de atualização e sabe atualizar somente as partes do componente que sofreu mudança.

Para um entendimento mais claro de como o React realiza essas alterações eficientemente, cabe-se explicar um pouco do funcionamento do DOM Real e o Virtual DOM, como pode ser apresentado a seguir.

O Modelo de Objeto de Documento (DOM) é uma interface de programação para documentos HTML, XML e SVG . Ele fornece uma representação estruturada do documento como uma árvore, conforme a Figura 1. O DOM define métodos que permitem acesso à árvore, para que eles possam alterar a estrutura, estilo e conteúdo do documento. O DOM fornece uma representação do documento como um grupo estruturado de nós e objetos, possuindo várias propriedades e métodos. Os nós também podem ter manipuladores de eventos que lhe são inerentes, e uma vez que um evento é acionado, os manipuladores de eventos são executados. Essencialmente, ele conecta páginas web a *scripts* ou linguagens de programação (WOOD et al., 2004).

Para renderizar um documento como uma página HTML, a maioria dos navega-

Figura 1 – Estrutura de árvore DOM



dores da Web usa um modelo interno semelhante ao DOM. Os nós de cada documento são organizados em uma estrutura de árvore, chamada árvore DOM, com o nó mais alto chamado ”Objeto de documento”. Quando uma página HTML é renderizada nos navegadores, o navegador baixa o HTML para a memória local e o analisa automaticamente para exibir a página na tela.

Quando uma página da Web é carregada, o navegador cria um DOM da página, que é uma representação orientada a objetos de um documento HTML, que atua como uma interface entre o JavaScript e o próprio documento e permite a criação de páginas da web dinâmicas (WOOD et al., 2004).

De acordo com Eisenman (2015), o Virtual DOM age como uma camada entre a descrição do desenvolvedor de como as coisas deveriam ser vistas e o funcionamento final para atualmente renderizar a aplicação dentro da página. Em vez de renderizar diretamente as mudanças na página, React computa as mudanças necessárias usando uma versão em memória do DOM real e re-renderiza somente o que é preciso.

Adicionalmente a isso, segundo Anthony Murray Nathaniel (2017), React trabalha diferentemente de muitos outros *frameworks frontend* JavaScript que ao invés de trabalhar com o DOM do navegador, ele contrói uma representação virtual do DOM. Possibilitando assim que desenvolvedores não manipulem diretamente o DOM real e sim o Virtual DOM e deixando a responsabilidade de gerenciar as alterações no DOM do navegador para si.

Anthony Murray Nathaniel (2017), afirmam que o Virtual DOM é uma árvore de

objetos JavaScript que representam o DOM real. Uma das razões interessantes para usar o Virtual DOM é a API oferecida. Além disso, quando se usa o Virtual DOM o desenvolvedor codifica como se estivesse recriando por inteiro o DOM a cada atualização.

A ideia de recriar o Virtual DOM a cada mudança de estados não torna o processo lento, pois Virtual DOM do React vem com importante implementação de otimizações de performance fazendo isso muito rápido. Essa rapidez é garantida, pois é utilizado eficientemente *diffing algorithm* para saber o que vai ser mudado, simultaneamente as subárvores do DOM são atualizadas e as essas atualizações são feitas em lote (ANTHONY MURRAY NATHANIEL, 2017).

2.5.2 JSX

Segundo Vipul (2016), JSX ou JavaScript XML é uma sintaxe XML que constrói a marcação em componentes React. Já de acordo com Holmes (2015), JSX é uma extensão XML para a especificação ECMAScript, combinando o componente lógico (JavaScript) e a marcação (DOM ou Native UI) dentro de um simples arquivo, ou seja, em vez de separar artificialmente as tecnologias ao colocar marcação e lógica em arquivos separados, o React separa as preocupações com unidades ligeiramente acopladas, chamadas ”componentes”que contêm ambos.

React funciona sem JSX, mas usando JSX os componentes React ficam mais legíveis e interpretativos semelhante a uma estrutura de algum outro elemento HTML. Além disso, reduz consideravelmente a quantidade de código escrito (VIPUL, 2016). Pode-se observar a distinção entre um componente escrito sem JSX nos códigos a seguir:

```
render: function(){
        return React.createElement('div',null,'Componente sem JSX')
}
```

E outro com JSX, conforme a seguir:

```
render: function(){
        return <div>Componente com JSX</div>
}
```

Vale lembrar que JavaScript não entende JSX, faz-se necessário converter o código escrito JSX para JavaScript antes de ser executado no ambiente. Para essa tarefa de conversão JSX para JavaScript, tem-se o Babel[3] que a realiza automaticamente.

[3] É um compilador JavaScript

Como pode ser visto acima nos dois exemplos, os benefícios de usar JSX são a legibilidade e fácil entendimento do código. A similaridade entre JSX e HTML significa que não desenvolvedores, como por exemplo, UI e UX **designers** em um time pode contribuir para um projeto usando JSX (VIPUL, 2016).

2.5.3 Componentes

Para Masiello (2017), constrói-se aplicações usando componentes modificáveis e modulares. Estes componentes representam parte da interface visual e são renderizados como tal. Na sua forma mais simples, são uma descrição de como renderizar.

Os componentes permitem que a interface do usuário, ou uma tela, seja divida em peças independentes e reutilizáveis pensando em cada peça isoladamente. Conceitualmente, os componentes são como funções de JavaScript. Eles aceitam entradas arbitrárias (chamados de ”props”) e retornam os elementos de React descrevendo o que deve aparecer na tela (OPENSOURCE, 2018a).

Existem duas formas para criação de um componente, ou seja, definir uma instância de ReactComponent. A primeira, é usando o método React.createClass() e a segunda, é usando classes do ES6 (ANTHONY MURRAY NATHANIEL, 2017).

No código que segue, tem-se um exemplo de um componente React. Esta função é um componente React válido porque aceita um único argumento de ”props”com dados e retorna um elemento React. Chamam-se esses componentes de ”funções”, porque são literalmente funções de JavaScript (OPENSOURCE, 2018a).

```
1 function exibeTitulo(props){
2         return <h1>Iniciando com React e {props.titulo}</h1>
3 }
```

Uma outra forma de representar componentes é utilizando classes ES6 (DEVELOPERMOZILLA, 2018), conforme o próximo código. São chamados de componentes de classes, pois têm recursos adicionais como: adicionar variáveis de estados e incorporar métodos de ciclo de vida (FACEBOOK, 2018).

```
1 class Bloco extends React.Component{
2  this.state = { variavel:1      }
3  render(){
4    return <h1>Iniciando com React e {this.props.titulo}</h1>
5  }
6 }
```

Vale ressaltar que o método *render()* é somente obrigatório em um componente React, ou seja, em componentes de classe, faz-se necessária a utilização do método render(). Sendo omitido em componentes de função.

Contudo, tem-se a seguinte ideia: quando tiver um componente onde não há variáveis de estado e nem métodos além do render(), deve-se criar um componente de função, caso contrário, cria-se um componente de classe.

2.5.4 Comportamento da aplicação: Props e State

Para Hudson (2016), quando se usa algum componente React é possível passar alguns dados de entrada para serem trabalhados nesse componente. Estas propriedades são chamadas de ”props” e os valores são somente leitura que definem pontos iniciais para um componente. Além disso, são imutáveis. Eles não devem ser atualizados pelo componente aos quais eles são passados (VIPUL, 2016). Se considerar um componente como uma ”função”, a props é equivalente a ”parâmetros” de uma função (ANTHONY MURRAY NATHANIEL, 2017).

Segundo Robbestad (2016), States são semelhantes a props, mas são destinados a variáveis que são disponíveis somente dentro de um componente. E são privados e controlados totalmente pelo próprio componente (OPENSOURCE, 2018a). Além disso, pode e será alterado dependendo das interações com o mundo exterior (VIPUL, 2016). Por fim, usa-se state quando se tem componentes *stateful*, ou seja, componentes que precisam manter dados dinâmicos (ANTHONY MURRAY NATHANIEL, 2017).

2.5.5 Métodos de Ciclos de Vida de um Componente

Cada componente tem vários métodos do ciclo de vida que podem ser reescritos para executar código em momentos específicos. Métodos prefixados com **will** são chamados logo antes de algo acontecer e o prefixados por **did** são chamados logo após algo acontecer (FACEBOOK, 2018).

Os métodos de ciclo de vida dos componentes podem ser categorizados em *Mounting*, *Updating*, *Unmounting* e *Error Handling* (FACEBOOK, 2018). A seguir serão apresentados somente os métodos de Mounting.

Os métodos que compõem esta categorias são chamados quando uma instância de um componente está sendo criada e inserida dentro do DOM. Pode-se listar a seguir:

constructor() é chamado antes de um componente React ser montado. Ao implementar o construtor para uma subclasse React.Component, faz-se necessário, antes de qualquer declaração, chamar **super(props)** para ter acesso às propriedades passadas para o componente, pois se não, **this.props** será indefinido no construtor, o que pode levar a erros. Além disso, é o lugar certo para inicializar states atribuindo um objeto a **this.state** (FACEBOOK, 2018).

componentWillMount() é invocado imediatamente antes de ocorrer a montagem do componente. É chamado antes de **render()** (FACEBOOK, 2018). Além disso, se dentro deste método for chamado **setState()**[4] não irá causar uma re-renderização do componente, pois o método render() receberá o state modificado (GACKENHEIMER, 2015).

O método **render()** é obrigatório em qualquer componete de classe e quando chamado ele analisa **this.props** e **this.state** para retornar elementos React , String ou números. Mais informações podem ser vistas no site oficial (FACEBOOK, 2018). Ele, também, deve retornar um único elemento filho, como por exemplo um elemento JSX, mas também pode retornar null ou false para indicar que não precisa renderizar nada (ROBBESTAD, 2016).

componentDidMount() é invocado imediatamente após um componente está montado. Vale ressaltar que chamar setState() neste método desencadeará uma renderização adicional, mas ocorrerá antes do navegador atualizar a tela (FACEBOOK, 2018). Pode-se fazer uso deste método para obter as informações dinâmicas que precisam ser exibidas em um componente após a renderização inicial do mesmo (VIPUL, 2016).

2.6 React Native

Segundo Eisenman (2015), React Native é um *framework* JavaScript para escrever real, nativamente renderizando aplicações móveis para iOS e Android. É baseado no React, biblioteca JavaScript do Facebook para construir interfaces de usuário, mas em vez de visar o navegador, visa plataformas móveis, ou seja, permite aos desenvolvedores escrever aplicações móveis que absorvem o visual verdadeiramente ”nativo”, tudo do conforto que a bilioteca JavaScript disponibiliza.

De acordo com Maharjan (2018), React Native é um *framework* moderno que permite que o JavaScript seja executado e interaja com iOS e *smartphones* Android da mesma maneira que o código nativo. Fornece oportunidades para organizações que desejam consolidar suas arquiteturas de aplicativos e adota uma abordagem ”aprenda uma vez, escreva em qualquer lugar”para o desenvolvimento de aplicativos para dispositivos. Permitindo criar uma UI móvel rica a partir de componentes declarativos (FACEBOOKNATIVE, 2018a).

Com o React Native, não se cria um ”aplicativo web móvel”, um ”aplicativo HTML5”ou um ”aplicativo híbrido”. Constrói-se um aplicativo móvel real que é indistinguível de um aplicativo criado usando Objective-C ou Java. O React Native usa os mesmos blocos de construção de UI fundamentais que os aplicativos regulares iOS

[4] Método para atribuir valor a variáveis de estado

e Android. Coloca-se apenas esses blocos de construção juntos usando JavaScript e React (FACEBOOKNATIVE, 2018a).

React Native é como React, mas usa componentes nativos em vez de componentes da Web como blocos de construção (FACEBOOKNATIVE, 2018a).

2.6.1 Arquitetura

Para um entendimento mais detalhado de como funciona React Native, cabe compreender três arquitetura: arquitetura de tempo de execução, arquitetura de construção e arquitetura de depuração(MAHARJAN, 2018).

Como o JavaScript não é uma linguagem que funciona nativamente no *smartphone*, deve-se usar uma técnica chamada *Bridging* para permitir que JavaScript seja executado e se comunicar com o processador de *smartphones*. Isso significa que o *smartphone* deve executar algo chamado 'JavaScript Core'. Que é basicamente um "ambiente de tempo de execução". Ambiente de tempo de execução não é nada mais que algum código que seja executado em cima do sistema operacional que permitirá que o JavaScript seja executado. Isso é feito carregando um código nativo que pode executar JavaScript e, em seguida, carregar algum JavaScript nela (que é enviado para o *smartphones* em um processo de compilação) (HEARD, 2018).

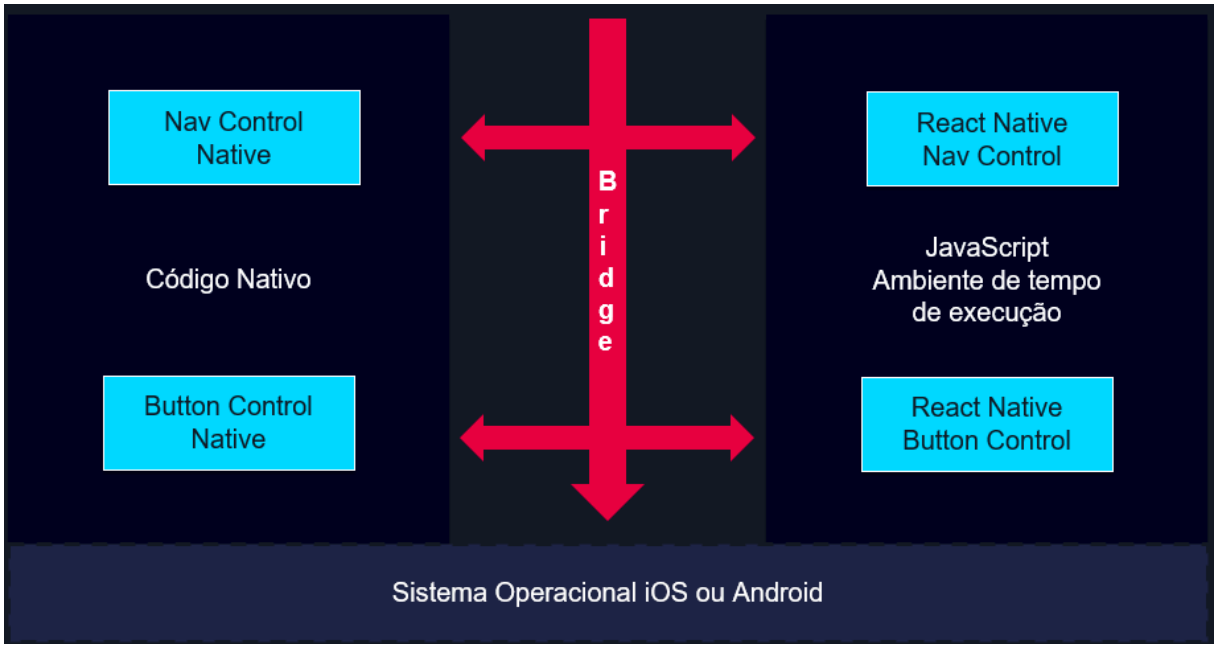
Para que o JavaScript execute um aplicativo nativo, ele ainda deve usar o código nativo subjacente. Na Figura 2, pode-se ver que cada controle React Native irá interagir com um componente de contrapartida nativo. React Native simplesmente permite que a ponte para a comunicação de dois sentidos aconteça. Portanto, o código nativo realmente está operando em seu próprio *thread*, o que significa que, se o código JavaScript do React Native demorar muito, a UI não irá pendurar ou diminuir a velocidade. Ele simplesmente liga de volta pela ponte usando eventos (MAHARJAN, 2018).

Uma vez entendido que o React Native utiliza um *bridging* para realizar a comunicação com a base nativa, cabe entender como isso tudo é configurado, ou seja, como tudo é construído.

Maharjan (2018) diz, que ao construir uma aplicação React Native, dois princípios devem ser seguindos:

- Ele executará um servidor web de Node localmente, que publica um arquivo de carga útil que contém todo o código de JavaScript do React Native;
- Em seguida, ele irá criar um projeto de aplicativo nativo puro (para iOS ou Android) e instalar isso no telefone. Este projeto está configurado para ter o ambi-

Figura 2 – Arquitetura central do React Native



ente de tempo de execução do JavaScript e em seguida, para baixar a carga útil gerada anteriormente no passo 1.

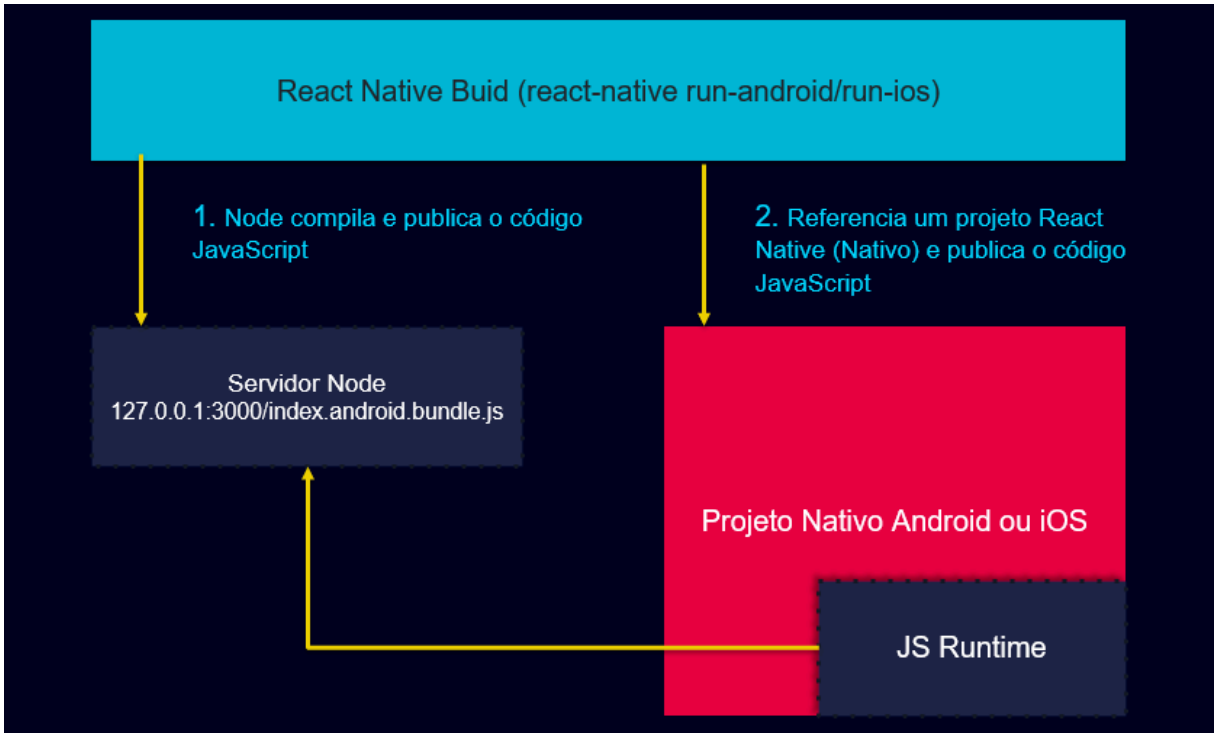
Esta carga útil é então entregue diretamente no ambiente de tempo de execução do JavaScript através do processo de compilação do React Native. Agora, tem-se o código de contêiner nativo e JavaScript que são necessários para alimentar o aplicativo React Native inteiro (HEARD, 2018). A Figura 3, mostra os dois passos: o servidor web com a carga útil e o projeto entregue ao *smartphone*.

De acordo com Heard (2018), React Native puxa o código JavaScript e o JavaScript não precisa ser compilado ou construído (é apenas escrito). Aqui está a maior vantagem do React Native. Em qualquer ponto 'depois' desencadeando uma compilação - que é lenta - pode-se recarregar o JavaScript atualizando o aplicativo React Native. React Native é embarcado com uma ferramenta *'debug time'*, disponível no menu do *smartphone*. Isso significa que se pode modificar um projeto React Native muito rápido. Essa capacidade de recarregar é chamada de *Hot Reloading* ou 'Recarga a quente'.

Para o processo de depuração de uma aplicação React Native, precisa-se fazer os seguintes passos:

- Definir a localização do código-fonte;
- Abrir um navegador que o depurador pode publicar informações de depuração .

Figura 3 – Processo de deploy da aplicação



React Native permite prototipagem rápida e uma velocidade inicial muito alta. As características básicas são relativamente fáceis de implementar. Uma vez que React Native usa JavaScript, os desenvolvedores podem trabalhar de forma mais rápida e eficiente, uma vez que não precisam reiniciar o aplicativo desenvolvido após cada atualização, eles podem simplesmente atualizar a página de visualização (MAHARJAN, 2018).

Mais informações de Debugging podem ser encontradas na documentação oficial Debugging [5].

2.6.2 Componentes React Native

Conforme Eisenman (2015), componentes React Native são largamente os mesmos componentes do React, com algumas diferenças importantes no que diz respeito a renderização e estilização. São, também, simples, reusáveis e objetos semelhantes a função que permitem ao desenvolver uma liberdade de definir o que deve ser renderizado (HOLMES; BRAY, 2015).

Em React para Web são renderizados elementos HTML normal, como por exemplo: , <p>, , <a>, etc. Com React Native, todos esse elementos são substituídos por componentes React específicos de cada plataforma. O mais básico é o

[5] https://facebook.github.io/react-native/docs/debugging.html

multiplataforma <View>, um flexível elemento UI que pode ser análogo a uma . No iOS, por exemplo, o componente <View> é renderizado para uma UIView, enquanto que no Android ele será renderizado para uma View (EISENMAN, 2015). Na Tabela 1, pode-se ver um comparativo de elemento utilizados no React com a equivalência no React Native de acordo com o objetivo de uso.

Tabela 1 – Equivalência de elementos React e React Native

Objetivo de uso	**React**	**React Native**
Organização de conteúdo		<View>
Parágrafo de um texto		<Text>
Listagens	,	<ListView>
Exibir imagens	<img>	<Image>

No código que segue é exibido um exemplo de um componente React Native mostrando na tela a mensagem **Renderizando um componente**, fazendo uso dos componentes <View> e <Text> utilizando JSX. Vale ressaltar que o método render() deve retornar somente um único elemento pai.

```
1 render(){
2         return (
3         <View>
4                 <Text>Renderizando um componente</Text>
5         </View>
6     )
7 }
```

De acordo com Eisenman (2015), quando se trabalha com React para Web, tipicamente são usados arquivos *stylesheet*[6] separados, que podem ser escritos em CSS, SASS ou LESS. React Native propõe uma abordagem radicalmente diferente, trazendo estilos inteiramente para dentro do mundo JavaScript e forçando a vinculação de estilos de objetos explicitamente para componentes.

Com React Native, não se usa uma linguagem especial ou sintaxe para definir estilos. Estiliza-se o aplicativo usando apenas o JavaScript. Todos os componentes principais aceitam um props chamado **style**. Os nomes e valores de estilo geralmente correspondem à forma como o CSS funciona na web, exceto os nomes são escritos usando a prática *camelCase*[7], por exemplo, **backgroundColor** em vez de **background-color** (FACEBOOKNATIVE, 2018b).

[6] Folhas de estilos

[7] É a denominação em inglês para a prática de escrever palavras compostas ou frases, onde cada palavra é iniciada ou não com maiúsculas e unidas sem espaços

Eisenman (2015), diz que estilos em linha são o caminho mais simples, sintaticamente, para estilizar um component em React Native, embora não seja geralmente o melhor caminho para estilos de componentes complexos. A sintaxe de estilo em linha é a mesma do React para o *web*.

A prop **style** pode ser um objeto JavaScript. Isso é o que se costuma usar para estilizar elementos, como pode se ver no exemplo a seguir:

```
renderTopico(options){
        return (
        <View style={{alignContent:'flex-start'}}>
                <Text style={{color:'#fff'}}>Estilizando um
                   componente</Text>
        </View>
    )
}
```

À medida que um componente cresce em complexidade, muitas vezes é mais limpo usar **StyleSheet.create** para definir vários estilos em um único lugar. O construtor StyleSheet.create é opcional, mas provê algumas vantagens. Garante que os valores sejam imutáveis. Por colocá-lo no final do arquivo, também, é garantido que eles são somente criados uma vez por aplicação e não em cada renderização (EISENMAN, 2015). A seguir, exemplo de uso do StyleSheet.create.

```
render(){
        return (
        <View style={styles.content}>
                <Text style={styles.titulo}>Usando StyleSheet.create
                   para estilizar</Text>
        </View>
    )
}
const styles = StyleSheet.create({
  content:{
    flex:1,
    paddingBottom: 10
  },
  titulo:{
        fontSize:10
  }
})
```

Pode-se, também, passar uma lista de estilos - o último estilo na lista tem precedência, para que possa usar isso para herdar estilos (FACEBOOKNATIVE, 2018b).

Desta forma é possível combinar ou contatenar estilos definidos diferentes em um único elemento. No código que segue é apresentado um exemplo do uso de lista de estilos.

```
render(){
        return (
        <View style={[styles.content, styles.box]}>
                <Text style={[styles.titulo,{color:'#6525'}]}>
                   Contatenando estilos</Text>
        </View>
    )
}
```

Uma das maiores mudanças quando se trabalha com estilos em React Native é o posicionamento. CSS suporta várias técnicas de posicionamento. A abordagem do React Native é mais focada no flexbox (EISENMAN, 2015).

Um componente pode especificar o layout de seus filhos usando o algoritmo Flexbox. Flexbox foi projetado para fornecer um layout consistente em diferentes tamanhos de tela (FACEBOOKNATIVE, 2018b). E é um modo de layout CSS3. Além disso, funciona da mesma maneira no React Native, como acontece no CSS na Web, com algumas exceções. Os padrões são diferentes, com **flexDirection** com parão para **column** em vez de **row** e o parâmetro flex apenas suportando um único número (FACEBOOKNATIVE, 2018b). Normalmente, usa-se uma combinação de **flexDirection**, **alignItems** e **justifyContent** para alcançar o layout ideal.

2.6.3 AsyncStorage

De acordo com Eisenman (2015), quando se constrói aplicações móveis, naturalmente é necessário aproveitar vantagens de API's de plataformas específicas. React Native faz isso para acessar recursos como: câmera, localização e armazenamento persistente. Estas API's são feitas disponíves para React Native através de módulos incluídos, que provem facilidade de uso de interfaces assincrónas JavaScript para essas funções.

Claro que React Native não incorpora todas as funcionalidades por padrão. Algumas API's requerem que seja escrito seu próprio módulo ou que faça uso de módulos escritos por outros e que podem ser encontradas na comunidade React Native.

Conforme Holmes (2015), React Native provê uma abstração sobre o mecanismo de armazenamento local para que não haja preocupação como salvar dados no dispositivo tanto no iOS como no Android. Esse tipo de armazenamento é um armazenamento de chave e valor que é global na aplicação, bem semelhante ao LocalStorage

na Web conhecido como AsyncStorage. Como o nome sugere, é assíncrono (EISENMAN, 2015).

No iOS, o AsyncStorage é suportado por código nativo que armazena pequenos valores em um dicionário serializado e valores maiores em arquivos separados. No Android, o AsyncStorage usará **RocksDB**[8] ou SQLite com base no que está disponível. O código JavaScript do AsyncStorage é uma fachada que fornece uma API de JavaScript clara, objetos de erro reais. Cada método na API retorna um objeto **Promise**[9] (ASYNCSTORAGE, 2018).

Sua utilização é feita a partir da inclusão na lista de módulos importados do React Native. Onde a chave de armazenamento usada pelo AsyncStorage pode ser qualquer string; é costume usar o formato @AppName:key, conforme código abaixo:

```
1 import {View, Text, AsyncStorage} from 'react-native'
2 .
3 .
4 .
5 var STORAGE_KEY = '@Vitrine:eventos'
```

Para salvar dados, usa-se o método **setItem(key: string, value: string)**. Este, define o valor de uma chave e invoca um retorno de chamada após a conclusão. Conforme o exemplo a seguir:

```
1 AsyncStorage.setItem(STORAGE_KEY, 'Valor de teste')
2 .then( () => console.log('Dados salvos com sucesso.'))
3 .catch(error => console.log('Erro ao salvar dados'))
```

Para recuperar dados, usa-se o método **getItem(key: string)**. Obtém um item para uma chave e invoca um retorno de chamada após a conclusão.

```
1 AsyncStorage.getItem(STORAGE_KEY)
2 .then( value => {
3         if(value != null){
4         console.log('Salvo no storage local.')
5     }
6 })
7 .catch(error => console.log('Erro ao recuperar dados'))
```

AsyncStorage, também, oferece métodos para deletar chaves, mesclar chaves e muito mais que pode ser visto na documentação oficial [10].

[8] É uma loja de persistência chave-valor para ambientes de armazenamento rápidos. Disponível em http://rocksdb.org/

[9] É um objeto usado para processamento assíncrono

[10] https://facebook.github.io/react-native/docs/asyncstorage.html

2.6.4 Persistência

De acordo com Waikar (2015), uma das decisões iniciais de um desenvolver ou de um time ao produzir uma aplicação de qualidade é escolher de um mecanismo de persistência de armazenamento.

Para Deka (2017), uma base de dados não relacional é um base de dados que não incorpora o modelo tabela/chave como o RBDMS [11] promove. Por outro lado, ela usa pares de chave e valor em documentos para armazenar dados. A mais popular emergente base de dados não relacional é chamada de NoSQL.

Bases de dados não relacionais usam modelos de dados agregados que habilitam o uso de dados estruturados para resolver problemas de domínio como modelado por desenvolvedores. Um agregado é uma coleção de dados que auxilia na manipulação do banco de dados armazenando informações sobre um grupo quando a unidade de dados reside em alguma máquina (DEKA, 2017).

2.7 Google Firebase

É uma plataforma poderosa do Google para armazenamento e sincronização de dados em tempo real. Provê uma variedade de soluções de desenvolvimento para acelerar a integração de recursos baseados em nuvem em aplicativos móveis e web (SMYTH, 2017). Além disso, provê infraestrutura necessária para construir grandes aplicativos, dando a possibilidade de crescimento e ganho com negócio de sucesso (MORONEY, 2017).

Segundo Cheng (2017), Firebase tem uma estrutura de base de dados única que é diferente de outras bases de dados. Cada base de dados Firebase é guardada como uma árvore de objeto JSON. Esta estrutura de árvore é muito flexível para todos os tipos de dados, como pode ser visto na Figura 4.

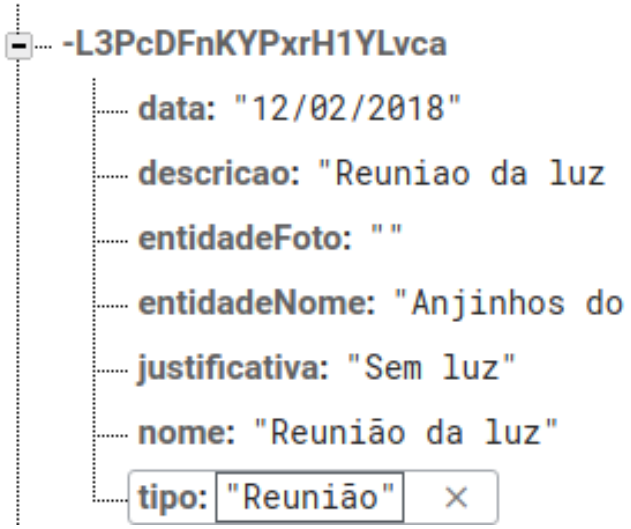
2.7.1 Produtos

Esta plataforma oferece serviços ou produtos para desenvolver e testar um aplicativo, objetivando o desenvolvimento de aplicativos com alta qualidade e ótimo tempo de resposta.

De acordo com Moroney (2017), Firebase tem 8 tecnologias que estão criadas para elevar a experiência de desenvolvimento de um aplicativo. Entretanto, atualmente têm mais dois produtos para auxiliar o desenvolvimento. Pode-se citar os produtos

[11] Um sistema de gerenciamento de banco de dados relacional (RDBMS) é um sistema de gerenciamento de banco de dados (SGBD) baseado no modelo relacional inventado por Edgar F. Codd, do Laboratório de Pesquisa San Jose da IBM

Figura 4 – Estrutura de base de dados Firebase



Cloud Firestore e o **Crashlytics**. Os principais serviços disponíveis estão citados a se abaixo:

Realtime Database é um banco de dados baseado em NoSQL hospedado na nuvem. Os dados são armazenados como JSON(JSON, 2018) e sincronizados em tempo real com todos os clientes conectados e está disponível quando não há conectividade de rede através de um cache local. Não há código do lado do servidor e níveis de acesso à base de dados; Todo código final está no cliente.

Sempre que há mudanças de dados na base, eventos são disparados no código cliente e podem ser manipulados e atualizados pela interface responsável. Ele conta com regras de linguagens baseadas em expressão, chamada de Firebase Realtime Database Security Rules, que define como os dados estruturados e como os usuários têm direitos para esses dados (MORONEY, 2017).

Firebase Authentication fornece serviços de *backend*, *Kit* de desenvolvimento fácil de usar e bibliotecas de interface de usuário prontas para autenticar usuários em aplicativos. Ele oferece suporte à autenticação por meio de senhas, números de telefone e provedores de identidade federados como Google, Facebook, Twitter entre outras. Além disso, ele integra com serviços do Firebase como o Realtime database onde é possível controlar como acessar quais dados (MORONEY, 2017).

Cloud Storage é um serviço de armazenamento de objetos poderoso, simples e econômico. Com os SDKs do Firebase para Cloud Storage, usa-se a segurança do Google para fazer o upload e o download de arquivos nos aplicativos Firebase, independentemente da qualidade da rede. Usa-se esses SDKs para armazenar imagens, áudio, vídeo ou outros conteúdos gerados pelo usuário. No servidor, é utilizado o Google Cloud Storage para acessar esses mesmos arquivos (FIREBASE, 2018).

Crash Reporting fornece relatórios detalhados dos erros no aplicativo. Para Moroney (2017), a razão número 1 de comentários ruins na *App Store* e *Play Store* são falhas inesperadas. Descobrir por que um aplicativo falhou e corrigir rapidamente é primordial, mas é um processo dificultoso se o app roda em um dispositivo que não é sabido o país ou cidade está longe. Crash Reporting ajuda com isso por disponibilizar um rastreamento de todas falhas no Firebase Console.

Test Lab para Android Oferece infraestrutura baseada em nuvem para testar os aplicativos para Android. Uma vez que a queixa mais comum, segundo Moroney (2017), dos desenvolvedores de aplicativos, em particular desenvolvedores Android, é que é muito difícil acessar todos os tipos de dispositivos que os usuários finais utilizam.

Cloud Functions com o Cloud Functions para Firebase, executa-se o código de *backend* automaticamente em resposta a eventos acionados pelos recursos do Firebase e pelas solicitações HTTPS. O código é armazenado na nuvem do Google e é executado em um ambiente gerenciado. Não há necessidade de gerenciar e dimensionar servidores próprios (FIREBASE, 2018).

Além de prover serviços que ajudam na construção de apps, Firebase tem um número de 9 tecnologias que podem ser usadas para auxiliar sistematicamente no engajamento de usuários gerando assim um crescimento ao app. Os principais serviços disponíveis estão citados a seguir:

Para Moroney 2017, no coração do Firebase está o Analytics. É uma solução absolutamente grátis para uso e fácil compreensão. Os recursos do Firebase são integrados ao Analytics, que fornece geração ilimitada de relatórios para até 500 eventos distintos, os quais podem ser definidos com o SDK do Firebase. Com os relatórios do Analytics, é possível entender claramente o comportamento dos usuários e podendo tomar decisões mais bem informadas sobre marketing e otimizações de desempenho do aplicativo (ANALYTICS, 2018).

O Firebase Cloud Messaging (FCM) é uma solução de mensagens entre plataformas que permite o envio confiável de notificações sem custo. Com essa ferramenta é possível notificar um app cliente de que novos e-mails ou outros dados estão disponíveis para sincronização. É possível, também, enviar mensagens de notificação para promover novas interações e a retenção de usuários. Para casos de uso como mensagens instantâneas, uma mensagem pode transferir um *payload* de até 4 KB para um app cliente (MESSAGING, 2018).

Dynamic Links são links que funcionam de maneira personalizada, em várias plataformas, mesmo que o app não esteja instalado. Quando um usuário abre um link dinâmico no iOS ou no Android, ele é levado diretamente ao conteúdo do link no aplicativo nativo do desenvolvedor. Se um usuário abrir o mesmo link dinâmico em

um navegador de computador desktop, ele será levado ao conteúdo equivalente no site do desenvolvedor.

Além disso, o Dynamic Links funciona em todas as instalações de app: se um usuário abre um link dinâmico no iOS ou no Android e não tem o aplicativo instalado, ele pode ser solicitado a instalá-lo. Após a instalação, o aplicativo é iniciado e pode acessar o link (DYNAMICLINKS, 2018).

O Firebase Invites é uma solução para indicações e compartilhamento de apps por e-mail ou SMS. Para personalizar a experiência do usuário do convite ou gerar links de modo programático, usa-se o Firebase Dynamic Links.

O Firebase invites facilita o processo de converter os usuários do seu app em excelentes divulgadores. Além disso, tem como base o Firebase Dynamic Links, o que garante aos destinatários dos links a melhor experiência possível na plataforma deles e nos apps que instalaram (INVITES, 2018).

A AdMob by Google é uma maneira fácil de gerar receita com publicidade segmentada nos aplicativos para dispositivos móveis. Ela é uma plataforma de publicidade móvel que pode ser usada para gerar receita com o aplicativo. O uso da AdMob com o Google Analytics para Firebase fornece recursos adicionais de dados e recursos de análise de aplicativos. O Firebase integra-se à AdMob sem exigir alterações na configuração existente (ADMOB, 2018).

3 ESTUDO DE CASO: VITRINE DE PROJETOS SOCIAIS

O aplicativo, aqui apresentado, foi idealizado objetivando atender às necessidades da APAR[1]. Entidade com trabalho voltado à crianças e adolescentes há 22 anos. Onde a mesma tem problemas de divulgação de eventos[2] para a comunidade onde é localizada e adjacências. Além disso, observou-se que a mesma precisa de ajuda com trabalho voluntário, ajuda financeira ou material permanente[3]. Diante disso, percebeu-se que algumas dessas necessidades também são de outras no Brasil.

Em virtude disso, como proposta de solução para essa problemática, foi escolhida a ferramenta React Native integrada com Firebase para criar um aplicativo nativo permitindo a publicação e exibição de eventos em tempo real das entidades para outras entidades e público em geral. Fazendo, com isso, que pessoas e outras Entidades também conheçam o trabalho destas, podendo a partir daí ganhar doadores, voluntários ou que Entidades possam implatar projetos que uma Entidade já executa, podendo, também, resolver problemas que outra Entidade já passou e resolveu, compartilhando experiências.

Contudo, a ideia principal do aplicativo é conectar pessoas com Entidades e Entidades com Entidades. Na Figura 5, é possivel entender a ideia central do aplicativo.

Figura 5 – Ideia principal do app.

O aplicativo foi criado e testado na plataforma Android, devido à restrições de acesso à plataforma iOS.

[1] Associação de Pais e Amigos Reviver
[2] Podem ser Projetos, Ações, Reuniões, Palestras
[3] Aquele de duração superior a dois anos. Exemplos: mesas, equipamentos de laboratórios, etc.

Cabe conceituar Doador,Voluntário e Entidade para entender o real papel no aplicativo.

- Doador: É aquele pode doar material, dinheiro, ou seja, a pessoa que pode ajudar com recursos que possui;
- Voluntário: É aquele que não é forçado, que só depende da vontade; espontâneo, ou seja, ajuda com trabalho pessoal sem ser pago para isso.
- Entidade: Instituição, sociedade, pessoa jurídica estabelecida para fins específico.

3.1 Modelagem

Nesta seção, utilizando conceitos de processos de engenharia de requisitos, são apresentadas as funcionalidades do aplicativo, ou seja, requisitos funcionais extraídos a partir da descrição mencionada anteriormente na introdução. É de extrema importância a listagem desses requisitos para o início da criação das telas do aplicativo. Além disso, serão apresentados os diagramas de caso de uso para ter um entendimento macro da solução.

3.1.1 Requisitos

Os requisitos listados a seguir surgiram a partir da ideia principal do aplicativo e do que ele poderia ser futuramente e também após conversas com membros da APAR para validá-los e sugerirem novos. Informando a regularidade dos requisitos pensados inicialmente. Diante disso, conclui-se que a aplicação deve permitir:

1. Que um usuário se cadastre como uma Entidade, Voluntário ou Doador.
    - O cadastro inicial não deve ser burocrático, apenas exigindo dados básicos de acesso e necessários para um contato posterior, como: Nome, email, telefone e senha. Para o cadastro de Entidade, um responsável deverá cadastrá-lá. Deve ser algum membro da diretoria. O cadastro deve ser finalizado, informando outras informações da Entidade, após o responsável entrar no aplicativo.
2. Que o usuário responsável pela entidade:
    - Cadastre eventos. Informando qual o tipo do evento (projeto, reunião, mutirão ou palestra), nome, data, descrição, justificativa e uma foto principal;

- Cadastre e exclua tipos de ajudas/doações que a entidade necessita. Informando apenas a descrição;
- Visualize pessoas que mostraram interesse em ser voluntário ou doador. Podendo aprovar ou não um interesse.
- Informe outros dados necessários de identificação da entidade como: uma foto da logomarca ou fachada da sede, CNPJ, endereço, cidade, estado, histórico, banco, conta corrente, agência e dados do representante legal. Tais como: nome, cargo na entidade, endereço e profissão.

3. Que o voluntário ou doador:
    - Visualize entidades cadastradas recentemente;
    - Marque entidade como favorita. Na listagem de entidades deverá ter um botão com a palavra **Seguir** em cada *card*[4] das entidades para o usuário clicar e a partir daquele momento receber informações dessa entidade ;
    - Visualize, em tempo real, eventos cadastrados pela entidades favoritas;
    - Mostre interesse em ser doador ou voluntário. Quando o usuário clicar no nome da entidade deverá ter acesso às informações detalhadas desta, podendo indicar interesse em ser volutário ou doador ou até mesmo seguí-la. Uma vez feito isso, o responsável pela entidade deverá visualizar esse usuário que mostrou interesse;
    - Visualize informações detalhadas de uma entidade. Como dados bancários, responsável legal, tipos de ajuda, histórico, eventos e localização.
    - Visualize informações detalhadas dos eventos.
4. Que um usuário seja responsável por apenas uma Entidade.
5. Que um usuário possa visualizar seus dados e alterar a senha de acesso, caso queira.
6. Que um usuário siga ou deixe de seguir qualquer entidade.

[4] É uma maneira conveniente de exibir conteúdos compostos por diferentes tipos de objetos.

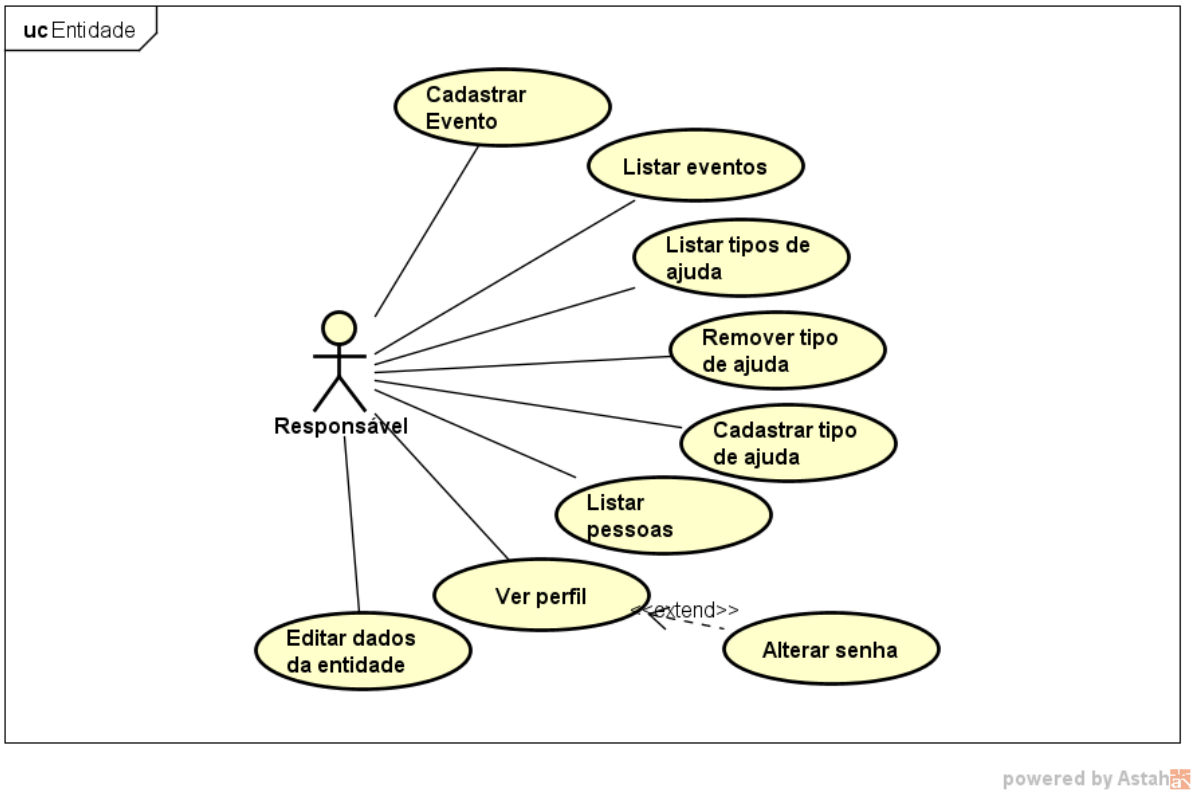
3.1.2 Diagramas

Após a validação dos requisitos, para um entendimento macro do problema, fez-se necessário criar dois diagramas de casos de uso e três de atividade, como segue.

O primeiro diagrama a ser criado foi o de casos de uso para a identificação dos atores envolvidos listando as interações com o aplicativo e modularizando a aplicação em níveis de acesso e funcionalidades.

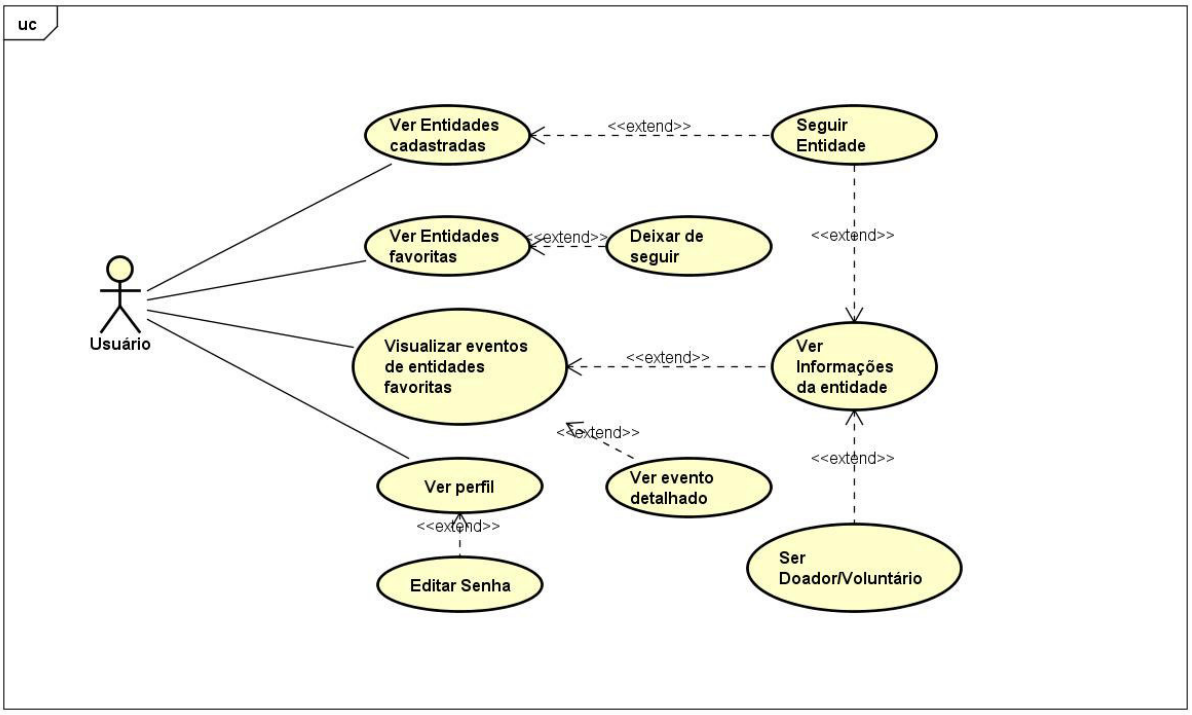
Na Figura 6, são exibidas todas as funcionalidades, listadas no item 2 da listagem anterior, pertinentes ao usuário nomeado aqui de **Responsável**, pois este é o representante digital de uma entidade no aplicativo.

Figura 6 – Caso de uso - Entidade.



Na Figura 7, são exibidas todas as funcionalidades pertinentes aos usuários nomeados aqui de **Doador** e **Voluntário**, representado na figura simplesmente por Usuário, pois ambas as nomeações compartilham de mesmas funções no aplicativo.

Figura 7 – Caso de uso - Usuário.



Os casos de uso apresentados anteriormente ajudaram a identificar quais funcionalidades cada tipo de usuário tem acesso. Entretanto, é preciso ainda ter uma visão do comportamento do aplicativo. Para isso, a Figura 8, apresenta todo o fluxo de atividades desde a inicialização do aplicativo pelo usuário, passando pela verificação de cadastro. Caso o usuário já tenha cadastro, ele será direcionado para a tela de login e consequentemente para a tela de acordo com o seu tipo. Se não tiver cadastro, ele será deverá escolher o tipo de cadastro. Após escolher o tipo de cadastro e preencher todos os dados necessários, esses dados são salvos no Firebase Database. Nesse momento, automaticamente já é criada uma sessão do usuário para que ele não precise passar pela tela de login. Em seguida será apresentada a tela de boas vindas, verificando o tipo de usuário e também será direcionado para tela inicial de acordo com seu tipo.

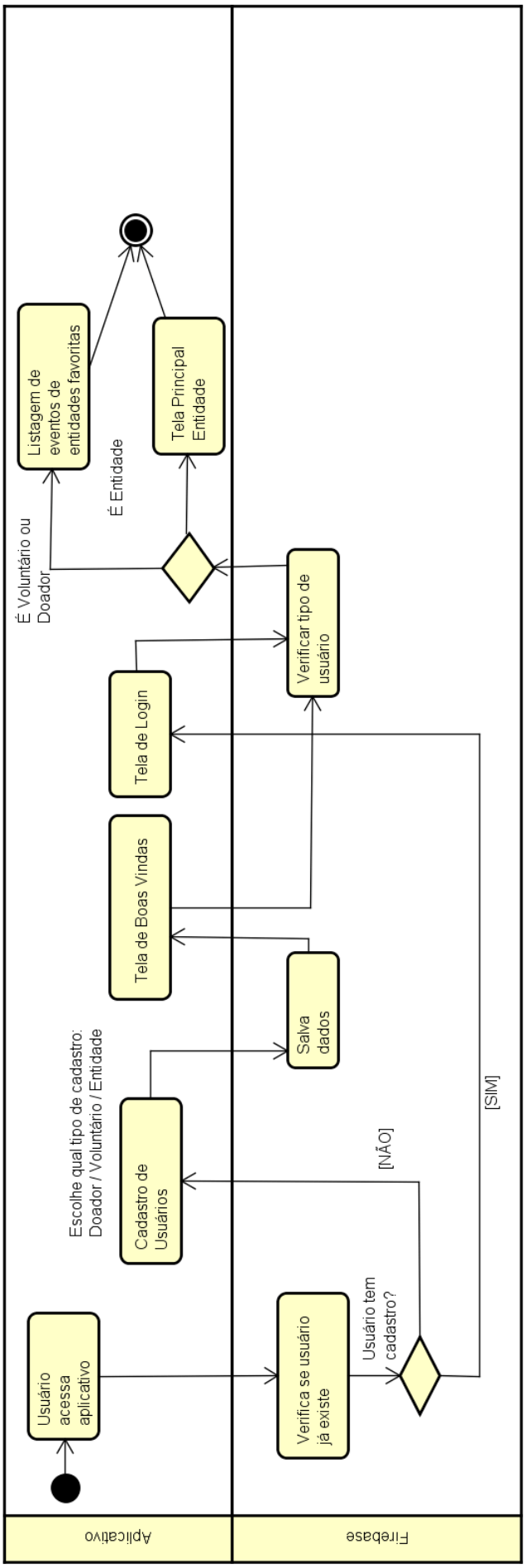
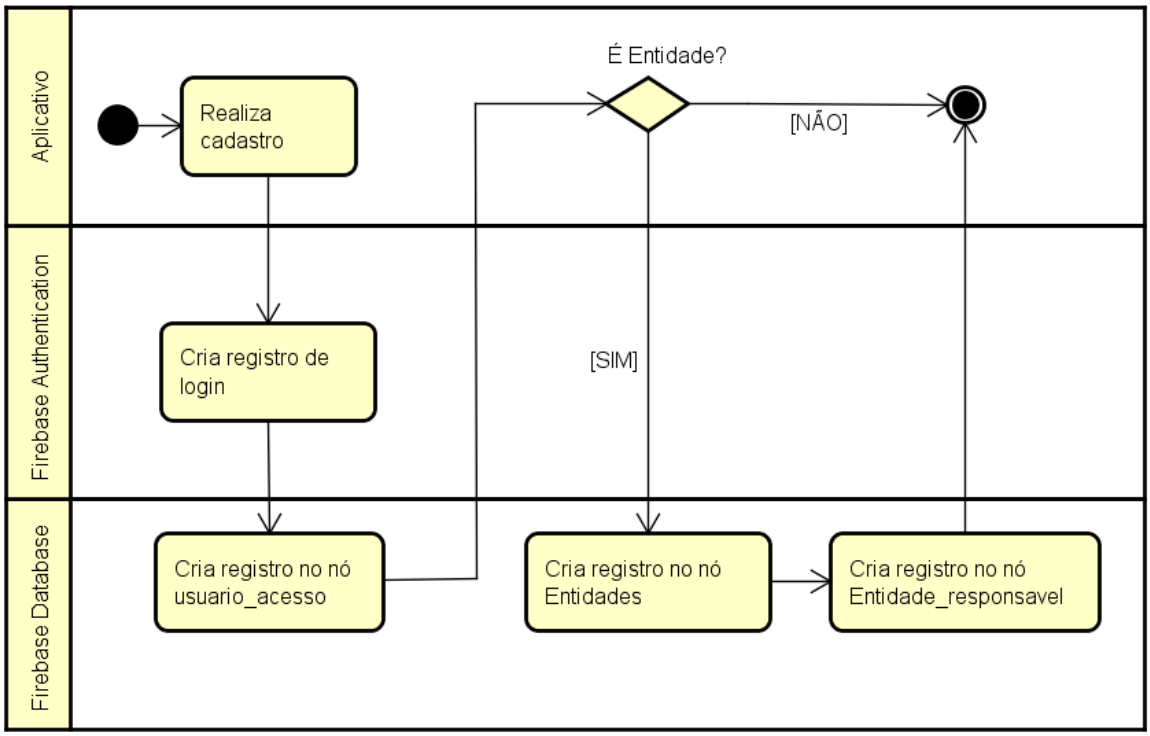


Figura 8 – Fluxo da aplicação.

A Figura 9, explica como é o processo de criação de registro de usuários. Nesse diagrama é possível analisar que o fluxo inicia no aplicativo logo após o usuário realizar o cadastro, em seguida, cria-se um registro no Firebase Authentication. Feito isso, é preciso salvar na **database** um registro referente ao acesso no nó usuario_acesso e o processo é finalizado caso o usuário seja um doador ou voluntário. Caso seja uma entidade, representado pelo seu responsável, ainda é criado na database um registro nos nós de entidades (onde são salvos dados da entidade) e de entidade_responsável (onde são salvos dados que relacionam o responsável com a entidade).

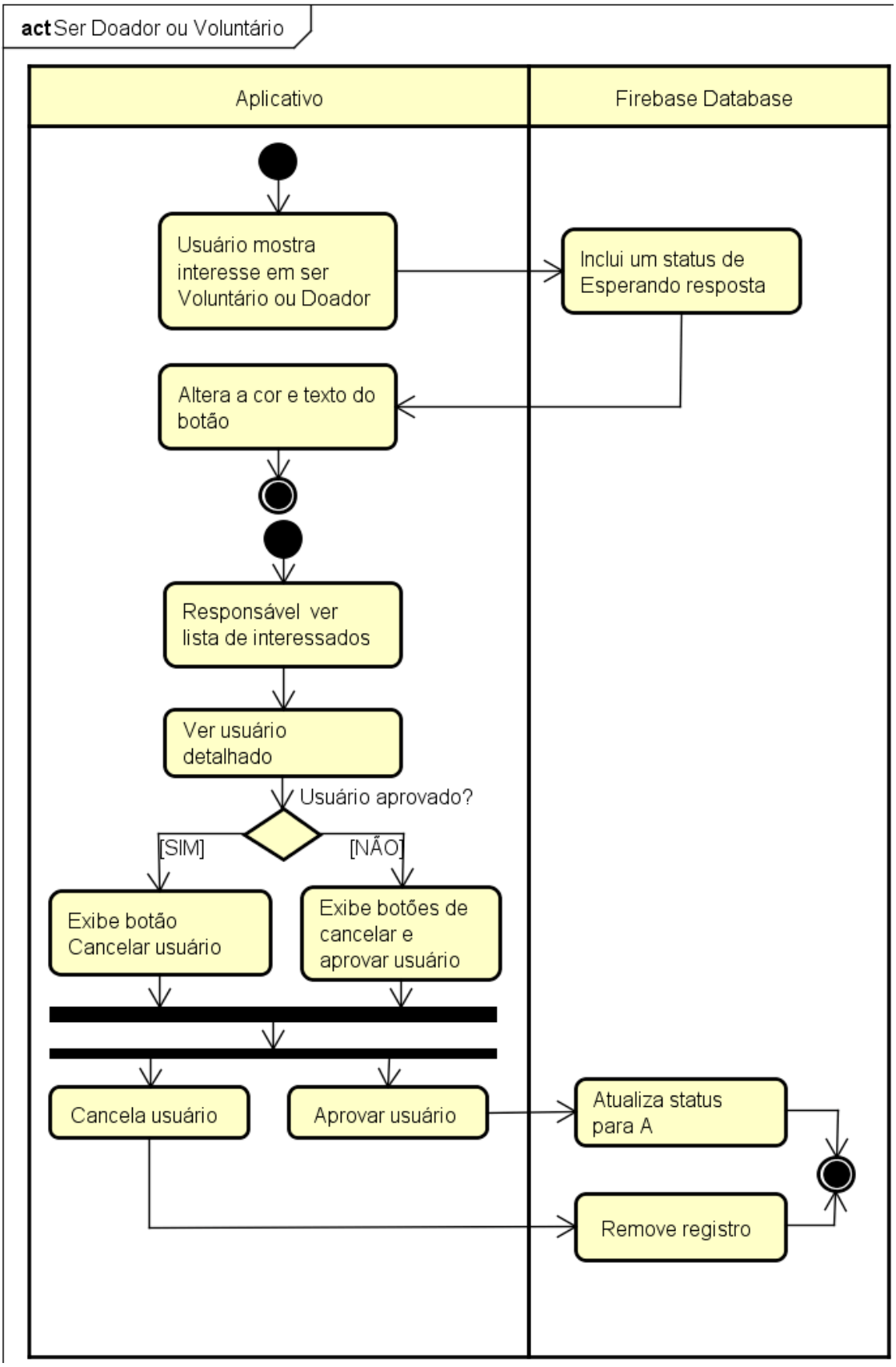
Figura 9 – Processo de criação de usuários



A Figura 10, explica o processo de validação dos usuários interessados em ser Doador ou Voluntário que é feito pelo responsável pela entidade. Esse processo inicia quando um usuário mostra interesse clicando no botão de **Ser Doador ou Voluntário**, na tela de detalhe de uma entidade. Feito isso é salvo um registro no Database e o botão muda para cor laranja e de texto (Quase Doador/Voluntário). Nesse momento, o fato de o usuário mostrar interesse não significa que já está efetivado, o responsável pela entidade precisa ativar esse interesse.

Na parte administrativa do responsável, na opção ”Pessoas”, são listados todos os interessados. Onde o responsável poderá cancelar ou aprovar o usuário. Cancelando um usuário, é removido o nó no Firebase Database. Aprovando um usuário, o status do interesse é apenas alterado para ”A”.

Figura 10 – Processo de Validação de interesse



3.2 Desenvolvimento

Nesta seção serão apresentados todos os conceitos envolvidos no desenvolvimento do aplicativo, indicando desde arquitetura até a estilização.

A arquitetura do aplicativo consiste na sincronização de informações, em tempo real, nos dispositivos conectados após alguma interação de qualquer usuário. Seja um simples cadastro de novo evento até um cadastro de uma nova entidade. Essa sincronização é garantida pelo próprio Firebase. Na Figura 11, é apresentado de maneira clara o processo de sincronização oferecido pelo Firebase.

Figura 11 – Arquitetura do Firebase

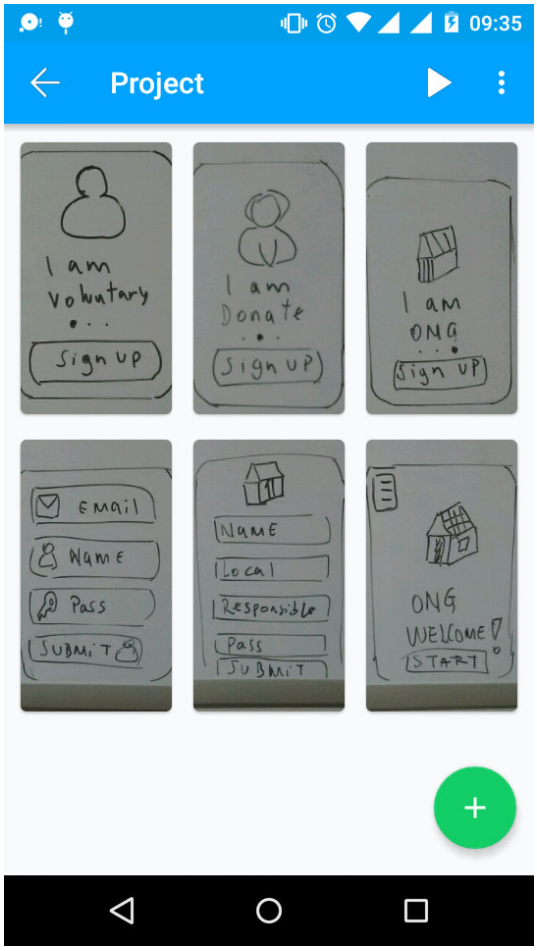
3.2.1 Ferramentas

A partir dos diagramas de atividades foi possível iniciar o processo de criação das telas do aplicativo. Inicialmente as telas foram pensadas e desenhadas no papel, posteriormente repassadas para um quadro branco (e esse processo de repetia a cada nova tela idealizada), mas faltava ter uma interação entre as telas para permitir uma visualização melhor do funcionamento do app.

Diante disso, para atender essa necessidade foi utilizada uma ferramenta de prototipação, chamada Marvel. Essa ferramenta é gratuita e está disponivel tanto para web quanto para dispositivos móveis. Auxiliando na simples criação, prototipação e colaboração.

Na Figura 12, é exibida a prototipação inicial do aplicativo. Contendo 6 telas, sendo as três imagens superiores foram propostas para a tela de login, as duas primeiras inferiores propostas para telas de cadastro de usuários e entidades, respectivamente e a última para a tela inicial da entidade. Vale lembrar que na seção de Resultados exibirá o resultado final da telas. A prototipação foi primordial para criação de todas as telas, pois com o protótipo pronto, sabia-se exatamente quais e onde os componentes iriam ser criados na tela, aumentando consideravelmente a produtividade da codificação.

Figura 12 – Prototipação do aplicativo



Para codificação, foi utilizando o editor de código Visual Studio Code, desenvolvido pela Microsoft para Windows, Linux e macOS. Ele inclui suporte para depuração, controle Git incorporado, realce de sintaxe, complementação inteligente de código, *snippets* e refatoração de código. Para uma maior produtividade utilizando esse editor, foram incluídas as extensões listadas abaixo:

- *React Native styled component Snippets*: Fornece sintaxe detalhada para estilizar componentes e folhas de estilos do React Native.

- *React Native Tools*: Esta extensão fornece um ambiente de desenvolvimento para projetos React Native. Usando-a, é possível depurar o código, executar rapidamente comandos React Native da paleta de comandos e usar o *IntelliSense* para procurar objetos, funções e parâmetros para as APIs React Native.

Após configurar o ambiente de desenvolvimento instalando as dependências necessárias, bastou instalar o React Native CLI[5] e criar o projeto com o comando **react-native init vitrinesocial**. É importante ressaltar que o aplicativo é divido em duas áreas:

1. Público: Área para Doadores e Voluntários acessarem;
2. Administrativo: Área para os responsáveis pelas entidades gerenciarem informações pertinentes a cada uma.

Utilizando o NPM, foi possível instalar alguns módulos necessários para o desenvolvimento e aprimoramento do aplicativo. Estes são listados a seguir:

- base-64: É um robusto codificador/decodificador base64, escrito em JavaScript. Foi utilizado para encriptar o email do usuário.
- firebase: Módulo responsável por manipular as funções com o Firebase. A versão utilizada foi a 4.8.1.
- react-native-elements: É um conjunto de ferramentas UI multiplataforma. A Versão utilizada foi a 0.18.5.
- react-native-image-picker: É um módulo React Native que permite usar UI nativa para selecionar uma foto ou vídeo da biblioteca do dispositivo ou diretamente da câmera. A Versão utilizada foi a 0.26.7.
- react-navigation: Módulo responsável pela navegação entre cenas do app. A Versão utilizada foi a 1.0.0-beta.21.

A cada nova implementação era necessário manter o controle de versionamento de código e do que deveria ser feito no aplicativo. Para auxiliar nesse processo foram utilizados o GIT para o controle de versão de arquivos e o GitHub para hospedagem gratuita do projeto. O Github oferece um recurso chamado **Projects**. Com este, foi possível gerenciar e dividir o projeto em pequenas tarefas com uma visão mais organizada do que realmente precisava ser feito, o que deveria ser priorizado e o que já tinha sido finalizado, pois ela utiliza a metodologia ágil Kanbam[6] auxiliando nesse processo.

[5] Interface de linha de comando

[6] É uma estrutura popular usada para implementar o desenvolvimento ágil de software

Como já foi mencionado na introdução, foi escolhido o Firebase, como servidor de backend, para ter acesso aos recursos disponibilizados por ele, fez-se necessário criar um projeto no console do Firebase. A seguir os passos necessários.

1. Requer ter uma conta da Google;
2. Acessar o console do Firebase;
3. Clicar em **Adicionar um projeto**
4. Informar o nome do projeto, escolher o país e clicar no botão **Criar Projeto**.

Após criar um projeto no Firebase, ele disponibiliza um código de incorporação para ser incluído no código cliente. Onde inclui os seguintes itens:

- apiKey – Core Firebase app;
- authDomain – Firebase Authentication.
- databaseURL – Firebase Realtime Database.
- projetId – Identificator Project.
- storageBucket – Firebase Storage.
- messagingSenderId - Firebase Cloud Messaging.

3.2.2 Firebase Database

Para gerenciamento e armazenamento de dados de cadastro foi o utilizado o Firebase Database com a estrutura mostrada na Figura 13. Vale lembrar que nessa figura são apresentados nós previamente criados oriundos de testes reais no app.

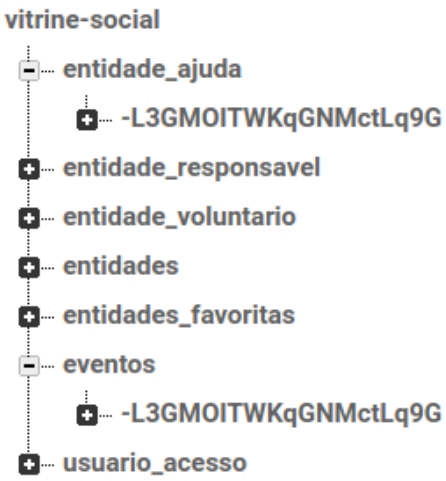
Para iniciar a interação com o Firebase Database, precisa-se pegar uma referência da database. Para isso, basta chamar o método database() do objeto global e chamar um caminho específico, como por exemplo, atribuir o caminho do database entidades a uma variável, como segue:

```
1 let refEntidades = firebase.database().ref(`entidades`)
```

Com a referência da database em mãos, é possível realizar qualquer outra operação, como *insert, update, delete e read*.

Os métodos de leituras utilizados no desenvolvimento foram:

Figura 13 – RealTime Database



- refEntidades.on(tipoEvento,funçãoDeCallBack()): É um método ouvinte, que ouve modificações no nó referenciado e envia mudanças para o cliente. Este método possui dois parâmetros:

  1. tipoEvento: Esse tipo de evento indica como o método ouvinte irá agir. Podendo agir das seguintes maneiras como é descrito na Tabela 2.

Tabela 2 – Tipos de eventos

Tipo Evento	Ação
value	Ouve e retorna dados quando houver qualquer mudança no database
child_added	Ouve e retorna toda vez que um novo registro for inserido na referência
child_removed	Ouve e retorna dados sempre que um nó é removido da referência

  2. funçãoDeCallBack(): É uma função de retorno onde pode ser manipulado o retorno da consulta no nó referenciado. Essa função retorna um objeto *DataSnapshot*. Esse *DataSnapshot* possui propriedades e métodos. É possível citar:
     - key: recupera a chave de identificação deste DataSnapshot. Em outras palavras é o ID do nó referenciado;
     - val(): É o método que recupera o valor de um DataSnapshot;
     - exists(): Verifica se o DataSnapshot contém algum dado.

- refEntidades.once(): É método que retorna todos os dados de uma única vez. Não possibilitando a manipulação em tempo real.

Para inserções e atualizações de informações na Firebase Database foram usados os seguintes métodos. Ambos os métodos recebem um objeto JavaScript como parâmetro.

- refEntidades.push(): Este método cria um valor único e randômico como chave do nó criado.
- refEntidades.set(): É método sobrescreve os dados no nós referenciado.

3.2.3 Firebase Storage

Conforme o requisito 3, primeiro e segundo tópico, é necessário o cadastro da imagem do evento e/ou cadastro da foto da entidade. Desta forma, necessitava-se salvar essas imagens em algum local. O Firebase dispõe de uma ferramenta íncrível para armazenar imagens, o Firebase Storage. Com esse recurso foi possível salvar imagens na nuvem e construir uma estrutura de armazenamento. Criando uma pasta específica para salvar imagens de entidades, eventos e usuários.

3.2.4 Firebase Authentication

Para gerenciamento de autenticação de usuário foi utilizado o *Firebase Authentication*. Foram utilizados os métodos a seguir:

- firebase.auth().createUserWithEmailAndPassword(email, password): Login de novos usuários.
- firebase.auth().signInWithEmailAndPassword(email, password): Login de usuários existentes.

Esses métodos após o login do usuário, permitiram acessar as informações básicas do perfil do usuário e controlar o acesso dele, pois cria uma sessão interna no app. Quando um usuário realiza o cadastro no aplicativo, salva informações iniciais, como Nome, Email e Telefone no Firebase Database, criando um registro no nó **usuario_acesso**. Sendo que a chave, deste nó criado, é o email criptografado do usuário para recuperação de informação posterior. Dentro desta chave é criado um nó randômico pelo próprio Firabase, através do método **push()**, criando os atributos: foto_perfil, nome, telefone e tipo que vai salvar o tipo do usuário. Esse tipo pode ser, como segue:

- D: Doador
- V: Voluntário

- E: Entidade

Finalizada a criação do nó usuario_acesso o Firebase, cria-se um registro em Authentication na aba Usuários, para gerenciar os acessos ao aplicativo. Uma vez feito isso, o usuário estará automaticamente logado no aplicativo sem precisar passar pela tela de login, pois o Firebase Authentication cria uma sessão internamente no aplicativo após um registro na Authentication.

3.3 Layout

O layout foi inicialmente criado utilizando apenas componentes do próprio React Native, com estilização própria embasada nos protótipos criados. Serviram de inspiração alguns aplicativos renomados, como o Facebook, G+ e Youtube, para disposição do *feed* de notícias e alguns elementos pontuais.

A paleta de cor utilizada em todo o app foi a do Color - Style - Material Design[7], facilitada pela extensão do navegador Google Chrome, chamada *Simple Material Design Palette*, disponível no Web Store.

Durante a pesquisa, apesar de não serem utilizadas na criação do layout, mas as ferramentas a seguir facilitam na criação e estilização de telas de um aplicativo fornecendo recursos impressionantes.

- Builderx.io: é uma ferramenta de design de tela que gera código Reative Native. Pode ser encontrado no site oficial BuilderX[8].
- NativeBase: ambiente nativo para criar aplicativos nativos. Pode ser encontrado no site oficial NativeBase.io[9].

[7] https://material.io/guidelines/style/color.html
[8] https://builderx.io/
[9] https://nativebase.io/

4 RESULTADOS

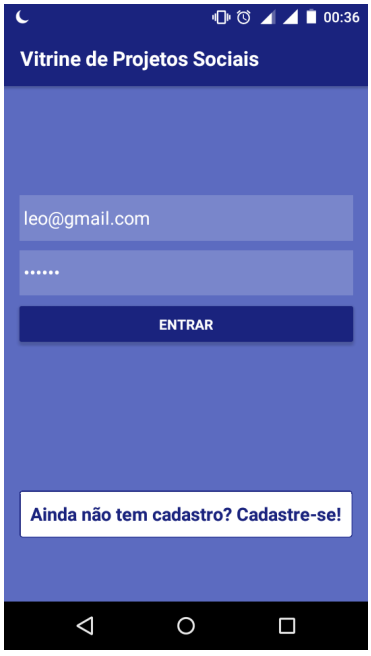
O produto final deste trabalho foi um aplicativo, ainda não totalmente finalizado, mas disponível para Android. Restam alguns detalhes para publicar oficial. De qualquer forma, já é possível realizar testes com a versão atual.

4.1 Telas do aplicativo

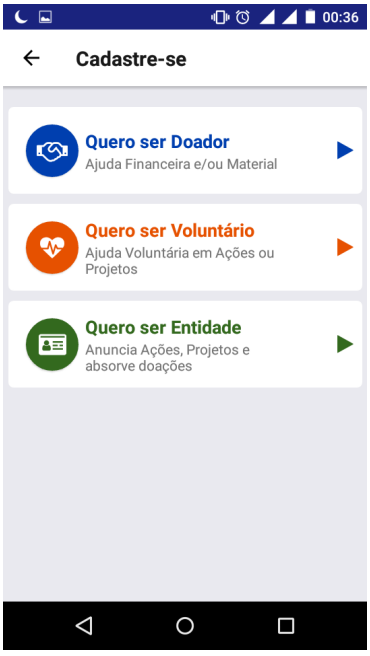
Cadastros e Login

A Figura 14a, apresenta o resultado da tela de Login. Onde o usuário poderá inserir email e senha cadastrados e clicar no botão Entrar. Após isso, é feita uma requisitação no Firebase para verificar se o usuário está cadastrado na base do Firebase. Caso possua cadastro irá ser direcionado para página inicial de acordo com o tipo de usuário. Se ainda não tiver cadastro, será exibida uma mensagem informativa. Além disso, poderá criar clicando no botão **Ainda não tem cadastro? Cadastre-se** para ser direcionado para tela de opção de cadastro. Nesta, o usuário vai escolher que tipo de cadastro ele quer, podendo ser **Doador**, **Voluntário** ou **Entidade**, conforme a Figura 14b. Ao finalizar o cadastro, o usuário visualizará a tela de boas vindas explicando as funcionalidades do app, conforme a Figura 14c.

Figura 14 – Tela Iniciais



(a) Tela de Login



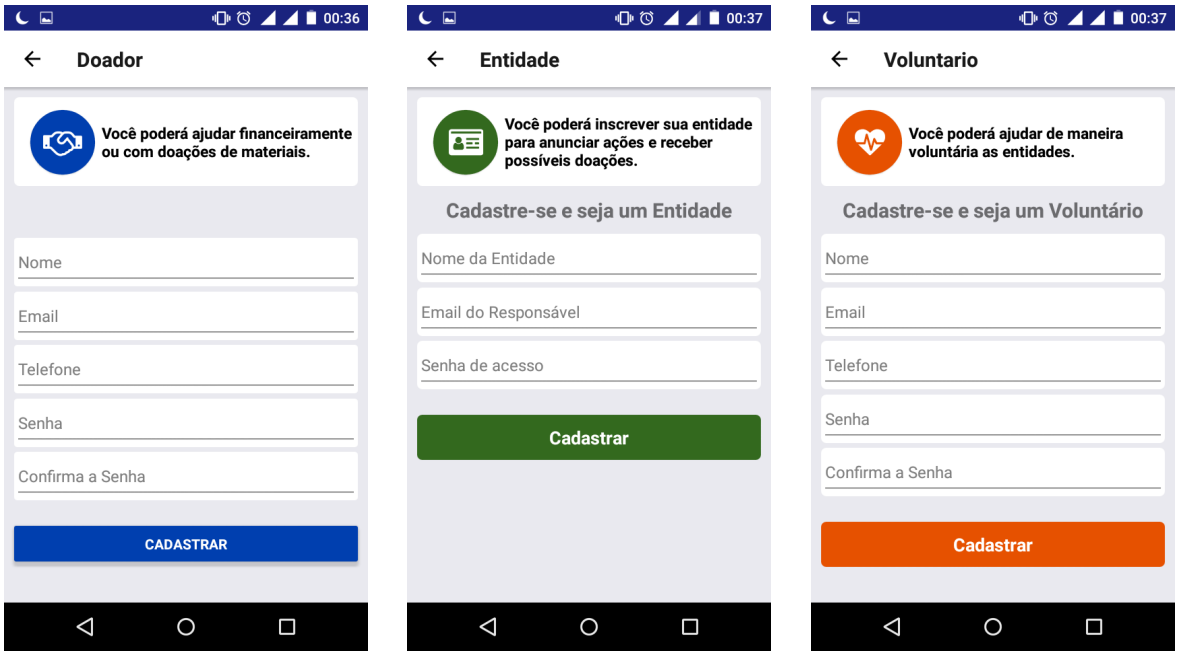
(b) Tela opções de cadastro

(c) Tela de Boas Vindas

Como fora mencionado, o usuário deverá escolher qual tipo de cadastro ele quer ter no aplicativo. A Figura 15, apresenta os 3 possíveis tipos de cadastro, com uma descrição no topo do formulário indicando qual forma de ajuda cada tipo de usuário pode oferecer.

- Doador: A Figura 15a, apresenta o cadastro de Doador com uma descrição e os campos nome, email, telefone, senha e confirma senha para serem preenchidos.
- Entidade: A Figura 15b, apresenta o cadastro de Entidade com uma descrição e os campos nome da Entidade, email do responsável, senha e confirma senha para serem preenchidos.
- Voluntário: A Figura 15c, apresenta o cadastro de Voluntário com uma descrição e os campos nome, email, telefone, senha e confirma senha para serem preenchidos.

Figura 15 – Telas de Cadastro



(a) Tela de Cadastro de Doador

(b) Tela de cadastro de Entidade

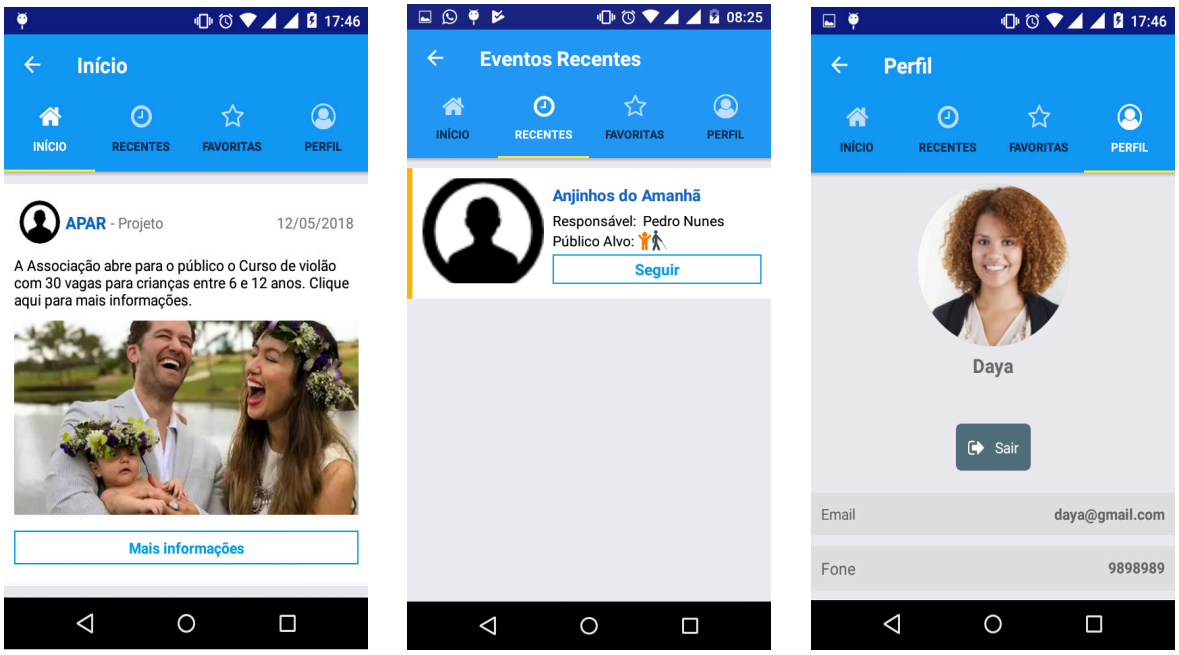
(c) Tela de Cadastro de Voluntário

Telas Pós Login - Doador e Voluntário

A seguir são apresentadas imagens de telas após o usuário realizar o login. Essas telas são semelhantes para o tipo Doador e Voluntário com poucas diferenças em funcionalidades.

- Feed de notícias: A Figura 16a, apresenta a tela inicial com as notícias, em tempo real, cadastradas pelas entidades. Nessa tela são exibidas as seguintes informações: Logomarca da entidade, tipo do evento, data do evento, descrição do evento, uma imagem principal do evento e um botão de mais informações que levará o usuário ao detalhamento dessa notícia.
- Entidades Recentes: A Figura 16b, lista todas as entidades cadastradas no aplicativo recentemente. Nesta, são exibidas a imagem da entidade, nome, reponsável, público alvo (ícones dinâmicos), uma cor padrão e um botão para seguir entidade.
- Perfil: A Figura 21, apresenta a tela de usuário. Mostrando a imagem de perfil do usuário, um botão para sair do aplicativo, email, telefone e senha.

Figura 16 – Telas Usuários



(a) Feed de notícias (b) Entidades recentes (c) Perfil do usuário

- Entidade Detalhe: A Figura 17a, apresenta todos os dados detalhados da entidade requerida pelo usuário, exibindo três abas, como segue:

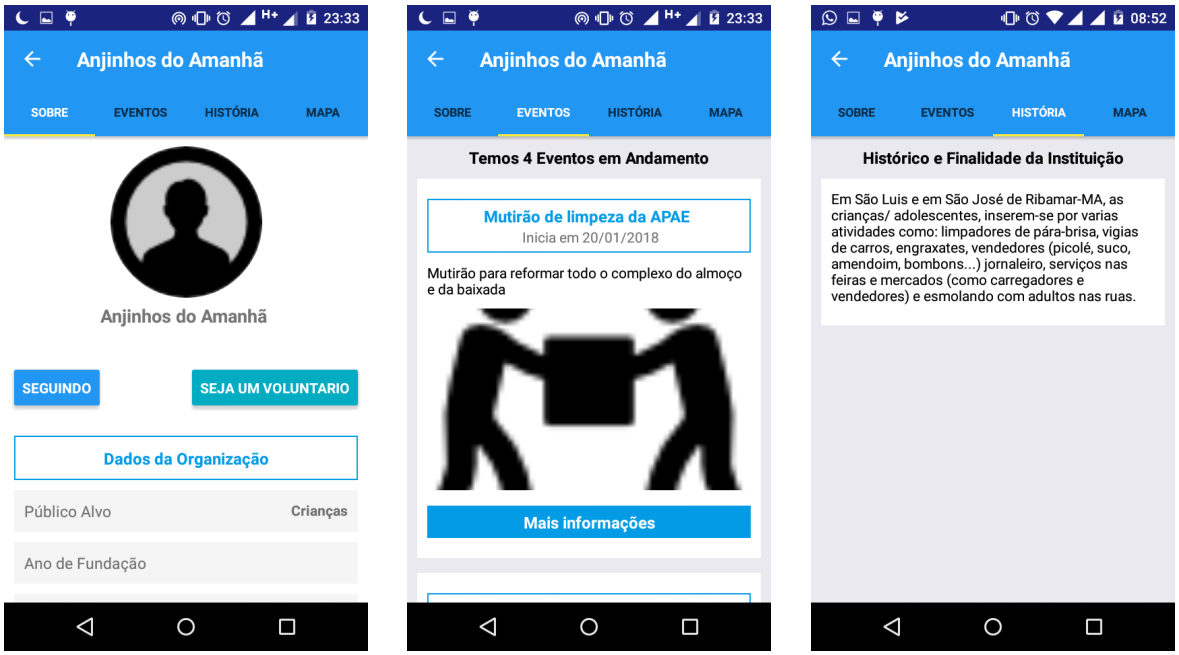
  1. Sobre: Nesta primeira aba, o usuário terá acesso a todas as informações legais da entidade. Nesta tela, também é possível ver as diferenças entre usuário do tipo Voluntário ou Doador, pelo fato de que dependendo do tipo de usuário, é exibido, no botão direito logo abaixo do nome da entidade, o texto **Seja um Doador** ou **Seja um Voluntário**. A exibição do texto e da funcionalidade é dinâmica de acordo com o tipo de usuário logado. Conforme a Figura 17a.

Além disso, para se tornar um Voluntário ou Doador, o responsável pela entidade deve aceitar a inscrição do usuário. Por isso, que uma vez que o usuário clica no botão, Quero ser um Voluntário/Doador, o botão muda de cor e de texto para amarelo e com o texto ”Quase Doador/Voluntário”. Aguardando assim a confirmação do responsável pela entidade, como pode ser visto na Figura 18a.

2. Eventos: Já nesta aba são exibidos todos os eventos realizados pela entidade com um total de eventos cadastrados no topo da página, conforme a Figura 19b.
3. História: Esta aba apresenta o histórico da entidade, desde sua fundação, lutas, conquistas e finalidade, conforme a Figura 17c.

- Evento Detalhe: A Figura 18b, apresenta um evento detalhado, exibindo além das informações prévias, outras mais detalhadas como: Justificativa, local, etc.
- Entidades Favoritas: A Figura 18c, exibe uma listagem de todas as entidades que o usuário marcou como favorita, ou seja, as que ele decidiu seguir. Nesta, ele poderá deixar de seguir qualquer entidade, basta clicar no botão **seguindo**.

Figura 17 – Telas detalhes da Entidade

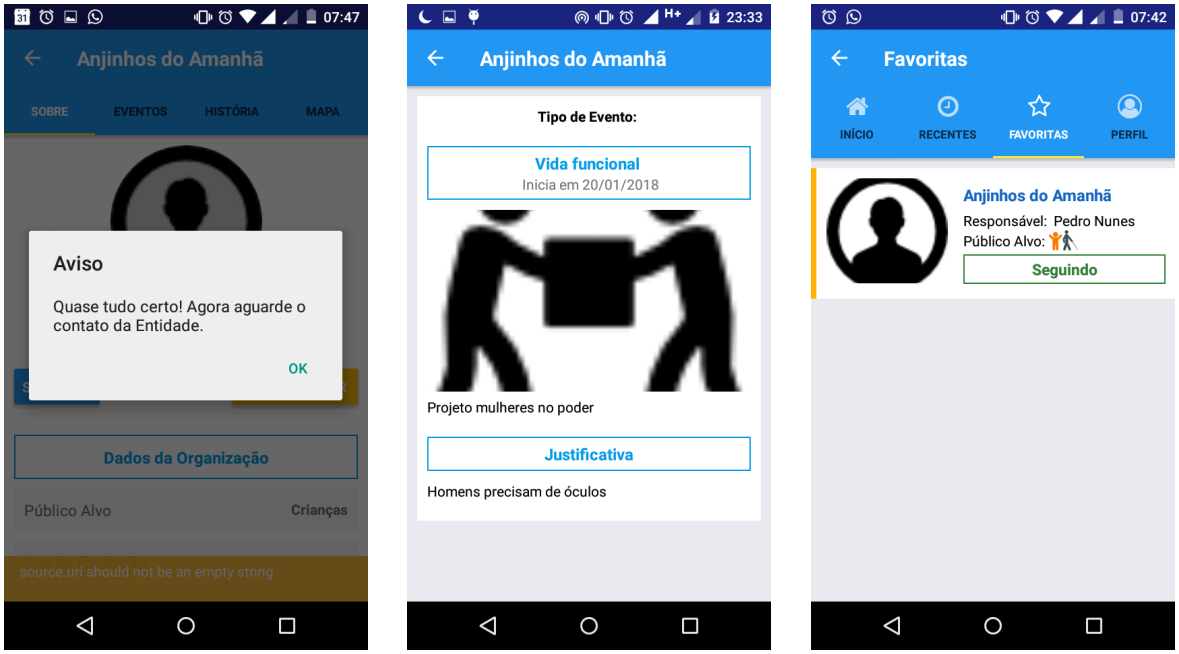


(a) Entidade detalhada (b) Eventos da entidade (c) Histórico da Entidade

Telas Pós Login - Entidade

A seguir são apresentadas imagens de telas após o usuário, responsável por uma Entidade, realizar o login.

Figura 18 – Telas do usuário

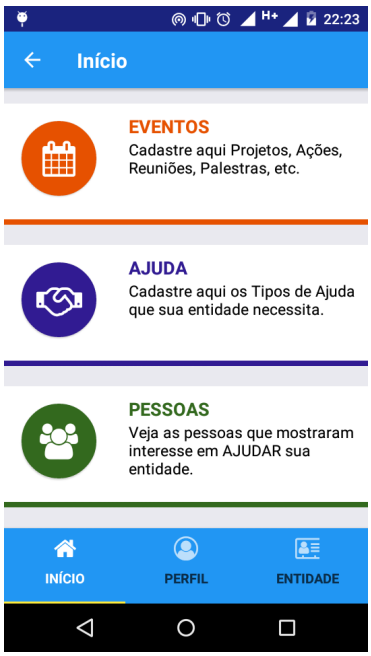


(a) Entidade Detalhada (b) Evento detalhado (c) Entidades Favoritas

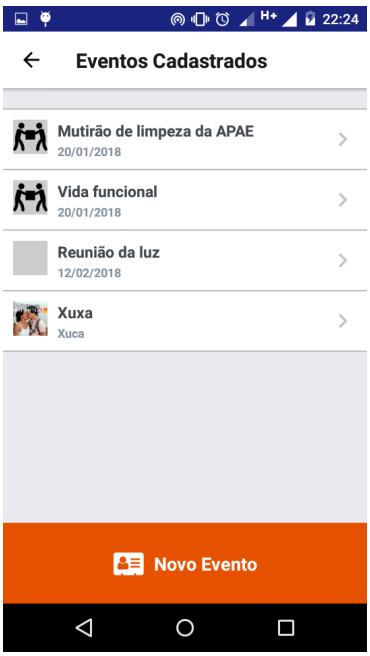
- Home Entidade: A Figura 20, apresenta as três abas:

  1. Início: Nesta aba são apresentadas as três funcionalidades principais deste usuário:
     - Eventos: Aqui é possível gerenciar eventos. Listar e cadastrar novos, conforme a Figura 19b. Ao clicar em qualquer evento, o usuário será levado para um detalhamento daquele evento;
     - Ajuda: É possível listar, deletar e cadastrar novos tipos de ajuda que a entidade precisa, conforme a Figura 19d. Ao clicar em qualquer tipo de ajuda, o aplicativo exibe uma alerta perguntando ao usuário se ele quer apagar aquele tipo de ajuda cadastrado;
     - Pessoas: Aqui é possível visualizar todas as pessoas que mostram interesse em ser Doador ou Voluntário. A partir daqui, a entidade poderá entrar em contato com os interessados, conforme a Figura 20b.
  2. Perfil: Aqui são exibidos dados do responsável. Semelhante a tla de perfil do usuário normal.
  3. Entidade: Nesta são exibidas as informações completas da entidade. É nessa tela que o responsável pela entidade deverá terminar o cadastro da entidade ou atualizar informações, como pode ser visto na Figura 20e.

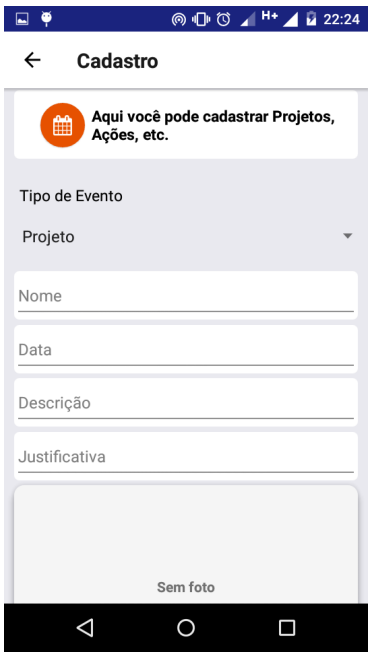
Figura 19 – Telas Entidade



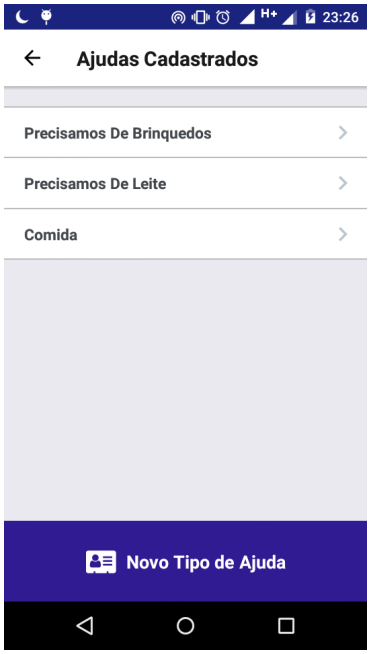
(a) Home Entidade



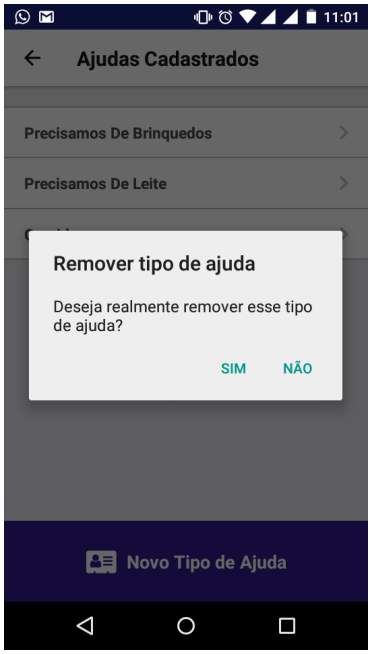
(b) Eventos Cadastrados



(c) Novo Evento

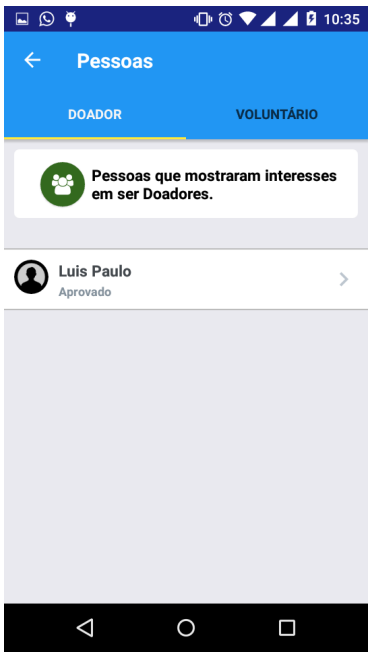


(d) Tipos de ajudas

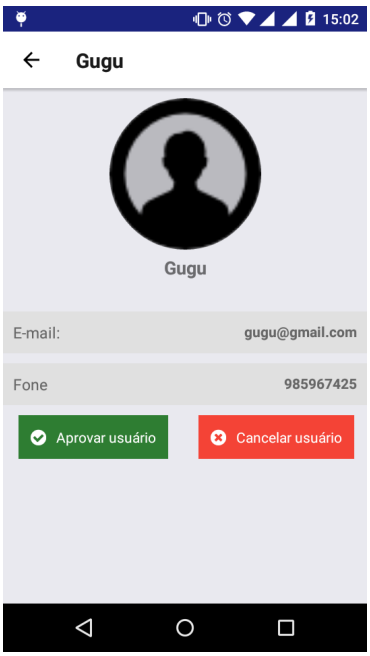


(e) Remover Ajuda

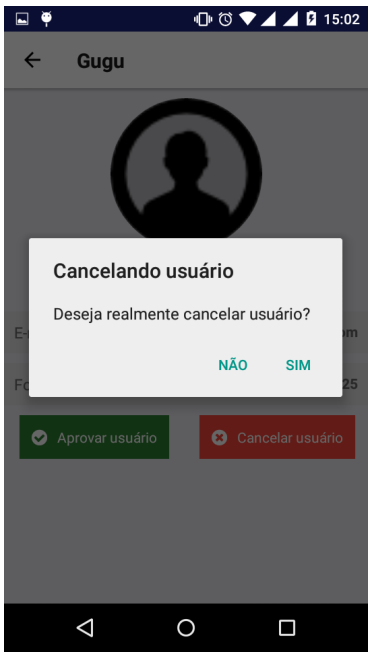
Figura 20 – Telas Entidade



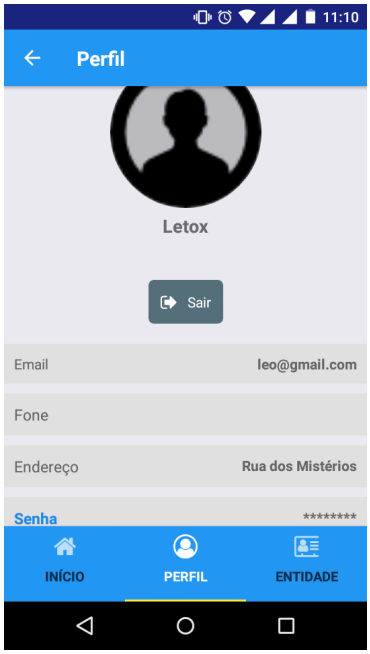
(a) Pessoas interessadas



(b) Pessoa detalhada



(c) Pessoa detalhada



(d) Perfil

(e) Editar entidade

4.2 Resultados do Questionário

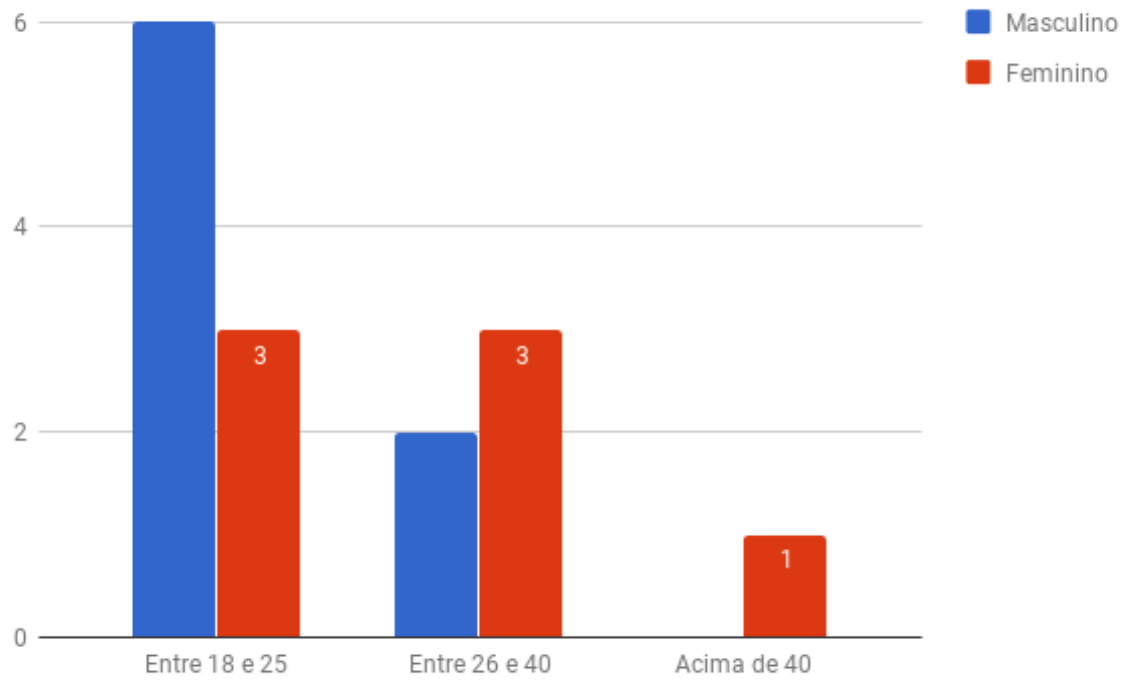
Buscando captar um *feedback* dos usuários e também ter uma estatítica do perfil dos usuários do aplicativo e do nível de satisfação, foi criado um questionário com três tipos de questões:

1. Perfil do Usuário: 3 perguntas de múltiplas escolhas;
2. Avaliação da usabilidade: 11 perguntas onde o usuário devia escolher e marcar com um ”X” a opção entre as possíveis: ”Concordo totalmente (CT)”, ”Concordo (C)”, ”Indeciso (I)”, Discordo (D)” e ”Discordo totalmente (DT)”.
3. Comentário: Uma pergunta subjetiva onde o usuário poderia colocar algum comentário adicional, seja sugestão ou crítica.

No Anexo A, tem-se o questionário aplicado com todas as questões elaboradas para avaliar a opinião do usuário.

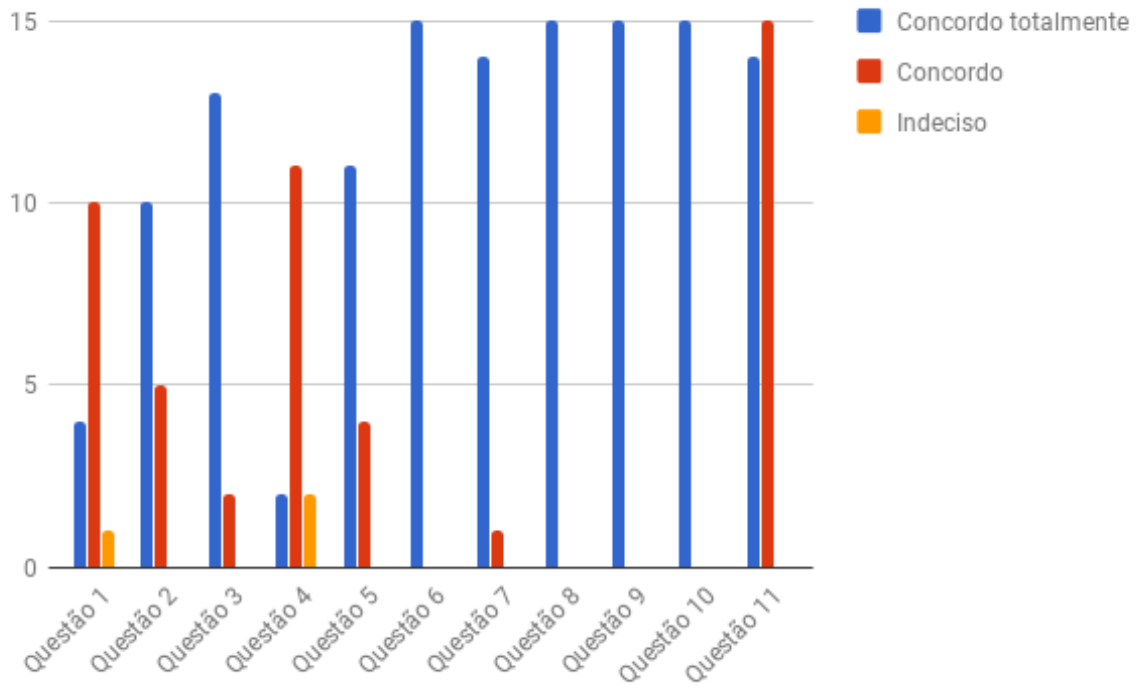
As informações recolhidas, a partir das respostas de 15 participantes, foram organizadas e representadas em dois gráficos, para um melhor entendimento dos resultados obtidos. A Figura 21, apresenta a estatística do perfil do usuário, mostrando que a maior parte dos participantes tem idade entre 18 e 25, e que são do sexos masculino.

Figura 21 – Perfil do usuário



Na Figura 22, é apresentado o grau de satisfação dos usuários por questão. Dessa forma, percebe-se que a maioria das questões foi classificada entre os critérios de ”Concordo totalmente”e ”Concordo”.

Figura 22 – Usabilidade

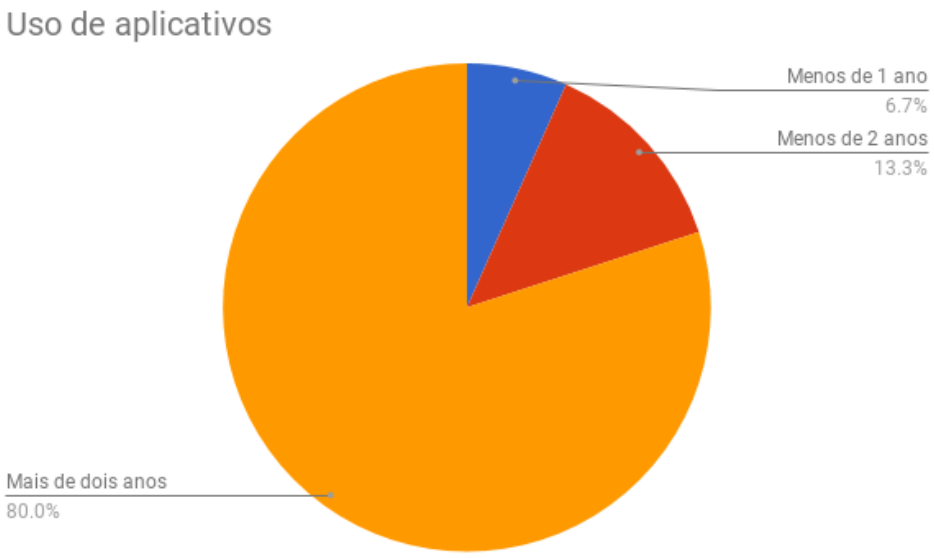


Já o gráfico 23, pode-se analisar que a maior parte dos usuários usam aplicativos móveis há bastante tempo (80% a mais de dois anos). Concluindo assim que o desenvolvimento e uso de apps ainda perdurará.

Além dos dados apresentados nos gráficos, os participantes descreveram na última questão algumas sugestões de melhorias, conforme abaixo:

- Colocar foto de perfil;
- Os eventos que não são fixos, ou seja, que são realizados apenas uma vez, como um fórum, gincana, seminário, etc, podem ter um ícone do tipo ”tenho interesse”(em participar, como os que aparecem nos eventos do Facebook) e que fiquem com um ”alerta”ou notificação para avisar ao usuário que a data da realização de tal evento se aproxima;
- Caso possível, assim que uma entidade precisar de uma grande colaboração/ajuda de recursos, possam enviar uma mesma mensagem pra todos os seus seguidores ficarem cientes, ao invés de entrar em contato com cada um por vez

Figura 23 – Tempo de uso de app



(mais ou menos como a ”transmissão”do WhatsApp, isso somente nos casos em que se trate de uma cooperação com grande número de pessoas).

- Quando tiver um grande número de eventos e entidades cadastradas seria bom uma barra de pesquisa, para o usuário pesquisar sobre determinado assunto de interesse ou mesmo já esteja atrás de um evento específico e possa achar com maior rapidez.

5 CONCLUSÃO

Com o que foi apresentado neste trabalho é possível concluir que React Native é uma ferramenta muito poderosa para criação de aplicativos móveis que rodam nativamente atigindo melhor usabilidade para usuário e mais, cria aplicativos para as plataformas iOS e Android usando apenas JavaScript. O JavaScript se mostrou tão versátil que também pode ser usado para criar aplicativos móveis, o que significa que é possível compartilhar componentes entre a web e as plataformas móveis.

Outro ponto importante, é o fato de o React Native realmente renderizar o uso das APIs de renderização padrão da plataforma hospedeira permitindo que ele se destaque da maioria dos métodos existentes de desenvolvimento de aplicativos multiplataforma, como Cordova ou Ionic. Os métodos existentes de escrita de aplicativos móveis usando combinações de JavaScript, HTML e CSS costumam renderizar usando **webviews**.

Foi demonstrado, também, que o Firebase é uma plataforma rica que disponibiliza vários recursos de **backend**. Fornecendo toda uma infraestrutura necessária para o desenvolvedor. Além disso, dispõe no seu plano gratuito um ambiente de testes que capacita o desenvolvedor a decidir se irá precisar de mais recursos ou não. Mostrou-se de fácil utilização de databases, armazenamento de imagens e um gerenciamento de autenticação robusto, tudo através do seu console.

Como trabalho futuro para uma evolução do app e possível publicação, pode-se citar as seguintes funcionalidades:

- Inclusão da funcionalidade de mapa: para colocar a localização da entidade e possívelmente um sistema de recomendação por localização;
- Login com Facebook, Google e outras redes sociais;
- Incluir funcionalidades de enviar notificações para usuários a cada novo cadastro de eventos de uma entidade favorita;
- Incluir funcionalidade para usuários se cadastrarem em um curso fornecido por uma entidade;
- Incluir o número de entidades recentes no menu;
- Incluir o número de eventos no menu;
- Utilizar layout do NativeBase;

- Feedback da entidade para o usuário;
- Comunicação entre as entidades e os usuários.

Contudo, é possível assegurar, que este aplicativo pode ser uma ferramenta importante para conectar pessoas com Entidades e vice versa. Auxiliando na visualização e divulgação de eventos que entidades desempenham no Brasil. Podendo desta forma, conseguir mais ajuda, reconhecimento e aceitação da comunidade em geral pelos trabalhos desempenhados que ajudam a mudar de maneira direta ou indiretamente a sociedade.

Referências

ADMOB. *AdMob*. 2018. <https://firebase.google.com/docs/admob/>. [Online; acessado 18/02/2018]. Citado na página 37.

ANALYTICS. *Google Analytics para Firebase*. 2018. <https://firebase.google.com/docs/analytics/>. [Online; acessado 18/02/2018]. Citado na página 36.

ANTHONY MURRAY NATHANIEL, L. A. A. *Fullstack React: The Complete Guide to ReactJS and Friends*. Fullstack.io, 2017. ISBN 0991344626,9780991344628. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=4266d31aa708d3e48fb1ea62933364f9>. Citado 6 vezes nas páginas 19, 20, 22, 23, 24 e 25.

ASYNCSTORAGE. *AsyncStorage*. 2018. <https://facebook.github.io/react-native/docs/asyncstorage.html>. [Online; acessado 14/02/2018]. Citado na página 33.

AXELSSON, O.; CARLSTRÖM, F. Evaluation targeting react native in comparison to native mobile development. 2016. Citado na página 15.

BARBOZA, M. Q. *As 100 melhores ongs do Brasil*. 2017. <http://epoca.globo.com/brasil/noticia/2017/08/100-melhores-ongs-do-brasil.html>. [Online; acessado 09/10/2017]. Citado na página 15.

CHENG, F. *Build Mobile Apps with Ionic 2 and Firebase: Hybrid Mobile App Development*. [S.l.]: Apress, 2017. Citado na página 34.

CHINNATHAMBI, K. *JavaScript Absolute Beginner's Guide*. 1. ed. Que Publishing, 2016. ISBN 0789758067,9780789758064. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=0e496c15f7ac3d7687f53001e8285557>. Citado na página 19.

COMPUTAÇÃO, C. D. C. D.; TOREZANI, H. C. Um framework para compartilhamento adaptativo de experiências familiares. 2015. Citado na página 20.

DEKA, G. C. *NoSQL: Database for Storage and Retrieval of Data in Cloud*. CRC Press, 2017. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=4103aa6fcef9a0ca4d2bd1d3efb0a58e>. Citado na página 34.

DEVELOPERMOZILLA. *Classes*. 2018. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>. [Online; acessado 04/02/2018]. Citado na página 24.

DJURUP, R. *Android Mobile Computing Using Samsung Tablets and Smartphones Running Android 2.3*. Rebidu, 2013. ISBN 9788777930355. Disponível em: <https://books.google.com.br/books?id=Oe1LYsZm6yYC>. Citado na página 18.

DYNAMICLINKS. *Firebase Dynamic Links*. 2018. <https://firebase.google.com/docs/dynamic-links/>. [Online; acessado 18/02/2018]. Citado na página 37.

EISENMAN, B. *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, 2015. ISBN 9781491929070. Disponível em: <https://books.google.com.br/books?id=274fCwAAQBAJ>. Citado 8 vezes nas páginas 21, 22, 26, 29, 30, 31, 32 e 33.

FACEBOOK. *State and Lifecycle*. 2018. <https://reactjs.org/docs/state-and-lifecycle.html>. [Online; acessado 07/02/2018]. Citado 3 vezes nas páginas 24, 25 e 26.

FACEBOOKNATIVE. *React Native*. 2018. <https://facebook.github.io/react-native/>. [Online; acessado 26/01/2018]. Citado 3 vezes nas páginas 15, 26 e 27.

FACEBOOKNATIVE. *Style*. 2018. <https://facebook.github.io/react-native/docs/style.html>. [Online; acessado 11/02/2018]. Citado 3 vezes nas páginas 30, 31 e 32.

FALCONER, A. P. A promessa do terceiro setor. *Centro de Estudos em*, 1999. Citado na página 16.

FERREIRA, F. M. G. et al. Desenvolvimento e aplicações de um framework orientado a objetos para análise dinâmica de linhas de ancoragem e de risers. Universidade Federal de Alagoas, 2005. Citado na página 20.

FIREBASE. *Firebase*. 2018. <https://firebase.google.com/>. [Online; acessado 27/01/2018]. Citado 3 vezes nas páginas 15, 35 e 36.

FLANAGAN, D. *JavaScript: The definitive guide: Activate your web pages*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 19.

GACKENHEIMER, C. *Introduction to React*. 1. ed. Apress, 2015. ISBN 1484212460,978-1-4842-1246-2,978-1-4842-1245-5. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=2F1302ACB53844126F4626D7F9FC615F>. Citado na página 26.

HEARD, P. *React Native Architecture : Explained*. 2018. <https://www.logicroom.co/react-native-architecture-explained/>. [Online; acessado 06/02/2018]. Citado 2 vezes nas páginas 27 e 28.

HOLMES, E.; BRAY, T. *Getting Started with React Native*. Packt Publishing, 2015. ISBN 9781785886232. Disponível em: <https://books.google.com.br/books?id=vSLICwAAQBAJ>. Citado 3 vezes nas páginas 23, 29 e 32.

HUDSON, P. *Hacking with React*. [s.n.], 2016. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=0e67d6c2ebdd326954c07000b83a7f78>. Citado 2 vezes nas páginas 21 e 25.

INVITES. *Firebase Invites*. 2018. <https://firebase.google.com/docs/invites/>. [Online; acessado 18/02/2018]. Citado na página 37.

JSON. *Json*. 2018. <https://www.json.org/json-pt.html>. [Online; acessado 26/01/2018]. Citado na página 35.

LOPES, S. *Aplicações mobile híbridas com Cordova e PhoneGap*. [S.l.]: Editora Casa do Código, 2016. Citado na página 14.

MAHARJAN, A. *Getting started with React Native: Core Architecture of React Native*. 2018. <http://jyaasa.com/blog/getting-started-with-react-native-core-architecture-of-react-native>. [Online; acessado 06/02/2018]. Citado 3 vezes nas páginas 26, 27 e 29.

MASIELLO, J. F. E. *Mastering React Native*. Packt Publishing, 2017. ISBN 1785885782,9781785885785. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=a3603a17841d7942288c1ad46d629edd>. Citado na página 24.

MESSAGING, C. *Firebase Cloud Messaging*. 2018. <https://firebase.google.com/docs/cloud-messaging/>. [Online; acessado 18/02/2018]. Citado na página 36.

MORONEY, L. *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. 1. ed. Apress, 2017. ISBN 978-1-4842-2942-2, 978-1-4842-2943-9. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=f2b26319bc8c9aebe626fcd4dbed2868>. Citado 3 vezes nas páginas 34, 35 e 36.

OPENSOURCE, F. *Components and Props*. 2018. <https://reactjs.org/docs/components-and-props.html>. [Online; acessado 04/02/2018]. Citado 2 vezes nas páginas 24 e 25.

OPENSOURCE, F. *React Js*. 2018. <https://reactjs.org/>. [Online; acessado 26/01/2018]. Citado na página 14.

PADILHA, A. V. et al. Usabilidade na web: uma proposta de questionário para avaliação do grau de satisfação de usuários do comércio eletrônico. Florianópolis, SC, 2004. Citado na página 71.

PATTNAIK, P.; MALL, R. *FUNDAMENTALS OF MOBILE COMPUTING, Second Edition:*. PHI Learning Private Limited, 2015. ISBN 9788120351813. Disponível em: <https://books.google.com.br/books?id=NREACwAAQBAJ>. Citado na página 18.

ROBBESTAD, S. A. *ReactJS Blueprints*. Packt Publishing, 2016. ISBN 1785886541,9781785886546. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=4aa7a1aa13fe55000903f2ffccd5856a>. Citado 3 vezes nas páginas 21, 25 e 26.

SMYTH, N. *Firebase Essentials - Android Edition:*. CreateSpace Independent Publishing Platform, 2017. ISBN 9781546660330. Disponível em: <https://books.google.com.br/books?id=9i4tDwAAQBAJ>. Citado na página 34.

SOLUçõES, E. D. *Pesquisa de uso de aplicativos no Brasil - Junho 2017*. 2017. <https://www.dubsolucoes.com/single-post/Pesquisa-de-uso-de-aplicativos-no-Brasil---Junho-2017>. [Online; acessado 07/10/2017]. Citado na página 14.

TECHNOPEDIA. *Cross-Platform Development*. 2017. <https://www.techopedia.com/definition/30026/cross-platform-development>. [Online; acessado 08/01/2018]. Citado na página 19.

VIPUL, P. S. *ReactJS by Example: Building Modern Web Applications with React: Get up and running with ReactJS by developing five cutting-edge and responsive projects*. Packt Publishing, 2016. ISBN 978-1-78528-964-4. Disponível em:

<http://gen.lib.rus.ec/book/index.php?md5=8f2479becdd3b0fbc75db1735db78dc5>. Citado 5 vezes nas páginas 21, 23, 24, 25 e 26.

WAIKAR, M. *Data-oriented Development with AngularJS: Write DSLs for your user interface code using AngularJS directives and add real-time capabilities to your applications using AngularFire's three-way data binding with Firebase*. Packt Publishing, 2015. ISBN 978-1-78439-805-7. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=8a7ff905b15148c907b0248a347d8a04>. Citado na página 34.

WOOD, L. et al. *Document Object Model (DOM) level 3 core specification*. [S.l.]: W3C Recommendation, 2004. Citado 2 vezes nas páginas 21 e 22.

ZAKAS, N. C. *Understanding ECMAScript 6. The definitive guide for Javascript developers*. No Starch Press, 2016. ISBN 978-1-59327-757-4. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=bfda74ab195c2f6bb9b563deb4e2a375>. Citado na página 20.

ANEXO A – Formulário de avaliação do Vitrine de Projetos Sociais

Abaixo estão listadas o Perfil do usuário e as afirmativas do formulário de avaliação empregados no protótipo do Vitrine de Projeto Sociais. Todas as afirmativas abaixo são respondidas através dos valores CT (Concordo totalmente), C (Concordo), I (Indeciso), D (Discordo) e DT (Discordo totalmente). Sendo escolhida a opção que for marcada com um “X”.

Perfil do Usuário

1. Qual sua faixa de idade?
    - Entre 18 e 25
    - Entre 26 e 40
    - Acima de 40
2. Qual seu sexo?
    - ( ) M
    - ( ) F
3. Há quanto tempo você utiliza aplicativos?
    - ( ) Menos de um ano
    - ( ) Menos de dois anos
    - ( ) Mais de dois anos

Comentário Adicional

Você tem algum comentário adicional sobre este aplicativo? Escreva sua resposta no espaço abaixo.

Tabela 3 – Avaliação da usabilidade

Experiência de navegação no aplicativo e aspectos visuais					
Questões	**CT**	**C**	**I**	**D**	**DT**
Este aplicativo tem uma apresentação gráfica agradável e legível					
Considero rápido o acesso às informações do aplicativo.					
Os recursos de navegação (menus, ícones, links e botões) estão todos claros e fáceis de achar					
Logo que entro no aplicativo, consigo identificar com facilidade as opções disponíveis (funcionalidades)					
Os títulos das abas deste aplicativo são muito intuitivos					
É fácil a navegação neste aplicativo					
A navegação do aplicativo é intuitiva					
O propósito/função do aplicativo é claro					
Interface do aplicativo se adapta a rotação da tela					
Os ícones possuem contraste suficiente em relação ao plano de fundo					
É fácil a utilização do aplicativo					

Fonte: Adaptado de Padilha (2004)

ANEXO B – Referências de material

- GitHub: react-native-material-design
- Canal Youtube:
  - Curso de React Native - Mario Díez
  - Firebase
- Curso gratuito:
  - Rocketseat Code
  - Mobile App Design: From Sketches to Interactive Prototypes na Udemy
- Curso pago: Desenvolvedor Multiplataforma Android/IOS com React e Redux