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

GABRIEL RODRIGUES DE ARAUJO

DESENVOLVIMENTO CROSS-PLATFORM COM REACT NATIVE: UM ESTUDO DE CASO DO APLICATIVO NAVEG

FORTALEZA

2019

GABRIEL RODRIGUES DE ARAUJO

DESENVOLVIMENTO CROSS-PLATFORM COM REACT NATIVE: UM ESTUDO DE CASO DO APLICATIVO NAVEG

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito à obtenção do título de Bacharel em Sistemas e Mídias Digitais. Área de concentração: Sistemas de Informação Multimídia.

Orientador: Prof. Dr. Windson Viana de Carvalho.

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A689d Araujo, Gabriel Rodrigues de.
Desenvolvimento cross-platform com react native: um estudo de caso do aplicativo Naveg / Gabriel Rodrigues de Araujo. – 2019.
67 f. : il. color.

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

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

CDD 302.23

GABRIEL RODRIGUES DE ARAUJO

DESENVOLVIMENTO CROSS-PLATFORM COM REACT NATIVE: UM ESTUDO DE CASO DO APLICATIVO NAVEG

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas e Mídias Digitais da Universidade Federal do Ceará, como requisito à obtenção do título de Bacharel em Sistemas e Mídias Digitais. Área de concentração: Sistemas de Informação Multimídia.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Windson Viana de Carvalho (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. José Gilvan Rodrigues Maia
Universidade Federal do Ceará (UFC)

_____
Prof. MSc. Antonio José Melo Leite Júnior
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Windson Viana de Carvalho, que se fez presente durante a maior parte do meu percurso na faculdade e, reconhecendo meu potencial, foi insistente nos conselhos e nas críticas.

À minha família, que sempre cultivou em mim o apreço pela leitura e estudos, e jamais me pressionou no âmbito escolar para que seguisse determinado caminho, mas, do contrário, me permitiu enxergar possibilidades amplas à minha frente, com dedicação e zelo por todos os meus atos.

À minha noiva, Carolina Castro, que é minha parceira desde o início e que ofereceu suporte nos momentos em que mais precisei, e teve toda a compreensão do mundo quando eu precisei, infelizmente, me ausentar.

Aos meus amigos Kevin Torres, Lucas Stark e Rister Saulo, irmãos de coração que resolvi adotar, e com os quais sei que posso contar pelo resto da vida.

Aos integrantes da equipe Sigma, Julianne Holanda, Hiago Rabelo e Thais Gouveia, que, além de grandes amigos, proporcionaram uma sinergia impressionante durante todo o período de desenvolvimento deste trabalho, fazendo com que os resultados fossem os melhores possíveis.

À Universidade Federal do Ceará e ao curso de Sistemas e Mídias Digitais, por abrirem suas portas e viabilizarem experiências únicas que, de outra forma, não seria possível de se experimentar.

Faço uma menção honrosa ao Chico (já falecido) e ao Pirata, mascotes caninos do curso de Sistemas e Mídias Digitais. A presença deles no bloco didático alegrou e trouxe paz às noites mais conturbadas.

RESUMO

O mercado vegetariano tem crescido nos últimos anos e isso, aliado ao aumento do uso de *smartphones* no Brasil, proporciona boas oportunidades de negócios. Neste contexto, este TCC apresenta o processo de conceituação e desenvolvimento do NaVeg, um aplicativo que busca auxiliar veganos e vegetarianos de Fortaleza a acompanhar os principais eventos, produtos e serviços relacionados a esse estilo de vida. A proposta do NaVeg é ser *cross-platform*, executando-se nos Sistemas Operacionais Android e iOS. Com o intuito de reduzir o tempo de desenvolvimento, foi adotado o *framework* React Native, que transforma código JavaScript em aplicativos nativos por meio da biblioteca React. São expostos neste relatório: os conceitos de desenvolvimento *cross-platform* e React Native, a metodologia que guiou a produção do NaVeg, a conceituação de sua proposta e os pontos essenciais do desenvolvimento, teste e exportação de um aplicativo *cross-platform* com React Native. São também descritas boas práticas de uso da ferramenta, e discutidas as dificuldades presentes nas etapas de construção do NaVeg. Por fim, é feito um comparativo entre o resultado final do aplicativo no Android e no iOS.

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

ABSTRACT

The vegetarian market has grown in recent years and this, coupled with the growth in the use of smartphones in Brazil, provides good business opportunities for entrepreneurs. In this context, this work presents the process of conceptualization and development of NaVeg, a mobile application that aims to help vegans and vegetarians from Fortaleza to discover the main events, products and services related to this lifestyle. NaVeg's intent is to be cross-platform, running on Android and iOS Operating Systems. In order to reduce development time, we adopted the React Native framework, which transforms JavaScript code into native applications through the React library. The concepts of cross-platform and React Native development, the methodology that guided the production of NaVeg, the conceptualization of its proposal and the essential points of developing, testing and exporting a cross-platform application with React Native are presented in this report. Good practices of the use of this tool are also described, and the difficulties in the construction of NaVeg are discussed. Finally, a comparison is made between the resulting apps on Android and iOS.

Keywords: Cross-platform. React Native. Vegetarianism.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

B2B	*Business to Business*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
HTML	*Hypertext Markup Language*
JSON	*JavaScript Object Notation*
JSX	*JavaScript XML*
SMD	Sistemas e Mídias Digitais
SO	Sistema(s) Operacional(is)
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, houve um aumento no número de adeptos do veganismo e vegetarianismo no Brasil. Segundo pesquisa realizada pelo IBOPE em maio de 2018, “14% da população se declara vegetariana”, o que representa um aumento de 75% em relação a 2012, quando pesquisa semelhante havia sido aplicada (SVB, 2018). Considerando apenas a região Nordeste, aqueles que se enxergam como vegetarianos, parcial ou totalmente, representam 17% da amostra.

Ainda conforme a pesquisa mais atual, empresários do setor apontam uma ampliação desse mercado na proporção de 40% ao ano pelos próximos anos. Esse crescimento afeta não somente o público vegetariano, mas também pessoas que têm buscado e adquirido hábitos alimentares mais saudáveis e sustentáveis. A pesquisa do IBGE mostra que:

> O mercado de produtos veganos atinge não somente veganos e vegetarianos, mas também uma parcela crescente da população que busca reduzir o consumo de carnes, leite/derivados e ovos, incluindo aqueles com algum grau de intolerância à lactose - que já atinge 70% dos adultos brasileiros. (SVB, 2018)

O mercado nacional tem acompanhado essas mudanças, fazendo surgir estabelecimentos focados nesse público, como restaurantes, lanchonetes e até mesmo grandes supermercados. No entanto, o segmento ainda não usufrui de forma efetiva da tecnologia que tem se destacado nesta década: os *smartphones*.

É inegável que os aparelhos celulares se tornaram itens essenciais do cotidiano. Em pesquisa realizada pelo IBGE em 2016 e divulgada em fevereiro de 2018, o celular estava presente em 92,6% dos domicílios pesquisados; 94,6% dos usuários da Internet se conectaram nela por meio de celular; e em 38,6% das residências pesquisadas, o celular foi o único meio de acesso à Internet utilizado (IBGE, 2018). É justo imaginar que, de 2016 a 2019, a presença dos *smartphones* na vida dos brasileiros se ampliou ainda mais.

Isso possibilitou o surgimento de diversos serviços e aplicações baseadas nos dispositivos móveis, dentre os quais se destacam os aplicativos de entrega de alimentos, como Uber Eats e iFood, este último tendo atingido, em 2018, o patamar de “empresa unicórnio” (FREITAS, 2018).

No entanto, existem poucas opções de aplicativos voltados para o público vegano e vegetariano do Brasil. De modo geral, essas aplicações têm enfoque na entrega de alimentos, o que já é realizado por serviços melhor consolidados, ou possuem caráter puramente informativo, sem a dinamicidade necessária para se tornar parte da rotina de seus usuários (ENCONTRO DIGITAL, 2017). Além disso, boa parte desses aplicativos são estrangeiros, não traduzidos para a Língua Portuguesa, o que representa obstáculo para a democratização de seu uso (SIGMAAPP, 2018).

No segundo semestre de 2018, a equipe Sigma App Development[1], doravante chamada apenas de Sigma, enxergou nessa carência uma boa oportunidade de negócios e produziu o NaVeg, aplicativo interessado em atender ao mercado referido anteriormente, com foco em Fortaleza e Região Metropolitana. O aplicativo foi desenvolvido como trabalho final da disciplina Projeto Integrado II, do curso de Sistemas e Mídias Digitais (SMD), da Universidade Federal do Ceará (UFC), que tem como principal objetivo a criação de um produto multimídia inovador.

No entanto, apesar do potencial de mercado, a criação do NaVeg esbarra em um fator dificultador: o custo de desenvolvimento. Desenvolver aplicações móveis não é tarefa trivial, e isso tem ainda mais impacto quando se considera a grande variedade de *smartphones* disponíveis no mercado, especialmente em relação a seus Sistemas Operacionais (SO).

Atualmente, o mercado de smartphones é constituído, majoritariamente, pelos SO Android, da Google, e iOS, da Apple. De acordo com estatísticas do site Net MarketShare (2019), o Android está presente em 70,22% dos aparelhos móveis de todo o mundo, enquanto o iOS abrange uma fatia de 28,17%, deixando apenas 1,61% a ser dividido entre outros sistemas. Se essas estatísticas forem filtradas para o Brasil, o uso do Android se acentua, representando 85,95% dos dispositivos, contra 13% do iOS (STATCOUNTER, 2019).

Essa lacuna não significa que o iOS possa ser relevado durante o desenvolvimento de um aplicativo, mesmo porque o dispositivo da Apple está associado a classes com maior poder aquisitivo, as mesmas classes que estão se adequando a novos estilos de vida sustentáveis. Negligenciar o sistema poderia acarretar em perda do potencial de mercado da aplicação. Com isso em vista, como seria viável desenvolver um aplicativo para esse público e que pudesse abranger os dois SO?

---

[1] Integrantes da equipe Sigma: Francisca Julianne Pessoa Holanda, Gabriel Rodrigues de Araujo, Hiago Bruno da Silva Rabelo e Thais Nascimento Gouveia.

Existem abordagens de desenvolvimento que buscam atacar essa deficiência: é o chamado desenvolvimento *cross-platform*[2]. Existem vários modos de se produzir um *software cross-platform*, mas todos eles têm o intuito de permitir que um aplicativo possa ser implantado em vários SO a partir de um único código base, muitas vezes em linguagem distinta da utilizada em qualquer um dos SO. Sua principal vantagem é a redução dos custos de desenvolvimento, tanto em relação à quantidade de pessoas envolvidas quanto em relação às futuras manutenções e expansões do aplicativo (HEITKÖTTER.; HANSCHKE; MAJCHRZAK, 2012, p. 3).

Existem variadas ferramentas de desenvolvimento *cross-platform*, algumas disponíveis gratuitamente, como é o caso do React Native. Este é um *framework* criado pelo Facebook em 2015 que utiliza JavaScript e a biblioteca React.js (doravante chamada apenas de React) para criar aplicativos praticamente nativos, tanto para Android como para iOS. O React Native tem crescido no mercado e adquirido cada vez mais importância, tornando-se uma alternativa viável para a implementação de aplicativos.

Tendo em vista os conceitos apresentados sobre o mercado atual de vegetarianismo e sobre o desenvolvimento *cross-platform*, o objetivo deste trabalho é relatar o processo de criação do aplicativo NaVeg com uso do *framework* React Native.

Os objetivos específicos são: compreender o React Native e suas tecnologias adjacentes; descrever as boas práticas de criação, organização e desenvolvimento de um aplicativo com React Native; relatar as dificuldades e desvantagens do uso do React Native; avaliar o esforço necessário para gerar o mesmo aplicativo em Android e iOS e comparar ambas as versões.

O restante deste trabalho está organizado em seis capítulos. No capítulo 2, conceituam-se o termo *cross-platform* e as tecnologias React e React Native. No capítulo 3, a metodologia que guia o desenvolvimento do projeto é brevemente apresentada e discutida. O capítulo 4 aborda a conceituação completa do NaVeg. No capítulo 5, são mostradas as etapas existentes na criação de um aplicativo com React Native, culminando em sua compilação para as plataformas Android e iOS. A seguir, no capítulo 6, é realizada uma análise entre os resultados finais de cada plataforma. O trabalho se conclui no capítulo 7, em que é feito um resumo do que foi apresentado no relatório e são sugeridos trabalhos para o futuro.

[2] Decidiu-se utilizar o termo em inglês em vez de sua tradução “multiplataforma”, pois este último carrega consigo um significado mais antigo, que não remete aos modelos atuais a que este trabalho se refere.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão explicados os principais termos e conceitos que permeiam este trabalho. Será esclarecido e significado de *cross-platform* as técnicas que podem ser utilizadas para desenvolver aplicativos com essa abordagem. Será apresentada uma explicação sobre a biblioteca React, com destaque ao seu uso de JSX e componentes. Por fim, será abordada a definição de React Native e como esse *framework* pode ser usado.

2.1 Cross-platform

Cross-platform é uma abordagem de desenvolvimento de software em que uma aplicação é desenvolvida uma única vez e pode ser utilizada ou transportada para diferentes sistemas. Esse tipo de metodologia surgiu para aumentar a produtividade dos desenvolvedores, reduzir a repetição de código e facilitar a criação de programas que sejam executados em várias plataformas, permitindo às empresas abrangerem uma gama maior de usuários (HEITKÖTTER.; HANSCHKE; MAJCHRZAK, 2012).

Cada SO, seja ele móvel ou não, possui capacidades, fluxos e linguagens próprias, que, na maioria das vezes, não são compatíveis ou consistentes com outras plataformas. O desafio do *cross-platform* é entender cada uma dessas plataformas e encontrar nelas pontos em comum que possam ser explorados. Assim, essa abordagem traz consigo, necessariamente, uma certa generalidade ao *software*, para que cada sistema usufrua da mesma qualidade, mas ao mesmo tempo precisa permitir que os desenvolvedores possam "desacoplar" suas aplicações e realizar ajustes finos focados em cada uma das plataformas (HEITKÖTTER.; HANSCHKE; MAJCHRZAK, 2012, p. 1-2).

É importante destacar que não existe apenas uma, mas várias maneiras de utilizar *cross-platform*. Por exemplo, é possível criar websites responsivos[3], que serão executados em um navegador, mas que se comportam e se assemelham a aplicativos nativos. É o caso, por exemplo, dos aplicativos como Google Drive, Google Documentos e Google Keep.

Outra técnica popular é programar com *frameworks* híbridos, dos quais se destacam Ionic[4] e PhoneGap[5]. "Essa abordagem é uma combinação de desenvolvimento web

[3] Responsividade é a capacidade de um website de se adaptar visualmente a diferentes tamanhos de tela.
[4] Ionic Framework - https://ionicframework.com/
[5] Adobe PhoneGap - https://build.phonegap.com/

e nativo, pois a aplicação é construída com técnicas web, mas é executada, renderizada e exibida como uma aplicação nativa utilizando WebView[6]" (DANIELSSON, p. 8, tradução nossa). Danielsson ainda explica que, apesar de menos custosos, os aplicativos híbridos não conseguem gerar a mesma experiência de aplicativos nativos para o usuário e, portanto, não podem ser seus substitutos definitivos (DANIELSSON, p. 8). Isso acontece porque os aplicativos executados por WebView não possuem acesso ao *hardware* do dispositivo ou possíveis otimizações advindas do SO (HEITKÖTTER.; HANSCHKE; MAJCHRZAK, 2012, p. 8).

Existe, ainda, o método do React Native, que utiliza a biblioteca React para produzir uma base de código única que é compilada para as plataformas alvo (neste caso, Android e iOS), as quais passam a compreender JavaScript. Isso permite que o desenvolvedor produza aplicações mais eficientes e que têm, de fato, uma sensação nativa, já que "os componentes JavaScript são declarados usando uma série de primitivas embutidas suportadas tanto por iOS como por Android" (DANIELSSON, p. 10, tradução nossa).

2.2. React

Em 2013, o Facebook lançou para o mundo, no formato de código-aberto, a biblioteca React, com a missão de facilitar o desenvolvimento de interfaces Web complexas, a exemplo do próprio Facebook. Antes de abrir o código para a comunidade, a empresa já trabalhava na React há alguns anos, buscando resolver um dos maiores problemas da Web: o custo operacional de manipular o Document Object Model (DOM) (DAWSON, 2014).

O DOM é uma estrutura em formato de árvore que descreve a organização e o estado atual dos elementos de um documento (MOZILLA, 2019). Quando um usuário acessa uma página Web, o DOM dessa página precisa ser criado para permitir que seu conteúdo seja exibido no navegador. Do mesmo modo, sempre que "a aplicação receber dados de um servidor, [...] o conteúdo do DOM é manipulado por JavaScript" (DANIELSSON, 2016, tradução nossa) e a aplicação se modifica novamente. O problema, como Danielsson explica, é que "[...] a manipulação do DOM é custosa, e isso se tornou um mantra" (DANIELSSON, 2016, tradução nossa), de modo que, por muito tempo, evitou-se, a todo custo, realizar essa manipulação.

[6] WebView – https://developer.android.com/reference/android/webkit/WebView

O Facebook enxergou nisso uma oportunidade e criou a React, biblioteca rápida, que otimiza a manipulação do DOM e encurta o código de aplicações Web. Isso é possível porque a React guarda uma cópia do DOM, e toda manipulação é realizada nessa cópia, em vez de utilizar o DOM diretamente. Quando o usuário interage com a página, o estado guardado na React se modifica, e somente a parte modificada é novamente renderizada no navegador (DAWSON, 2014).

2.2.1 JSX

Normalmente, a React é usada em conjunto com JavaScript XML (JSX), uma extensão da sintaxe do JavaScript que se assemelha bastante ao HTML. Um exemplo da notação JSX é mostrado no Código-fonte 1. Nele, observa-se que o valor da constante *element* não é *string*, e tampouco HTML, mas uma estrutura JSX. Essa estrutura é um elemento da React que poderá ser renderizado na interface do usuário e, então, se transformará em HTML (neste caso, a *tag* <h1>Hello, world!</h1>).

Código-fonte 1 – Exemplo de JSX

```
1 const element = <h1>Hello, world!</h1>
```

Fonte: transcrito da documentação da React (https://reactjs.org/docs/introducing-jsx.html)

JSX pode conter expressões JavaScript, como cálculos matemáticos e operações com *strings*. Todo JSX, após compilado, se transforma em uma função normal de JavaScript, e gera objetos JavaScript. No exemplo a seguir, mostra-se como esse recurso é utilizado. Neste exemplo, o código irá renderizar a frase *Hello, Josh Perez* a partir da constante *element*, que recebe o valor da constante *name*.

Código-fonte 2 – Expressões JavaScript em JSX

```
1 const name = 'Josh Perez';
2 const element = <h1>Hello, {name}</h1>;
3
4 ReactDOM.render(
5   element,
```

```
  document.getElementById('root')
);
```

Fonte: transcrito da documentação da React (https://reactjs.org/docs/introducing-jsx.html)

Quando o JSX surgiu, houve desconforto entre programadores mais experientes, pois essa sintaxe faz com que o código JavaScript se misture com HTML e CSS, algo que por muito tempo foi considerado má prática de projeto, pois dificultava a manutenção do código. JSX traz uma abordagem diferente: "Ao invés de separar artificialmente tecnologias colocando marcação e lógica em arquivos separados, a React separa **conceitos** por meio de unidades fracamente acopladas chamadas 'componentes', que contêm ambas [marcação e lógica] " (REACT, 2019, tradução nossa, ênfase nossa).

Isso tem semelhanças com as classes na Orientação a Objetos, e o seu principal benefício é possibilitar a criação de componentes pequenos, de fácil manutenção, e que encapsulam em si tudo o que é necessário para descrevê-los, seja funcional ou visualmente.

2.2.2 Componentes

O principal conceito da React é que toda a interface é subdividida em componentes. Estes são pequenas partes de código reutilizável que reúnem funcionalidades e estruturas intimamente relacionadas, e que estão, conceitualmente, isoladas do restante da interface (REACTJS, 2019). Por exemplo, o campo de pesquisa no cabeçalho de um site pode ser entendido como um componente. Ele não precisa conhecer o funcionamento ou estrutura de outras partes da interface.

Um componente pode ser formado por outros componentes, assim como uma função pode chamar outras funções. Conceitualmente, a própria aplicação pode ser considerada um componente que agrega todos os outros componentes que existirão. Essa abstração torna fácil e intuitiva a criação e a fragmentação das partes do projeto.

Componentes são formados por métodos, *props* e *states*. Estes últimos são semelhantes aos atributos de uma classe na Orientação a Objetos. *Props* guardam valores próprios do componente, como cores e *labels*, e dificilmente mudam. *States* guardam valores dinâmicos que representam o estado do componente em dado momento. O estado de *focus* em um campo de formulário é um exemplo de *state*.

2.3 React Native

React Native é a ferramenta que usufrui de todos os benefícios da biblioteca React (otimização do DOM, componentes, etc.) e os utiliza para criar interfaces de aplicativos *cross-platform*. O React Native executa uma instância do JavaScriptCore[7] dentro da aplicação, e dessa forma ele consegue renderizar no dispositivo componentes específicos de cada plataforma.

React Native descreve uma série de componentes genéricos, como <View>, que são entendidos tanto por iOS (renderizado como UIView) quanto por Android (renderizado como View). Além disso, existem componentes específicos para cada plataforma, como <DatePickerIOS>, que renderiza no iOS um seletor de datas. Como a React não está rigidamente ligada ao DOM, ela é então capaz de encapsular qualquer tipo de sistema de visualização (OCCHINO, 2015), tornando-a flexível e virtualmente capaz de gerar interfaces para qualquer plataforma.

> Uma das melhores partes dessa abordagem é que nós podemos adotá-la incrementalmente, construindo novos produtos em cima dela e convertendo produtos antigos para usá-la onde e quando necessário. Assim como não precisamos reescrever o Facebook.com para começar a usar React em alguns pontos, nós não precisamos refazer todos os nossos aplicativos mobile do Facebook para começarmos a tirar proveito dos benefícios do React Native. (OCCHINO, 2015, tradução nossa)

Como apontam Axelsson e Carlström (2016), o objetivo do React Native não é ser uma ferramenta de template, em que se escreve o código uma única vez e ele é executado da mesma forma em todas as plataformas. Pelo contrário, a sintaxe do React Native reconhece que as plataformas-alvo são diferentes, e que isso é algo que deve ser explorado, em vez de contornado, a fim de melhorar a experiência do usuário, sem que para isso o desenvolvedor precise aprender novas técnicas ou linguagens.

O *framework* possui características que podem ser enxergadas como negativas, dependendo do tipo de aplicação que se pretende criar. Pelo fato de ele utilizar uma base de código compartilhada para dois SO, sendo esse código criado de maneira a contemplar ambas

[7] JavaScriptCore é a *engine* por trás dos navegadores que utilizam Webkit, que, por sua vez, é uma *engine open source* utilizada por navegadores como Safari e Opera.

as plataformas de forma genérica, o aplicativo resultante não possuirá otimizações específicas para essas plataformas, ocasionando desempenhos menores do que os de aplicativos nativos (DANIELSSON, p. 30).

Além disso, é notório que o React Native precisa utilizar diversas dependências para construir suas aplicações. Isso faz com que o *software* resultante consuma mais espaço em disco de forma desnecessária, pois essas dependências carregam códigos que nem sempre são utilizados em sua totalidade.

Por fim, de maneira mais abrangente, o React Native é um *framework* relativamente recente e que está em constante evolução; *bugs* são esperados na ferramenta, sejam eles grandes ou pequenos, e isso pode trazer barreiras durante o desenvolvimento do produto.

3 METODOLOGIA

Como exposto anteriormente, o curso de SMD possui em sua grade curricular a disciplina Projeto Integrado II. Seus principais objetivos são: desenvolvimento de produtos/serviços multimídia; oferecer aos alunos vivência de processos; estabelecer relacionamentos de equipe e fornecer bases para o Trabalho de Conclusão de Curso. Por ser uma das últimas cadeiras do curso, entende-se que os alunos acumularam aptidões em sua trajetória acadêmica que lhes permitem criar produtos inovadores de qualidade profissional e grande valor de mercado.

A disciplina teve seu formato atual elaborado por professores do SMD (COUTINHO et al., 2016). Esse formato torna imperativo o uso da metodologia de Design Thinking por todas as equipes e, portanto, será a metodologia descrita, resumidamente, neste trabalho. Uma explicação mais detalhada sobre *Design Thinking* pode ser encontrada no Apêndice A deste relatório.

3.1 Design Thinking

Design Thinking é um modelo mental iterativo que estabelece alguns processos de *design* para se projetar a solução de um problema (ESCOLA DESIGN THINKING, 2016, p. 15). Esses processos não têm rigidez em sua ordem, podendo-se retornar a etapas anteriores sempre que preciso. Ele valoriza a experiência e a colaboração de pessoas, entendendo que erros são naturais e podem funcionar como testes. Por isso, todos os participantes são incentivados a prestar sua colaboração, trazendo várias perspectivas ao problema enfrentado.

O *Design Thinking* possui diversas etapas, as quais podem variar entre diferentes autores. Para Ambrose e Harris (2011), essas etapas são: Definição, Pesquisa, Ideação, Prototipação, Seleção, Implementação e Aprendizagem. Durante a disciplina, todas elas foram contempladas de alguma forma. Porém, por limitação do escopo deste trabalho, será dado enfoque à etapa de Implementação; algumas etapas serão condensadas (Definição, Pesquisa e Seleção), e outras, ainda, serão eliminadas (Ideação, Prototipação e Aprendizagem).

3.2 Organização da equipe

É válido citar que, além da metodologia da própria disciplina, a equipe se inspirou no Scrum para trazer características de metodologias ágeis para sua organização interna.

Scrum é uma metodologia ágil para gestão de projetos de software e se baseia em pequenos ciclos, chamados *sprints*, para realizar um desenvolvimento iterativo do produto (DESENVOLVIMENTOÁGIL, 2014). Antes de um *sprint*, realiza-se uma reunião para se definir quais funcionalidades serão desenvolvidas dentro daquele ciclo. Scrum engloba práticas voltadas para a rapidez e eficiência das ações da equipe, para a participação ativa do cliente, para a capacidade de reagir a mudanças durante a execução do projeto.

A equipe utilizou essas características como base, mas sem rigidez no formato. Periodicidade de reuniões, uso de *sprint* e documentação cíclica foram algumas dessas características.

Os integrantes se dividiram em quatro comissões, cada uma com gerente, responsável por monitorar e tomar decisões relativas à comissão, e um subgerente, visto como um auxiliar às atividades do gerente. O objetivo dessa disposição foi distribuir a carga de trabalho e potencializar as habilidades individuais dos membros.

No Quadro 1, está disposta a organização da equipe:

Quadro 1 – Divisão de comissões da equipe Sigma

Comissão	**Gestão do Projeto**	**Programação**	**Direção de arte**	**UX**
Gerente	Hiago Rabelo	Gabriel Araujo	Julianne Holanda	Thais Gouveia
Subgerente	Gabriel Araujo	Hiago Rabelo	Thais Gouveia	Julianne Holanda

Fonte: elaborado pela equipe Sigma (http://sigmaapp.com.br/projeto/projeto-do-produto/).

4 NAVEG

Nesta seção, será feita uma descrição do produto deste trabalho. O intuito é estabelecer os problemas que o aplicativo NaVeg busca resolver, e como isso pode ser realizado em termos de funcionalidade. Apresentam-se os resultados da pesquisa realizada pela equipe durante a disciplina que visou entender o mercado local e a familiaridade e aceitabilidade do público com um possível novo aplicativo.

Por fim, a solução é descrita de forma completa, com a elaboração de um modelo conceitual, um esquema de arquitetura e a enumeração das principais funcionalidades do aplicativo.

4.1 Definição do problema

Tomando como base o contexto do mercado descrito na introdução deste trabalho, o problema que a equipe almejou atacar foi a dificuldade que o público vegano e vegetariano tem de descobrir lugares e eventos relacionados com o vegetarianismo, tendo que recorrer normalmente a indicações de amigos e redes sociais. Enxergou-se ser difícil, também, conhecer os tipos de pratos servidos nos restaurantes e suas informações nutricionais.

Inicialmente, não foi elencado um cliente específico. Portanto, o público-alvo, composto pela população vegetariana e vegana de Fortaleza e Região Metropolitana, foi considerado como cliente em potencial.

4.2 Pesquisa de mercado

Tendo em vista os problemas levantados pela equipe, e sua possível solução, foi realizada uma pesquisa de mercado para validar o levantamento inicial e entender se o recurso proposto combateria, de fato, os problemas citados. A pesquisa foi aplicada por meio de um questionário online, via Google Forms[8], divulgada em grupos de Fortaleza no Facebook e por e-mail. Ela foi disponibilizada durante alguns dias de setembro de 2018 e seus resultados completos podem ser encontrados no site da equipe Sigma por meio do endereço http://sigmaapp.com.br/tarefas/pesquisa/.

[8] Google Forms - Ferramenta do Google para criação de formulários e captação de dados.

Em resumo, a pesquisa coletou dados de 44 participantes, quantidade que, apesar ser grande em relação a respostas *online*, não pode ser generalizada para a população de Fortaleza. Identificou-se, na maioria das respostas, falta de conhecimento de aplicativos do ramo e aqueles que eram conhecidos tinham relação com entrega de alimentos em domicílio. A pesquisa percebeu, ainda, que algumas pessoas não utilizaram nenhum aplicativo do meio por falta de necessidade do usuário. O estudo detectou, ainda, que a experiência de uso de aplicativos vegetarianos foi apontada como negativa, pois nenhum dos usuários entendeu que os aplicativos utilizados atenderam suas necessidades com excelência.

Apesar dos resultados do estudo apontarem para falta de interesse e necessidade de um aplicativo como o proposto pela equipe Sigma, considerou-se que as pesquisas recentes sobre o mercado vegetariano no Brasil (SVB, 2018) não devem ser ignoradas e que apresentam boas perspectivas para quem quer empreender na área. Portanto, julgou-se admissível que a equipe arriscasse o desenvolvimento de um aplicativo para esse setor, mesmo a pesquisa não apresentando os resultados positivos esperados.

4.3 Análise de concorrência

Em paralelo à pesquisa, a equipe realizou uma análise de soluções digitais com propostas semelhantes à do NaVeg. Diversos aplicativos e plataformas Web, tanto brasileiros como estrangeiros, foram encontrados, mas a análise se limitou a quatro deles, que foram escolhidos por terem *ranking* mais elevado em pesquisas no Google e por seus nomes serem consistentemente citados em artigos e *posts* na Web. Os resultados da análise são condensados no Quadro 2, abaixo.

Quadro 2 – Comparativo entre os aplicativos analisados

Aplicativo	**Pontos positivos**	**Pontos negativos**
BeVeg	Público diversificado; Design limpo; Funcionalidades claras; Integrado com Google Maps; Modelo B2B para cadastro de delivery.	Design apresenta inconsistências; Aplicativo descontinuado.

Velivery	Identificação por selos (vegetariano, vegano, sem lactose, sem glúten); Disponível para Fortaleza; Realiza busca por bairro; Permite montagem de prato.	Ainda em versão Beta; Poucas opções de restaurante; Não possui visão de mapa para ver os restaurantes próximos.
HappyCow	Visualização de mapa; Opções de avaliação e compartilhamento dos restaurantes; Design simples; Navegação intuitiva.	Aplicativo pago; Idioma: apenas inglês.
Vanilla Bean	Visualização de mapa; Filtros e busca avançada; Design simples; Navegação intuitiva.	Não funciona em vários países.

Fonte: elaborado pelo autor, com base em pesquisa da equipe Sigma (http://sigmaapp.com.br/tarefas/pesquisa/)

A comparação foi realizada pela própria equipe, que buscou informações sobre os aplicativos e utilizou cada um deles em seus próprios *smartphones*. Foram verificados aspectos de interface, como clareza visual, consistência e arquitetura de informação. Em relação aos aspectos funcionais, a análise verificou a corretude das funcionalidades propostas nos aplicativos.

Por fim, a avaliação dos aplicativos concorrentes permitiu que a equipe adquirisse uma visão mais ampla sobre o problema que almejava resolver e gerasse novas ideias para a proposta do aplicativo.

4.4 Proposta do produto

O objetivo do NaVeg é fornecer uma ferramenta que auxilie vegetarianos e veganos (e pessoas com interesses similares a este estilo de vida) a descobrir e acompanhar os principais eventos, produtos e serviços relacionados, além de dicas de adaptação para aqueles que também querem aderir a estes movimentos. O aplicativo deve, ainda, permitir que o

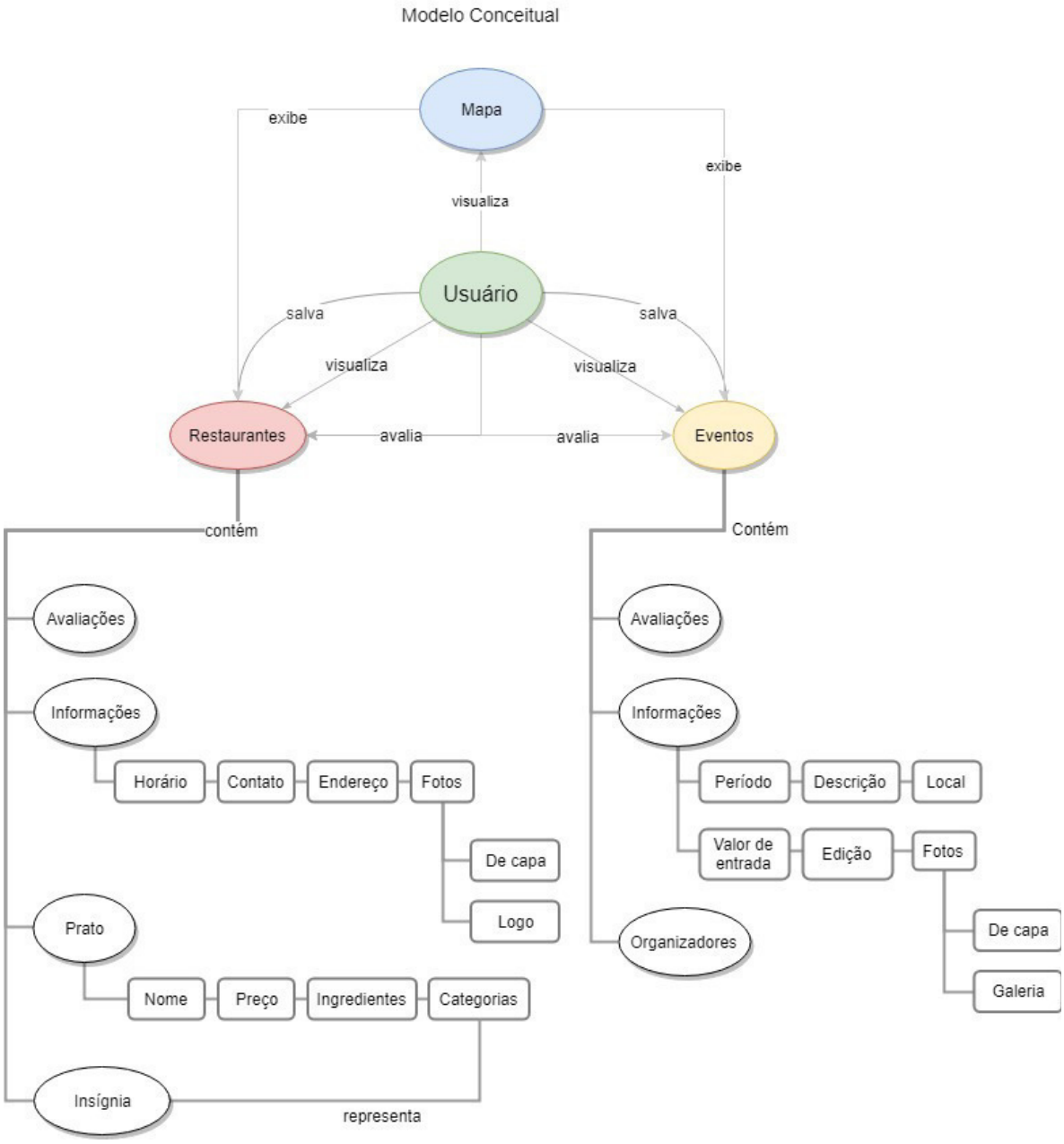
usuário obtenha informações nutricionais sobre produtos fornecidos por restaurantes e outros estabelecimentos, e possa realizar pedidos desses produtos para entrega (*delivery*).

Aplicativos similares ao NaVeg geralmente têm um enfoque informacional ou de prestação de serviço para entrega de alimento; o diferencial do NaVeg é o enfoque em eventos e na programação cultural que possa vir a interessar o público-alvo.

4.5 Descrição do sistema

Na Figura 1, apresenta-se o modelo conceitual do NaVeg, elaborado pelos membros da equipe em conjunto.

Figura 1 – Modelo Conceitual do NaVeg



Fonte: equipe Sigma (http://sigmaapp.com.br/projeto/projeto-do-produto/)

O modelo conceitual possibilita relacionar as principais funcionalidades do aplicativo:

- Listar eventos veganos e vegetarianos;
- Listar restaurantes veganos e vegetarianos;
- Acessar informações de eventos e de restaurantes;

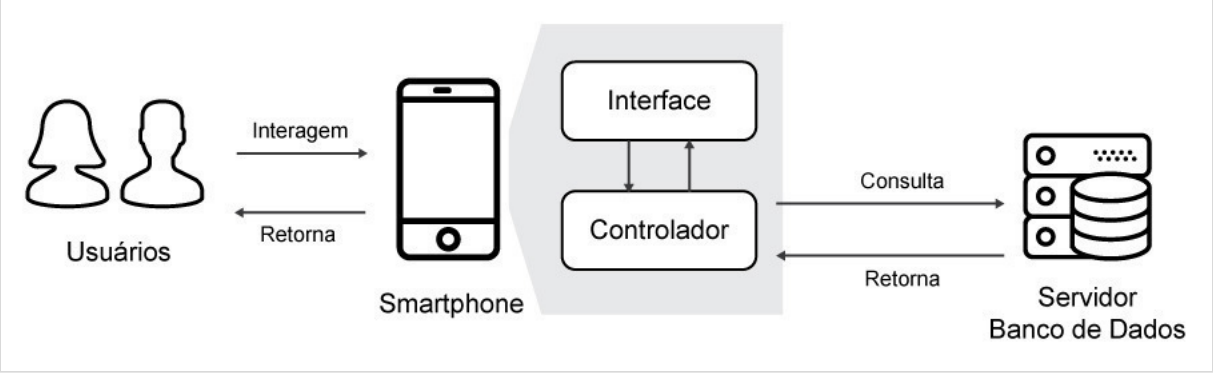
- Avaliar eventos e restaurantes, atribuindo notas;
- Comentar em eventos e restaurantes;
- Salvar eventos e restaurantes como favoritos;
- Visualizar mapa, que exibe eventos e restaurantes.

Também é possível identificar os agentes que utilizarão o sistema:

- Usuários do aplicativo: pessoas que fazem parte do público-alvo;
- Donos(as) de restaurantes: pessoas responsáveis por fornecer as informações dos restaurantes listados no sistema;
- Organizadores(as) de eventos: pessoas responsáveis por fornecer as informações dos eventos listados no sistema;
- Administradores do sistema: pessoas que têm acesso ao sistema e podem cadastrar as informações fornecidas pelos donos de restaurantes e organizadores(as) de eventos.

Além disso, foi elaborado um modelo de Arquitetura do Sistema, que permite enxergar a forma como as tecnologias usadas se relacionam e se comunicam. É válido ressaltar que, por causa do tempo limitado da disciplina, a equipe decidiu limitar o escopo inicial do projeto ao desenvolvimento para Android e isso foi refletido na arquitetura montada.

Figura 2 – Arquitetura do sistema



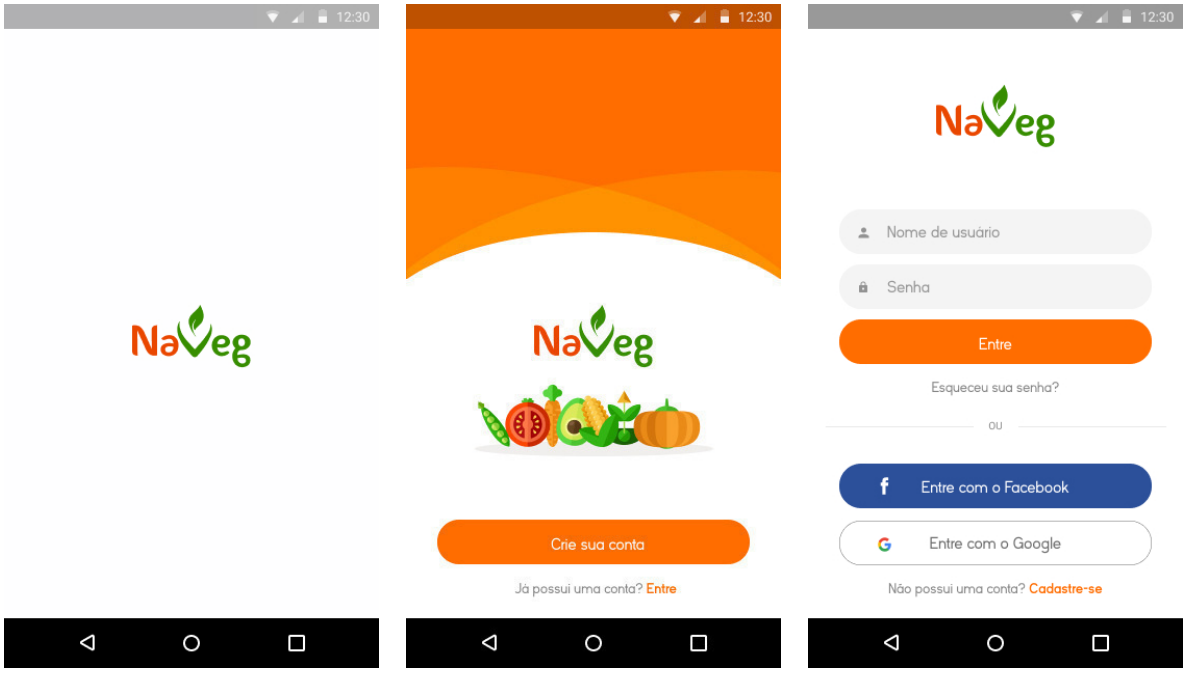
Fonte: equipe Sigma (http://sigmaapp.com.br/projeto/projeto-do-produto/)

4.6 Interface

Com base nas informações anteriores, a Comissão de Direção de Arte criou as telas que compõem a interface do NaVeg. Como guia, foram utilizadas as diretrizes do Material Design[9], da Google.

A Figura 3 mostra as telas introdutórias do NaVeg, compostas pela *Splash Screen*, apresentada ao usuário enquanto o aplicativo é carregado, pela tela de *login* e pela tela de cadastro, apresentada somente quando o usuário não possui *login* cadastrado.

Figura 3 – Interface do NaVeg: Telas introdutórias



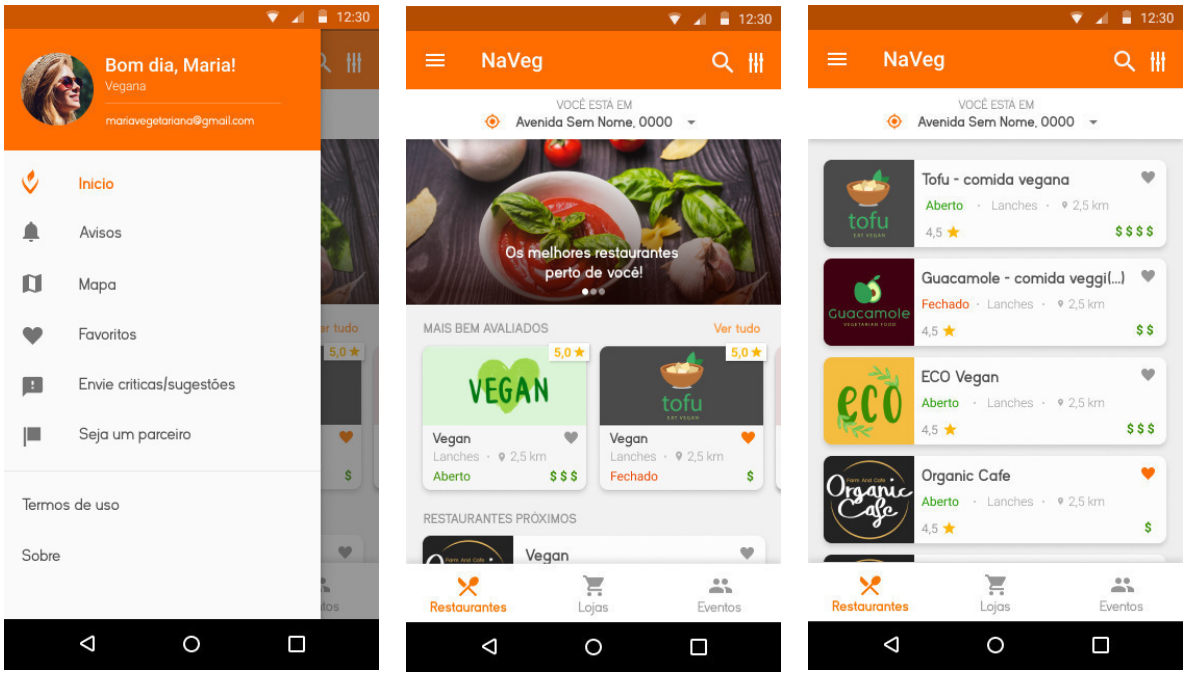
(a) *Splash screen* (b) Tela de *login* (c) Tela de cadastro

Fonte: montagem elaborada pelo autor com base em *mockups* da equipe Sigma.

Na Figura 4a é exibido o menu principal do NaVeg, também chamado de *DrawerMenu*[10]. As Figuras 4b e 4c mostram a tela principal do aplicativo, na qual é apresentada uma lista dos restaurantes próximos ao usuário.

[9] Material Design - https://material.io/design/. Acessado em 15 jun. 2019.
[10] *Drawer* é o nome dado para menus com esse comportamento e layout.

Figura 4 – Interface do NaVeg: Menu e listagem de restaurantes

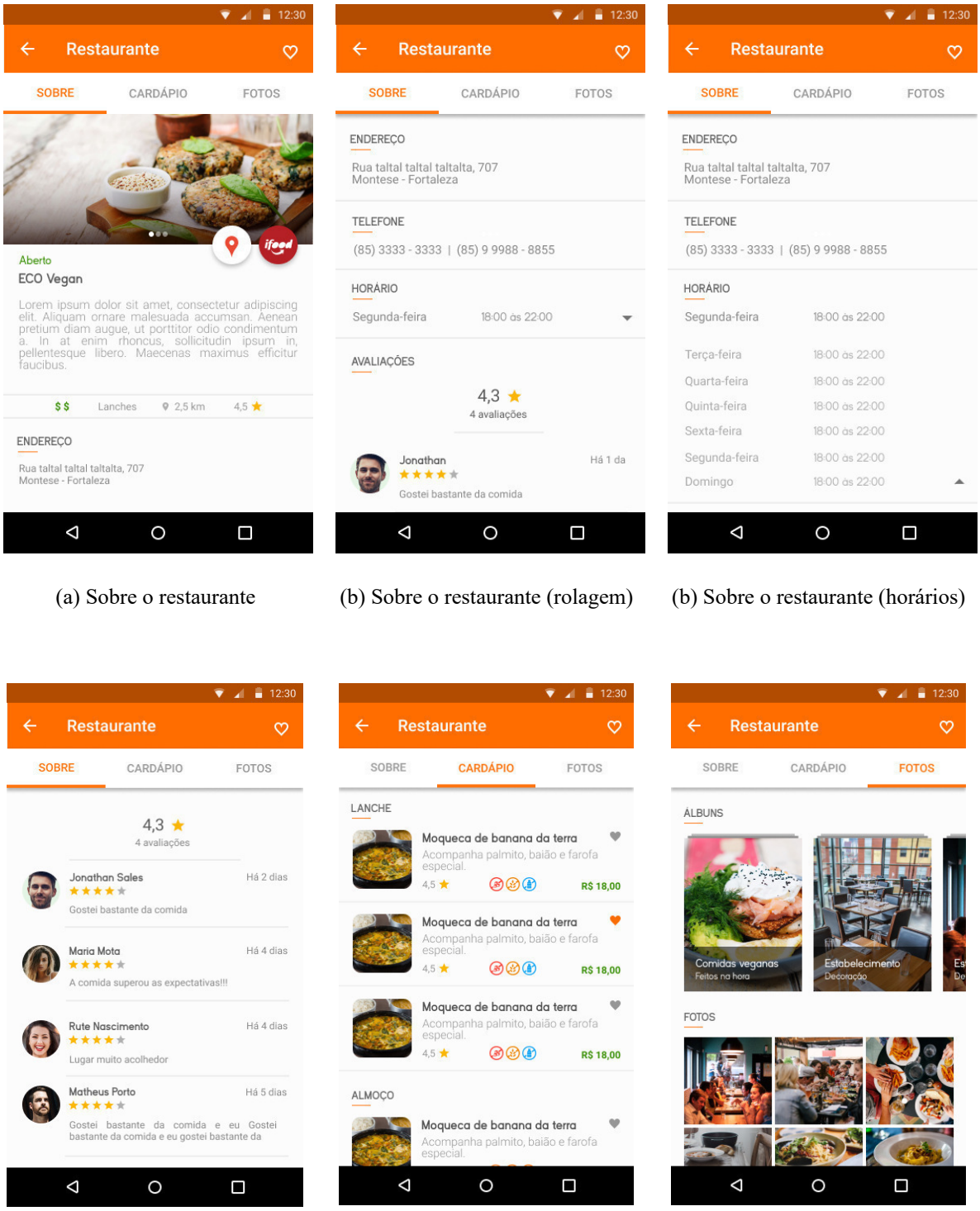


(a) Menu principal (b) Listagem de restaurantes (c) Listagem de restaurantes (rolagem)

Fonte: montagem elaborada pelo autor com base em *mockups* da equipe Sigma.

A Figura 5 mostra, em detalhes, as várias partes que compõem a tela de um restaurante.

Figura 5 – Interface do NaVeg: Telas do restaurante



(a) Sobre o restaurante

(b) Sobre o restaurante (rolagem)

(b) Sobre o restaurante (horários)

(d) Sobre o restaurante (avaliações)

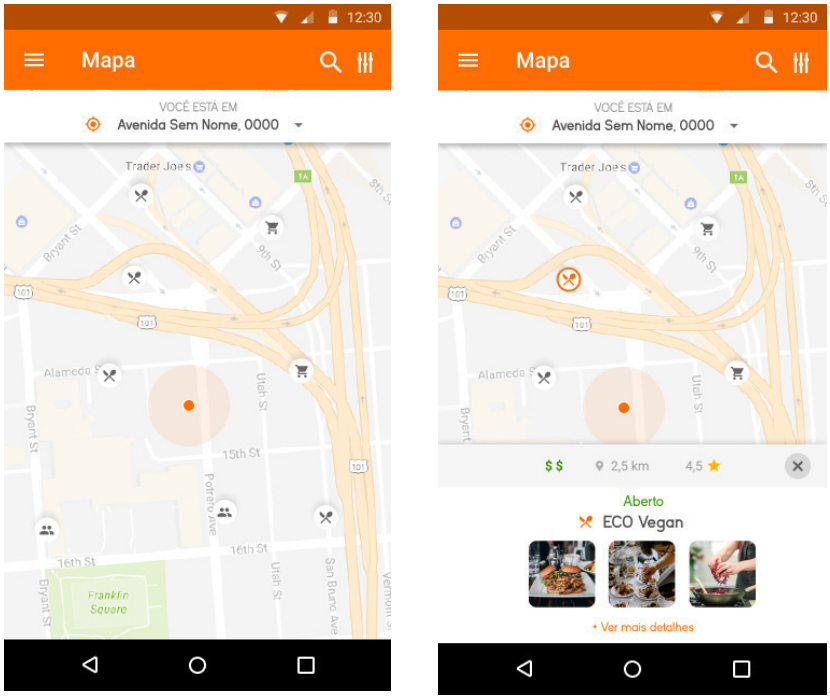
(e) Cardápio do restaurante

(f) Galeria de fotos do restaurante

Fonte: montagem elaborada pelo autor com base em *mockups* da equipe Sigma.

A Figura 6 mostra a visualização de restaurantes, eventos e outros estabelecimentos em um mapa, que está presente no NaVeg.

Figura 6 – Interface do NaVeg: Mapa



(a) Sobre o restaurante (avaliações) (b) Cardápio do restaurante

Fonte: montagem elaborada pelo autor com base em *mockups* da equipe Sigma.

5 DESENVOLVIMENTO DO PRODUTO

Esta seção relata o desenvolvimento do NaVeg com React Native. São abordados diversos aspectos, como as configurações iniciais do projeto, organização de diretórios, boas práticas utilizadas e dificuldades enfrentadas durante as fases de escrita e *deploy* (implantação) da aplicação.

É importante destacar que, devido ao tempo limitado em que o projeto foi desenvolvido (cerca de quatro meses, correspondentes ao tempo da disciplina), as únicas telas que puderam ser concluídas foram: Splash Screen (Figura 3a), Menu Principal (Figura 4a), Listagem de Restaurantes (Figuras 4b e 4c), e as telas do restaurante (Figura 5).

5.1 Primeiros passos com React Native

Segundo a documentação oficial do React Native[11], existem duas formas de utilizar o *framework*: pelo React Native CLI e pelo Expo CLI. Esta última é a mais indicada para desenvolvedores inexperientes com a tecnologia e que desejam iniciar a programação rapidamente, pois não requer configurações robustas, sendo necessária apenas a instalação do ambiente Node.js[12] e do próprio Expo CLI, na linha de comando (Windows) ou terminal (macOS e Linux). Esse foi o método escolhido pela equipe.

Expo é uma ferramenta de linha de comando que encapsula projetos React Native em "projetos Expo", que possuem todos os recursos do *framework*, mas dispostos de forma mais organizada. Além disso, o Expo acrescenta ao desenvolvedor recursos de *debug* da aplicação e visualização do aplicativo em tempo real. Este último é possível por meio do Expo Client, um aplicativo disponível para Android e iOS que utiliza a rede Wi-Fi para criar uma conexão direta entre *smartphone* e ambiente de desenvolvimento.

Uma característica notável do Expo é que ele não permite misturar código React Native com código nativo (iOS ou Android). Portanto, a única forma de realizar otimizações específicas para cada plataforma é "ejetar" o projeto, ação que, neste contexto, significa transformar um projeto React Native em um projeto do Android Studio ou do XCode, os ambientes de desenvolvimento do Android e do iOS, respectivamente.

---

[11] React Native Docs - https://facebook.github.io/react-native/docs/getting-started. Acessado em 13 jun. 2019.
[12] Node.js, ambiente de execução JavaScript - https://nodejs.org/en/. Acessado em 13 jun. 2019.

5.1.1 Configuração do ambiente

A instalação do Expo CLI se resume a abrir a linha de comando do computador e, considerando o Node.js como já instalado, executar o comando:

```
npm install -g expo-cli
```

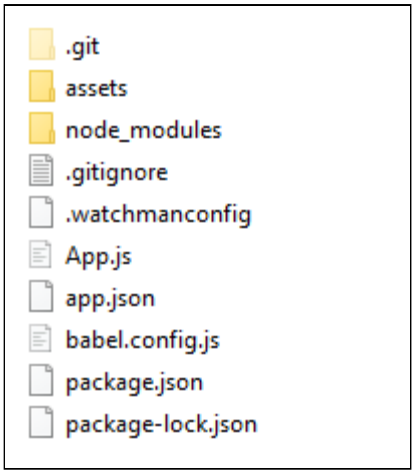
O Expo e suas dependências serão instalados. Após isso, pela linha de comando, escolhe-se o diretório que conterá o projeto e executa-se o comando:

```
expo init nome_do_projeto
```

No caso deste trabalho, “nome_do_projeto” foi substituído por “naveg”. A execução do comando anterior leva a um menu de opções, em que se pode escolher o template do projeto. Neste caso, escolheu-se o template *blank* (em branco). A partir disso, o Expo CLI lidará com a configuração inicial e *download* de dependências do novo projeto.

Ao fim desse processo, terão sido criados arquivos e pastas que seguem a estrutura exibida na Figura 7.

Figura 7 – Estrutura inicial de diretórios de projeto Expo



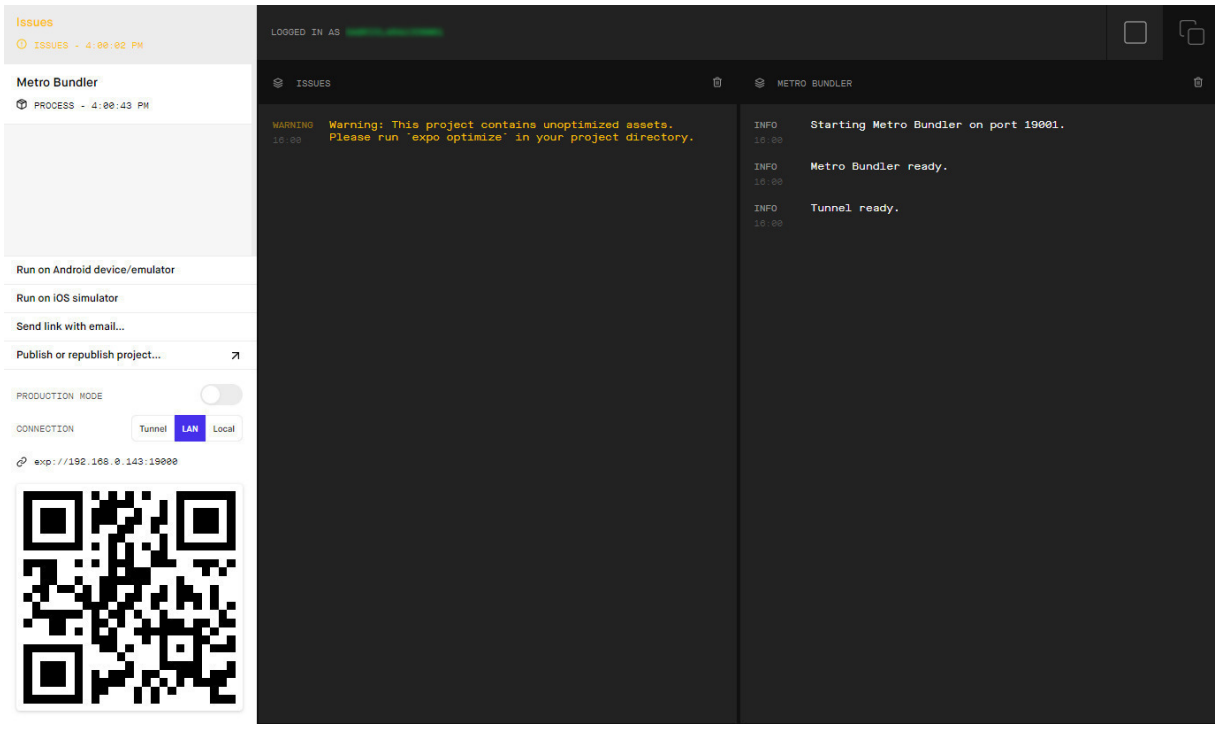
Fonte: elaborado pelo autor.

Para inicializar os recursos do Expo, basta executar o comando a seguir (no diretório raiz do projeto):

```
expo start
```

O servidor do Expo iniciará sua execução. Será aberta uma página local no navegador; ela indica o estado do servidor e da aplicação. A página contém opções de compartilhamento e publicação do aplicativo, bem como consoles que exibem mensagens diversas, como avisos, atualizações e erros.

Figura 8 – Página web do servidor Expo

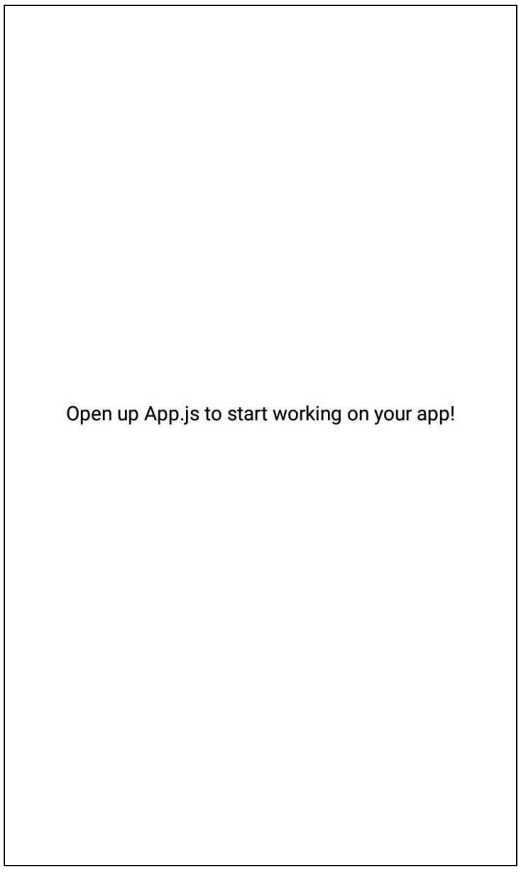


Fonte: elaborado pelo autor

Outro recurso importante dessa página é o *QR Code*[13] localizado no canto inferior esquerdo. Por meio dele, o aplicativo Expo Client pode se conectar ao servidor e exibir a aplicação em tempo real. Na Figura 9, é mostrada a captura de tela de um *smartphone* executando este projeto pelo Expo. Observa-se que a tela exibe apenas uma mensagem; essa é a única coisa implementada no template *blank*.

[13] QR Code – Wikipédia - https://pt.wikipedia.org/wiki/C%C3%B3digo_QR. Acessado em 20 jun. 2019.

Figura 9 – Aplicativo executado pelo Expo Client



Fonte: elaborado pelo autor

A partir desse momento, qualquer alteração no projeto será detectada pelo servidor, que reconstruirá o aplicativo no Expo Client. Virtualmente, o servidor não precisa ser desligado em nenhum momento do desenvolvimento.

5.1.2 Controle de versão

Para realizar o versionamento do código, a Comissão de Programação criou um repositório no site *Bitbucket*[14]. A plataforma é semelhante ao *Github* e foi escolhida por permitir a criação de repositórios privados gratuitamente (neste ano, o Github também adicionou essa possibilidade[15]).

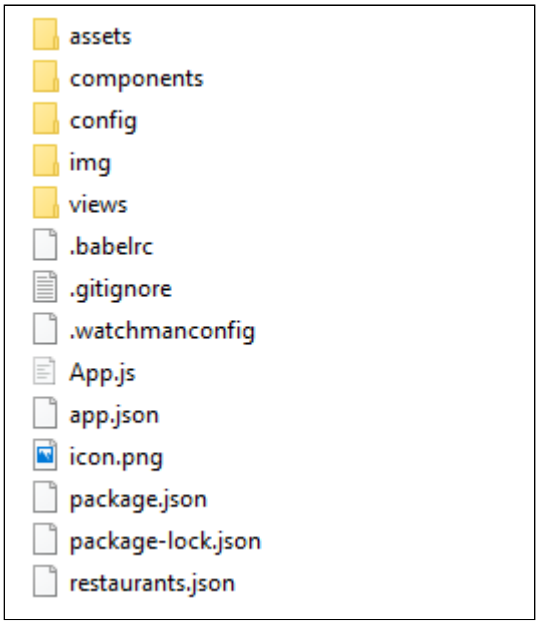
[14] Bitbucket. Disponível em: https://bitbucket.org/. Acessado em 13 jun. 2019.
[15] "GitHub agora permite repositórios privados em contas gratuitas". https://tecnoblog.net/274127/microsoft-github-repositorios-gratuitos-privados/. Acessado em 13 jun. 2019.

Posterior ao término da disciplina, a equipe decidiu tornar o repositório público. Ele pode ser acessado pelo *link* (https://bitbucket.org/gabrielaraujo9001/naveg/src/master/).

5.1.3 Organização de arquivos e pastas

Além da estrutura padrão do Expo, foram criadas novas pastas para facilitar a organização dos arquivos. A Figura 10 mostra a nova estrutura.

Figura 10 – Organização de pastas do NaVeg



Fonte: elaborado pelo autor.

No diretório raiz, encontram-se arquivos de configuração do Expo, além dos arquivos *App.js*, *app.json* e *package.json*, que serão abordados adiante. O arquivo *icon.png* é o ícone do aplicativo no celular e o arquivo restaurants.json foi usado pela equipe para armazenar dados de testes provisórios, e continua no projeto apenas por referência.

A pasta *assets* contém arquivos exportados das interfaces, como fontes e ícones. De modo semelhante, a pasta *img* guarda arquivos de imagem da interface. As pastas *components* e *views* possuem, ambas, componentes do React Native; os da primeira são aqueles com funções específicas, como *NavegIcon*, responsável por desenhar ícones na tela; os da segunda formam as telas do aplicativo, podendo agregar diversos componentes.

A pasta *config* lida com a configuração do banco de dados Firebase[16], o qual será tratado posteriormente neste trabalho.

5.1.4 Configurações (app.json e package.json)

Dois arquivos se responsabilizam pela configuração do projeto: *app.json* e *package.json*. O primeiro lida com as configurações do aplicativo que serão vistas e usadas pelo Sistema Operacional, tais como nome, descrição e versão. O segundo lida com configurações usadas pelo Expo, definindo, dentre outras coisas, as dependências do projeto. Ele é usado para realizar o deploy do aplicativo.

Código-fonte 3 – app.json: configurações do projeto

```
{
  "expo": {
    "name": "NaVeg",
    "description": "NaVeg - Sigma App Development",
    "slug": "naveg",
    "privacy": "public",
    "sdkVersion": "30.0.0",
    "platforms": ["android"],
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./icon.png"
    ...
  }
}
```

Fonte: elaborado pelo autor.

Código-fonte 4 – package.json: configurações do projeto

```
{
  "name": "naveg",
  "main": "node_modules/expo/AppEntry.js",
  "private": true,
```

[16] "Firebase - Google." https://firebase.google.com/. Acessado em 14 jun. 2019.

```
  ...
  "dependencies": {
    "expo": "^30.0.1",
    "firebase": "^5.5.9",
    "react": "16.3.1",
    ...
  }
}
```

Fonte: elaborado pelo autor.

5.2 App.js

O primeiro arquivo do NaVeg a ser carregado é o *App.js*. Ele define o componente *App*, responsável por inicializar parâmetros importantes anteriores à execução do programa. Esse arquivo permite analisar pontos essenciais sobre o React Native que se repetem em várias outras partes do código. O Código-fonte 5, a seguir, revela trechos do arquivo.

Código-fonte 5 – App.js: criação do componente *App*

```
import React from 'react';
import { Font } from 'expo';
import { Splash, DrawerMenu } from './views';

export default class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {hasLoaded: false};
    }
    ...
    componentDidMount() {
        this.timeoutHandle = setTimeout(()=>{
            this.setState({hasLoaded: true});
        }, 3000);
    }

    render() {
```

```
18          if(this.state.hasLoaded) {
19              return (<DrawerMenu />);
20          } else {
21              return (<Splash />);
22          }
23      }
24  }
```

Fonte: elaborado pelo autor.

O início do código mostra a importação de recursos necessários ao componente. Essa importação se refere apenas a este componente, não sendo compartilhada com outros, que devem realizar processo similar para acessar os mesmos recursos. Um ponto interessante do trecho em questão é que o React Native possibilita que componentes de um módulo sejam importados separadamente, como é o caso de *Font*, do módulo *expo*, não sendo necessário, portanto, carregar o módulo inteiro. A partir deste momento, será considerado, neste trabalho, que todo componente utilizado no código foi devidamente importado.

Após a importação, ocorre a declaração do componente *App*. Todo componente do React Native é uma classe que estende *React.Component*. Isso concede ao componente a capacidade de realizar a gerência de seu ciclo de vida. Essa habilidade é adquirida por meio de métodos como *constructor*, *componentDidMount* e *render*.

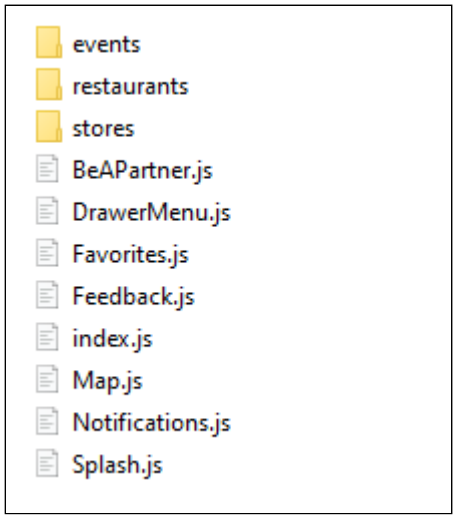
O *constructor* é responsável pela inicialização dos *props* e *states* do componente. Neste caso, o único *state* é o atributo *hasLoaded*, usado para indicar o carregamento da aplicação. O *componentDidMount* é chamado imediatamente após o carregamento completo do aplicativo, neste caso apenas modificando o valor de *hasLoaded* após três segundos. Todo *state* deve ser inicializado no método *constructor* e modificado por meio da função *setState*.

O método *render* desenha o valor de seu retorno (normalmente expressões JSX) na tela. No componente *App*, esse método renderiza a tela *DrawerMenu* ou *Splash* (advindas da pasta *views*), dependendo do valor de *hasLoaded*. Sempre que um *state* é modificado, o método *render* de seu componente é executado, gerando novo desenho da tela.

5.3 Views

O exemplo de código anterior revela a renderização de telas, ou *views*, do NaVeg. Como é observado na terceira linha do Código-fonte 5, elas são componentes presentes no diretório *views* do NaVeg, cuja estrutura é mostrada na Figura 11.

Figura 11 – Estrutura da pasta *views*



Fonte: elaborado pelo autor.

Apesar de haver arquivos referentes às *views Splash* e *DrawerMenu*, eles não são diretamente referenciados no código. Ocorre que, quando um diretório é referenciado no React Native, o *framework* busca o arquivo *index.js*. Como visto no Código-fonte 6, este faz a exportação das *views* do diretório em que se encontra. Assim, o componente *App* pode acessar todas elas, mesmo sem conhecer seus caminhos específicos. O método, considerado boa prática, distribui melhor as responsabilidades da aplicação e facilita sua manutenção.

Código-fonte 6 – views/index.js: exportação de *views*

```
1 ...
2 export DrawerMenu from './DrawerMenu.js';
3 export Splash from './Splash.js';
```

Fonte: elaborado pelo autor.

5.3.1 Splash

Splash é uma *view* simples, mas revela mais características sobre o React Native. Seu conteúdo pode ser visto no Código-fonte 7, a seguir:

Código-fonte 7 – Splash.js

```
...
export default class Splash extends React.Component {
  render() {
    return (
      <View style={styles.background}>
        <Image style={{width: 114, height: 60, marginBottom: 30}}
source={require('./../img/logo.png')} />
      </View>
    );
  }
}

const styles = StyleSheet.create({
  background: {
    backgroundColor: '#fff',
    height: '100%',
    alignItems: 'center',
    justifyContent: 'center',
    width: '100%'
  }
});
```

Fonte: elaborado pelo autor.

O componente *Splash* renderiza dois componentes básicos do React Native, <View> e <Image>, estilizados de maneiras distintas. Em <Image>, as propriedades de estilo são aplicadas diretamente no componente, sendo esta prática chamada de *inline*. Em <View>, as propriedades originam-se da constante *styles*, que é um objeto da classe *StyleSheet*, outro componente do React Native. Em ambos os casos, as propriedades são atributos de objetos JavaScript. A nomenclatura deles se assemelha ao CSS para facilitar o aprendizado.

Em geral, recomenda-se utilizar o segundo método, pois ele facilita a organização do código, melhora sua legibilidade, possibilita a criação de uma hierarquia de atributos dentro de um mesmo objeto e permite que esse objeto esteja alocado em arquivo diferente.

5.3.2 DrawerMenu

O componente *DrawerMenu* propicia a análise de um aspecto essencial de qualquer aplicação: a navegação. No React Native, o modo mais fácil de realizar navegação ou transição entre telas é utilizar o módulo React Navigation[17], que encapsula as formas mais comuns de navegação.

O Código-fonte 8 mostra a criação de uma navegação do tipo *Drawer* por meio do referido módulo. *Drawer* é menu escondido na região esquerda da tela, que se expande quando o usuário arrasta o dedo na interface, da esquerda para a direita. A Figura 4a deste trabalho mostra um exemplo desse menu.

Código-fonte 8 – DrawerMenu.js: criação da navegação *Drawer*

```
1  const RootDrawer = createDrawerNavigator(
2    {
3      Home: {
4        screen: RootTabs,
5        navigationOptions: {
6          title: 'Início',
7          drawerIcon: ({ tintColor }) => <NavegIcon name="naveg"
   height={23} width={15} fill={tintColor} />,
8        }
9      },
10     ...
11     Map: {
12       screen: Views.Map
13     },
14     ...
15   },
16   {
```

[17] React Navigation. Disponível em: https://reactnavigation.org/. Acessado em: 15 jun. 2019.

```
17      initialRouteName: 'Home',
18      ...
19    }
20  );
```

Fonte: elaborado pelo autor.

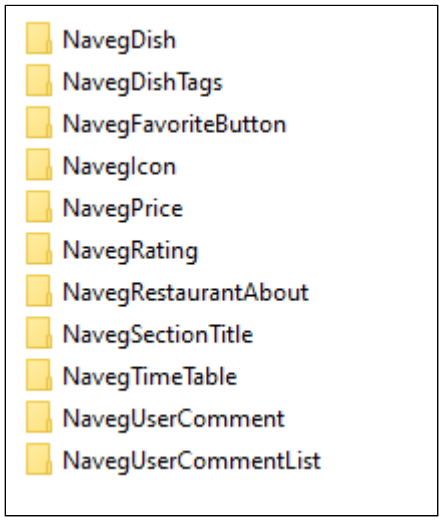
O método *createDrawerNavigator* retorna um componente com todas as particularidades de um *Drawer*. A função aceita parâmetros que definem quais são as rotas (*views*) acessíveis por esse menu e permite personalizar características dessas rotas, destacando-se, no código, o título da *view* e o ícone (gerado pelo componente *NavegIcon*, próprio deste projeto) que a representa.

Observa-se que a rota inicial, “Home”, é definida pelo valor de *RootTabs*. Apesar de não estar visível no trecho de código anterior, *RootTabs* é uma constante que gera uma navegação do tipo *Bottom Tab Navigation*, também do React Navigation. Esta é composta por abas localizadas na região inferior da interface, como exemplificam as Figuras 4b e 4c, presentes na seção 4.6 deste trabalho. Conclui-se, portanto, que é possível criar estruturas de navegação mais complexas, sempre aninhando componentes de navegação dentro de outros.

5.4 Componentes

O NaVeg possui alguns componentes que se diferenciam das *views* por terem responsabilidades pequenas e específicas. Como visto na Figura 12, eles são onze, e estão localizados no diretório *components*. Os componentes têm o prefixo “Naveg” para evitar conflito com outros módulos.

Figura 12 – Componentes do NaVeg



Fonte: elaborado pelo autor.

A distribuição em subdiretórios facilita a organização e possibilita que o componente seja referenciado por sua pasta, em vez de um arquivo específico, visto que cada um tem seu próprio *index.js*. Além do arquivo citado, cada componente possui um *styles.js*, contendo um objeto *StyleSheet* que centraliza os estilos do componente. É importante entender que esse objeto de estilo precisa ser exportado para que outros arquivos possam referenciá-lo, como mostrado na linha 9 do Código-fonte 9.

Código-fonte 9 – NavegFavoriteButton/styles.js: exportação dos estilos do componente

```
1 const styles = StyleSheet.create({
2   buttonContainer: {
3     alignItems: 'center',
4     justifyContent: 'center',
5     padding: 10
6   }
7 });
8
9 export default styles;
```

Fonte: elaborado pelo autor.

5.5 Firebase

Algumas funcionalidades do NaVeg, como cadastro de restaurantes e suas informações institucionais, presumem integração com algum tipo de Banco de Dados. A solução encontrada que melhor se encaixa com a lógica do React Native foi o Firebase, da Google. Em realidade, o Firebase não é simplesmente um Banco de Dados, mas um contêiner para aplicativos que oferece diversos serviços, incluindo configuração remota, integração com o Google Analytics[18], monitoramento de desempenho e outras soluções que facilitam a gerência da operação de aplicativos.

Os dados do Firebase são armazenados como JSON e são reativos, de forma que, quando há mudança em algum dos dados, o servidor gera um sinal imediato para todas as aplicações conectadas àquele banco. Isso significa que, ao se trabalhar com os *states* do React Native, é possível criar aplicações que se atualizam imediatamente, sem a necessidade de reiniciar o aplicativo.

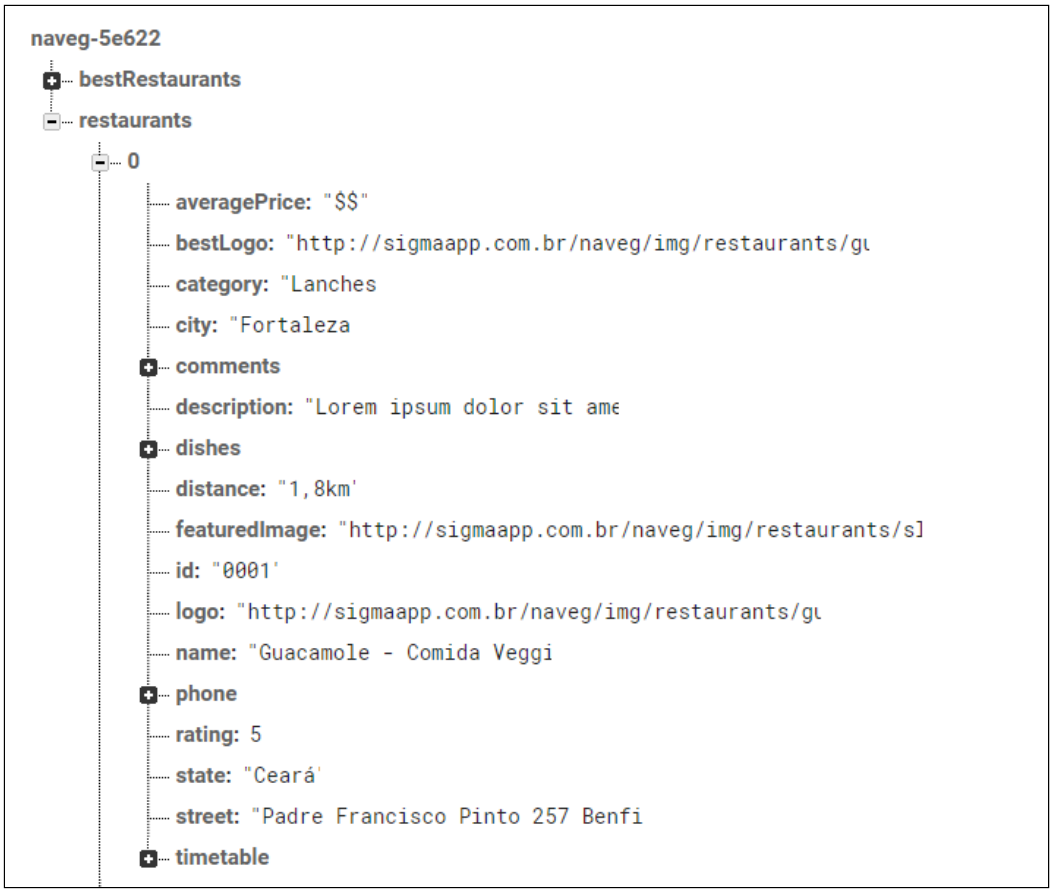
5.5.1 Criação do Firebase Database

Para utilizar os recursos do Firebase só é necessário possuir uma conta da Google. Acessando-se o Firebase Console[19], é possível criar um novo projeto do Firebase, ação que também cria um banco de dados vazio. A partir de então, pode-se preencher esse banco com uma estrutura arbitrária definida pelo desenvolvedor.

Apenas para fins de ilustração, a Figura 13 mostra uma parte dessa estrutura (todos os dados são fictícios, servindo somente como exemplo).

[18] Google Analytics. Disponível em: https://analytics.google.com/analytics/web/. Acessado em: 14 jun. 2019.
[19] Firebase Console - https://console.firebase.google.com/

Figura 13 – Recorte da estrutura do banco de dados



Fonte: elaborado pelo autor.

5.5.2 Conexão com o banco

A configuração da conexão do aplicativo com o Firebase é simples e ocorre no diretório *config/Firebase*, de modo que pode ser acessada por qualquer *view* necessária. O Código-fonte 10 apresenta essa configuração (dados reais do banco omitidos).

Código-fonte 10 – config/Firebase: conexão com Firebase

```
1  const firebaseConfig = {
2    apiKey: "chave-de-api",
3    authDomain: "url-de-autenticacao",
4    databaseURL: "url-do-banco",
5    storageBucket: "url-de-armazenamento"
6  }
7
```

```
const Firebase = firebase.initializeApp(firebaseConfig);
export default Firebase;
```

Fonte: elaborado pelo autor.

5.5.3 Uso do banco

No NaVeg, o banco é utilizado para compor a *view* que lista os restaurantes. Para que isso aconteça, é necessário que uma referência ao banco seja instanciada na *view*. O Código-fonte 11 mostra como essa referência é obtida e como ela pode ser usada em conjunto com um *state* do componente *RestaurantListView* para gerar tornar a interface reativa.

Código-fonte 11 – RestaurantListView.js: uso dos dados do banco

```
...
let ref = Firebase.database().ref(); // Captura referência do banco

export class RestaurantListView extends React.Component {
  ...
  constructor(props) {
    super(props);
    this.state = {loaded: false, restaurantes: null};
  }

  databaseLoaded = (snapshot) => {
    this.setState({
      loaded: true,
      restaurantes: snapshot.child('restaurants').val()
    });
  }

  componentWillMount() {
    ref.on('value', this.databaseLoaded);
  }
  ...
}
```

Fonte: elaborado pelo autor.

Se a obtenção dos dados for bem sucedida, o *state restaurantes* é preenchido com uma estrutura JSON que pode ser facilmente acessada durante o ciclo de vida do componente.

5.6 Deploy

Na seção 4.6 deste relatório, foi afirmado que o escopo do projeto entregue na disciplina de Projeto Integrado II se limitou ao Android. Neste trabalho, no entanto, entende-se que é necessário estender esse escopo, a fim de comparar os aspectos *cross-platform* do desenvolvimento com React Native. Dessa forma, faz-se necessário realizar o *deploy* do aplicativo nas plataformas Android e iOS.

Considera-se, ainda, que não foi desenvolvida nenhuma funcionalidade além das que já estavam contempladas ao fim da disciplina de 2018.

5.6.1 Android

Pelo fato de o NaVeg ser um projeto Expo, a geração do aplicativo para Android (*.apk*) é facilitada por meio dos seus serviços em nuvem. É exigido somente que se possua uma conta ativa no site do Expo (https://expo.io/) e que o arquivo *app.json* informe um *package* válido, como exposto no Código-fonte 12. *Package* é uma *string* que identifica, de forma exclusiva, um aplicativo. Ele é geralmente definido como uma URL invertida, como é o caso do exemplo mostrado.

Código-fonte 12 – app.json: *package* para Android

```
1 ...
2 "android": {
3   "package": "br.com.sigmaapp.naveg"
4 },
5 ...
```

Fonte: elaborado pelo autor.

Para gerar uma *build* do aplicativo, deve-se acessar seu diretório raiz por meio da linha de comando ou terminal e executar o comando a seguir:

```
expo build:android
```

É solicitado um *login* no serviço do Expo. Após autenticação, a compilação do aplicativo será iniciada. O processo, que dura alguns minutos, pode ser acompanhado no navegador, por meio de um *link* informado no próprio terminal. Por fim, o arquivo *.apk* poderá ser baixado, distribuído e instalado em *smartphones*.

5.6.2 iOS

O processo para gerar o aplicativo para iOS é, virtualmente, o mesmo descrito anteriormente. Faz-se necessário adicionar no arquivo *app.json* um *bundleIdentifier*, que desempenha exatamente a mesma função do *package* descrito anteriormente.

Código-fonte 13 – app.json: *bundleIdentifier* para iOS

```
...
"ios": {
  "bundleIdentifier": "br.com.sigmaapp.naveg"
},
...
```

Fonte: elaborado pelo autor.

De modo semelhante à versão para Android, executa-se o comando a seguir para iniciar a compilação:

```
expo build:ios
```

Isso acarreta uma questão que pode representar dificuldade para desenvolvedores principiantes: esse comando, no Expo, necessita de autenticação com uma Conta de Desenvolvedor da Apple, e esta só pode ser adquirida mediante pagamento de US$ 99,00[20] anuais. Esse tipo de conta é exigido porque o Expo, no momento de escrita deste trabalho, não

[20] US$ 99,00, valor aplicado no momento de escrita deste trabalho. Disponível em: https://developer.apple.com/support/purchase-activation/. Acessado em: 17 jun. 2019.

gera *builds* para iOS que possam ser distribuídos individualmente (EXPO, 2019), apenas publicados na App Store, ação que, de fato, necessita de uma Conta de Desenvolvedor.

Uma maneira de contornar esse obstáculo é realizar a ejeção do projeto do Expo, para que ele se torne um projeto do XCode[21], ambiente de desenvolvimento de *softwares* da Apple. Isso permitirá que o aplicativo seja transformado em um *.ipa*. No entanto, esse método não soluciona totalmente o problema, pois, por restrições de segurança da Apple, não é possível instalar esse *.ipa* em dispositivos físicos, dado que ele não estará assinalado com uma chave de desenvolvedor, como constatado pelo autor deste trabalho após diversas tentativas.

Assim, torna-se dispensável a descrição das etapas de geração de um arquivo *.ipa*, pelo XCode, neste trabalho. No entanto, é útil descrever o processo de ejeção de projeto Expo, pois, após isso, o aplicativo pode ser usado pelos simuladores do XCode, para fins de teste.

5.6.2.1 Ejeção do projeto Expo

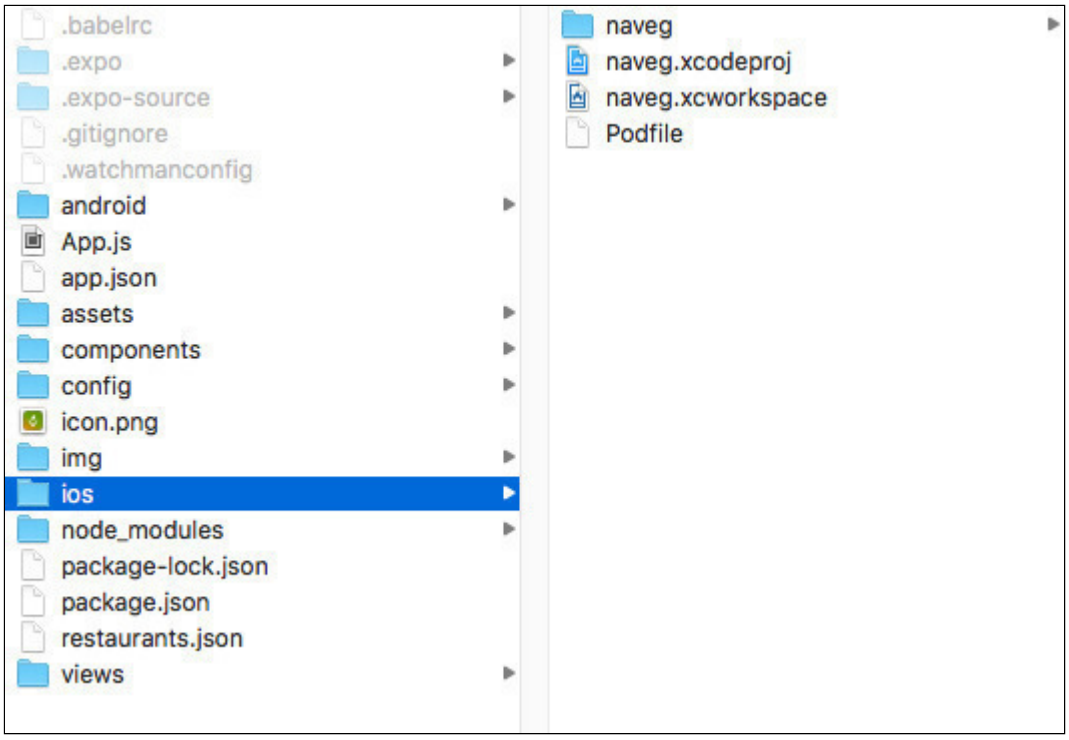
Um projeto Expo é ejetado por meio do comando a seguir, que cria uma nova organização de pastas e arquivos do projeto, própria do XCode:

```
exp detach
```

Observa-se que a sintaxe do comando é *exp*, nome utilizado anteriormente pela interface de comandos. Se o comando for bem sucedido, a pasta *ios* será criada no diretório raiz do projeto, contendo um arquivo de extensão *.xcodeproj*, que deve ser aberto no XCode, e um Podfile para instalar dependências.

[21] Xcode - Apple Developer. https://developer.apple.com/xcode/. Acessado em 19 jun. 2019.

Figura 14 – Diretório *ios* após ejeção do projeto



Fonte: elaborado pelo autor.

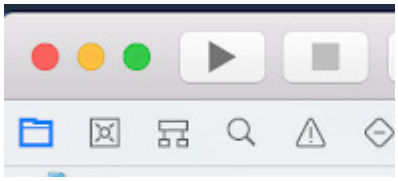
5.7.2.2 Uso do simulador

Com o projeto devidamente ejetado, deve-se abrir o arquivo *naveg.xcodeproj* no XCode. Ele levará um tempo para indexar todos os arquivos. Quando isso acontecer, pelo terminal, deve-se executar o comando a seguir na pasta *ios* a fim de ativar o servidor do Expo.

```
npm run start
```

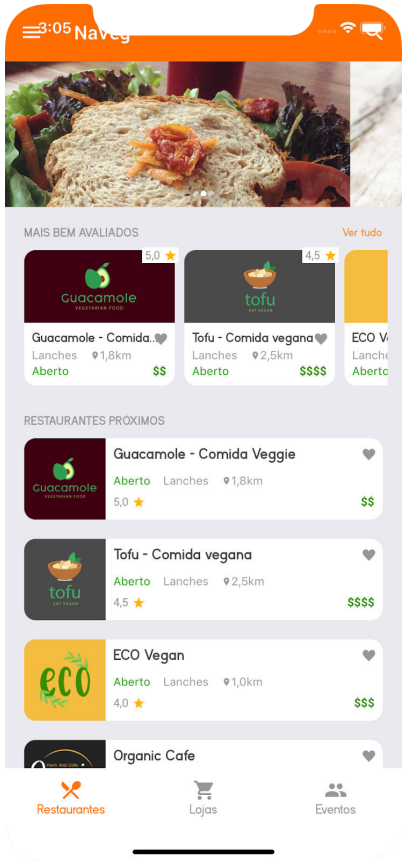
Com isso feito, é possível gerar o build do aplicativo no Simulador (iPhone X) clicando no botão *play* do XCode, localizado na parte superior do programa, como exibido na Figura 15.

Figura 15 – Botão *play* do XCode



Fonte: elaborado pelo autor.

Figura 16 – NaVeg sendo executado no simulador do XCode



Fonte: elaborado pelo autor.

Apesar de a aplicação ser executada com sucesso, nota-se uma falha: a interface do aplicativo é escondida pelos elementos físicos do aparelho (iPhone X). Para solucionar isso, utiliza-se o <SafeAreaView>, componente do React Native específico para dispositivos da Apple. Ele faz com que os elementos da interface da aplicação se acomodem dentro de uma área útil, que varia entre os dispositivos.

O componente é adicionado ao retorno do método *render* do *DrawerMenu*, como visto no Código-fonte 14.

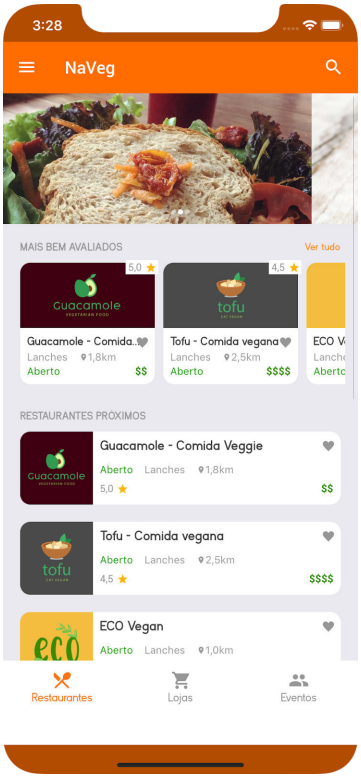
Código-fonte 14 – DrawerMenu.js: adição de <SafeAreaView>

```
...
render() {
  return (
  <SafeAreaView style={{ flex: 1, backgroundColor: "#B24C00" }}>
    <ThemeContext.Provider value={getTheme(uiTheme)}>
      <RootDrawer />
    </ThemeContext.Provider>
  </SafeAreaView>
  );
}
...
```

Fonte: elaborado pelo autor.

Após a mudança, repete-se o processo de ejeção do projeto. Realizado novo teste no simulador, observa-se que a interface foi consertada.

Figura 17 – Interface ajustada no iOS



Fonte: elaborado pelo autor.

É preciso entender o que essa mudança representa para o projeto, especificamente para sua versão Android. Como demonstrado, <SafeAreaView> foi criado para resolver um problema originado pelo iPhoneX, cujos componentes de *hardware* "invadem" a tela do aparelho. De acordo com instruções do React Native, "isso [o componente] detecta se o aplicativo está sendo executado em um iPhoneX e, se estiver, garante que o conteúdo não está oculto por elementos de *hardware*" (REACT NATIVE, 2019).

Entende-se, então, que quando esse componente é compilado para Android, o React Native verifica a informação da plataforma e renderiza um simples <View> no lugar de <SafeAreaView>. Esse comportamento é padrão no *framework*: componentes particulares de uma plataforma não têm efeito na outra, e isso é construído de forma a não ocasionar ajustes por parte do desenvolvedor.

6. ANÁLISE ENTRE PLATAFORMAS

Um dos objetivos deste trabalho é a realização de uma análise comparativa entre os aplicativos gerados para Android e iOS. Esse é um aspecto importante, pois permite que o *framework* utilizado possa ter sua eficiência avaliada.

No entanto, como visto na seção 5.6 deste trabalho, os resultados de *deploy* para as duas plataformas foram divergentes; com o Android, pode-se gerar um arquivo *.apk* e instalá-lo em *smartphones*; no entanto, com o iOS, apesar de o arquivo *.ipa* ter sido gerado, sua instalação não foi possível, devido a restrições da Apple, podendo-se apenas executar o aplicativo em um simulador do XCode. Dessa forma, a análise proposta neste trabalho não deve ser considerada como completa, e necessita de estudos posteriores.

6.1 Programação

No momento de escrita deste trabalho, o NaVeg é composto por: 13 componentes autorais simples, dos quais se destaca *NavegIcon*, responsável por imprimir ícones na tela; 16 *views*, com a ressalva de que algumas ainda não foram devidamente implementadas e outras necessitam de otimização, seguindo as boas práticas descritas neste trabalho; e recursos que constroem a identidade visual do aplicativo, como 2 famílias de fontes e uma série de ícones no formato SVG. Considerando somente os arquivos do próprio NaVeg, descartando, por exemplo, arquivos do *Git* e do Node.js, o projeto possui 84 arquivos que consomem um espaço de armazenamento de 784 KB (*kilobytes*), como mostrado na Figura 20.

Figura 18 – Espaço em disco dos arquivos do projeto

Fonte: elaborado pelo autor.

Com relação à infraestrutura, o aplicativo é composto por: um banco de dados Firebase, em nuvem, com dados de teste relativos a oito restaurantes fictícios, e um servidor de arquivos, provisório, localizado na mesma hospedagem do site da equipe e com o intuito apenas de testes. O referido servidor contém as imagens de exemplo dos restaurantes, como galeria de fotos, miniatura e cardápio.

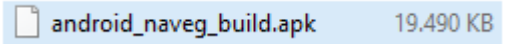
Em relação às versões Android e iOS, identifica-se que o aplicativo requer mudanças mínimas para funcionar em ambas as plataformas. No caso do NaVeg, como seu desenvolvimento inicial foi pautado no Android, posteriormente foi preciso adicionar novas configurações aos arquivos *app.json* e *package.json*, a fim de que o *deploy* para iOS fosse bem-sucedido.

Quanto ao funcionamento da aplicação, a única disparidade encontrada entre as plataformas foi o problema de sobreposição do *hardware* do iPhoneX com elementos da interface. A solução foi a adição de um componente próprio do React Native, o <SafeAreaView>, que resolveu o problema de forma simples, sem afetar o funcionamento do aplicativo no Android.

6.2 Builds

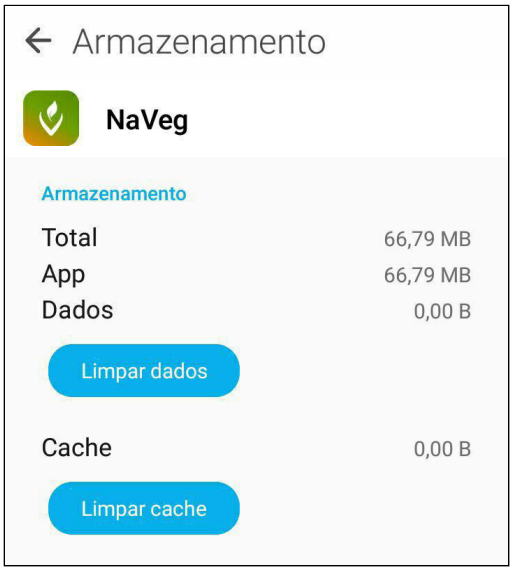
No âmbito dos arquivos gerados pelo *deploy* da aplicação, revela-se que o *.apk* ocupa somente 19 MB (*Megabytes*) de armazenamento em disco. Porém, quando instalado no dispositivo, esse valor cresce para quase 67 MB, um aumento de quase três vezes o tamanho original, sem contar com os dados de *cache* que poderão ser baixados durante o uso do aplicativo.

Figura 19 – Espaço em disco ocupado pelo *.apk*



Fonte: elaborado pelo autor.

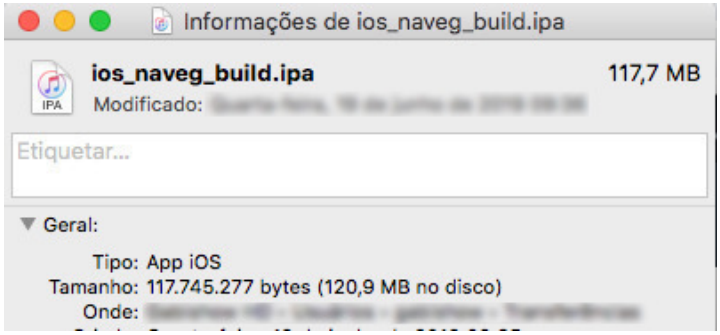
Figura 20 – Espaço em disco ocupado pelo *.apk* instalado



Fonte: elaborado pelo autor.

O *.ipa* gerado para iOS se mostrou mais pesado, ocupando cerca de 117 MB em disco. O espaço ocupado após instalado não pode ser avaliado, pois o simulador não fornece tal informação.

Figura 21 – Espaço em disco ocupado pelo *.ipa*



Fonte: elaborado pelo autor.

6.3 Interface

Foram realizados testes com três usuários com o intuito de avaliar a interface do aplicativo. Por terem sido testes informais, sem o aparato necessário para coletar dados precisos, e tendo o autor deste trabalho apenas escrito as observações feitas pelos usuários durante os testes, os resultados não devem ser considerados completos ou abrangentes. É

essencial, portanto, que novos testes sejam realizados no futuro, seguindo formatos já consolidados no mercado e abrangendo maior amostra de usuários.

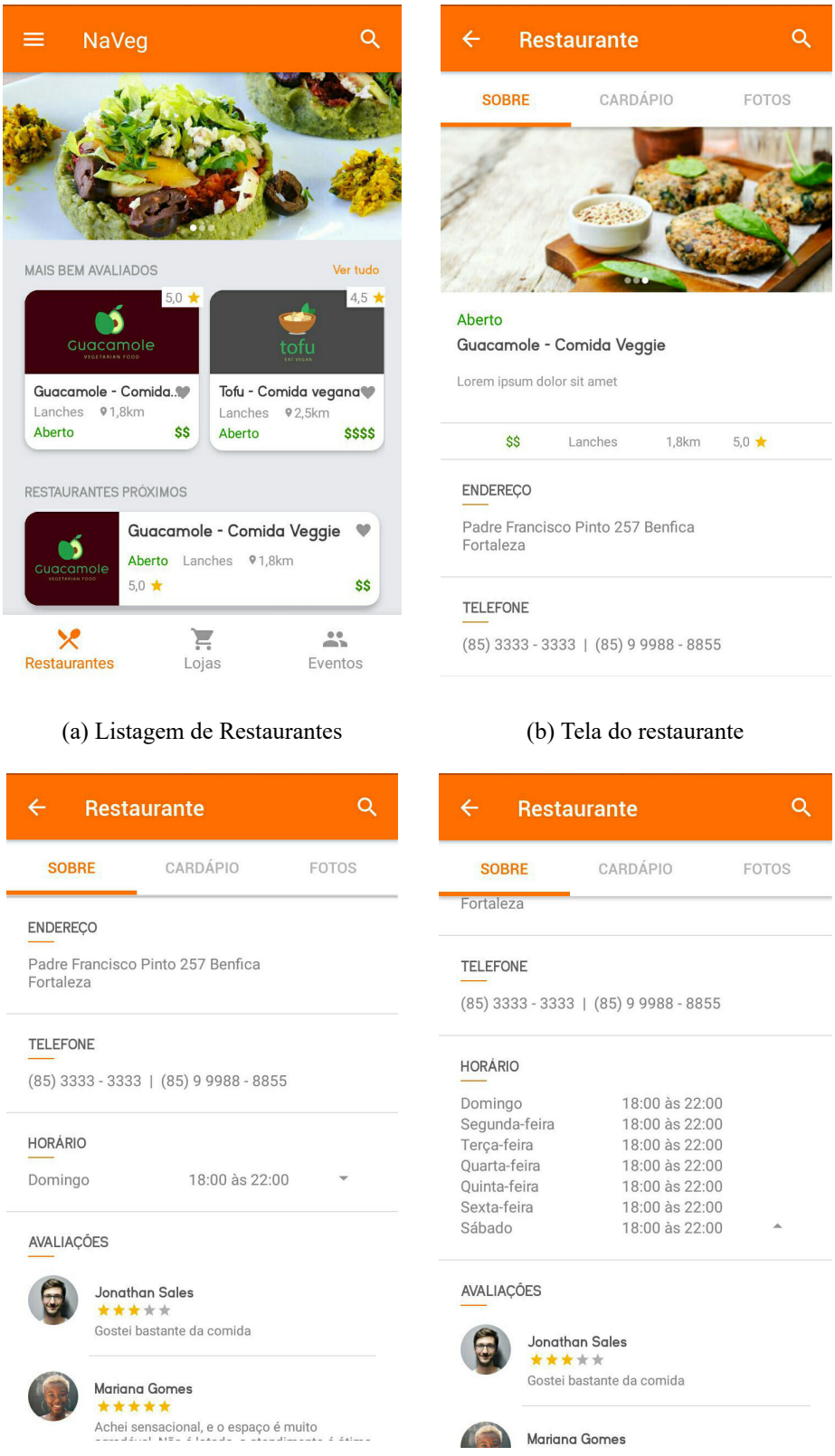
Dos usuários que utilizaram o NaVeg, dois deles são alunos do SMD, e o outro é colega de trabalho do autor deste relatório. A cada um deles, foram apresentadas as tarefas de (1) descobrir, no aplicativo, qual o horário de funcionamento, no dia de sexta-feira, do restaurante melhor avaliado e (2) enviar crítica ou sugestão para os criadores do aplicativo.

Novamente, é válido ressaltar que esses testes foram realizados exclusivamente no Android, por dificuldades já comentadas anteriormente.

6.3.1 Primeira tarefa

O fluxo ideal para que o usuário tenha êxito nesta tarefa é ele partir da tela inicial, de listagem de restaurantes, e selecionar o primeiro que é mostrado em destaque, na lista "Mais bem avaliados" (Figura 22a). Ao executar essa ação, a tela referente a este restaurante é exibida (Figura 22b). Nesta tela, pode-se encontrar a informação de horários apenas aplicando uma rolagem para baixo (Figura 22c); é mostrado o horário referente a domingo, com um ícone ao lado indicando a expansão de uma lista. O toque nesse ícone revela a lista de horários em sua totalidade, incluindo a informação requerida na tarefa.

Figura 22 – Fluxo da primeira tarefa



(a) Listagem de Restaurantes

(b) Tela do restaurante

(c) Tela do restaurante, com horários

(d) Horários expandidos

Fonte: elaborado pelo autor.

Essa sequência de passos foi facilmente entendida pelos três usuários, embora tenha havido confusão em relação ao *slideshow* de imagens na tela inicial (Figura 22a), que foi percebido, inicialmente, como uma lista dos melhores restaurantes. Nenhum dos usuários chegou a tocar nesse *slideshow*, pois antes de tomarem essa ação, o rótulo "Mais bem avaliados" foi identificado; no entanto, essa incerteza momentânea foi relatada e vista como um possível problema para pessoas menos experientes com tecnologia.

Um problema identificado pelos três usuários foi a dificuldade de tocar no ícone que expande a lista de horários. A área clicável desse botão, de fato, é pequena, o que atrapalha o *touch* na tela. Levantou-se, como possível solução, um aumento da área clicável, sem que o usuário perceba que o botão possui área maior do que a visível.

Por fim, os usuários do SMD sugeriram, como melhoria para a interface, que a lista de horários fosse disposta próxima ao nome do restaurante, pois é uma informação amplamente procurada, segundo eles.

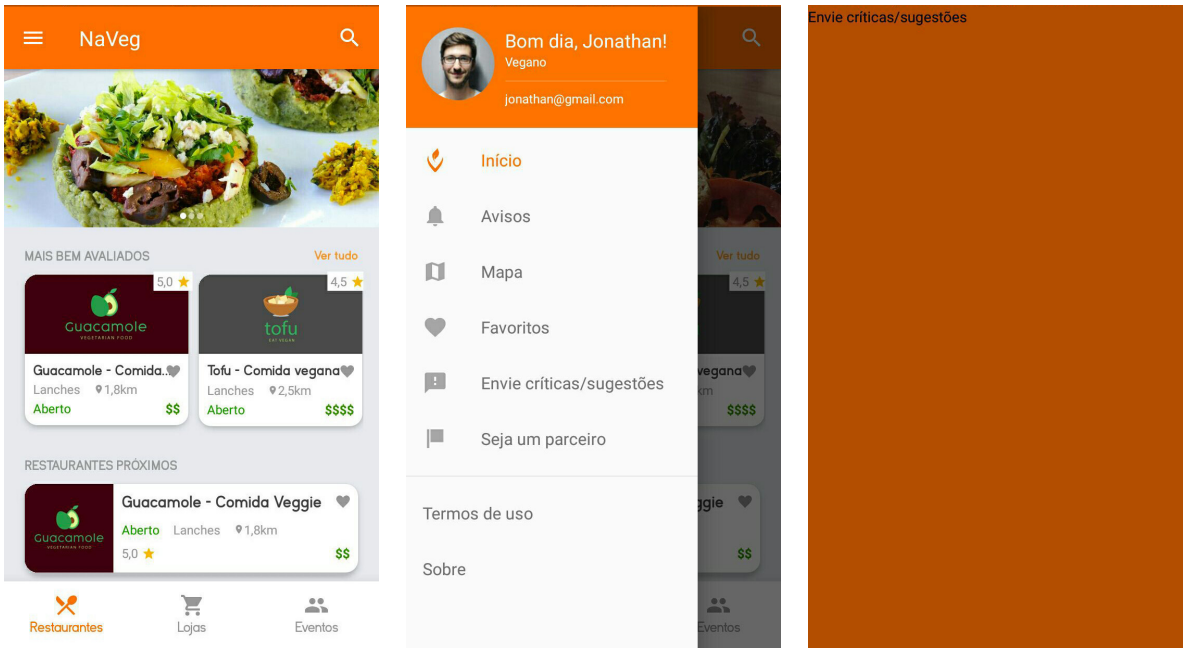
6.3.2 Segunda tarefa

A segunda tarefa exige mais raciocínio. Seu intuito é validar se o menu *Drawer* é entendido facilmente e seus rótulos estão intuitivos. A sequência esperada do usuário é tocar no ícone de sanduíche (três traços horizontais empilhados), abrindo o *Drawer* (Figura 23b), e, em seguida, selecionar o item "Envie críticas/sugestões" (Figura 23c). É válido observar que essa página não havia sido concluída até a escrita deste trabalho.

No caso dos três usuários que realizaram a tarefa, o primeiro movimento realizado foi tentar acessar essa opção por meio das abas localizadas na parte inferior do aplicativo, apesar de não haver nenhuma opção relacionada à tarefa. Isso pode indicar que o senso comum atual é de que as ações de aplicativos estejam posicionadas nessa região. Faz-se necessário realizar testes mais objetivos para averiguar essa hipótese.

Apesar desse problema, quando os usuários alcançaram, enfim, o *Drawer*, o reconhecimento da opção do envio de críticas foi rápido, e a página de envio foi alcançada sem problemas.

Figura 23 – Fluxo da segunda tarefa



(a) Tela inicial (b) Menu Drawer (c) Envio de críticas/sugestões

Fonte: elaborado pelo autor.

6.3.3 Comentários gerais

De modo geral, a interface se mostrou intuitiva e fluida. A resposta das ações dos usuários, como abertura do *DrawerMenu* e clique nos botões de favoritar restaurantes, tiveram respostas rápidas do sistema. Em um dos testes, porém, o aplicativo teve um atraso no carregamento e acabou exibindo a lista de restaurantes, sem antes passar pela tela *Splash*.

7. CONSIDERAÇÕES FINAIS

O crescimento atual do mercado vegano e vegetariano no Brasil produz oportunidades de empreendedorismo nesse ramo. Com base nessa premissa, a equipe Sigma iniciou o desenvolvimento do aplicativo NaVeg, utilizando o *framework* React Native. Este trabalho mostrou todo o processo de conceituação, pesquisa e desenvolvimento do aplicativo, buscando trazer enfoque para os aspectos *cross-platform* do *framework*. Foi mostrado como iniciar a programação de um aplicativo em React Native por meio do Expo, reunindo as principais características e boas práticas desse desenvolvimento.

De modo geral, observou-se que o uso do *framework* facilitou a criação do aplicativo e aumentou a produtividade. Para principiantes no React Native, sua documentação oficial pode se mostrar complexa, mas existem diversos outros canais em que se pode obter informações sobre a prática da ferramenta. O Expo foi essencial para o andamento do projeto, visto que possibilitou ótimos recursos de testes e depuração do código.

Por fim, tentou-se gerar arquivos executáveis para as plataformas Android e iOS. Nesta última, o processo não teve êxito, por causa de restrições quanto ao tipo de conta da Apple necessária para gerar uma *build* que possa ser instalada em dispositivos físicos. Devido a essa situação, a comparação entre os resultados das duas plataformas foi dificultada, e os dados finais não podem ser considerados como precisos.

O estado atual de desenvolvimento do NaVeg encontra-se da mesma forma como foi entregue na disciplina de Projeto Integrado II. Até o momento de escrita deste trabalho, a equipe não pode evoluir o projeto, pois os membros decidiram focar na conclusão da faculdade. Dessa forma, foram implementadas somente as telas *Splash*, listagem de restaurantes, conteúdo dos restaurantes e o *Drawer Menu*.

Como trabalhos futuros, pretende-se (1) implementar o restante das funcionalidades elencadas, como o mapa e as telas de eventos e lojas; (2) realizar testes do aplicativo com o público-alvo, de maneira a colher dados sobre a interface e validar a corretude e completude das funcionalidades do sistema e (3) buscar uma maneira de exportar efetivamente o aplicativo para iOS.

REFERÊNCIAS

AMBROSE, G.; HARRIS, P. **Design Th!nking**. AVA Publishing SA. Lausanne, Suíça. 2011.

AXELSSON, O.; CARLSTRÖM, F. **Evaluation Targeting React Native in Comparison to Native Mobile Development - An assessment of Developer Impression, User Experience and System Performance**. Department of Design Sciences, Faculty of Engineering LTH, Lund University, Lund, Suécia, 2016.

COUTINHO, E. F. et al. **Applying Design Thinking in Disciplines of Systems Development**. Instituto UFC Virtual, Universidade Federal do Ceará, Fortaleza, Ceará, Brasil, 2016.

DANIELSSON, W. **React Native application development: A comparison between native Android and React Native**. Institutionen för datavetenskap. Linköpings universitet, Linköpings, Suécia. 2016.

DAWSON, C. **JavaScript's History and How it Led To ReactJS**, 2014. Disponível em: <https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/>. Acesso em: 01 jun. 2019.

DESENVOLVIMENTOÁGIL. **SCRUM**, 2014. Disponível em: <https://www.desenvolvimentoagil.com.br/scrum/>. Acesso em: 10 set. 2018.

ENCONTRO DIGITAL. **Aplicativos ajudam quem é vegetariano ou vegano**, 2017. Disponível em: <https://www.revistaencontro.com.br/canal/atualidades/2017/11/aplicativos-ajudam-quem-e-vegetariano-ou-vegano.html>. Acesso em 14 jun. 2019.

EISENMAN, B. **Learning react native: building native mobile apps with javascript**. O'Reilly Media, Inc. 2015.

ESCOLA DESIGN THINKING. Mini Toolkit Design Thinking, 2016. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/4160905/mod_resource/content/1/Mini%20Toolkit%20Design%20Thinking.pdf>. Acesso em: 02 jun. 2019.

EXPO. **Building Standalone Apps**, 2019. Disponível em: <https://docs.expo.io/versions/v33.0.0/distribution/building-standalone-apps/>. Acesso em: 19 jun. 2019.

FREITAS, T. **Novo unicórnio brasileiro: iFood recebe investimento de US$ 500 milhões**, 2018. Disponível em: <https://www.startse.com/noticia/startups/57508/novo-unicornio-brasileiro-ifood-recebe-investimento-de-us-500-milhoes>. Acesso em: 26 mai. 2019.

HANSSON, N; VIDHALL, T. **Effects on performance and usability for cross-platform application development using React Native**. Institutionen för datavetenskap. Linköpings universitet, Linköpings, Suécia. 2016.

HEITKÖTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Evaluating cross-platform development approaches for mobile applications. In: **International Conference on Web Information Systems and Technologies**. Springer, Berlin, Heidelberg, 2012. p. 120-138.

IBGE. **Smartphones**. 2018. Disponível em: <https://agenciadenoticias.ibge.gov.br/agencia-sala-de-imprensa/2013-agencia-de-noticias/releases/20073-pnad-continua-tic-2016-94-2-das-pessoas-que-utilizaram-a-internet-o-fizeram-para-trocar-mensagens>. Acesso em: 05 mai. 2019.

MOZILLA. **Document Object Model (DOM)**. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>. Acesso em: 01 jun. 2019.

NET MARKETSHARE. **Operating System Market Share**. Disponível em: <https://bit.ly/2M8ZC3y>. Acesso em: 26 mai. 2019.

OCCHINO, T. **React Native: Bringing modern web techniques to mobile**, 2015. Disponível em: <https://code.fb.com/android/react-native-bringing-modern-web-techniques-to-mobile/>. Acesso em: 02 jun. 2019.

REACT NATIVE. **Supporting safe areas**. Disponível em: <https://reactnavigation.org/docs/en/handling-iphonex.html>. Acesso em: 02 jun. 2019.

REACTJS. **Getting Started**. Disponível em: <https://reactjs.org/docs/getting-started.html>. Acesso em: 02 jun. 2019.

SIGMAAPP. **Momento 1 – Projeto do Produto**, 2018. Disponível em: <http://sigmaapp.com.br/projeto/projeto-do-produto/>. Acesso em: 14 jun. 2019.

STATCOUNTER. **Mobile Operating System Market Share Brazil**, 2019. Disponível em: <http://gs.statcounter.com/os-market-share/mobile/brazil>. Acesso em: 05 mai. 2019.

SVB. **Mercado Vegetariano**. Disponível em: <www.svb.org.br/vegetarianismo1/mercado-vegetariano>. Acesso em: 12 set. 2018.

APÊNDICE A – DESIGN THINKING

Design Thinking é um modelo mental iterativo que estabelece alguns processos de *design* para se projetar a solução de um problema (ESCOLA DESIGN THINKING, 2016, p. 15). Esse ato pode ser desafiador se os processos não estiverem claros e esquematizados, mas ao mesmo tempo o *Design Thinking* não é rígido na ordem desses processos, permitindo que as pessoas voltem a etapas anteriores sempre que necessário, o que confere a iteratividade anteriormente citada.

Esse método é pautado na experiência entre humanos e se vale da empatia e da colaboração. Ele entende que erros são naturais e importantes, pois funcionam como testes e possibilitam chegar a soluções melhores. Além disso, o *Design Thinking* incentiva que cada membro da equipe seja atuante e forneça a sua perspectiva sobre o problema a ser enfrentado. Ele colabora para que aptidões de diferentes pessoas se tornem ferramentas valiosas à disposição da equipe.

Essa visão se encaixa bem com a do SMD, que forma profissionais multidisciplinares para atuarem em diversos setores do mercado. Sobre diferentes perspectivas e multidisciplinaridade, a Escola Design Thinking (2016) explica que:

> Quando uma pessoa pensa em um problema, ela certamente tem uma visão única sobre ele, mas se multiplicarmos os olhares, teremos diferentes perspectivas, chegando mais perto do que é esse problema na realidade. Essa diversidade de olhares somada a multidisciplinaridade nos permite enxergar oportunidades e soluções que jamais seriam possíveis se geradas por uma só cabeça. (ESCOLA DESIGN THINKING, 2016, p. 11)

Portanto, é evidente que essa é uma abordagem que agrega aos alunos do SMD e pode gerar bons resultados.

Coutinho et al. (2016) definem 7 etapas que devem ser contempladas até a conclusão do projeto da disciplina, sendo elas:

1. Definição: é o *briefing* inicial do projeto e suas informações precisam delimitar o problema de forma completa. Nessa etapa, deve-se buscar resposta para as perguntas: Qual o problema? Como será resolvido? Por quê? Onde? A quem se destina? Quem é o cliente? Perguntas além destas são bem-vindas;
2. Pesquisa: é o momento de coletar informações sobre o problema e suas adjacências. Pode ser uma pesquisa qualitativa ou quantitativa, contanto que

ela consiga identificar os possíveis usuários ou consumidores do produto, suas características, dados demográficos, hábitos de consumo, e quaisquer informações que forem importantes para o problema específico;

3. Ideação: realizada a pesquisa, os dados são analisados e a equipe começa a gerar ideias sobre como resolver o problema (com base nos dados levantados na pesquisa). É comum o uso de *brainstorming*[22] nessa etapa. Se a equipe julgar necessário, pode ser realizada uma nova pesquisa para colher mais dados;
4. Prototipação: criação de testes e protótipos de baixo custo e rápida execução, a fim de validar as ideias geradas na etapa anterior;
5. Seleção: escolha da melhor solução a ser desenvolvida, com base nos protótipos;
6. Implementação: a implementação é a parte mais longa do projeto, pois é quando se inicia o desenvolvimento real do produto. Essa etapa também envolve a entrega final do produto para o cliente;
7. Aprendizagem: com o projeto concluído, faz-se necessário obter *feedbacks* do cliente sobre o produto para determinar se a solução aplicada foi a mais correta e atendeu às necessidades iniciais. Além disso, a própria equipe deve se reunir e conversar sobre as dificuldades que ocorrem durante a execução do projeto. Assim, cada membro tem a oportunidade de refletir sobre seus erros e, portanto, melhorar em projetos futuros.

No decorrer da disciplina, cada uma dessas etapas foi contemplada em diversos momentos. Um dos meios para marcar o fim de cada etapa foi o uso de blog próprio da equipe, no qual foram feitas postagens ao longo do semestre para atualizar o *status* do projeto.

Como o escopo deste relatório enfatiza o aspecto *cross-platform* da implementação do aplicativo NaVeg, nem todas as etapas do *Design Thinking* serão relatadas. As etapas de Definição, Pesquisa e Seleção serão abordadas de forma sucinta, pois entende-se que é relevante apresentar o conceito do aplicativo, o público-alvo e o mercado; as etapas de Ideação, Prototipação e Aprendizagem não estarão presentes; e a etapa de Implementação será destacada.

[22] Tempestade de ideias (tradução livre), processo que consiste em gerar múltiplas ideias, sem que nenhuma seja julgada ou inviabilizada imediatamente.