

UNIVERSIDADE FEDERAL DE VIÇOSA  
CAMPUS DE RIO PARANAÍBA  
SISTEMAS DE INFORMAÇÃO

RÔMULO PAULO BERNARDINO SILVA

**DESENVOLVIMENTO DE UMA APLICAÇÃO  
MULTIPLATAFORMA COM A MEAN STACK E O IONIC**

RIO PARANAÍBA  
2018

RÔMULO PAULO BERNARDINO SILVA

DESENVOLVIMENTO DE UMA APLICAÇÃO MULTIPLATAFORMA  
COM A MEAN STACK E O IONIC

Monografia apresentada à Universidade Federal de Viçosa como parte das exigências para a aprovação na disciplina Trabalho de Conclusão de Curso II

Orientador: Prof<sup>a</sup>. Dr<sup>a</sup>. Adriana Zanella Martinhago

RIO PARANAÍBA

2018

*Aos meus pais.*

# Agradecimentos

Aos meus pais e à minha família por sempre terem me apoiado e estado comigo quando sempre precisei e por sempre demonstrarem seu amor e confiança. Obrigado aos meus queridos amigos e à minha querida, Laura Ananias Ferreira Navarro, por sempre estarem ao meu lado, tanto nos bons quanto nos maus momentos ao longo dessa jornada. Obrigado à Nataniele Barbosa Sousa (*in memoriam*), por ter sido uma pessoa tão especial pela qual serei eternamente grato. Obrigado à minha orientadora, Prof<sup>a</sup>. Adriana Zanella Martinhago, pela sua paciência e pelo seu apoio. Meus mais sinceros agradecimentos a todos que de alguma forma estiveram comigo durante todos esses anos nessa querida instituição. Obrigado!

*"Computer Science is no more about computers than Astronomy is about telescopes.  
- E. W. Dijkstra"*

# Resumo

Este trabalho propõe a criação de uma aplicação multiplataforma para auxiliar os usuários do Refeitório Universitário da Universidade Federal de Viçosa - Campus Rio Paranaíba no de sentido proporcionar uma alternativa quanto às formas de acesso aos cardápios disponibilizados pelo restaurante. No atual contexto tecnológico, os smartphones e as aplicações, tanto web quanto móveis, exercem um papel fundamental no que diz respeito aos meios de propagação de informação e de interação de usuários. Com base nisso, a aplicação é disponibilizada via web desktop, web móvel e aplicação instalável no sistema operacional móvel *Android*. Por meio da possibilidade de acesso via web, é permitido que os usuários de outros sistemas operacionais móveis, como *iOS* ou *Windows 10 Mobile*, por exemplo, possam ter acesso aos recursos da aplicação. O sistema tem como ênfase mostrar informações adicionais aos usuários, como fotografias dos alimentos e descrição sobre os mesmos, notificar os usuários previamente a cada horário de refeição informando sobre o cardápio atualizado e permitir os feedback dos mesmos por meio de comentários sobre os cardápios ofertados. A aplicação é dividida em servidor, aplicação administrativa e aplicação pública. Para a construção do servidor e da aplicação administrativa foi utilizado a *MEAN Stack*, com *Node.js*, *Express*, *MongoDB* e *Angular*. Já para a aplicação pública foi utilizado o *Ionic*. A aplicação administrativa permite que os administradores do sistema manipulem os registros presentes na base de dados, como gerenciar alimentos e programar cardápios. Estes são então disponibilizados por meio da aplicação pública, que notifica os visitantes sobre novos cardápios e informações sobre o Refeitório Universitário, via web e móvel.

**Palavras-chaves:** Aplicações Multiplataforma, Aplicações Móveis, *MEAN Stack*, *Ionic*, Aplicações para Restaurantes

# Abstract

This paper proposes the creation of a multiplatform application to assist the users of the Student Restaurant of the Universidade Federal de Viçosa - Rio Paranaíba Campus in order to provide an alternative to access the menus made available by the restaurant. In the current technological context, smartphones and applications, both web and mobile, play a key role in terms of the means of information dissemination and user interaction. Based on this, the application is made available via web desktop, mobile web and installable application on Android mobile operating system. Through the possibility of web access, users of other mobile operating systems such as iOS or Windows 10 Mobile, for example, can be granted access to the features of the application. The emphasis of the system is to show additional information to users, such as food photographs and descriptions, to notify users in advance of each meal time, informing them about the updated menu, and to allow their feedback through comments on the menus offered. The application is divided into server, administrative application and public application. For the construction of the server and the administrative application was used the MEAN Stack, with Node.js, Express, MongoDB and Angular. For the public application Ionic was used. The administrative application allows system administrators to manipulate the records in the database, such as managing food and scheduling menus. These are then made available through the public application, which notifies visitors about new menus and information about the Student Restaurant, via the web and mobile.

**Key-words:** Multiplatform Applications, Mobile Applications, MEAN Stack, Ionic, Applications for Restaurants

# Listas de ilustrações

Figura 1 – Estrutura de uma aplicação nativa. . . . .	19
Figura 2 – Sequência básica de requisição/resposta em um modelo cliente/servidor. . . . .	22
Figura 3 – Estrutura de uma aplicação web móvel. . . . .	22
Figura 4 – Estrutura de uma aplicação híbrida. . . . .	24
Figura 5 – Divergência entre desenvolvedores <i>front-end</i> e <i>back-end</i> ao longo do tempo. . . . .	26
Figura 6 – Impacto dos <i>frameworks</i> nos dois aspectos separados do desenvolvimento web. . . . .	27
Figura 7 – Exemplo de funcionamento de uma aplicação utilizando a <i>MEAN Stack</i> . . . . .	28
Figura 8 – Exemplos de endpoints de uma aplicação <i>REST</i> . . . . .	28
Figura 9 – Estrutura de uma aplicação <i>MEAN</i> . . . . .	29
Figura 10 – Formato de armazenamento de documentos no <i>MongoDB</i> . . . . .	30
Figura 11 – Definições de esquemas no <i>Mongoose</i> . . . . .	31
Figura 12 – Exemplo de criação de um servidor em <i>Node.js</i> . . . . .	32
Figura 13 – Instanciação do objeto do <i>Express</i> . . . . .	33
Figura 14 – Estrutura de um método típico no ambiente do <i>Express</i> . . . . .	34
Figura 15 – Exemplo de dois módulos independentes em uma aplicação do <i>Angular</i> . . . . .	35
Figura 16 – Representação da sintaxe de template do <i>Angular</i> . . . . .	36
Figura 17 – Definição das rotas em uma aplicação <i>Angular</i> . . . . .	37
Figura 18 – Arquitetura de uma aplicação <i>Apache Cordova</i> . . . . .	38
Figura 19 – Estrutura de uma aplicação <i>Ionic</i> . . . . .	39
Figura 20 – Tela inicial e menu lateral do aplicativo <i>BandejãoUFRJ</i> . . . . .	41
Figura 21 – Tela inicial do aplicativo RU-UFU. . . . .	43
Figura 22 – Telas de seleção de horário da refeição e listagem dos alimentos. . . . .	44
Figura 23 – Aplicação do <i>Consulta RU/SISRU UFV-CRP</i> . . . . .	45
Figura 24 – Aplicativo <i>UFV mobile</i> . . . . .	46
Figura 25 – Etapas de desenvolvimento do trabalho. . . . .	48
Figura 26 – Diagrama de caso de uso do sistema. . . . .	50
Figura 27 – Diagrama entidade relacionamento da aplicação. . . . .	51
Figura 28 – Arquitetura do sistema. . . . .	54
Figura 29 – Estrutura de diretórios. . . . .	56
Figura 30 – Definição da estrutura de diretórios do sistema. . . . .	57
Figura 31 – Definição de variáveis de ambientes. . . . .	58
Figura 32 – Definição das configurações de ambiente e <i>middlewares</i> do <i>Express</i> . . . . .	59
Figura 33 – Criação da conexão com o <i>MongoDB</i> por meio do <i>Mongoose</i> . . . . .	60
Figura 34 – Iniciando o servidor com o <i>Express</i> . . . . .	60

Figura 35 – Criação dos esquemas do <i>Mongoose</i> . . . . .	61
Figura 36 – Rotas principais da aplicação. . . . .	61
Figura 37 – Métodos <i>HTTP</i> das rotas. . . . .	62
Figura 38 – Exemplo de método que trás todos os registros de um documento. . . . .	62
Figura 39 – Demais roteamentos presentes na aplicação do servidor. . . . .	63
Figura 40 – Tela de <i>login</i> da aplicação administrativa. . . . .	64
Figura 41 – Exemplos de notificações de validações. . . . .	64
Figura 42 – Página <i>Dashboard</i> de acordo com a visão do administrador. . . . .	65
Figura 43 – Módulo de Gerenciamento de Alimentos. . . . .	66
Figura 44 – Módulo de Gerenciamento de Cardápios. . . . .	67
Figura 45 – Módulo de Gerenciamento de Cardápios Ativos. . . . .	68
Figura 46 – Módulo de Gerenciamento de Usuários. . . . .	69
Figura 47 – Tela de perfil do usuário. . . . .	69
Figura 48 – Estrutura visual da aplicação pública/móvel mostrada em diferentes versões. . . . .	71
Figura 49 – Tela de informações sobre o alimento, na versão do navegador <i>Google Chrome</i> para a plataforma <i>Android</i> . . . . .	72
Figura 50 – Tela de comentários sobre um cardápio ativo, na versão do <i>Apple Safari</i> para a plataforma <i>iOS</i> . . . . .	73
Figura 51 – Tela de busca de alimentos ativos para uma determinada data. . . . .	74
Figura 52 – Exemplo de tarefa do <i>Gulp</i> de <i>build</i> para o ambiente de Teste. . . . .	76

# **Lista de tabelas**

Tabela 1 – Comparação entre recursos acessíveis em cada abordagem. . . . .	18
Tabela 2 – Requisitos para desenvolvimento nos principais sistemas operacionais móveis . . . . .	20
Tabela 3 – Descrição das ações possíveis ao administrador. . . . .	53
Tabela 4 – Definição dos ambientes existentes no sistema. . . . .	58
Tabela 5 – Lista de comandos e ações presentes na aplicação. . . . .	75

# Sumário

<b>1</b>	<b>Introdução . . . . .</b>	<b>12</b>
1.1	Objetivo Geral . . . . .	13
1.2	Objetivos Específicos . . . . .	13
1.3	Estrutura do Trabalho . . . . .	14
<b>2</b>	<b>Referencial Teórico . . . . .</b>	<b>15</b>
2.1	Aplicações Móveis, Saúde e Alimentação . . . . .	15
2.2	Abordagens no Desenvolvimento Móvel . . . . .	17
2.2.1	Aplicações Nativas . . . . .	19
2.2.2	Aplicações Web . . . . .	20
2.2.3	Aplicações Híbridas . . . . .	23
2.3	Desenvolvimento <i>Full-stack</i> . . . . .	25
2.4	A <i>MEAN Stack</i> . . . . .	27
2.4.1	<i>MongoDB</i> . . . . .	29
2.4.2	<i>Node.js</i> . . . . .	31
2.4.3	<i>Express</i> . . . . .	33
2.4.4	<i>Angular</i> . . . . .	34
2.5	Desenvolvimento Híbrido . . . . .	37
2.5.1	<i>Apache Cordova</i> . . . . .	37
2.5.2	<i>Ionic Framework</i> . . . . .	39
<b>3</b>	<b>Trabalhos Relacionados . . . . .</b>	<b>41</b>
3.1	Aplicativo para o Restaurante Universitário da Universidade Federal do Rio de Janeiro ( <i>BandejãoRU</i> ) . . . . .	41
3.2	Aplicativo para o Restaurante Universitário da Universidade Federal de Uberlândia ( <i>RU-UFU</i> ) . . . . .	42
3.3	Aplicação web para o Refeitório Universitário da Universidade Federal de Viçosa - Campus Rio Paranaíba ( <i>Consulta RU/SISRU UFV-CRP</i> ) . . . . .	44
3.4	Aplicativo <i>Android</i> para a Universidade Federal de Viçosa ( <i>UFV mobile</i> ) .	46
<b>4</b>	<b>Métodos . . . . .</b>	<b>48</b>
<b>5</b>	<b>Resultados e Discussões . . . . .</b>	<b>50</b>
5.1	Modelagem . . . . .	50
5.2	Arquitetura . . . . .	54
5.3	<i>Scaffolding</i> . . . . .	55
5.4	Ambientes . . . . .	58
5.5	Servidor . . . . .	59
5.6	Base de Dados e <i>API</i> . . . . .	60
5.7	Aplicação Administrativa . . . . .	63

5.8	Aplicação Pública . . . . .	70
5.9	Automação . . . . .	73
5.10	<i>Deploy</i> . . . . .	76
<b>6</b>	<b>Conclusões</b> . . . . .	<b>77</b>
6.1	Trabalhos Futuros . . . . .	78
	<b>Referências</b> . . . . .	<b>79</b>
	<b>Apêndices</b>	<b>84</b>
	<b>APÊNDICE A Documentação de requisitos</b> . . . . .	<b>85</b>

# 1 Introdução

Os dispositivos móveis são itens no dia a dia das pessoas que se tornaram tão comuns quanto essenciais. Segundo o The Economist (2015), no mundo todo há 2 bilhões de pessoas que possuem um smartphone<sup>1</sup> com conexão à internet e interação por meio de comando por toque e é previsto que até o final da década este número alcançará a marca de 4 bilhões, onde só em 2015 foram vendidos cerca de 500 milhões somente na China.

Esse crescimento acelerado também foi impulsionado pelo investimento no setor móvel. De acordo com o Boston Consulting Group (2015), de 2009 a 2013 foram investidos US\$ 1,8 trilhão no setor a fim de melhorar a infraestrutura em todo o mundo. O aumento da velocidade de conexão juntamente com a redução do preço por megabyte, aliados à ampliação do uso de redes Wi-Fi em casas e escritórios, permitiu ampliar o poder computacional dos telefones a patamares antes vistos somente em grandes centros de processamento de dados.

Os smartphones atraem muito da atenção em questão de investimentos e pessoal de criação, essencialmente por terem se tornado uma peça chave dentro da tecnologia da informação (ESTADÃO, 2015). Isso pode ser observado em quantidade de aplicativos disponibilizados. Segundo dados de Statista (2016a), a *App Store*, loja de aplicativos da Apple possui cerca de 2 milhões de aplicativos disponíveis e a *Google Play*, maior loja em número de aplicativos, oferece por volta de 2,2 milhões, sendo essas as duas maiores com uma grande margem se comparado aos 669.000 da *Windows Store*, 600.000 da *Amazon Appstore* e 234.500 da *BlackBerry World*. Esses números refletem na arrecadação nessas lojas de aplicativos. Em 2015 a *Google Play* arrecadou US\$ 6 bilhões com compras em aplicativos, o que é relativamente bem menor se comparado com os US\$ 20 bilhões arrecadados pela *App Store* (STATISTA, 2016b)(STATISTA, 2016c).

O contexto supracitado também leva à explicação de uma estratégia de desenvolvimento chamada *mobile first*, um conceito em experiência do usuário (*User Experience* ou *UX*) que coloca os dispositivos móveis como alvo central no desenvolvimento de sítios web, o que significa desenvolver uma interface que atenda primeiro à experiência on-line em dispositivos móveis antes da versão desktop. Isto se baseia essencialmente no fato de que as pessoas estão realizando mais atividades com seus dispositivos, o que levou as empresas a mudarem sua estratégia para uma abordagem que atenda primeiro a esse novo contexto (GRAHAM, 2012).

---

<sup>1</sup> Conforme especificado no sítio do Senado Federal, palavras estrangeiras já incorporadas à Língua Portuguesa não são grafadas em itálico (SENADO, 2015)(SENADO, 2016).

De forma geral, em um contexto de franco crescimento do uso de dispositivos móveis onde os quais têm cada vez maior poder de processamento, a adaptação de serviços até pouco antes mais tradicionais tende a tomar um sentido igualmente proporcional. Esse fato se faz presente em vários setores da sociedade, seja na educação, atividades físicas ou alimentação. A necessidade de ter informações de imediato com alguns toques no celular, ou mesmo ser notificado quando algum fato novo acontecer, se tornou algo bastante comum.

Um restaurante universitário oferece vantagens para aqueles que fazem uso deste. A praticidade e a comodidade, a variedade de alimentos e o preço atrativo das refeições são pontos diferenciais para que haja um grande volume de pessoas que adotam esse tipo de restaurante como local para fazerem suas principais refeições diárias. Com isso, ter informações de imediato sobre o que será oferecido no restaurante se torna algo bastante importante.

No contexto em que se insere a temática desse trabalho, essa necessidade é observável no que se trata de haver uma solução que possibilite ter acesso a informações sobre um serviço que tem grande impacto na comunidade no qual o mesmo se faz presente. Ter de prontidão informações sobre o que será oferecido no Refeitório Universitário (RU) para alguma data em questão é algo relativamente básico quanto se trata de um restaurante desse tipo.

Baseado nisso, pode-se observar a necessidade de fornecer mais opções de acesso e ampliar a divulgação dos cardápios do restaurante, de forma a permitir aos usuários do mesmo saberem de antemão o que será ofertado.

## 1.1 Objetivo Geral

O objetivo do trabalho é desenvolver uma aplicação multiplataforma disponibilizada via web móvel, web desktop e *Android* para oferecer um método alternativo de acesso aos cardápios pelos usuários do Refeitório Universitário da Universidade Federal de Viçosa – Campus Rio Paranaíba.

## 1.2 Objetivos Específicos

- Modelar o sistema.
- Implementar a aplicação multiplataforma, de forma a ser acessível pela web móvel e desktop e pelo aplicativo do sistema operacional *Android*.
- Distribuir a aplicação on-line via sítio web e aplicação móvel.

### 1.3 Estrutura do Trabalho

Este trabalho é dividido em cinco capítulos. No Capítulo 2 é apresentado o referencial teórico para permitir que haja um entendimento acerca dos conceitos aqui abordados. O Capítulo 3 traz os trabalhos relacionados no qual é realizado um paralelo entre outros trabalhos similares desenvolvidos, apontando aspectos similares e diferentes entre eles. O Capítulo 4 traz os métodos utilizados no desenvolvimento do trabalho, direcionando ao Capítulo 5, que traz os resultados e discussões do trabalho. O Capítulo 6 apresenta um resumo geral do que foi feito e abre espaço para discutir possíveis trabalhos futuros relacionados a este. Ao final seguem as referências aqui utilizadas e o Apêndice A com a documentação de requisitos do sistema.

## 2 Referencial Teórico

Neste capítulo são abordados os conceitos teóricos que sustentam o desenvolvimento do trabalho e que permitem um maior entendimento em relação ao tema proposto. Inicia-se por uma abordagem sobre a relação do uso dos dispositivos móveis com a alimentação e saúde, mostrando como o uso de aplicativos influenciam no cotidiano das pessoas.

Em seguida são levantados os tipos de abordagens inerentes ao desenvolvimento móvel, sendo elucidadas as vantagens e desvantagens de cada um dos tipos de desenvolvimento, levando em consideração fatores atuais e históricos no cenário de desenvolvimento de aplicações móveis, recursos disponíveis em questão de hardware e software atuais, além de maiores detalhes sobre ambos os tipos.

Na seção seguinte são levantados os tipos de abordagens no cenário do desenvolvimento web, sendo realizado um paralelo entre a abordagem desenvolvimento típica e mais tradicional com a *full-stack*.

As seções seguintes entram no escopo do método de desenvolvimento *full-stack* denominado *MEAN Stack*, ilustrando cada um de seus componentes: o *MongoDB* o *Node.js*, o *Express* e o *Angular*. Por fim são abordadas as ferramentas utilizadas no desenvolvimento de aplicações móveis híbridas com o *Cordova* e o *Ionic*.

Os conceitos aqui expostos são detalhados de forma a permitir que se conheça um pouco melhor sobre os mesmos e que possa realizar um posterior aprofundamento em cada tópico abordado.

### 2.1 Aplicações Móveis, Saúde e Alimentação

O desenvolvimento tecnológico e científico nas últimas décadas simplificou a forma como as pessoas fazem sua alimentação, proporcionando economia de tempo e dinheiro com alimentos rápidos e de fácil acesso. Isso, à priori, deveria proporcionar uma alimentação mais saudável e rica em nutrientes, de maneira a aumentar a longevidade e bem-estar da população. Entretanto isso não é o que ocorre de fato, sendo observado um grande consumo de gorduras e sódio, além da diminuição de consumo de frutas e hortaliças e aumento do sedentarismo (FERREIRA, 2010).

Como coloca Proença (2010), com a modernidade e o desenvolvimento de novas tecnologias as pessoas têm procurado maior diversidade, seja esta em relação a novos produtos, serviços ou locais de alimentação a fim de tornar essa atividade mais simples. Com as refeições por muitas vezes sendo realizadas fora de casa, isso acaba levando a

um consumo exagerado de alimentos pouco saudáveis, ricos em calorias, açúcares, sal, gorduras, sódio e pobre em fibras e outros recursos importantes para a saúde, isso quando comparado com a alimentação feita em casa (BEZERRA; SICHERI, 2010).

Os dispositivos móveis, com atenção especial ao smartphone, juntamente com o advento da internet, têm proporcionado melhorias significativas na forma como as pessoas lidam com a alimentação e estilo de vida, aumentando em grande escala a viabilidade de acesso a uma enorme gama de informações e também ferramentas para auxílio de cuidados com a saúde (CRUZ; NACIF, 2015).

O desenvolvimento dos smartphones levou a um grande aumento do número de aplicativos disponíveis, onde muitos desses oferecem uma abordagem relativamente nova para intervenção na saúde e estilo de vida, de forma amplamente acessível e com uma alta relação de custo-benefício (AZAR et al., 2013).

Segundo Liu et al. (2011), tais aplicativos são enquadrados nos chamados *m-health apps*, que visam utilizar da plataforma rica em conteúdo do smartphone para oferecer opções de mudanças no estilo de vida dos usuários. Como apontam Martín et al. (2014) e Klasnja e Pratt (2012) muitos desses aplicativos são voltados para dietas, controle de calorias e motivação para exercícios físicos, sendo os de ajuda no controle de peso e conselhos sobre uma alimentação correta as mais procuradas até a data em questão.

El-Gayar et al. (2013) também mostram em seu trabalho como é difundido o uso de aplicações móveis no autocontrole de diabetes no dia a dia dos portadores da enfermidade.

Segundo Klasnja e Pratt (2012) os smartphones são um meio bastante atrativo de disponibilizar intervenções no que diz respeito à saúde e também alimentação. Os fatores são, devido ao fato de sua grande adoção aliado ao fato de haver um crescente aumento no poder computacional dos mesmos; pelo fato das pessoas levarem os dispositivos consigo para onde vão; ao fato da ligação com que as pessoas têm com seus celulares; e recursos baseados em informações pessoais do usuário. Sendo assim, os smartphones atuais representam uma plataforma ideal para a propagação de serviços relacionados a estilo de vida, como um tipo de *personal trainer*, de forma acessível para praticamente todo mundo, tendo assim um substancial impacto positivo na vida de muitas pessoas (KRANZ et al., 2013).

Baseado nisso, é possível afirmar que há uma estreita relação entre usuários de restaurantes e os cardápios dos mesmos devido ao fato das pessoas estarem mais interessadas em como se alimentam. Segundo Lessel et al. (2012) o fato de os cardápios de restaurantes estarem em formato digital ao invés do convencional de papel se mostra como um diferencial no aspecto de escolha dos usuários, por dar mais opções na hora da escolha dos pratos, onde um dos pontos que faz presente é a qualidade nutricional dos mesmos. A posse de informações permite aos usuários ter mais controle sobre aquilo que irão consumir, tornando assim um aspecto altamente atrativo para um restaurante em

conquistar novos clientes.

Na seção seguinte serão introduzidos e detalhados os conceitos relacionados ao desenvolvimento móvel, considerando os subtipos existentes na área, além de seus aspectos gerais e específicos.

## 2.2 Abordagens no Desenvolvimento Móvel

No que concerne o desenvolvimento móvel de forma geral, pode-se definir dois tipos de abordagens: o nativo e o multiplataforma. No desenvolvimento nativo as aplicações são desenvolvidas de maneira mais específica, sendo assim dependente de plataforma. Utiliza-se de todo o poderio computacional dos dispositivos alvos, tendo acesso aos seus mais específicos recursos. Dentro da abordagem multiplataforma se encontram dois subtipos: a web e a híbrida. Ambas fazem uso de tecnologias web para centralizar o desenvolvimento e poupar recursos de tempo e pessoal, além de buscar se aproximar da experiência de uso de uma aplicação nativa (ANGULO, 2014).

De forma sucinta, no que se refere desenvolvimento para dispositivos móveis, pode-se listar três tipos de abordagem possíveis (MOUBRAY et al., 2015):

- **Aplicações nativas** são compiladas na linguagem nativa do ecossistema em que a mesma se encontra (*Apple iOS* ou *Android*, por exemplo), instaladas no dispositivo e acessadas normalmente através de ícones dispostos na tela inicial do aplicativo, sendo estes aplicativos distribuídos através de lojas específicas de cada sistema operacional (*App Store*, da Apple e *Google Play*, da Google, por exemplo) (BUDIU, 2013).
- **Aplicações web** não se tratam de uma aplicação móvel propriamente dita e sim de sítios web codificados em tecnologias web e que são acessadas através de um navegador convencional (BUDIU, 2013).
- **Aplicações híbridas** são uma junção do desenvolvimento de aplicações nativas com as tecnologias de desenvolvimento web, como *HTML5*, *CSS3* e *JavaScript*, e que têm o aspecto muito similar a uma aplicação nativa propriamente dita, incluindo distribuição em lojas de aplicativos e acesso a recursos de hardware do dispositivo (COSTA, 2013).

A abordagem utilizada no trabalho é a híbrida, onde é criada uma aplicação móvel responsiva que é também distribuída como uma aplicação móvel instalável.

A Tabela 1 mostra de forma resumida os principais recursos acessíveis por uma aplicação móvel, comparando cada uma das abordagens de desenvolvimento em relação à disponibilidade de cada um deles.

Tabela 1 – Comparaçāo entre recursos acessíveis em cada abordagem.

	<b>Nativo</b>	<b>Web</b>	<b>Híbrido</b>
<b>Recursos do App</b>			
Gráficos	APIs Nativas	HTML, Canvas, SVG	HTML, Canvas, SVG
Desempenho	Rápido	Lento	Lento
Look and feel nativo	Nativo	Emulado	Emulado
Distribuição	Appstore	Web	Appstore
<b>Acesso ao Dispositivo</b>			
Câmera	Sim	Não	Sim
Notificações	Sim	Não	Sim
Contatos, calendário	Sim	Não	Sim
Armazenamento offline	Armazenamento seguro de arquivos	SQL compartilhado	Sistema de arquivos seguro, SQL compartilhado
Geolocalização	Sim	Sim	Sim
<b>Gestos</b>			
Arrastar	Sim	Sim	Sim
Pinça, espalhar	Sim	Não	Sim
<b>Conectividade</b>			
Online e offline	Principalmente online	Online e offline	Online e offline
<b>Habilidades de Desenvolvimento</b>			
ObjectiveC, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript	HTML5, CSS, Javascript

Fonte: Adaptado de Korf e Oksman (2016).

Angulo (2014) define que é importante ressaltar que um código multiplataforma possivelmente necessite de adaptações (dependendo da natureza da aplicação) ao ser implementado nas diferentes plataformas a serem usadas e, não significando assim, que todo código funcionará da exata forma em todos os dispositivos. Isso também fica claro no fato de aplicações web e híbridas poderem ser recusadas por lojas de aplicativos, como a Apple, por exemplo, que demanda que as mesmas proporcionem uma experiência similar a um aplicativo nativo, onde seja apresentado alguma diferença em relação à experiência em um navegador web comum e que se adequem ao ecossistema do sistema operacional *iOS*, de modo que o usuário se sinta “em casa” ao fazer uso dos aplicativos (TRICE, 2012).

Como será abordado ao decorrer deste capítulo, já existem *frameworks*<sup>1</sup> e ferramentas que proporcionam maneiras mais objetivas e simplificadas no que diz respeito ao desenvolvimento multiplataforma. Além disso, há os *plug-ins*<sup>2</sup> do *Apache Cordova* (os quais serão abordados mais adiante), que proporcionam o acesso aos recursos do dispositivo, se aproximando mais de uma aplicação nativa (APACHE, 2016).

Nas subseções seguintes serão apresentados de forma mais ampla os conceitos referentes a cada tipo de abordagem referente ao desenvolvimento móvel.

<sup>1</sup> Um *framework* é um ambiente de software que provém recursos com o objetivo de facilitar o desenvolvimento de aplicações, produtos e soluções.

<sup>2</sup> Também conhecidos como componentes extras ou módulos de extensão, têm o objetivo de adicionar funcionalidades a um sistema.

### 2.2.1 Aplicações Nativas

Aplicações nativas dispõem de uma gama de recursos disponibilizados pelas entidades proprietárias dos ecossistemas móveis, onde os mesmos oferecem recursos como *SDKs* (*Software Development Kits*) e documentação especialmente direcionada para a plataforma em questão. Além disso, estas aplicações se utilizam de todos os recursos do dispositivo no qual se encontram instaladas fazendo que o fator de desempenho seja mais elevado nesse tipo de aplicação. Neste caso, diferentemente de uma aplicação web comum, a nativa pode usar recursos de sensores, câmera, *GPS*, agenda telefônica, acelerômetro, dentre outros, podendo contar também com o armazenamento off-line de dados e enviar notificações de maneira nativa, sem uso de recursos externos como bibliotecas e *frameworks* (BUDIU, 2013).

Para o armazenamento de dados e informações de forma local, a aplicação pode utilizar arquivos ou mesmo um banco de dados *SQLite*, o qual é utilizado por uma grande gama de desenvolvedores e permite que os mesmos façam consulta diretamente na aplicação ou por linha de comando, através de consultas *SQL* (SQLITE, 2012).

Vale ressaltar também que a interface de uma aplicação nativa é mais integrada ao sistema em que se encontra, devido ao fato de que recursos gráficos como botões, listas e menus são emprestados à aplicação, causando uma melhor impressão e conforto ao usuário (ANDRADE et al., 2012).

Na Figura 1 pode ser observada a arquitetura em que funciona uma aplicação móvel nativa, tendo em destaque as duas plataformas mais proeminentes no mercado atualmente, *Android* e *iOS* (COSTA, 2013).

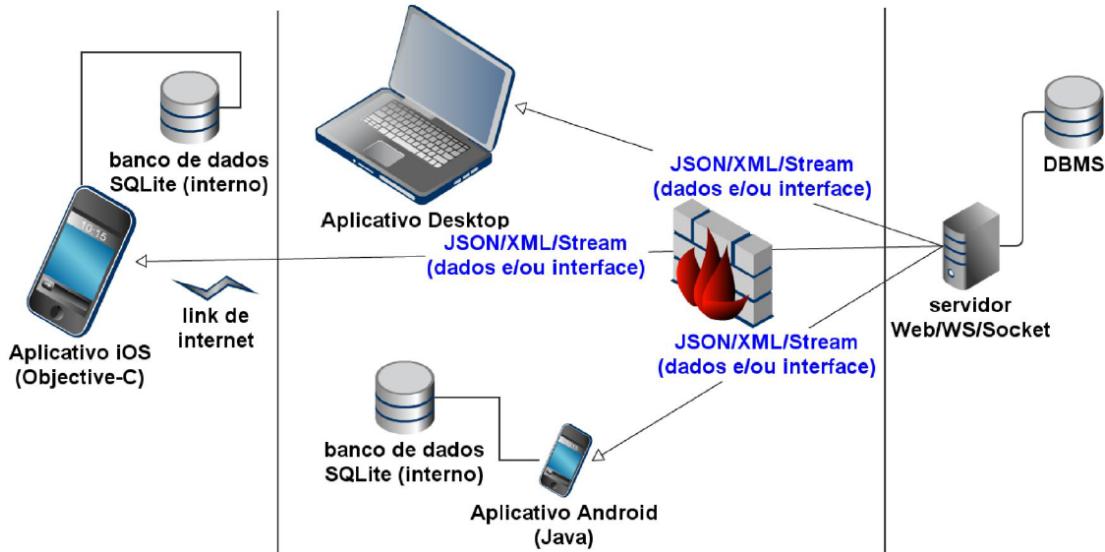


Figura 1 – Estrutura de uma aplicação nativa.

Fonte: Andrade et al. (2012)

É possível observar algumas desvantagens no desenvolvimento nativo, especialmente em relação ao desenvolvimento híbrido. Ao passo que o nativo oferece inúmeras vantagens em questão de desempenho e maior acesso a recursos do dispositivo, há certa limitação no que diz respeito à flexibilidade no desenvolvimento e distribuição da aplicação. Ao ser estabelecido que uma aplicação será desenvolvida para uma plataforma específica, possivelmente serão utilizadas ferramentas também específicas, mais adequadas àquela plataforma escolhida. Isso pode ser um problema quando posteriormente for necessário criar uma aplicação para outra plataforma, gerando custos adicionais em desenvolvimento, além de ser necessária uma nova compilação e distribuição. Isso já não ocorre em aplicações híbridas, onde há necessidade de codificar e compilar uma vez, variando na distribuição das plataformas escolhidas (BEZERRA; SCHIMIGUEL, 2016).

Outro fator importante que acarreta um ponto negativo no desenvolvimento nativo é a necessidade de domínio de cada plataforma desejada, ou ao menos a necessidade de haver desenvolvedores suficientes para suprir essa necessidade técnica. Para uma empresa de pequeno a médio porte, como uma *startup*<sup>3</sup>, por exemplo, isso acaba não sendo muito viável, pois seria necessário uma quantidade de recursos além do que normalmente é acessível para empresas desse porte (CHARLAND; LEROUX, 2011). Na Tabela 2 pode ser observado em mais detalhes o que é necessário para uma empresa disponibilizar uma aplicação que sirva aos sistemas operacionais mais populares atualmente.

Tabela 2 – Requisitos para desenvolvimento nos principais sistemas operacionais móveis

Sistema Operacional Móvel	Sistema Operacional de Desenvolvimento	Softwares/IDEs	Linguagens de Programação
<i>iOS</i>	macOS somente	Xcode	Objective-C, Swift
<i>Android</i>	Windows/macOS/Linux	NetBeans/Android Studio/Eclipse (sem suporte oficial)	Java
<i>Windows 10 Mobile</i>	Windows na maioria	Visual Studio	C#/.NET

Fonte: Adaptado de Charland e Leroux (2011) e Pierre et al. (2015).

Na Tabela 2 também pode ser observado que há a necessidade do uso de ferramentas específicas para cada ambiente de desenvolvimento. Isso acarreta à necessidade de conhecimentos específicos para cada uma respectivamente, se tornando um ponto negativo em relação a abordagem híbrida utilizando tecnologia web que necessita do desenvolvimento de um código que será adaptado para diferentes plataformas (KARADIMCE; BOGATINOSKA, 2014).

## 2.2.2 Aplicações Web

Aplicações web no geral são desenvolvidas em *HTML5*, *CSS3* e *JavaScript* e usam do poder de alcance da *World Wide Web* de forma a permitir o acesso ao seu conteúdo

<sup>3</sup> Uma *startup* consiste em uma empresa de pequeno porte que busca rápido crescimento e um modelo de negócios inovador.

através de qualquer dispositivo, seja esse móvel ou não (MOUBRAY et al., 2015). O funcionamento de uma aplicação web se faz pelo modelo cliente/servidor sobre o padrão de comunicação *Hypertext Transfer Protocol (HTTP)*. Neste padrão, o processo mais básico de funcionamento do sistema consiste em requisição/resposta (*request/response*, em inglês), onde um cliente realiza uma solicitação ao servidor e o mesmo responda de forma que o primeiro entenda, geralmente constituindo de uma página web (NIXON, 2014). A Figura 2 detalha o funcionamento do modelo.

Nixon (2014) lista as etapas do processo de requisição e resposta da seguinte forma:

1. O usuário insere o endereço *http://servidor.com* na barra de endereços do navegador.
2. O navegador então procura pelo endereço *IP* referente à *servidor.com*.
3. O navegador realiza a requisição pela página inicial do endereço informado.
4. A requisição é propagada pela internet até chegar ao servidor web onde está localizada a página.
5. O servidor web então procura pela página no disco rígido.
6. O servidor recebe a página e devolve para o navegador do usuário.
7. O navegador então exibe a página web.

Esta sequência representa o processo de requisição/resposta para páginas estáticas, sem processamento no lado do servidor. Neste segundo tipo, visto em páginas dinâmicas, pode ser necessário processar código no lado do servidor, como um código em *PHP*, o qual pode conter requisições ao SGBD, que pode ser *MySQL*, para então responder à solicitação do usuário (NIXON, 2014).

No que concerne aplicações web para dispositivos móveis, estas são acessadas como um sítio web comum, por meio de um navegador instalado no mesmo sendo o funcionamento basicamente o mesmo de como seria em um navegador desktop, como é mostrado na Figura 3.

Entretanto, os leiautes das aplicações supracitadas, em dispositivos móveis, devem ser desenvolvidos por meio da técnica de design responsivo (ANDRADE et al., 2012). O conceito de design responsivo se faz em desenvolver leiautes de páginas web que funcionem em várias telas e dispositivos e desse modo fazer com que a página se adapte às especificações do ambiente pelo qual está sendo acessada, como tamanho da tela, tipos de entradas e capacidades do navegador ou dispositivo, tudo por meio de *HTML5* e *CSS3*, sem a necessidade de usar tecnologias *back-end* ou processamento no lado do servidor (FRAIN, 2015).

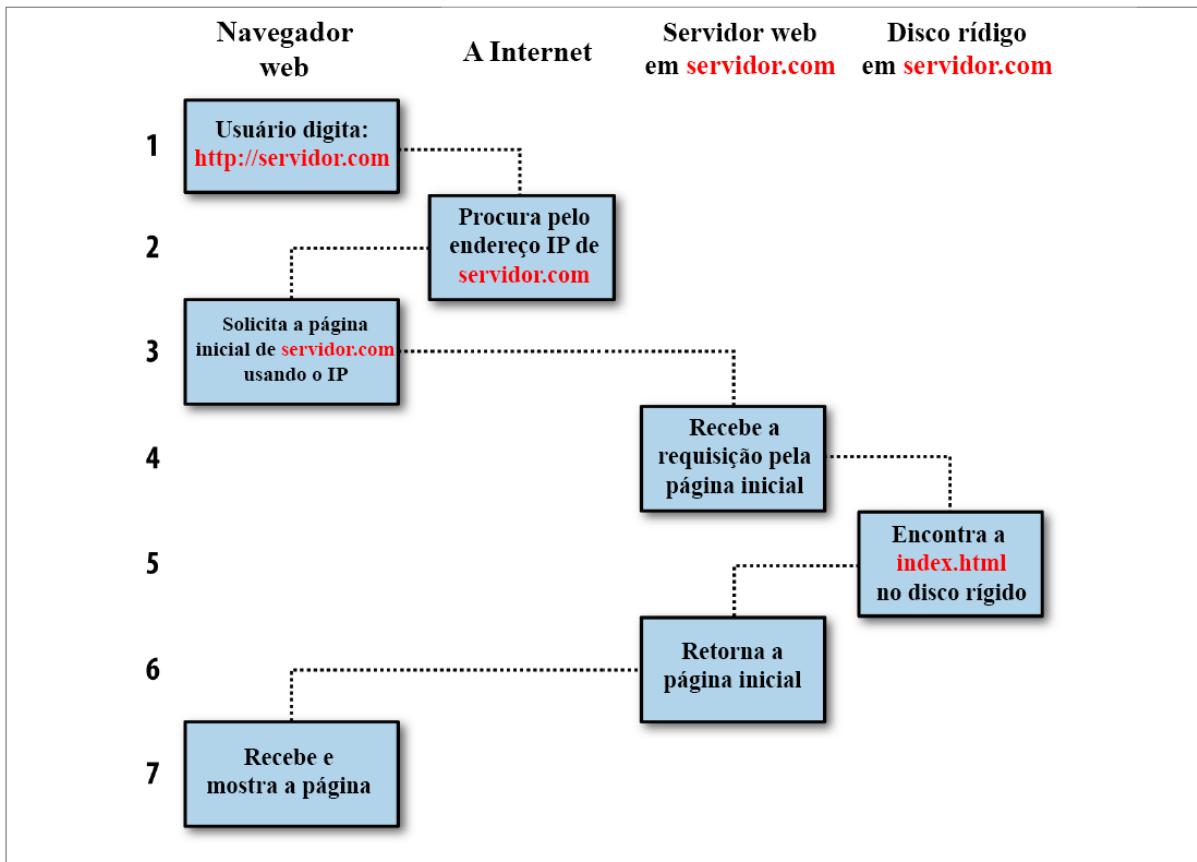


Figura 2 – Sequência básica de requisição/resposta em um modelo cliente/servidor.

Fonte: Adaptado de Nixon (2014).

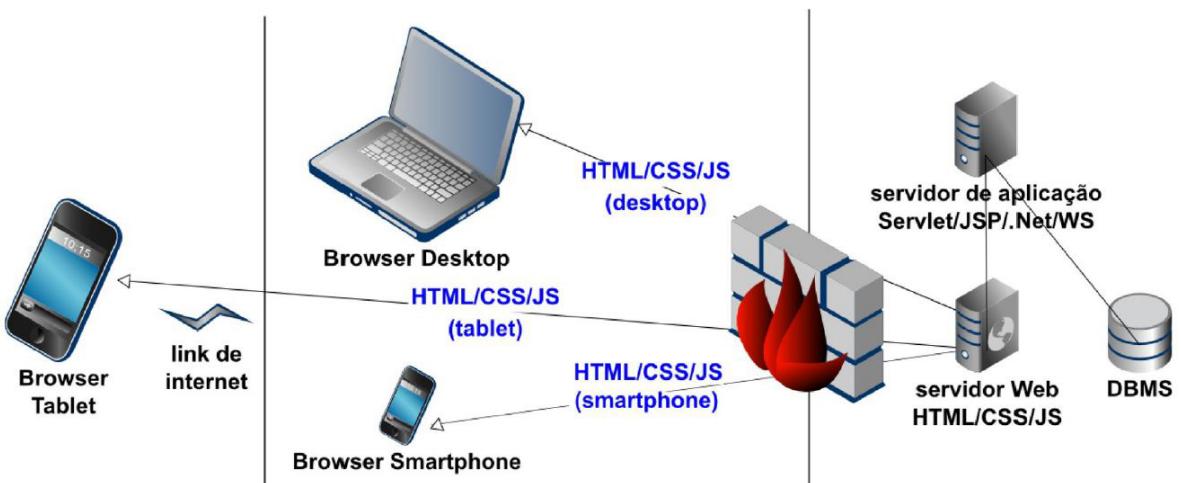


Figura 3 – Estrutura de uma aplicação web móvel.

Fonte: Andrade et al. (2012).

O desenvolvimento do *HTML5* resultou em uma versão muito mais adaptável ao ambiente móvel. As melhorias trazidas pela versão trouxeram consigo uma abertura para o desenvolvimento de aplicações que funcionem em qualquer navegador que interprete a linguagem, seja desktop ou móvel. Com isso, as aplicações se tornaram mais robustas, dispensando mesmo tecnologias de terceiros, podendo contar com recursos para execução de áudio e vídeo, por exemplo, o que permite maior interesse em desenvolvimento para esse tipo ambiente (XINO GALOS et al., 2012).

### 2.2.3 Aplicações Híbridas

Uma aplicação móvel híbrida pode ser definida como uma aplicação web incorporada em um contêiner (dito *wrapper*) nativo, provendo acesso a recursos do dispositivo no qual está instalada. Sendo assim, aproveita-se o melhor dos dois mundos, ao permitir uma grande abertura a customizações e acesso a recursos de dados do *back-end* (provados pela plataforma nativa), com o acesso universal provido pela web (provados pela plataforma da *World Wide Web*) (AMATYA; KURTI, 2014). As aplicações web são então visualizadas através de um *WebView*, que constitui uma tela de navegador que renderiza páginas web dentro do *wrapper* da aplicação nativa, onde é então associada às bibliotecas que são relacionadas com os recursos nativos (MADAUDIO; SCANDURRA, 2013).

Sendo assim, estes aplicativos proporcionam para os usuários uma alta similaridade com aplicações nativas propriamente ditas, como o fato de poderem ser baixadas de lojas de aplicativos, serem armazenadas no dispositivo e acessadas da mesma forma que é feita com aplicações nativas (KARADIMCE; BOGATINOSKA, 2014). Este fator se mostra como uma vantagem em relação às aplicações web acessadas diretamente pelo navegador, pelo fato dos usuários não perceberem estas como aplicações nativas e não permitir o acesso a recursos do dispositivo, como o sistema de arquivos e o acelerômetro, por exemplo. O uso de um *wrapper* nativo para conter a aplicação web permite que o usuário se sinta familiarizado com a interface, além de permitir que o desenvolvedor personalize os nomes e ícones à sua maneira e apenas carregue a aplicação web dentro do contêiner nativo (COSTA, 2013).

A Figura 4 ilustra o funcionamento de uma aplicação híbrida. Estas funcionam como uma aplicação web ou nativa comum, realizando comunicação com o servidor, sendo trafegados dados ou descrição dos componentes da aplicação. Isso também estabelece uma vantagem em relação a uma aplicação nativa, onde quando se faz necessária uma alteração que afete somente a interface do usuário a mesma pode ser feita somente no servidor, evitando assim um novo processo de validação e publicação na loja de aplicativos da plataforma (ANDRADE et al., 2012).

Devido ao fato que os desenvolvedores desejam criar mais escrevendo menos código, o reuso e a reciclagem do mesmo para várias plataformas se torna algo essencial,

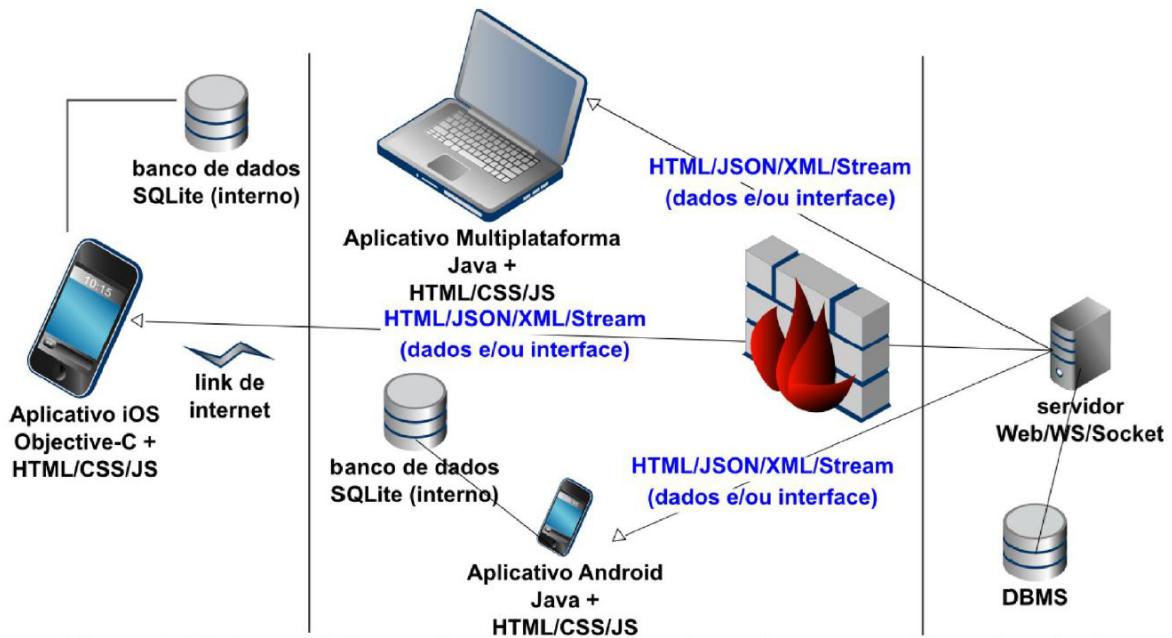


Figura 4 – Estrutura de uma aplicação híbrida.

Fonte: Andrade et al. (2012).

pois evita-se de escrever o código todo do zero novamente para cada plataforma, bastando somente realizar adaptações necessárias. Isso se torna um grande diferencial no desenvolvimento híbrido, pois o código escrito em *HTML*, *CSS* e *JavaScript* é reutilizado em diferentes plataformas, permitindo criar uma espécie de “ponte” entre o navegador web a as *APIs* (*Application Programming Interfaces*) do dispositivos podendo aproveitar de todos os recursos e potencial dos mesmos. O fator de poupar de tempo se faz ainda mais evidente na possibilidade do uso de *frameworks* para desenvolvimento (como o *Apache Cordova*, por exemplo) com tecnologias web que oferecem interface amigável ao usuário e acesso aos recursos do dispositivo. Além disso, existe a possibilidade de testar a aplicação diretamente no navegador, justamente por se tratar de uma aplicação web (KARADIMCE; BOGATINOSKA, 2014).

De acordo com Madaudo e Scandurra (2013), existem também outras abordagens e ferramentas para o desenvolvimento híbrido, como *Adobe Air*, *Appcelerator Titanium* e *Corona SDK*, por exemplo que permitem escrever o código em linguagem abstrata e então compilar para o código nativo. Por exemplo, o *Titanium* permite que a aplicação seja codificada em *JavaScript* e compilada para código nativo, convertendo o código em classes e objetos da plataforma em questão. Diferentemente de *frameworks* como *PhoneGap* e *Cordova*, por exemplo, que utilizam somente o *WebView* para renderizar o código do aplicativo. Contudo, esse modelo de desenvolvimento de aplicações híbridas não será abordado neste trabalho, sendo o mesmo voltado para o desenvolvimento utilizando tecnologias web.

Como visto anteriormente na Tabela 1, o desenvolvimento de aplicações híbridas, ainda que seja uma abordagem melhor que a relacionada somente com a criação de uma aplicação móvel responsiva, possui suas desvantagens em relação à abordagem nativa, principalmente no que diz respeito ao desempenho. Entretanto, suas vantagens se fazem suficientes para que seja levada em consideração ao desenvolver aplicações móveis. Em seu trabalho, Palmieri et al. (2012) elucidam e resumem os benefícios que esse tipo de abordagem proporciona:

- Redução na quantidade de habilidades necessárias para o desenvolvimento de aplicações devido ao uso de linguagens em comum;
- Redução da quantidade de código devido à sua reusabilidade na compilação para diferentes plataformas que serão atendidas;
- Redução do desenvolvimento e de custos relacionados à manutenção em longo prazo;
- Redução na necessidade de conhecimento de diferentes *APIs*, sendo necessário somente o conhecimento da *API* da ferramenta a ser utilizada.

A abordagem por desenvolvimento híbrido foi escolhida para este trabalho devido à sua abrangência de diversas plataformas, podendo a aplicação ser disponibilizada online e para *Android* de imediato, com a possibilidade de expansão para plataformas como *iOS* ou *Windows 10 Mobile* facilitada para trabalhos futuros.

### 2.3 Desenvolvimento *Full-stack*

No que diz respeito à estrutura de uma aplicação web, esta pode ser dividida em duas partes: o *front-end* e o *back-end*.

O *front-end* consiste na apresentação da interface que o usuário vê e interage, consistindo de menus, imagens, formulários, botões e afins. É desenvolvida por meio da tríade padrão das tecnologias web, *HTML*, *CSS* e *JavaScript*, sendo operada através do navegador web.

O *back-end* consiste em três partes distintas e relacionadas, sendo estas o servidor, a aplicação e o banco de dados (*database* ou *DB*). As tecnologias usadas geralmente são o *Apache Web Server* como exemplo de servidor; o *PHP*, *Ruby*, *Python* etc., para a aplicação; e *MySQL*, *MongoDB*, *PostgreSQL*, dentre outros, para o banco de dados (ABDULLAH; ZEKI, 2014).

Dentro desse contexto, podem ser considerados dois tipos de abordagens no que diz respeito aos tipos de desenvolvimento: o tradicional e o *full-stack*. Dentro do contexto tradicional há desenvolvedores especializados em cada etapa da aplicação (GIRDLEY,

2014). Já no desenvolvimento *full-stack* um desenvolvedor é responsável por toda aplicação, iniciando do banco de dados e do servidor web, até a interface do usuário no *front-end* (FEKETE, 2014). Esse tipo de abordagem se faz muito útil e necessária em *startups* e pequenas empresas, pois o orçamento por muitas vezes limitado demanda que os desenvolvedores adquiram conhecimento necessário para entregar um produto ou serviço prestado por tais empresas (WESTBERG, 2014).

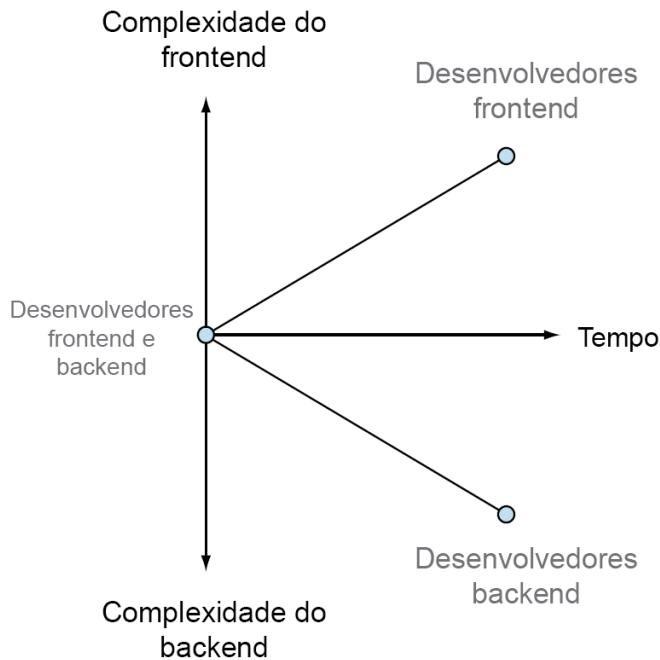


Figura 5 – Divergência entre desenvolvedores *front-end* e *back-end* ao longo do tempo.

Fonte: Adaptado de Holmes (2016).

Historicamente, o contexto do desenvolvimento *full-stack* tem sua origem nos primórdios da *World Wide Web*. Segundo Holmes (2016), no surgimento da web não havia muitas expectativas em relação aos sítios web em geral. Sendo assim também não havia muita preocupação com a apresentação das páginas web, e o desenvolvimento se concentrava mais na parte do código básico em *HTML* e linguagem de processamento no servidor, como o *Perl*, por exemplo. Com o crescimento da internet e o aumento do suporte ao *CSS* e *JavaScript* nos navegadores, tornou-se maior a preocupação com a apresentação das páginas web, gerando uma exigência maior em relação ao tempo que era dedicado a melhor essa questão. Este foi o ponto que criou um discernimento entre desenvolvedor *back-end* e *front-end*, onde o primeiro se concentrou mais nas mecânicas por trás das aplicações, enquanto o segundo manteve o foco em criar uma boa experiência para o usuário e a adaptação a diferentes navegadores. A Figura 5 mostra essa separação em relação ao tempo.

Esse contexto mudou durante os anos 2000 devido à popularização de bibliotecas e *frameworks* tanto para *front-end* quanto para *back-end* (como a *jQuery*, uma biblioteca *JavaScript* voltada para o *front-end*, e o *CodeIgniter*, um *framework* para *PHP* e *Ruby on*

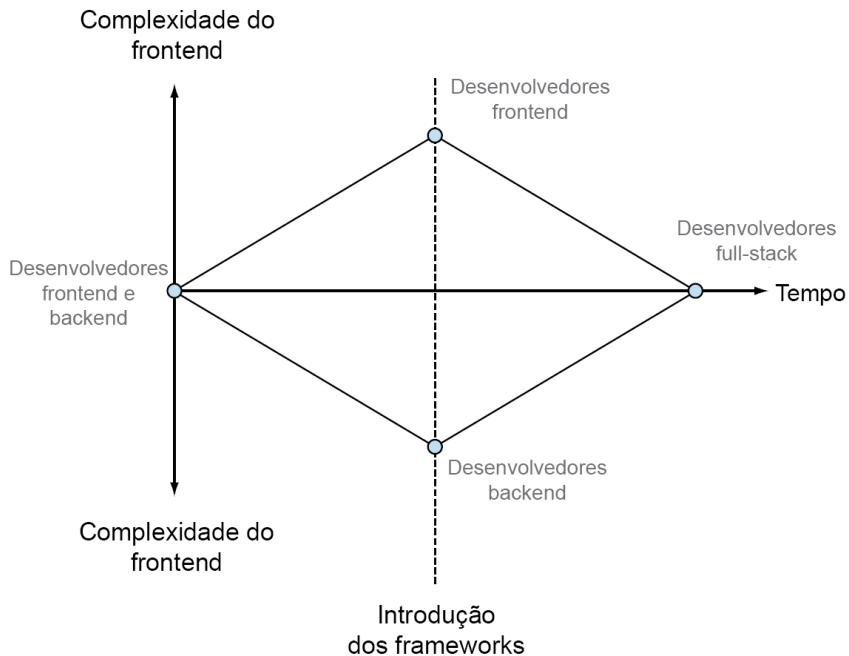


Figura 6 – Impacto dos *frameworks* nos dois aspectos separados do desenvolvimento web.

Fonte: Adaptado de Holmes (2016).

*Rails*) que visavam tornar o desenvolvimento mais rápido e fluído, diminuindo a necessidade de conhecimento técnico aprofundado nas tecnologias usadas. Isso criou então uma tendência para o ressurgimento de desenvolvedores *full-stack*, como mostra na Figura 6 (HOLMES, 2016).

Isso ilustra uma tendência de que, com o uso de ferramentas modernas e *frameworks*, passa a não ser necessária a escolha de um lado para ser um bom desenvolvedor web, mostrando que a vantagem do uso de *frameworks* se faz ao permitir ao desenvolvedor ser produtivo e ter uma noção do funcionamento da aplicação como um todo e como ela se integra adequadamente (HOLMES, 2016).

Nas próximas seções são abordadas de forma sucinta e direta algumas das tecnologias necessárias para o desenvolvimento do *front-end*, *back-end* e banco de dados da aplicação.

## 2.4 A *MEAN Stack*

A *MEAN Stack* consiste em uma abordagem de desenvolvimento *full-stack* que engloba quatro tecnologias: *MongoDB*, *Express*, *Angular* e *Node.js*. Esta *stack* contempla os três componentes de uma aplicação, sendo o *MongoDB* responsável pela parte do banco de dados, o *Node.js* e *Express* pelo *back-end* e o *Angular* pela parte do *front-end*.

A parte do *back-end* da aplicação como um todo tem o servidor construído sobre o *Node.js* contendo o *Express* como uma extensão do mesmo de modo a permitir a

definição de uma *API REST* (*Representational State Transfer*, Transferência de Estado Representacional) que exponha os *endpoints*, ou pontos de acesso (CANTELON et al., 2014).

Na aplicação do *Node.js* também se faz presente uma ferramenta que possibilite a interação com o banco de dados, como o *Mongoose ODM*, que fazendo essa ponte entre ambos. Já a parte do *front-end* é construída através de uma *SPA* (*Single Page Application*, Aplicação de Página Única) utilizando o framework *Angular* e que faz interação com a *endpoints* expostos no servidor (CANTELON et al., 2014)(HOLMES, 2016). A estrutura de uma aplicação construída utilizando a *MEAN Stack* é ilustrada na Figura 7.

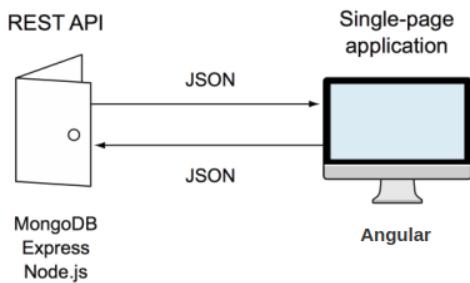


Figura 7 – Exemplo de funcionamento de uma aplicação utilizando a *MEAN Stack*.

Fonte: Adaptado de Holmes (2016).

Como apontam Holmes (2016) e Cantelon et al. (2014), a comunicação de uma aplicação com a *API* no servidor em uma aplicação que segue o modelo *REST* se dá essencialmente utilizando-se dos verbos *HTTP*, como *GET*, *POST*, *PUT* e *DELETE*, por exemplo, para se comunicar com o mesmo. Esta comunicação é concretizada ao enviar requisições utilizando *payloads* em formato *JSON* (*JavaScript Object Notation*) que levam e trazem informações de e para o servidor.

De acordo com Tilkov (2008) e Ferreira (2017), a comunicação é concretizada nos *endpoints* mapeados no lado da *API* através de um canal de recursos unificado denominado *URI* (*Universal Resource Identifier*, Identificador de Recurso Universal) que consiste essencialmente na identificação de um endereço para onde irão as requisições enviadas ao servidor. A Figura 8 ilustra exemplos de *endpoint* de uma aplicação no modelo *REST*.



Figura 8 – Exemplos de endpoints de uma aplicação *REST*.

Fonte: Adaptado de Tilkov (2008).

A definição da estrutura de funcionamento da *MEAN Stack* pode ser observada na Figura 9. O lado do cliente contém a lógica do *front-end* e é utilizada para realizar as requisições à *API REST* e mostrar os resultados das mesmas. Ao enviar a requisição ao servidor *Node.js*, o *framework Express* intercepta essa requisição e processa a mesma por meio de uma função definida na aplicação.

O *framework* então irá utilizar os recursos para buscar ou enviar os dados ao servidor *MongoDB*, dependendo do caso. Ao terminar esse processamento, o *Express* irá devolver os dados ao cliente em formato *JSON* para o *Angular* que irá processar e mostrar os dados ao usuário de acordo com as regras da aplicação (PHAM, 2016).

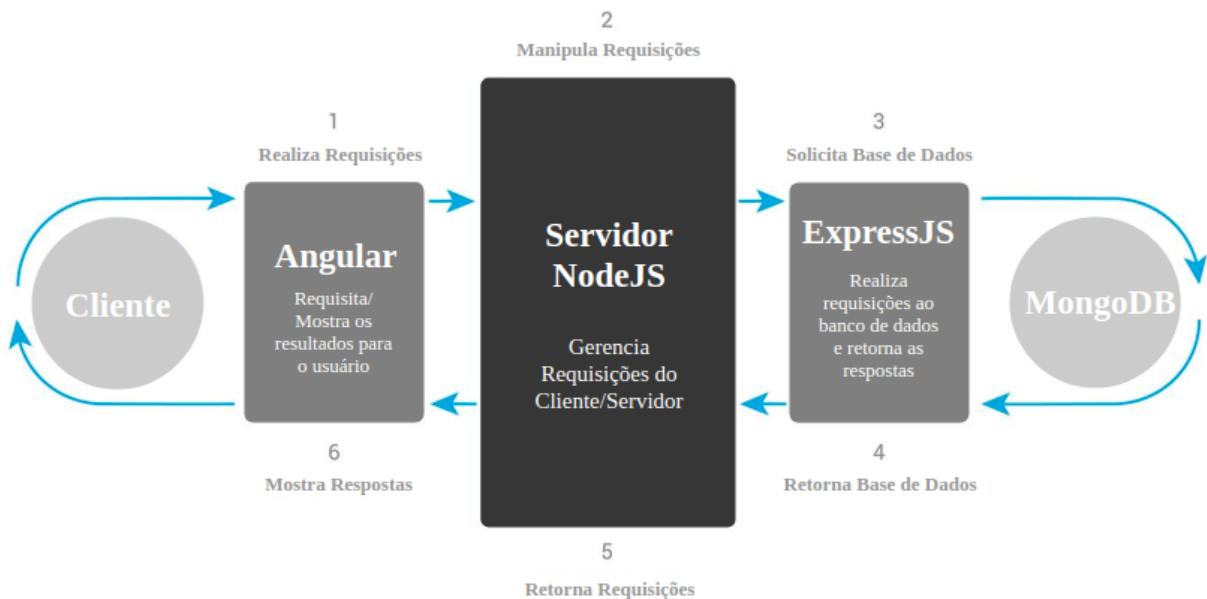


Figura 9 – Estrutura de uma aplicação *MEAN*.

Fonte: Adaptado de Bacancy (2016).

Nas próximas seções são abordados os conceitos irentes à cada uma das tecnologias presentes na *stack*.

#### 2.4.1 *MongoDB*

Desenvolvido pela *MondoDB Inc.*, o *MongoDB* é um sistema de gerenciamento de banco de dados do tipo *NoSQL* (*Not Only SQL*, Não Somente *SQL*), sendo assim não-relacional, multiplataforma e orientado a documentos (MONGODB, 2018a).

Utiliza o formato *BSON*, que representa um *Binary JavaScript Object Notation* (JSON Binário), do tipo chave-valor, para armazenamento de dados como documentos, que são então agrupados em coleções, de forma similar aos esquemas dos bancos de dados relacionais. Esses documentos podem ter similaridades mas não necessariamente precisam ser iguais, podendo ter propriedades distintas uns dos outros. Além disso, os campos

presentes em um documento salvo em uma coleção do *MongoDB* podem possuir *arrays* de outros documentos e de outras coleções (PHAM, 2016)(HOLMES, 2016).

```
var mydoc = {
  _id: ObjectId("5099803df3f4948bd2f98391"),
  name: { first: "Alan", last: "Turing" },
  birth: new Date('Jun 23, 1912'),
  death: new Date('Jun 07, 1954'),
  contribs: [ "Turing machine", "Turing test", "Turingery" ],
  views : NumberLong(1250000)
}
```

Figura 10 – Formato de armazenamento de documentos no *MongoDB*.

Fonte: Holmes (2016).

Como mostra a Figura 10, o armazenamento de documentos no *MongoDB* se faz por objetos no formato de objeto do *JavaScript*. Uma propriedade importante que está sempre presente é o `_id`, que representa um identificador único atribuído a todo documento registrado na base do *MongoDB* (HOLMES, 2016).

Diferentemente de sistemas de gerenciamento de banco de dados do tipo relacional, os esquemas ou documentos no *MongoDB* são flexíveis, isto é, podem ser alterados mesmo após ter sido definidos, possibilitando também que sejam gravados registros sem mesmo ter definido a estrutura de uma coleção. Isso permite que as coleções sejam auto-descritivas dentro de uma aplicação, sendo também bastante útil caso seja necessário adicionar algum novo recursos posteriormente à implementação dos mesmos (MONGODB, 2018b).

De acordo com Holmes (2016) e Banker et al. (2016), uma grande vantagem do *MongoDB* é que é possível utilizar mais de um índice de busca tornando as buscas mais eficientes e mais rápidas. Porém seu uso deve ser evitado essencialmente em operações transacionais, isto é, onde várias operações são feitas separadamente em uma mesma transação. Se caso uma falhar, todas as outras irão falhar também. No *MongoDB* isso não é possível, sendo as operações independentes. Se caso uma falhar, esta será ignorada e as outras serão continuadas normalmente.

Em aplicações maiores e mais robustas são necessárias estruturas mais consistentes e bem definidas. Para isso existe a opção de modelagem de bases de dados construídas utilizando o *MongoDB*. Como define Holmes (2016) a modelagem permite construir a estrutura dos dados, adicionar validações, realizar consultas, definir campos obrigatórios e tipos dados que determinados campos irão suportar.

O *Mongoose ODM* (*Object Data Modeling*, Modelagem de Objeto de Dados, em tradução livre) é uma ferramenta de modelagem de dados do *MongoDB* para *Node.js*. Esta permite estruturar e manipular os documentos através dos esquemas do *Mongoose*, como mostra a Figura 11.

Sendo assim, além de ferramenta de modelagem dos esquemas, o *Mongoose* serve

```

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});

```

- (a) Definição de um esquema denominado *blogSchema*.

```

var animalSchema = new Schema({
  name: String,
  type: String,
  tags: { type: [String], index: true } // field level
});

animalSchema.index({ name: 1, type: -1 }); // schema level

```

- (c) Definição de um índice composto no esquema *animalSchema*, indicando que o nome deverá estar em ordem decrescente.

```

animalSchema.query.byName = function(name) {
  return this.find({ name: new RegExp(name, 'i') });
};

var Animal = mongoose.model('Animal', animalSchema);
Animal.find().byName('fido').exec(function(err, animals) {
  console.log(animals);
});

```

- (b) Execução de ação de busca por nome sobre um esquema denominado *Animal*.

Figura 11 – Definições de esquemas no *Mongoose*.

Fonte: Automattic (2018)

como um intermediário entre a aplicação do *Node.js* e o banco do *MongoDB*, permitindo acessar e manipular os registros presentes na base de dados à medida que são requisitados e alterados na mesma.

#### 2.4.2 *Node.js*

O *Node.js*, ou simplesmente *Node*, é uma plataforma orientada a eventos assíncrona que permite desenvolvimento de aplicações escaláveis do lado do servidor utilizando *JavaScript* (HERRON, 2013)(CANTELON et al., 2014).

Criado por Ryan Dahl em 2008, é executado sobre a plataforma *V8* do *Google*, um motor de código aberto desenvolvido pela *Google*. Mesmo tendo o *.js* no nome, o *Node.js* tem a maior parte do seu código desenvolvido em *C++*, sendo este correspondendo por 60% dos códigos fonte do software (HERRON, 2013).

Devido ao fato de utilizar uma arquitetura orientada a eventos e um modelo de E/S não-bloqueante, o *Node.js* é ideal para aplicações em tempo real e com uso intensivo de dados (HOLMES, 2016).

De acordo com Tesanovic (2018), o processo do *Node.js* em si é executado em uma *thread*, porém com múltiplas *threads* executando por trás desta permitindo a manipulação várias requisições simultaneamente, o que diferente de modelos baseados em *thread* única.

Por ter um modelo assíncrono e não-bloqueante, todas as funções, definidas como *callbacks*, são delegadas a um *event loop* que permanece executando durante o tempo ativo do processo do *Node.js*. Estas funções são então podem ser delegadas a múltiplas *threads* que ficam responsáveis pela execução das tarefas bloqueantes do sistema, ou seja, que iria fazer com que o processo da aplicação interrompesse sua execução (TESANOVIC, 2018).

O *Node.js* é bastante utilizada na construção de servidores web, devido essencialmente à facilidade do seu uso para este propósito. Diferentemente de outras plataformas que necessitam de um *software* para construção da aplicação e outro para a hospedagem, como uma aplicação em *PHP* que necessita ser hospedada em um servidor do *Apache HTTP Server*, por exemplo, no *Node.js* a aplicação e o servidor são a mesma coisa (CANTELON et al., 2014). A Figura 12 ilustra o funcionamento de um servidor *HTTP* construído com *Node.js*.

```
var http = require('http');
var server = http.createServer();

server.on('request', function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
})

server.listen(3000);
console.log('Servidor executando em http://localhost:3000');
```

Figura 12 – Exemplo de criação de um servidor em *Node.js*.

Fonte: Adaptado de Cantelon et al. (2014).

Neste caso é um exemplo bem simples, onde toda vez que uma requisição for feita ao servidor, ele retornará sempre Olá Mundo como resposta. Em aplicações maiores as requisições são manipuladas das mais diversas formas, dependendo do que está sendo buscado ou enviado ao servidor (CANTELON et al., 2014).

Toda aplicação criada com o *Node.js* tem uma padronização, contida em um arquivo do tipo *JSON* chamado *package.json*. Esse arquivo define os recursos existentes na aplicação atual em que se está trabalhando, qual é o ponto de entrada da aplicação, isto é, à partir de qual arquivo será iniciado a aplicação do *Node.js*, e as suas dependências (CANTELON et al., 2014).

Em aplicações reais, a necessidade de recursos mais complexos e melhores se fazem necessários. Estes recursos são muitas vezes desenvolvidos por terceiros para que possam ser utilizados por outras pessoas em suas próprias aplicações, fazendo-se necessário um meio de distribuir e acessar de forma padronizada esses recursos, sendo estes então chamados de dependências pelas aplicações que os utilizam (HERRON, 2013)

Como define Holmes (2016), a fim de padronizar a forma de distribuição desses recursos faz-se necessário um gerenciador de dependências. Um recurso utilizado para tal finalidade é o *Node Package Manager (NPM)*, ou Gerenciador de Pacotes do Node. O *NPM* é instalado no sistema operacional destino e pode ser utilizado via linha de comando.

Além de gerenciar dependências, o *NPM* é utilizado como o gerenciador da aplicação em si. É possível iniciar uma nova aplicação do *Node.js* pelo comando `npm init`, ou `npm install <nome_da_dependência> --save` para salvar uma nova dependência, por exemplo, e adicionar a referência da mesma no arquivo `package.json`, ou também adicionar scripts que servem como atalhos para funcionalidades na aplicação, como o `npm start`, que irá iniciar a aplicação do *Node.js* à partir do seu ponto de entrada (NPM, 2018).

Todas as dependências de uma aplicação do *Node.js* ficam salvas em um diretório `node_modules`, localizado na raiz da aplicação. Sempre que uma aplicação é distribuída e outro usuário deseja iniciá-la, terá somente que utilizar o comando `npm install` para baixar as dependências e então iniciar a aplicação pelo comando direto do *Node.js* ou pelo script do *NPM* (NPM, 2018).

#### 2.4.3 Express

Criado originalmente em 2009 por T. J. Holowaychuk e atualmente um projeto da Fundação *Node.js* e mantido pela empresa StrongLoop, o *Express* é um *framework* para o *Node.js* que possui um conjunto de recursos e funcionalidades robustas para o desenvolvimento de aplicações web que permitem auxiliar na organização das aplicações web no lado do servidor. Isto é concretizado por meio de recursos como roteamento, exposição de rotas de métodos para requisições *HTTP* e de *middlewares* (STRONGLOOP, 2018).

Como mostra a documentação oficial do *framework Express* providenciada por StrongLoop (2018), o *framework* possui em sua composição três componentes principais, sendo eles o da aplicação, o da requisição e o da resposta. O objeto da aplicação é uma instanciação direta do *Express* por meio do método principal `express()`. Geralmente se utiliza uma variável chamada `app` para se nomear esse objeto e é por meio dela que são acessadas as funcionalidades presentes no *Express*, como mostra a Figura 13.

```
var express = require('express');
var app = express();
```

Figura 13 – Instanciação do objeto do *Express*.

Fonte: Adaptado de StrongLoop (2018).

O componente de requisição é criado toda vez que uma requisição é feita ao servidor contendo o *Express*. Nele são contidas as informações referentes à requisição *HTTP* enviada ao servidor, como cabeçalho, corpo e parâmetros, por exemplo. Já o componente de resposta é criado juntamente com a requisição e serve para retornar os dados para o cliente que fez a requisição inicialmente (STRONGLOOP, 2018).

Toda as requisição enviada a uma aplicação contendo o *Express* é manipulada por rotas e *middlewares*. As rotas são as definições dos *endpoints* da aplicação, ou seja, as *URIs* ou caminhos presentes na aplicação que irão redirecionar para métodos que irão manipular as requisições que chegam ao servidor, podendo existir vários manipuladores para uma mesma rota.

Os manipuladores são definidos pelos *middlewares* presentes na aplicação. Esses métodos recebem a requisição, a resposta, e um parâmetro chamado `next`, que é utilizado para dar sequência à execução do encadeamento de *middlewares*, caso esse exista. Um método deve sempre encerrar a requisição enviando a resposta ao cliente ou chamar o próximo método por meio do `next()`. *Middlewares* externos podem ser carregados na aplicação por meio do método `use()`, acessível por meio do objeto `app`, figurando como `app.use()`. A Figura 14 ilustra a estrutura completa de um método típico dentro do *Express* (STRONGLOOP, 2018).

```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

Figura 14 – Estrutura de um método típico no ambiente do *Express*.

Fonte: Adaptado de StrongLoop (2018).

Neste caso o método pode ser dividido em quatro partes distintas. O `app`, que define o objeto da aplicação; o `get` que representa o método *HTTP* que deverá ser usado para acessar aquela rota; o `/` que representa o caminho do endpoint na aplicação; e o método que fará a manipulação da requisição. Por convenção utiliza-se os parâmetros `req`, `res` e `next`, sendo este último obrigatório somente em casos onde o mesmo seja utilizado para chamar outros *middlewares* dentro da aplicação (STRONGLOOP, 2018).

#### 2.4.4 Angular

Criado e mantido pela *Google* e pela comunidade, o *Angular* é uma reconstrução do *framework* *AngularJS* e consiste em uma plataforma e um *framework* para construção *front-end* utilizando *HTML* e *TypeScript*<sup>4</sup> (GOOGLE, 2018b).

Uma aplicação construída utilizando o *Angular* possui em sua consistência blocos de construção chamados de *NgModules*. Esses módulos representam as funcionalidades existentes dentro de uma aplicação. Além desses módulos, uma aplicação deverá ter sempre um módulo *root* denominado *AppModule* que é responsável pelo *bootstrap* da aplicação, ou seja, o módulo que será o ponto de partida na construção da aplicação (GOOGLE, 2018b). A Figura 15 ilustra exemplos de módulos em uma aplicação do *Angular*.

<sup>4</sup> <https://www.typescriptlang.org/>

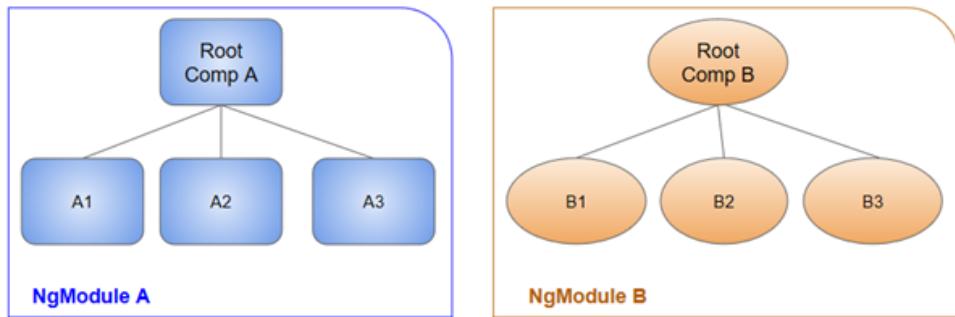


Figura 15 – Exemplo de dois módulos independentes em uma aplicação do *Angular*.

Fonte: Google (2018b).

Como define a documentação oficial do *Angular* providenciada por Google (2018b), outros dois recursos importantes em uma aplicação *Angular* são os serviços e os componentes. Os componentes definem as chamadas *views*, que correspondem aos componentes que são apresentados na tela e que são modificados dependendo das regras da aplicação e dos dados. Assim como o módulo, é obrigatória a presença de pelo menos um componente *root* na aplicação.

Já os serviços proveem funcionalidades utilizadas por toda aplicação e que serão injetadas diretamente nos componentes. Sua utilização tem ênfase em permitir a reusabilidade de código de forma modular e eficiente. Ambos os tipos são simplesmente classes, porém utilizando decoradores como `@Component` e `@Injectable`, que indicam que tal classe é um componente ou um serviço, respectivamente (GOOGLE, 2018b).

De forma similar as dependências do *Node.js*, importadas pelo *NPM*, o *Angular* também pode conter dependências externas. Essas dependências são módulos ou aplicações do *Angular*, registradas no arquivo `package.json` e gerenciadas pelo *NPM*.

Um ponto importante do *Angular* são os templates. Um template é bastante similar a conteúdo *HTML* comum, porém contendo a sintaxe de template do *Angular*, que altera o conteúdo que será exibido na tela baseado no estado atual da aplicação, na lógica e nas regras da mesma. O conteúdo da página *HTML* é alterado por meio de interpolação das propriedades e os eventos são atrelados a métodos, ambos presentes no componente ao qual o template está atrelado. A Figura 16 exemplifica o uso do template no *Angular*.

O conteúdo do template por ser alterado por meio de diretivas e *pipes*, que permitem processar o conteúdo que será apresentado na tela. Como por exemplo as diretivas `*ngIf`, que só apresentará determinado conteúdo sob uma determinada condição dentro da aplicação, e `*ngFor`, que percorre um *array* presente no componente e apresenta o conteúdo do mesmo na tela. Um *pipe* como o `date`, por exemplo, é utilizado para transformar o conteúdo de uma determinada propriedade no formato padrão definido de datas. *Pipes* e diretivas podem ser criados para atender as demandas dentro de uma aplicação (GOOGLE, 2018b).

```

<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>

```

Figura 16 – Representação da sintaxe de template do *Angular*.

Fonte: Google (2018b).

Para navegar em uma aplicação do *Angular* são utilizadas as rotas, providas pelo módulo nativo do *Angular* chamado **Router**, que dão acesso a estados possíveis da aplicação. As rotas são definidas seguindo as convenções padrão dos navegação de web sítios, como ao inserir uma *URL* (*Uniform Resource Locator*, Localizador Uniforme de Recursos) na barra de navegação, clicando em algum link interno ou usando os botões de avançar e voltar presentes no navegador. Se uma determinada rota depende de funcionalidades de outros módulos, a rota irá esse módulo por meio do *lazy loading*<sup>5</sup> e carregará o mesmo dinamicamente. A Figura 17 ilustra o uso das rotas em uma aplicação.

Ao definir as rotas na aplicação, devem ser especificados quais componentes correspondem a qual *URL*, podendo passar parâmetros para aquele estado correspondente ou não. No template *HTML* as rotas são então acessadas por intermédio de um *routerLink*. É necessária a utilização da diretiva **<router-outlet>** para determinar para onde o conteúdo requisitado no estado da aplicação será carregado dinamicamente assim que a rota for acessada pelo usuário (GOOGLE, 2018b).

Toda manipulação e execução das aplicações construídas utilizando o *framework* são realizadas por meio da *Angular CLI* (*Command Line Interface*, Interface de Linha de Comando). Assim como outras *CLIs*, a *Angular CLI* permite que por meio de comandos do terminal sejam executadas ações como iniciar, executar e construir uma aplicação, por exemplo (GOOGLE, 2018a).

A *Angular CLI* é instalada através da linha de comando e é dependente do *Node.js* e também do *NPM*. Sua instalação é feita pelo comando `npm install -g @angular/cli`, onde o parâmetro `-g` indica que a instalação será global e não somente para o projeto corrente onde esteja sendo utilizada, podendo ser acessada via linha de comando em qualquer outro diretório (GOOGLE, 2018a).

<sup>5</sup> Carregamento de componentes sob demanda, isto é, somente quando forem requisitados dentro do contexto.

```

const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id', component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];

```

(a) Definição das rotas no módulo *root*.

```

template: `
<h1>Angular Router</h1>
<nav>
  <a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>
  <a routerLink="/heroes" routerLinkActive="active">Heroes</a>
</nav>
<router-outlet></router-outlet>

```

(b) *Links* para rotas no template *HTML*.Figura 17 – Definição das rotas em uma aplicação *Angular*.

Fonte: Google (2018b)

Nas próxima seção são abordados os conceitos irentes à cada as tecnologias de desenvolvimento híbrido.

## 2.5 Desenvolvimento Híbrido

Esta seção aborda as tecnologias de desenvolvimento híbrido móvel que serão utilizadas no desenvolvimento da aplicação.

### 2.5.1 Apache Cordova

O *Apache Cordova* é um *framework* de código aberto destinado desenvolvimento móvel multiplataforma por meio das tecnologias web *HTML5*, *CSS3* e *JavaScript*. O *Cordova* incorpora as aplicações a *wrappers* destinados a cada plataforma, de modo a oferecer acesso a recursos do dispositivo como os sensores, dados do dispositivo, status da rede, dentre outros (CORDOVA, 2016).

O uso do *framework* é recomendado nos casos onde o desenvolvedor deseja ampliar sua aplicação para múltiplas plataformas sem a necessidade de rescrever todo o código, sendo também recomendável quando há o interesse em usar o *WebView* para acessar

os recursos do dispositivo através das *APIs* dos mesmos. Além disso, também pode ser utilizado quando o desenvolvedor pretende distribuir uma aplicação web por diversas lojas de aplicativos (CORDOVA, 2016).

A arquitetura de uma aplicação *Cordova* é constituída por diversos componentes. A Figura 18 mostra de uma forma geral como é a estrutura de uma aplicação desenvolvida usando o *framework*.

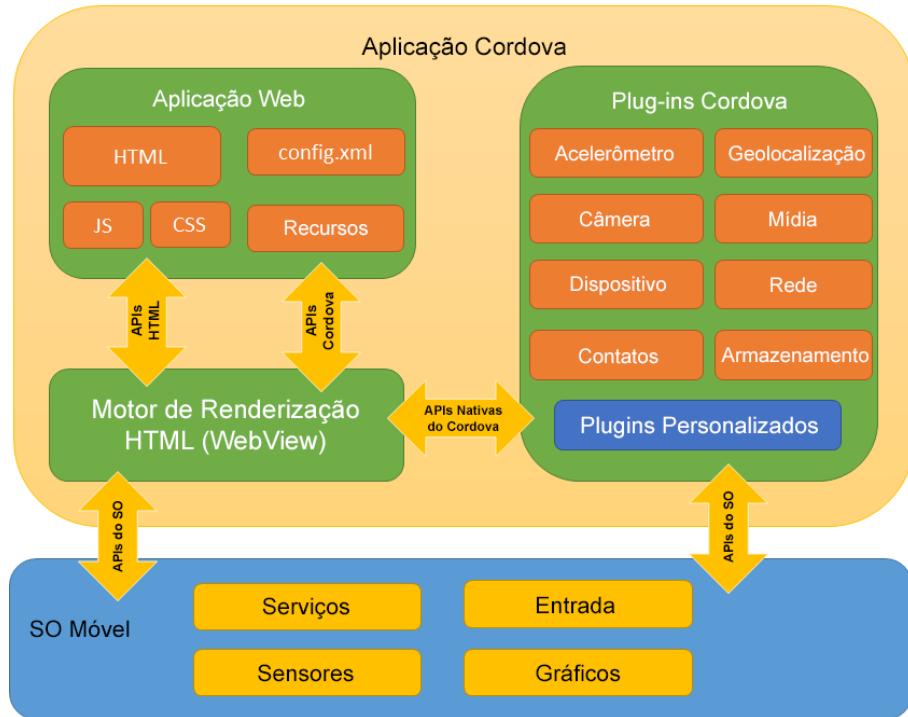


Figura 18 – Arquitetura de uma aplicação *Apache Cordova*.

Fonte: Adaptado de Cordova (2016).

A aplicação é dividida em duas partes principais: a aplicação web e os *plug-ins*. Na parte da aplicação web, se situa o código da aplicação que contém o arquivo `index.html`, os arquivos de folhas de estilo *CSS*, arquivos de script em *JavaScript*, imagens e qualquer outra mídia que se faça necessária para a execução da aplicação. Nesta parte também está localizado um arquivo de extrema importância chamado `config.xml`, o qual detém informações importantes sobre os parâmetros para o correto funcionamento da aplicação.

A outra parte está relacionada aos *plug-ins* utilizados pelo *framework*, sendo estes uma parte de grande importância no ecossistema de uma aplicação *Cordova*. É por intermédio deles que é realizada a comunicação da aplicação com o dispositivo (CORDOVA, 2016).

O projeto *Apache Cordova* mantém uma ampla gama de *plug-ins* chamados de *Core Plugins* que permitem o acesso aos recursos mais básicos do dispositivo, como a câmera, contatos, acelerômetro, dentre outros. Além desses *plug-ins* mantidos pelo projeto, há

também aqueles desenvolvidos por terceiros que permitem estender os recursos acessíveis nos mais diferentes dispositivos e plataformas (CORDOVA, 2016).

### 2.5.2 Ionic Framework

O *Ionic Framework* consiste em um *SDK* que permite ao desenvolvedores criar aplicações móveis híbridas utilizando tecnologias de desenvolvimento web, sendo estas *HTML*, *CSS* e *JavaScript*, possibilitando que haja uma curva de aprendizado menor devido à familiaridade com as tecnologias (IONIC, 2018a).

Como mostra Wilken (2016), o *Ionic* é essencialmente uma combinação dos recursos web para a criação de aplicações híbridas de forma rápida e com componentes elaborados. Utiliza-se do *Angular* para a construção da aplicação web e o *Cordova* para incorporá-la em uma aplicação nativa. A interação entre esses dois *frameworks* definem o fluxo de funcionamento das aplicações criadas com *Ionic*, como mostra a Figura 19

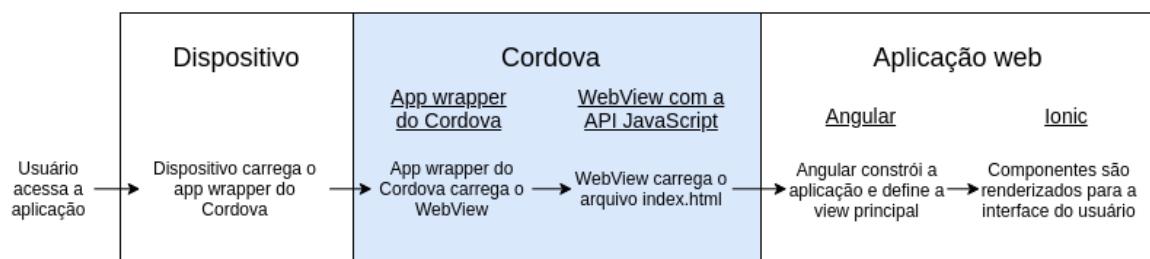


Figura 19 – Estrutura de uma aplicação *Ionic*.

Fonte: Adaptado de Wilken (2016).

Na ação de o usuário acessar a aplicação instalada no dispositivo, este requisita o *wrapper* do *Cordova*. O *wrapper* é uma aplicação responsável por carregar o código da aplicação web e servir como ponte para os recursos nativos do dispositivo e carregar o *WebView*.

Através do *WebView* é disponibilizada a *API* do *JavaScript*, que passa a ter acesso tanto à aplicação quanto ao dispositivo. A aplicação do *Angular* é então executada sobre o *WebView*, ficando esta responsável pelas funcionalidades e pelas rotas que serão acessadas pelo usuário. A parte do *Ionic* é construída sobre a aplicação do *Angular*, fornecendo os recursos de interface e experiência do usuário, além de fornecer utilitários que facilitam desde a pré-visualização à distribuição da aplicação (WILKEN, 2016).

Sendo assim, o *framework* tem em seu objetivo a construção das interfaces das aplicações móveis, visando essencialmente a interação da interface do usuário e o *look and feel* da mesma. Isso implica no fato de que o *Ionic* não é um substituto dos *frameworks* *JavaScript*, como o *Cordova*, por exemplo, mas sim para ser utilizado juntamente com este. O *Ionic* utiliza dos recursos do *framework* *Cordova* para gerar e executar as aplicações

nos dispositivos móveis e também para acessar os recursos destes aparelhos através dos *plug-ins* do *framework* (IONIC, 2018a).

De forma similar à *Angular CLI* descrita anteriormente, toda manipulação e execução das aplicações construídas utilizando o *Ionic* são realizadas por meio da *Ionic CLI*, de modo que sejam permitidas ações como iniciar, executar e construir uma aplicação, por exemplo (IONIC, 2018a).

A *Ionic CLI* também é instalada através da linha de comando e é dependente do *Node.js* na versão 6 *LTS* (*Long Term Support*, Suporte de Longo Prazo) ou superior e também do *NPM* em sua versão 3 ou superior. Sua instalação é feita através do comando `npm install -g ionic@latest`, onde o parâmetro `-g` indica que a instalação será global e poderá ser acessada de qualquer diretório (IONIC, 2018b).

### 3 Trabalhos Relacionados

Este capítulo aborda os trabalhos relacionados a este aqui desenvolvido, relacionando os pontos comuns e diferentes entre eles.

#### 3.1 Aplicativo para o Restaurante Universitário da Universidade Federal do Rio de Janeiro (*BandejãoRU*)

Desenvolvido para *Android* por Souza (2015) e disponibilizado para download na *Google Play*, o aplicativo *BandejãoRU* tem um visual bem simples e objetivo, mostrando na tela inicial os cardápios para os horários do dia, como pode ser observado na Figura 20.

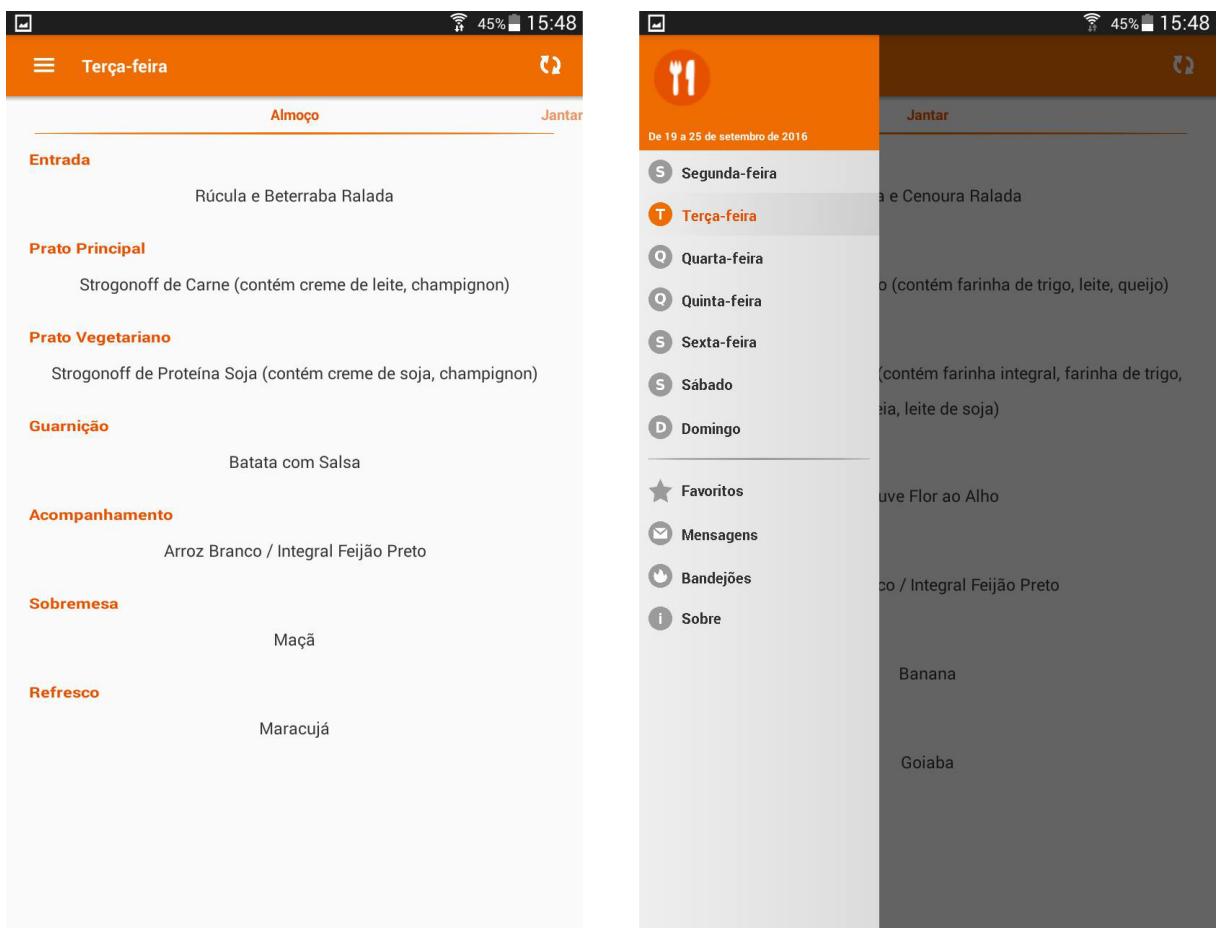


Figura 20 – Tela inicial e menu lateral do aplicativo *BandejãoUFRJ*.

Fonte: Souza (2015).

A aplicação apresenta também um menu lateral que pode ser acessado ao arrastar o dedo na tela do dispositivo (Figura 20). Nesse menu é mostrado os dias da semana

atual, além de informações gerais como mensagens do aplicativo, a opção de inserir um determinado alimento nos favoritos e ser notificado quando há a presença dele no cardápio, endereços dos RUs e informações do desenvolvedor. O aplicativo exibe notificações sobre o alimento contido nos favoritos quando o mesmo será oferecido no dia em questão.

O aplicativo desenvolvido também permite que sejam visualizados os cardápios para cada dia da semana, porém é possível selecionar qualquer data passada ou futura. Entretanto, não conta com a opção de favoritos definida pelo usuário, podendo ser considerado para alguma alteração futura. Apesar de o aplicativo *Bandejão UFRJ* trazer alguns detalhes a mais sobre determinados pratos, o mesmo não contém imagens ou mesmo informações mais detalhadas, como descrição ou informações dietéticas, o que torna esse fator um diferencial do trabalho aqui desenvolvido.

Devido ao fato de ausência de publicação científica a respeito do aplicativo acima discutido, não há muitas informações divulgadas a respeito do desenvolvimento do mesmo e, apesar de aparentemente parecer uma aplicação desenvolvida nativamente, não é possível afirmar com exatidão se fora mesmo esse o método adotado, inviabilizando assim fazer um paralelo com o método adotado para o desenvolvimento deste trabalho.

### 3.2 Aplicativo para o Restaurante Universitário da Universidade Federal de Uberlândia (*RU-UFGU*)

Dantas (2016) criou o aplicativo *RU-UFGU*, desenvolvido para o sistema operacional *Android* e disponibilizado para download na *Google Play*. O aplicativo atende os refeitórios universitários dos campi da Universidade Federal de Uberlândia. A Figura 21 apresenta a tela inicial do aplicativo.

O objetivo do aplicativo *RU-UFGU* é listar os refeitórios daquela universidade, mostrando os respectivos cardápios dos alimentos que serão fornecidos no almoço e jantar durante a semana em questão, como pode ser observado na sequência ilustrada na Figura 22.

Além de mostrar os cardápios, o aplicativo também fornece informações gerais sobre os RUs, como endereços, horários de funcionamento, estatísticas gerais e notificações sobre o que será servido. O aplicativo também conta com propaganda, cujo objetivo é obter fundos a serem doados para a APA – Associação de Proteção Animal<sup>1</sup>, e uma funcionalidade de alerta para os ditos “pratos delícia da semana” que informa ao usuário quando haverá refeição especial durante a semana, como lasanha e estrogonofe, por exemplo.

Como informado pelo autor do aplicativo *RU-UFGU*, o mesmo foi desenvolvido de

<sup>1</sup> <http://www.apauberlandia.org.br/>

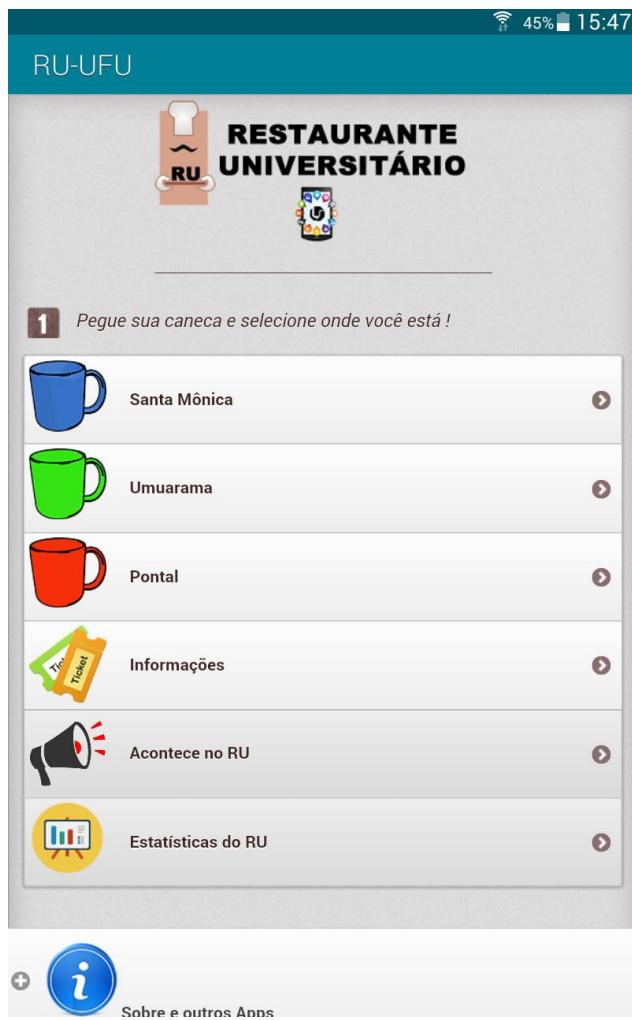


Figura 21 – Tela inicial do aplicativo RU-UFU.

Fonte: Dantas (2016).

forma nativa para *Android*, usando a linguagem *Java* e a *IDE Eclipse* por meio do *plug-in Android Development Tools (ADT)*. É importante ressaltar também que tal *plug-in* não tem mais suporte oficial, devido ao fato que a *IDE (Integrated Development Environment, Ambiente de Desenvolvimento Integrado)* oficial de desenvolvimento para *Android* é agora a *Android Studio* (EASON, 2015).

O trabalho desenvolvido conta com pontos similares e também pontos diferentes do supracitado. Ele também conta com informações gerais sobre o RU, mostra notificações para checagem do cardápio e lista os alimentos a serem oferecidos.

Entretanto, o aplicativo conta com maiores detalhes, mostrando informações visuais e descrições sobre os alimentos, por meio de fotos e textos informativos, além de um link externo para que o usuário obtenha mais informações a respeito do que é servido, caso haja interesse em conhecer melhor um determinado alimento.

Outro ponto que difere os dois aplicativos é que este é desenvolvido de forma híbrida e multiplataforma por meio da *MEAN Stack* e do *Ionic*, utilizando tecnologias

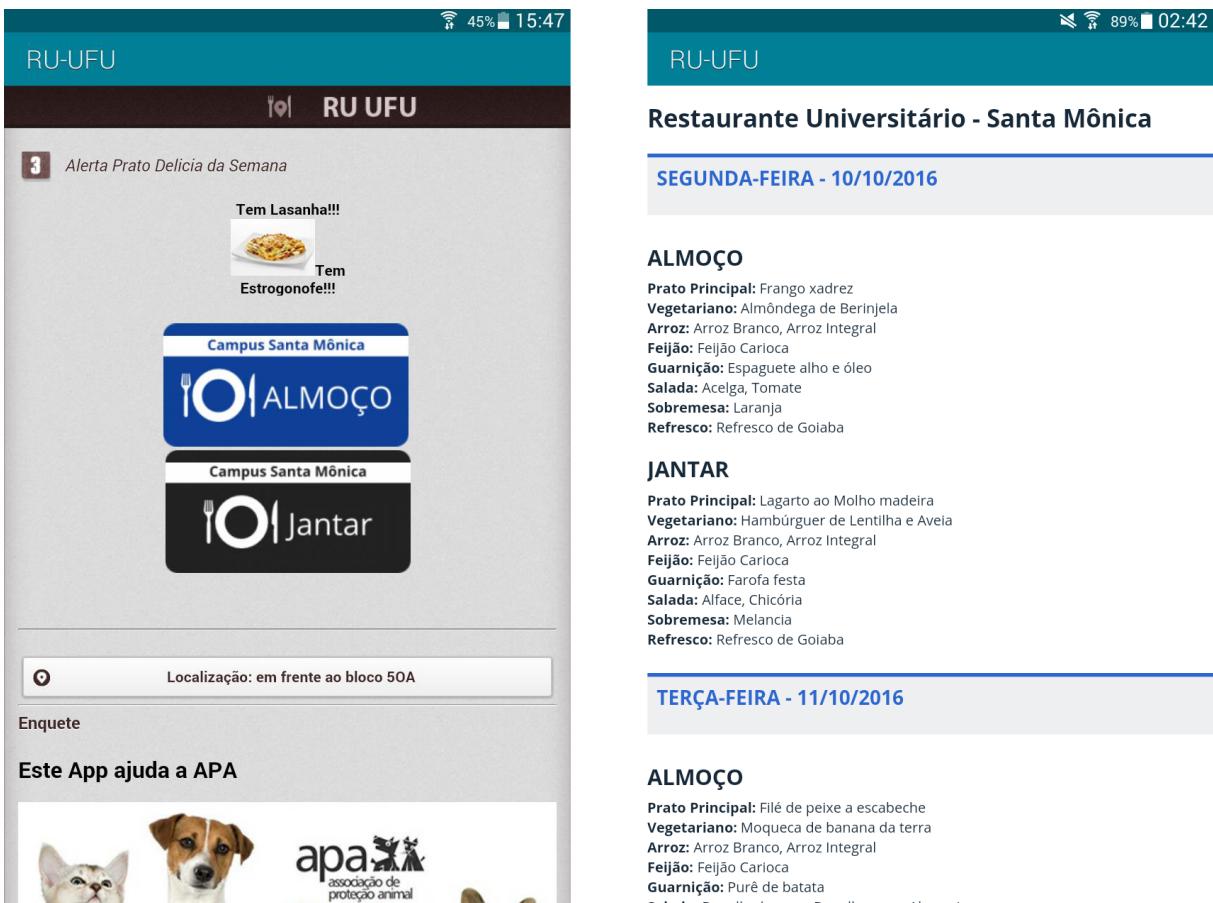


Figura 22 – Telas de seleção de horário da refeição e listagem dos alimentos.

Fonte: Dantas (2016).

de desenvolvimento web *HTML5*, *CSS3* e *JavaScript* e integrando à plataforma móvel através dos *plug-ins* do *Cordova*, tendo uma maior amplitude de dispositivos atendidos.

### 3.3 Aplicação web para o Refeitório Universitário da Universidade Federal de Viçosa - Campus Rio Paranaíba (*Consulta RU/SISRU UFV-CRP*)

Desenvolvida pela UFV (2017), consiste em uma aplicação web dividida em dois subsistemas: o de consulta aos cardápios, denominado *SISRU UFV-CRP*<sup>2</sup>, e o de acesso aos dados de usuário, denominado *Consulta RU*<sup>3</sup>.

O subsistema *SISRU UFV-CRP* é utilizado para gerenciamento dos cardápios e disponibilização do mesmo. Este possui a parte de gerenciamento que é restrita ao gestores da instituição e a parte pública acessível de forma aberta. Na parte pública da aplicação

<sup>2</sup> <https://consultaru.crp.ufv.br/login/>

<sup>3</sup> <https://sisru.crp.ufv.br/>

são apresentados os cardápios cadastrados para as datas que podem ser selecionadas em um calendário apresentado na página inicial.

Já o subsistema *Consulta RU* permite aos usuários do RU a consultar seus dados, como saldo e extrato, além de conter links para os sistemas de visualização de cardápios e de adição de créditos. A Figura 23 ilustra as interfaces da aplicação.

**(a) Tela do subsistema público do *Consulta RU*.**



**(b) Tela de login do subsistema *SISRU UFV-CRP*.**



**(c) Tela de início do subsistema *SISRU UFV-CRP*.**



Figura 23 – Aplicação do *Consulta RU/SISRU UFV-CRP*.

Fonte: UFV (2017)

A aplicação foi desenvolvida somente para a plataforma web, com a utilização da linguagem *PHP*, diferenciando-se da aplicação apresentada neste trabalho que é desenvolvida de forma híbrida.

Por ter acesso amplo aos recursos de informação internos da universidade, a aplicação utiliza de dados específicos do aluno que acessa a mesma, como o número de matrícula e nome completo, por exemplo, além de mostrar o saldo e extrato de uso de créditos no RU da instituição.

O trabalho aqui desenvolvido foca essencialmente na disponibilização de uma alternativa de acesso para os usuários do RU, tendo como diferencial em apresentar informações adicionais sobre os alimentos, permitir acessos pelos usuários aos cardápios em múltiplas

plataformas, possibilitar o feedback em relação às refeições oferecidas, integrar anúncios com objetivos sociais e notificar os usuários seja pela web ou dispositivos móveis.

### 3.4 Aplicativo *Android* para a Universidade Federal de Viçosa (*UFV mobile*)

Desenvolvido pelo Departamento de Tecnologia da Informação - DTI (2018) com a colaboração do Laboratório de Desenvolvimento para Dispositivos Móveis (LABD2M), ambos pertencentes à UFV, o *UFV mobile* é um aplicativo para dispositivos móveis disponível para o sistema operacional *Android*.

Seu ponto principal é a integração de alguns dos principais serviços e sistemas disponibilizados pela universidade que são voltados para os estudantes de forma centralizada e unificada em uma mesma aplicação. A Figura 24 ilustra a interface da aplicação.



(a) Tela de início da aplicação.

(b) Seção da aplicação referente ao Refeitório Universitário.

Figura 24 – Aplicativo *UFV mobile*.

Fonte: DTI (2018)

Um ponto diferente entre os trabalhos é que a aplicação foi desenvolvida de forma nativa para a plataforma *Android*, diferenciando-se deste trabalho, que adota a abordagem híbrida.

De forma similar à aplicação web descrita anteriormente, o *UFV mobile* também tem acesso amplo aos recursos de informação internos da universidade. Isso permite ao aplicativo utilizar os dados específicos do aluno, como o número de Cadastro de Pessoa Física (CPF), nome completo, carga horária e afins. Por contar com uma seção voltada ao RU, o mesmo também conta com a apresentação de saldo e extrato de uso do mesmo.

Como descrito anteriormente no paralelo feito com a outra aplicação, o trabalho aqui desenvolvido tem como foco o fornecimento de uma alternativa de acesso para os usuários do RU da UFV no Campus de Rio Paranaíba, tendo como diferencial a apresentação de informações adicionais sobre os alimentos, a possibilidade de acesso pelos usuários aos cardápios em múltiplas plataformas, anúncios com finalidade social e a viabilização de meio de feedback para com relação às refeições oferecidas.

## 4 Métodos

Neste capítulo são abordados os métodos que foram utilizados para realização do trabalho. O desenvolvimento do projeto consistiu em três etapas dependentes entre si, como pode ilustrar a Figura 25.

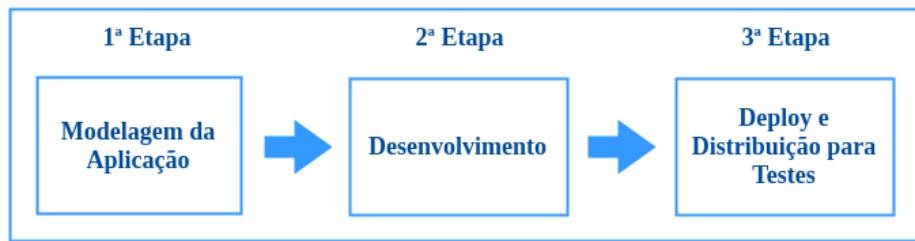


Figura 25 – Etapas de desenvolvimento do trabalho.

Fonte: Autoria própria.

A primeira etapa consistiu em um levantamento dos principais meios existentes pelos quais são divulgados os cardápios pela universidade. Foram então levados em consideração os arquivos *PDF* disponibilizados no site da instituição para compreender a estrutura na qual são disponibilizados os cardápios, para então serem tomadas as decisões em relação aos requisitos e posteriormente a modelagem da aplicação.

Após serem analisados os documentos e tendo sido alinhado com os objetivos propostos pelo trabalho, foram levantados os requisitos funcionais que deveriam existir no sistema e que foram utilizados posteriormente na modelagem do mesmo. O documento simplificado de requisitos pode ser visualizado no Apêndice A.

Na modelagem foi utilizado o diagrama de caso, presente na especificação da *UML* (*Unified Modeling Language*, Linguagem de Modelagem Unificada). Como definem Booch et al. (1998) em seu trabalho de especificação da *UML*, os diagramas de caso de uso são utilizados para modelar o aspecto dinâmico de um sistema. Utiliza-se de atores e relacionamentos entre os mesmos para permitir a visualização e a compreensão do sistema como um todo juntamente com os subsistemas e classes constituintes, apresentando uma visão externa de como esses elementos podem ser utilizados dentro de um determinado contexto e quais são os seus comportamentos.

Definidos os requisitos e criado o modelo da aplicação, iniciou-se então a etapa de desenvolvimento. Para esta finalidade foram utilizados os seguintes recursos para a configuração do ambiente de desenvolvimento:

- **Ferramentas e serviços**

- **Sistema Operacional:** Ubuntu 16.04 LTS (Long Term Support) (<https://www.ubuntu.com/download/desktop>)
- **IDEs:** JetBrains WebStorm (<https://www.jetbrains.com/webstorm/>) e Visual Studio Code (<https://code.visualstudio.com/>)
- **Hospedagem do projeto:** Atlassian Bitbucket (<https://bitbucket.org/>)
- **Sistema de controle de versão:** Git (<https://git-scm.com/>)
- **Hospedagem da aplicação:** Umbler (<https://www.umbler.com/>)
- **Hospedagem do banco de dados:** Umbler (<https://www.umbler.com/>)

- **Software**

- **Gerenciamento de dependências:** NPM 3.10.10 (<https://www.npmjs.com>)
- **Desenvolvimento:**
  - \* MongoDB: v3.4.10 (<https://www.mongodb.com/>)
  - \* Node.js: v6.11.2 LTS (<https://nodejs.org>)
  - \* Yeoman: 2.0.0 (<http://yeoman.io/>)
  - \* Angular CLI: 1.7.3 (<https://cli.angular.io/>)
  - \* Ionic CLI: 3.20.0 (<https://ionicframework.com/>)
- **Automação de tarefas:** Gulp 1.4.0 (<https://gulpjs.com/>)

É importante ressaltar que sempre deverá existir um processo *daemon*<sup>1</sup> do *MongoDB* executando na máquina. Este processo pode ser iniciado, no *Ubuntu*, pelo comando `sudo service mongod start` no terminal.

Além dos recursos essenciais informados acima, foram utilizadas inúmeras bibliotecas e dependências dentro dos módulos da aplicação, tornando assim um número elevado de recursos. Neste caso, não serão discorridas sobre cada biblioteca em particular, mas somente daquelas que são essenciais para a compreensão do trabalho aqui apresentado.

Após o desenvolvimento da aplicação, foram geradas as versões prontas por meio dos mecanismos de automação do *build* sendo então enviada para o servidor remoto e a aplicação nativa distribuída para testes de desenvolvimento.

---

<sup>1</sup> Processo que é executado em segundo plano.

# 5 Resultados e Discussões

Neste capítulo são ilustrados os passos e etapas seguidos para a construção do sistema, sendo abordadas as etapas que incluem a modelagem, desenvolvimento e *deploy*<sup>1</sup> da aplicação.

## 5.1 Modelagem

A aplicação desenvolvida neste trabalho, doravante denominada *RU Mobile*, consiste substancialmente de três módulos, o servidor, a aplicação pública e a aplicação administrativa. Os dois módulos de interação por parte dos usuários são o módulo *Público Web/Mobile*, que representa a aplicação híbrida pública apresentada ao usuário final, e o *Módulo Admin*, que é a aplicação onde os usuários administradores gerenciam os cardápios e todos os seus componentes relativos, além dos próprios usuários. As duas aplicações em questão interagem com o servidor via *API*, para realizar buscas e manipular os dados.

Com isso, é possível definir e distinguir o que cada módulo possui de possíveis interações, associadas a seus respectivos atores, assim como mostra o diagrama de caso de uso ilustrado pela Figura 26.

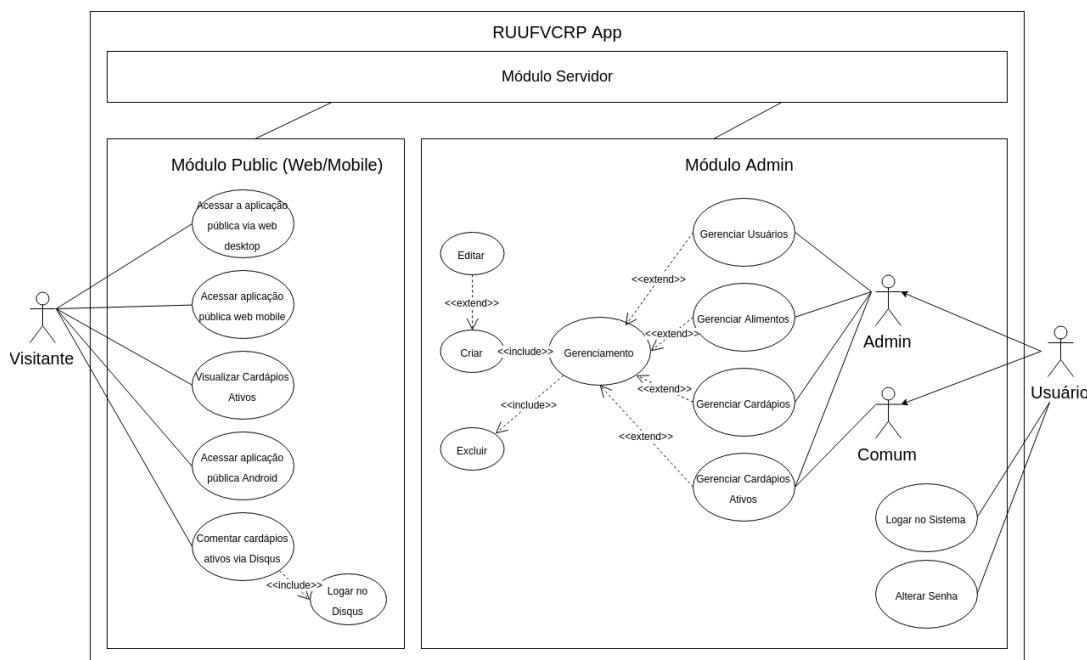


Figura 26 – Diagrama de caso de uso do sistema.

Fonte: Autoria própria.

<sup>1</sup> Processo de distribuição da aplicação, ou seja, enviar para o servidor e disponibilizar para ao público.

No módulo Público Web/Mobile, tem-se o ator **Visitante**, cujas ações possíveis são acessar a aplicação por meio da interface web desktop, web móvel e aplicação *Android*. Isso permite ao usuário visitante ter acesso aos cardápios ativos para a data atual que estejam cadastrados no sistema. Além disso, é possível a este ator comentar nos cardápios ativos, a fim de deixar sua opinião avaliativa ou não sobre os mesmos. Isso se faz possível através da plataforma  *Disqus*<sup>2</sup>, que consiste em um serviço on-line voltado para discussões e integração de comentários em sítios web dos mais diversos tipos e ramos de atuação. Para este módulo do sistema, não são salvos registros na base de dados, logo não existem entidades relacionadas ao visitante que acessa a aplicação.

No Módulo Admin, têm-se dois atores: o **Admin** e o **Usuário**. Ambos são cadastrados no sistema de forma similar, diferenciando-se sumariamente no papel exercido por cada um dentro do mesmo. Esse papel é denominado como *role* (ou um cargo) e representa o nível de permissão dentro da aplicação. Neste caso são definidas as roles de **admin** e **user**.

Essas *roles* têm implicação direta no que cada usuário pode e o que não pode fazer dentro da aplicação, sendo essas ações atreladas às manipulações das entidades presentes neste módulo.

Existem quatro entidades presentes na aplicação, sendo elas: Alimentos, Cardápios, Cardápios Ativos e Usuários. A Figura 27 apresenta o diagrama entidade relacionamento, ilustrando o que cada uma representa dentro do sistema.

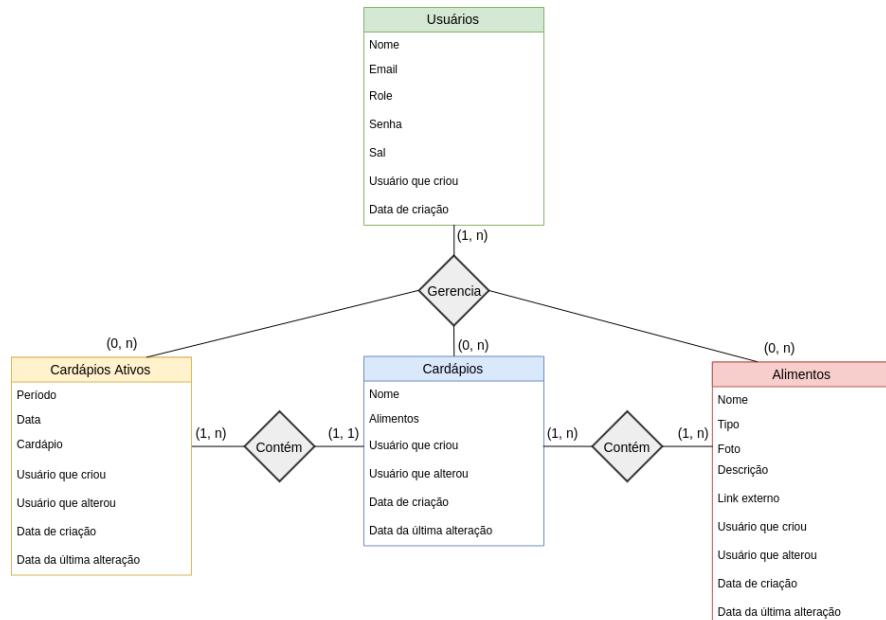


Figura 27 – Diagrama entidade relacionamento da aplicação.

Fonte: Autoria própria.

<sup>2</sup> <https://disqus.com/>

A entidade de Alimentos mantém os dados essenciais para o registro dos alimentos na base do sistema. Esses dados são compostos por: nome, tipo, foto, descrição, link externo. Todos são de caráter obrigatório, à exceção do link externo, por ser entendido que a ausência do mesmo não representa impacto negativo na percepção das informações relacionadas a determinado alimento.

Outro ponto importante é que o tipo do alimento é restrito aos nove tipos disponibilizados pelo Refeitório Universitário da UFV - Campus Rio Paranaíba, sendo eles: prato principal, opção vegetariana, guarnição, arroz, feijão, sopa, salada, suco e sobremesa. A restrição existente na entidade é que só é possível existir um mesmo nome de alimento para um determinado tipo.

Além disso, existem também os dados relacionados à manipulação dos registros cadastrados. Estes dados consistem em: usuário que criou, data que foi criado, usuário que alterou e data que foi alterado, sendo estes dois últimos para casos onde o registro tenha sofrido alguma alteração. Esse dados também estão presentes nas entidades de cardápios e cardápios ativos. Na entidade de usuário existe somente quem criou e quando o fez.

A entidade de Cardápios consiste em uma espécie de *template*, ou modelo, com a finalidade de agilizar o processo de programação de cardápios. A fim de evitar que seja preciso selecionar todos os alimentos no momento que for ativar um cardápio, usa-se, então, um cardápio previamente definido.

O registro de um cardápio possui um nome e um conjunto de alimentos contendo um cada tipo, podendo ser mais de um em caso de saladas. Somente o campo de nome é obrigatório e deve haver pelo menos um alimento presente no cardápio. O nome pode ficar a critério de quem estiver cadastrando, podendo ser um código, um nome que siga algum critério de padronização ou mesmo algum nome mais abrangente. A restrição da entidade é que não é possível existir dois cardápios com o mesmo nome.

A entidade de Cardápios Ativos representa os cardápios programados atualmente no sistema. São estes que o usuário da aplicação do Módulo Público irá obter quando fizer acesso ao mesmo. O registro desta entidade consiste em período ativo, podendo ser almoço ou jantar, data que estará ativo e o cardápio, onde todos os campos são de caráter obrigatório. A restrição da entidade é que só é possível existir um cardápio ativo para um determinado período em uma determinada data.

A entidade de Usuários mantém os usuários cadastrados na aplicação e que possuem acesso à mesma. Os registro de um usuário é composto por: nome, e-mail, que é o critério utilizado para acessar a aplicação, senha, *role* e sal, onde todos os campos são obrigatórios. O sal (ou *salt*, em inglês) é um campo que mantém um dado aleatório para ser utilizado juntamente com a senha de modo a fim de obter proteção adicional de acesso.

O cadastro de usuários é feitos somente por algum outro usuário dentro do sistema, logo não existe confirmação de e-mail ou recuperação de senha, mas somente alteração desta última pelo usuário logado. Esta entidade possui duas restrições, sendo uma delas que um usuário não pode deletar a sua própria conta, garantindo, assim, que sempre haverá pelo menos um usuário com permissão administrativa dentro do sistema e a outra que só é possível existir um usuário cadastrado para cada e-mail.

Com base nisso, é possível observar a relação entre as entidades presentes no sistema. A entidade Alimento é referenciada pela entidade Cardápio e a entidade Cardápio é referenciada pela entidade Cardápios Ativos. Além disso, a entidade Usuários atua como uma proteção de manipulação destas, ou seja, é necessário que tenha acesso ao sistema para cadastrar, editar ou excluir qualquer registro.

Sendo assim, as *roles* têm implicação direta no que cada usuário pode e o que não pode fazer dentro da aplicação. O administrador do sistema, neste caso, possui acesso amplo dentro da aplicação, salvas limitações existentes no modelo, como mostra a Tabela 3:

Tabela 3 – Descrição das ações possíveis ao administrador.

Ação	Descrição
Gerenciar Alimentos	Cadastrar, alterar ou excluir alimentos, desde que estes não estejam associados a pelo menos um cardápio
Gerenciar Cardápios	Cadastrar, alterar ou excluir cardápios, desde que estes não estejam associados a pelo menos um cardápio ativo
Gerenciar Cardápios Ativos	Cadastrar, alterar ou excluir cardápios ativos
Gerenciar Usuários	Cadastrar ou excluir usuários, além de poder alterar a própria senha

Fonte: Autoria própria.

Por outro lado, o usuário comum tem acesso mais restrito, podendo somente:

- Visualizar Alimentos
- Visualizar Cardápios
- Gerenciar Cardápios Ativos
- Alterar a própria senha

A utilização de permissões de acesso se faz presente devido a um fator de controle do sistema para casos onde seja incumbido a um usuário comum a manipular os cardápios ativos. Isso permite que a atribuição seja delegada a outras pessoas além do administrador ou administradores do sistema.

Um outro ponto importante quanto ao sistema são os recursos gerenciados externamente, sendo estes as notificações, as propagandas para web, as propagandas para dispositivos móveis e a análise e moderação de comentários.

As notificações são gerenciadas pelo administrador na plataforma *OneSignal*<sup>3</sup>, tendo a plataforma administrativa própria do serviço sido preterida ao invés de integração via *API* com o Módulo Admin devido à sua robustez e variedade de recursos. As propagandas são gerenciadas através das plataformas *Google Adsense*<sup>4</sup>, para web, e *Google Admob*<sup>5</sup>, para móvel. Os comentários são gerenciados por meio do painel administrativo da plataforma do *Disqus*, sendo possível neste caso analisar os comentários quanto ao seu conteúdo, podendo obter feedback dos usuários e realizar moderação para comentários que sejam considerados inapropriados para serem exibidos na aplicação.

## 5.2 Arquitetura

Do aspecto arquitetural, seguindo o descrito no modelo, o sistema possui em sua composição três patamares distintos, correspondendo ao banco de dados, *back-end* e *front-end*. A Figura 28 representa graficamente a relação entre estes componentes.

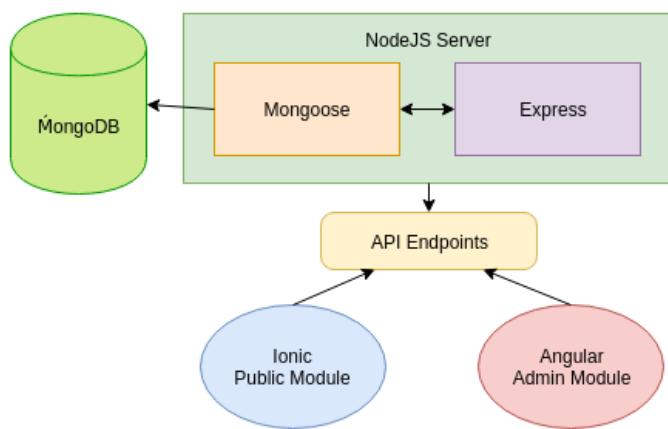


Figura 28 – Arquitetura do sistema.

Fonte: Autoria própria.

Os dados da aplicação são mantidos em um banco de dados do *MongoDB* seguindo as estrutura prevista nas entidades presentes no modelo, sendo estas representadas por meio de esquemas.

A aplicação do servidor é construído utilizando o *Node.js*, sendo responsável por manter a lógica do sistema e interagir com a base do *MongoDB*. O servidor possui em

<sup>3</sup> <https://onesignal.com/>

<sup>4</sup> [https://www.google.com/intl/pt-BR\\_br/adsense/start/](https://www.google.com/intl/pt-BR_br/adsense/start/)

<sup>5</sup> <https://www.google.com.br/admob/>

sua constituição o *Express*, que visa facilitar a criação do mesmo, de modo a agilizar a construção da *API*. O *Mongoose* exerce um papel crucial na composição do servidor, pois é por intermédio do mesmo que é possível construir os esquemas que fazem referência direta à forma que os dados são armazenados na base do *MongoDB*. Além disso, o mesmo serve como um intermediador entre o servidor e o banco de dados, pois ao acessar um *endpoint* na *API* exposta pelo servidor, é através dos métodos do *Mongoose* que se é concretizado o acesso ao *MongoDB*.

A parte da *API* é uma abstração em relação à exposição dos *endpoints* dentro do servidor, pelos quais se fazem possíveis as interações com a aplicação, seja para alterar qualquer registro de cardápios ou adicionar usuários, por exemplo.

Quanto a parte do *front-end* da aplicação, a mesma é dividida em dois módulos, como descrito anteriormente no modelo. O módulo de acesso público é construído utilizando o *Ionic*, à partir do qual é gerada uma aplicação que atenda a todos as plataformas abordadas no trabalho: web, web móvel e *Android*. Já o módulo administrativo é construído utilizando o *Angular* na versão 5, que tem acesso restrito mediante a e-mail e senha e interage com a *API* presente no servidor.

Contudo, se fazem necessárias adaptações para a aplicação web, de forma a garantir que a interface esteja de acordo com a plataforma alvo e a aplicação *Android*, pois é necessário realizar o *build*<sup>6</sup> e *deploy* da aplicação de forma separada, a fim de gerar o arquivo .apk de instalação.

Toda o sistema é hospedado em um mesmo servidor, de modo que seja composta a aplicação completa final. Somente o arquivo instalável *Android* é distribuído por outros meios, porém ainda assim acessando a *API* presente neste mesmo servidor.

### 5.3 *Scaffolding*

Para a construção da aplicação também se faz necessária a estruturação do projeto e a divisão de diretórios e subdiretórios, sendo tal processo chamado de *scaffolding*. A Figura 29 ilustra a estrutura geral da aplicação.

O sistema como um todo tem sua localização na raiz, indicado na imagem por /. Contém seu próprio arquivo *package.json* e o diretório *node\_modules*, indicando que o mesmo contém um projeto do *Node.js* configurado. À partir deste ponto têm-se três outros diretórios que representam distintas partes dentro do sistema, sendo estes a */server*, */client* e */dist*.

O diretório */server*, é onde estão localizados todos os arquivos referentes à aplicação do servidor, como os modelos da base de dados, rotas da *API* e o serviço de autenticação.

<sup>6</sup> Termo utilizado para indicar o processo de compilação do projeto ou a versão após o processo.

/			
package.json e /node_modules			
/server	/client		/dist
- index.js ... 	/public	/admin	/\${ambiente}
	package.json /node_modules	package.json /node_modules	/server /client /admin /public /mobile

Figura 29 – Estrutura de diretórios.

Fonte: Autoria própria.

ção, por exemplo. Já a pasta `/client` contém os os módulos do *front-end*, contendo dois subdiretórios: `/admin` e `/public`, sendo a primeira destinada à aplicação administrativa e segunda à pública web/móvel.

Em ambos os diretórios é possível identificar seus respectivos arquivos `package.json` e os diretórios `node_modules`, indicando que são duas aplicações distintas que estão inseridas no sistema como um todo. Por fim, tem-se o diretório `/dist`, que é destinado a conter as aplicações após o processo de *build*, estando os arquivos já minificados *Processo de redução do tamanho dos arquivos de código por meio da remoção de espaços em branco e renomeação de funções, por exemplo.* e destinados à distribuição. Esse diretório é composto por subdiretórios destinados à cada tipo de ambiente, discutidos no próximo tópico, à exceção do ambiente de desenvolvimento, que utiliza os arquivos do diretório principal.

A estrutura adotada do projeto raiz e `/server`, é similar a um projeto criado utilizando o *Yeoman*<sup>7</sup>, construído à partir de um gerador chamado *Angular Fullstack Generator*<sup>8</sup>. Essa ferramenta gera a estrutura inicial da aplicação contendo diversos recursos, servindo como base de orientação para configuração deste projeto.

Já para a aplicação `/admin` e para a `/public`, são utilizados o *Angular CLI* e a *Ionic CLI*, respectivamente. Esses recursos permitem a geração automatizada da estrutura de um novo projeto, além de possibilitar a inserção facilitada de novos componentes em um projeto já existente.

A Figura 30 ilustra a estrutura completa dos diretórios e subdiretórios presentes no sistema, mostrando os arquivos gerados tanto na aplicação raiz quanto nos módulos que compõe o sistema.

<sup>7</sup> <http://yeoman.io/>

<sup>8</sup> <https://github.com/angular-fullstack/generator-angular-fullstack>



Figura 30 – Definição da estrutura de diretórios do sistema.

Fonte: Autoria própria.

## 5.4 Ambientes

Uma questão importante é a configuração de ambientes da aplicação, que é influenciada pelo destino alvo de *deploy* da mesma. A definição e divisão de ambientes se faz necessária para manter um controle sobre as versões que são liberadas de um determinado estado do sistema. Neste sistema, são definidos quatro ambientes, como definido na Tabela 4.

Tabela 4 – Definição dos ambientes existentes no sistema.

Ambiente	Objetivo
Desenvolvimento (development ou dev)	Utilizado durante o desenvolvimento, com recursos e parâmetros locais
Teste ( <i>test</i> )	Utilizado durante o desenvolvimento, com recursos e parâmetros refletindo as configurações de um servidor remoto
Homologação ( <i>staging</i> )	Utilizado para testes com usuários externos ao sistema, a fim de se obter avaliações e receptividade em relação ao mesmo
Produção ( <i>production</i> ou <i>prod</i> )	Utilizado para o momento que o sistema é liberado para acesso ao público

Fonte: Autoria própria.

Cada ambiente depende de configurações específicas determinadas por variáveis definidas externamente. Essas variáveis são *URLs* de *APIs* de serviços externos e códigos de acesso às mesmos, por exemplo, e são armazenadas em formato *JSON*. A Figura 31 ilustra alguns exemplos de definições de variáveis de ambiente.

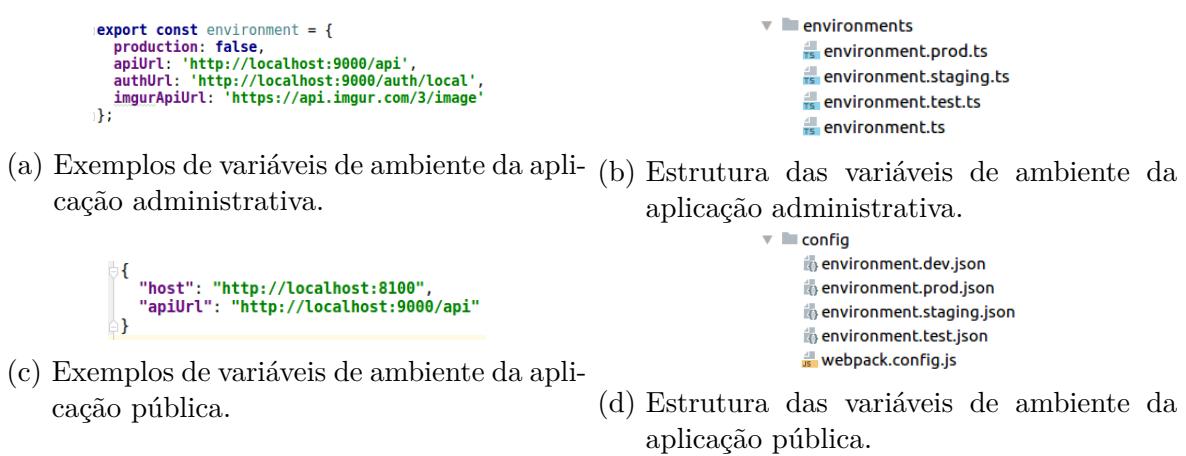


Figura 31 – Definição de variáveis de ambientes.

Fonte: Autoria própria.

Essas variáveis são acessadas no momento de execução e/ou *build* do sistema. Passando uma definição do tipo `NODE_ENV=test`, por exemplo, irá definir que o ambiente

escolhido será o de Teste, logo as variáveis que serão carregadas serão a deste ambiente em específico.

## 5.5 Servidor

A construção do servidor dá-se essencialmente pela configuração dos *middlewares* do *Express* e a conexão com o banco de dados do *MongoDB*. Este processo é simplificado graças à vasta gama de *middlewares* disponíveis para o *Express* e ao *Mongoose*.

Tanto o *Express* quanto o *Mongoose*, são instalados via *NPM* por se tratarem de dependências que serão utilizadas dentro de uma aplicação do *Node.js*. Para tal, é necessário o comando `npm install express mongoose --save`. Com isso, essas dependências serão salvas no arquivo `package.json` e serão baixadas para o diretório `node_modules`.

As configurações do *Express* ficam localizadas em `/server/config/express.js`, onde são definidos os parâmetros de ambiente e *middlewares*, como mostra a Figura 32.

```
const env = app.get('env');

if(env === 'development') {
    app.set('appPath', path.join(config.root, 'client/public/www'));
    app.set('adminPath', path.join(config.root, 'client/admin/dist'));
}

if(env === 'test' || env === 'staging' || env === 'production') {
    app.use(favicon(path.join(config.root, 'client/admin', 'favicon.ico')));
    app.set('appPath', path.join(config.root, 'client/public'));
    app.set('adminPath', path.join(config.root, 'client/admin'));
}

if(env === 'staging' || env === 'production') {
    app.get('*', (req, res, next) => {
        if (req.headers['x-forwarded-proto'] !== 'https') {
            res.redirect("https://" + req.headers.host + req.url);
        } else {
            next();
        }
    });
}

app.use(express.static(app.get('appPath')));
app.use(express.static(app.get('adminPath')));
app.use(morgan('dev'));

app.set('views', `${config.root}/server/views`);
app.engine('html', require('ejs').renderFile);
app.set('view engine', 'html');
app.use(shrinkRay());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(methodOverride());
app.use(cookieParser());
app.use(passport.initialize());
app.use(cors());
```

Figura 32 – Definição das configurações de ambiente e *middlewares* do *Express*.

Fonte: Autoria própria.

O *Express* utiliza os métodos `set()` e `use()` para definir variáveis internas e utilizar os *middlewares* externos, respectivamente. Feito isso, é necessária então a criação da conexão com o *MongoDB*, por meio do *Mongoose*, localizado em `/server/app.js`, como mostra na Figura 33.

```

mongoose.connect(config.mongo.uri, config.mongo.options);
mongoose.connection.on('error', function(err) {
  console.error(`MongoDB connection error: ${err}`);
  process.exit(-1);
});

```

Figura 33 – Criação da conexão com o *MongoDB* por meio do *Mongoose*.

Fonte: Autoria própria.

Neste ponto basta então iniciar o servidor adicionando ao mesmo arquivo o que consta no exemplo da Figura 34.

```

const app = express();
const server = http.createServer(app);
require('../config/express').default(app);
require('../routes').default(app);

function startServer() {
  app.rumobile = server.listen(config.port, config.ip, function() {
    console.log('Express server listening on %d, in %s mode',
      config.port, app.get('env'));
  });
}

```

Figura 34 – Iniciando o servidor com o *Express*.

Fonte: Autoria própria.

Feito isso, é necessário somente executar `npm start` pelo terminal dentro do diretório raiz e o servidor será iniciado.

## 5.6 Base de Dados e API

Para manipular os dados presentes na base do sistema, é importante definir a camada de acesso ao mesmos. Um dos passos é definir os esquemas do *Mongoose*, que refletem a forma como os dados serão estruturados na base. Isso evidencia a maleabilidade dos documentos do *MongoDB*, onde é possível alterar a estrutura dos documentos sempre que for necessário.

Como existem quatro entidades presentes no modelo, os mesmos são mapeados para os esquemas como representado na Figura 35.

É importante notar que os índices informados nos esquemas `FoodSchema`, `MenuSchema` e `ActiveMenuSchema` são referentes às restrições previstas no modelo do sistema, sendo estas de nome e tipo únicos, nome único e data e período únicos, respectivamente.

Criados os esquemas, é necessária a exposição das rotas tanto da *API* quanto as que serão utilizadas para direcionar o usuário corretamente quando o mesmo acessar a aplicação pública ou administrativa.

No arquivo `/server/routes.js` são definidas as rotas principais da aplicação, como é ilustrado na Figura 36.

```

let FoodSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'O nome do alimento deve ser informado.']
  },
  kind: {
    type: String,
    enum: ['SALADA', 'PRATO PRINCIPAL', 'VEGETARIANO', 'GUARNICAO', 'ARROZ',
    'FEIJAO', 'SUCO', 'SOBREMESA', 'SOPA'],
    required: [true, 'O tipo do alimento deve ser informado.']
  },
  photo: {
    type: String,
    required: [true, 'A foto do alimento deve ser informada.']
  },
  desc: {
    type: String,
    required: [true, 'A descrição do alimento deve ser informada.']
  },
  externalLink: {
    type: String,
    required: false
  },
  createdBy: {
    type: String,
    required: true
  },
  createdAt: {
    type: String,
    required: true
  },
  updatedBy: {
    type: String,
    required: false
  },
  updatedAt: {
    type: String,
    required: false
  }
});

FoodSchema.index({name: 1, kind: 1}, {unique: true});

let UserSchema = new Schema({
  name: String,
  email: {
    type: String,
    lowercase: true
  },
  role: {
    type: String,
    default: 'user'
  },
  password: {
    type: String,
  },
  salt: String
});

```

(a) Esquema de usuários.

```

let ActiveMenuSchema = new Schema({
  period: { type: String, required: true },
  date: { type: Date, required: [true, 'Um data deve ser informada.'] },
  menu: {
    type: Schema.Types.ObjectId,
    ref: 'Menu',
    required: [true, 'Um modelo de cardápio deve ser informado']
  },
  createdBy: {
    type: String,
    required: true
  },
  createdAt: {
    type: String,
    required: true
  },
  updatedBy: {
    type: String,
    required: false
  },
  updatedAt: {
    type: String,
    required: false
  }
});

ActiveMenuSchema.index({period: 1, date: 1}, {unique: true});

```

(b) Esquema de alimentos.

```

let MenuSchema = new Schema({
  name: { type: String, required: [true, 'Um nome deve ser informado.']},
  foods: [{ type: Schema.Types.ObjectId,
    ref: 'Food',
    required: [true, 'Os alimentos do cardápio devem ser informados.']
  }],
  createdBy: {
    type: String,
    required: true
  },
  createdAt: {
    type: String,
    required: true
  },
  updatedBy: {
    type: String,
    required: false
  },
  updatedAt: {
    type: String,
    required: false
  }
});

MenuSchema.index({name: 1}, {unique: true});

```

(c) Esquema de cardápios.

(d) Esquema de cardápios ativos.

Figura 35 – Criação dos esquemas do *Mongoose*.

Fonte: Autoria própria.

```

app.use('/api/activeMenus', require('./api/activeMenu'));
app.use('/api/menus', require('./api/menu'));
app.use('/api/foods', require('./api/food'));
app.use('/api/users', require('./api/user'));
app.use('/api/today', require('./api/today'));

app.use('/auth', require('./auth').default);

```

Figura 36 – Rotas principais da aplicação.

Fonte: Autoria própria.

Neste caso, as rotas relacionadas às entidades estão dentro da rota principal /api, além da rota /api/today que é utilizada na aplicação pública para trazer os cardápios ativos para a data atual.

É possível observar que na definição das rotas, é utilizado o `require('nome da rota')`, importando de um arquivo que está localizado em outro diretório dentro da estrutura da API. Essa importação se refere a um arquivo `index.js` que exporta os métodos *HTTP* possíveis para uma determinada rota, como é ilustrado na Figura 37.

```
router.get('/', auth.isAuthenticated(), controller.index);
router.get('/:id', auth.isAuthenticated(), controller.show);
router.post('/', auth.hasRole('admin'), controller.create);
router.put('/:id', auth.hasRole('admin'), controller.upsert);
router.patch('/:id', auth.hasRole('admin'), controller.patch);
router.delete('/:id', auth.hasRole('admin'), controller.destroy);
```

Figura 37 – Métodos *HTTP* das rotas.

Fonte: Autoria própria.

Os métodos *HTTP* têm associado a si uma função presente em um `controller` onde são realizadas as interações com a base de dados, por meio do esquema do *Mongoose*. A Figura 38 representa uma consulta onde são trazidos todos os registros presentes em um determinado documento.

```
export function index(req, res) {
  return SchemaName.find().exec()
    .then(respondWithResult(res))
    .catch(handleError(res));
}
```

Figura 38 – Exemplo de método que trás todos os registros de um documento.

Fonte: Autoria própria.

Além das rotas associadas às entidades, existe a rota /auth, destinada a autenticação por meio do *Passport.js*<sup>9</sup>, verificando se o usuário está logado ou não, assinatura do *token JWT (JSON Web Token)*<sup>10</sup> após o usuário fazer *login* e verificação se o usuário tem permissão necessária para acessar determinada rota.

Com base nisso, é possível exemplificar o processo ocorrido no servidor quando o usuário acessa a aplicação administrativa. O usuário ao acessar a página de *login* e logar com sucesso, é criado um *token*<sup>11</sup> *JWT* que é armazenado no *Local Storage*<sup>12</sup> do usuário. Este *token* então será utilizado sempre que for feita uma requisição a um *endpoint* da API, sendo passado no cabeçalho da requisição.

As verificações no momento de acesso a algum dos *endpoints* são baseados no tipo da requisição realizada. Se esta for do tipo *GET*, passará então pelo método de

<sup>9</sup> <http://www.passportjs.org>

<sup>10</sup> <https://jwt.io/>

<sup>11</sup> Consiste em um *hash* criptografado que indica se o usuário tem a permissão para acessar o sistema ou não.

<sup>12</sup> Armazenamento de recursos no navegador do cliente.

autenticação denominado `isAuthenticated()`. Se a requisição for do tipo onde é passado um corpo na mesma, como *POST* ou *PUT*, por exemplo, é necessária a verificação `hasRole('admin')`, indicando que ação demanda uma permissão do tipo administrativa. Essa verificação garante que somente um usuário do tipo administrador possa manipular a base, seja para criar ou alterar algum registro existente.

Por fim, existem também as definições das rotas estáticas, que indicam os caminhos para a aplicação pública e administrativa, e as que definem todas as rotas não permitidas. Neste segundo cenário, quando o usuário ou aplicação cliente tentar acessar a rota `/api` ou as rotas de *assets*<sup>13</sup>, por exemplo, o mesmo será encaminhado para uma página de erro `404`, indicando que o conteúdo não existe. Todas as demais rotas são redirecionadas para a aplicação pública. A Figura 39 mostra estas definições.

```
// Todas as rotas não definidas ou de assets devem retornar 404
app.route('/:url(api|auth|components|app|bower_components|assets)/*')
  .get(errors[404]);

// Rotas da aplicação administrativa
app.route('/admin')
  .get((req, res) => {
    res.sendFile(path.resolve(`${app.get('adminPath')}/index.html`));
  });
app.route('/admin/*')
  .get((req, res) => {
    res.sendFile(path.resolve(`${app.get('adminPath')}/index.html`));
  });

// Todas as outras rotas deve redirecionar para a aplicação pública
app.route('*')
  .get((req, res) => {
    res.sendFile(path.resolve(`${app.get('appPath')}/index.html`));
  });
}
```

Figura 39 – Demais roteamentos presentes na aplicação do servidor.

Fonte: Autoria própria.

Com isso, a aplicação do *back-end* está concluída e com a *API* exposta para ser utilizada nas aplicações do *front-end*.

## 5.7 Aplicação Administrativa

O próximo ponto na construção do sistema é a criação da aplicação administrativa, que permite que sejam criados e gerenciados os dados para serem acessados pela aplicação pública e que são entregue ao usuário visitante.

A aplicação administrativa é construída utilizando *Angular* na versão 5 com uso da *Angular CLI*, tendo como base o projeto de código aberto *Start Angular*<sup>14</sup> para a sua estruturação e configuração de alguns componentes. Seu acesso é por meio da rota contendo o link principal do servidor acrescido de `/admin`, indicando que está acessando esta aplicação em específico.

<sup>13</sup> Recursos estáticos da aplicação, como imagens e fontes tipográficas, por exemplo.

<sup>14</sup> <https://github.com/start-angular/SB-Admin-BS4-Angular-6>

Ao acessar a primeira vez a aplicação administrativa, tem-se a tela de *login* contendo os campos de e-mail e senha, como mostra a Figura 40.



Figura 40 – Tela de *login* da aplicação administrativa.

Fonte: Autoria própria.

Ao tentar logar com e-mail ou senha errados ou mesmo algum erro no servidor, uma mensagem aparece no canto superior da tela informando do ocorrido. Essas notificações são providas pela biblioteca *Toastr*<sup>15</sup> e são utilizadas nas validações por toda a aplicação. A Figura 41 mostra algumas dessas notificações.

Após logar na aplicação, o usuário é redirecionado para a página inicial, denominada de *Dashboard*, como ilustra a Figura 42.

A estrutura visual da aplicação é dividida em três componentes principais: o cabeçalho, o menu lateral e o corpo, onde são renderizadas as rotas dos componentes. O

<sup>15</sup> <https://github.com/Foxandxss/angular-toastr>

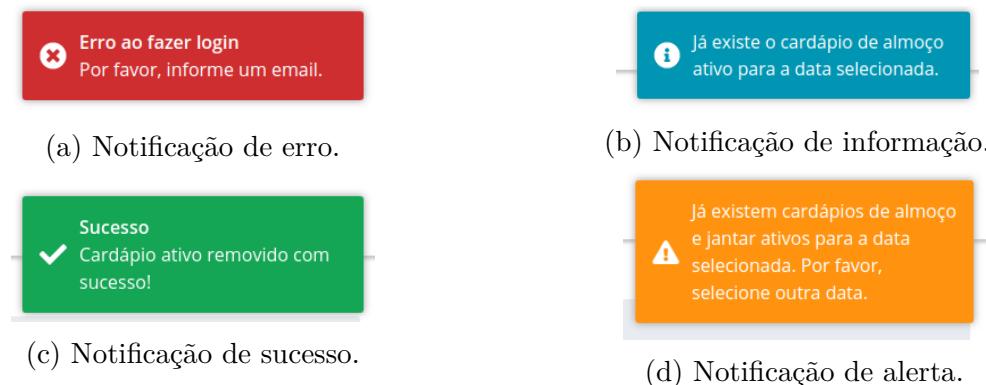


Figura 41 – Exemplos de notificações de validações.

Fonte: Autoria própria.

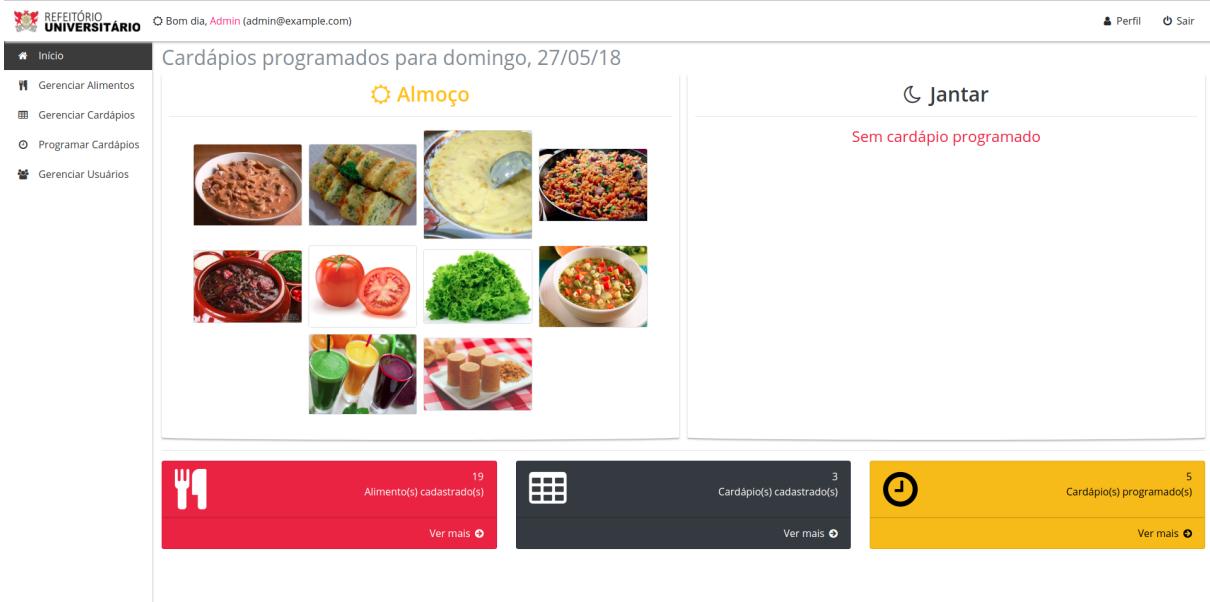


Figura 42 – Página *Dashboard* de acordo com a visão do administrador.

Fonte: Autoria própria.

cabeçalho contém informações pertinentes ao usuário, como nome e e-mail e acesso a uma página de perfil que permite a alteração da senha, além da opção de sair da aplicação.

A navegação é viabilizada pelo menu lateral localizado na parte esquerda, contendo os links de acesso às rotas da aplicação. Os nomes presentes nas rotas e quais rotas são acessíveis depende da permissão do usuário. O *admin* tem a opção de gerenciar todos os componentes, incluindo os usuários, e ao usuário somente visualizar alimentos, visualizar cardápios e gerenciar cardápios ativos.

No corpo da página do *Dashboard* são exibidos os cardápios ativos para a data atual ou, caso tenha passado o horário de jantar, os cardápios ativos do dia seguinte. Na parte inferior estão presentes as contagens de alimentos, cardápios e cardápios ativos atualmente cadastrados no sistema.

No módulo de Gerenciamento de Alimentos, tem-se primeiro a página de listagem dos mesmos e então o módulo de cadastro, que é utilizado tanto para cadastrar quanto pra editar, como mostra a Figura 43.

Em todos os módulos há a presença do campo de busca para filtrar os elementos presentes na tabela. Já o botão de cadastro no canto superior direito acima da tabela e o botão de editar e deletar presentes em cada linha das tabelas estão disponíveis somente para o administrador.

Na tela de cadastro são dispostos os elementos presentes na entidade de alimentos, incluindo uma imagem que é enviada via *API* para o *Imgur*<sup>16</sup>, um serviço de hospeda-

<sup>16</sup> <https://imgur.com/>

(a) Tela de listagem de alimentos.

(b) Tela de cadastro/alteração de alimentos.

Figura 43 – Módulo de Gerenciamento de Alimentos.

Fonte: Autoria própria.

gem de imagens gratuito, que retorna a *URL* que é salva na aplicação. Isso garante um controle maior sobre os links das imagens, além de poupar espaço em disco deixando essa funcionalidade com um serviço à parte que garante alta disponibilidade.

O módulo de Gerenciamento de Cardápios segue o mesmo padrão descrito anteriormente, como ilustra a Figura 44.

Na tela de cadastro e alteração são dispostos os elementos que compõem os cardápios, sendo o nome obrigatório e que tenha ao menos um alimento informado. A seleção de saladas é o único campo onde é possível de informar mais de um componente.

Seguindo o mesmo padrão, tem-se o módulo de Gerenciamento de Cardápios Ativos, ilustrado pela Figura 45.

(a) Tela de listagem de cardápios.

(b) Tela de cadastro/alteração de cardápios.

Figura 44 – Módulo de Gerenciamento de Cardápios.

Fonte: Autoria própria.

Na tela de cadastro e alteração de cardápio ativo é possível selecionar os elementos presentes no mesmo. Ao selecionar uma data é apresentado na tela um calendário para a seleção da mesma e ao fazer isso uma chamada é feita ao serviço verificando quais cardápios ativos já existem para aquele dia e se é possível cadastrar no mesmo. Ao alterar um cardápio ativo somente o cardápio pode ser alterado, mantendo assim a mesma data e o mesmo período.

Para o módulo de Gerenciamento de Usuários existem algumas diferenças, onde só é possível cadastrar ou remover um usuário, como mostra a Figura 46. Ao remover um usuário é mostrado um *modal*<sup>17</sup> perguntando ao usuário que está realizando a ação se o mesmo tem certeza disto. Caso esteja tentando remover a própria conta é então mostrada uma notificação informando não ser possível realizar tal procedimento.

<sup>17</sup> Tela sobresalente à tela principal que cobre a mesma exibindo o conteúdo solicitado.

Data	Período	Alimentos	Criado Por	Criado Em	Atualizado Por	Atualizado Em	Oções
quinta-feira, 31/05/18	Almoço	Strogonofe de Carne, Torta de Legumes, Polenta, Arroz Carreteiro, Feijoada, Tomate, Alface, Sopa de Legumes, Suco Variado, Paçoca	Admin	quinta-feira, 31/05/18	Admin	quinta-feira, 31/05/18	
quinta-feira, 31/05/18	Jantar	Isca Suína, Torta de Legumes, Creme de Milho, Arroz Simples, Feijoada, Tomate, Alface, Sopa de Legumes, Suco Variado, Paçoca	Admin	quinta-feira, 31/05/18	Admin	quinta-feira, 31/05/18	
sexta-feira, 01/06/18	Almoço	Strogonofe de Carne, Torta de Legumes, Polenta, Arroz Carreteiro, Feijoada, Tomate, Alface, Sopa de Legumes, Suco Variado, Paçoca	Admin	quinta-feira, 31/05/18	Admin	quinta-feira, 31/05/18	

(a) Tela de listagem de cardápios ativos.

\* Data  
2018-06-05

\* Cardápio  
CR-38796

Periodo  
Almoço

Alimentos presentes no cardápio  
Strogonofe de Carne, Torta de Legumes, Polenta, Arroz Carreteiro, Feijoada, Tomate, Alface, Sopa de Legumes, Suco Variado, Paçoca

(b) Tela de cadastro/alteração de cardápios ativos.

Figura 45 – Módulo de Gerenciamento de Cardápios Ativos.

Fonte: Autoria própria.

No caso do gerenciamento de usuários, a única alteração possível é a da senha, que pode ser feita ao clicar no botão **Perfil**, como ilustra a Figura 47, sendo possível realizar essa ação em qualquer parte da aplicação devido ao mesmo estar presente no cabeçalho. Existe também a restrição de que a senha tenha pelo menos 6 caracteres.

A tela de perfil é acionada por um *modal*, sendo permitido ao usuário somente alterar a própria senha, como descrito previamente.

Com isso, tem-se a aplicação administrativa definida e todas as alterações relacionadas aos cardápios são refletidas na aplicação pública, tanto web, quanto na aplicação móvel, por estar manipulando diretamente o serviço.

Nome	Email	Role	Criado Por	Criado Em	Opções
Admin	admin@example.com	Admin			
RUUFVCRP	rufvcrp@gmail.com	Admin			
Test User	test@example.com	Usuário			

3 total

(a) Tela de listagem de usuários.

\* Nome:

\* Email:

\* Role:

\* Senha (mínimo 6 caracteres):

\* Confirmar Senha:

\* Campo de preenchimento obrigatório.

**Cadastrar Usuário** **Voltar**

(b) Tela de cadastro de usuários.

Figura 46 – Módulo de Gerenciamento de Usuários.

Fonte: Autoria própria.

**Perfil**

**Admin**  
admin@example.com  
Admin

Alteração de Senha

\* Senha Antiga:

\* Nova Senha (mínimo 6 caracteres):

\* Confirmar Nova Senha:

\* Campo de preenchimento obrigatório.

**Alterar Senha** **Cancelar**

Figura 47 – Tela de perfil do usuário.

Fonte: Autoria própria.

## 5.8 Aplicação Pública

A construção e manipulação da aplicação pública é feita por meio da *Ionic CLI*, sendo o projeto iniciado através do comando `ionic start`. Este cria a estrutura inicial da aplicação, baseada no modelo definido, sendo neste caso um modelo de *tabs*, ou abas.

Por ser uma aplicação híbrida, a mesma pode ser acessada pela aplicação instalável no sistema *Android*; navegador móvel no *Android*, *iOS*, *Windows*, dentre outros; e navegador desktop. A diferença no aspecto e apresentação em cada plataforma são básicas, alterando essencialmente a disposição dos elementos em relação às versões móvel e desktop.

A estrutura da aplicação é dividida em três partes principais: o cardápio de almoço, cardápio de jantar e informações. Esses três componentes são acessados diretamente pelas abas da aplicação, como mostra a Figura 48. Para a simulação nativa no *iOS* foi utilizado o *Ionic DevApp*<sup>18</sup>.

Os alimentos são apresentados por meio de um componente `ion-card`, contendo nome, título e um link para informações relacionadas ao mesmo.

Além dos componentes principais existem outros três componentes secundários que permitem ao usuário interagir com o sistema: informações sobre o alimento, comentários e calendário dos cardápios para outros dias.

O componente de informações é acessado ao clicar no link `Mais informações` localizado logo após o título e traz a descrição e link externo sobre o alimento, como ilustra a Figura 49.

<sup>18</sup> <https://ionicframework.com/docs/pro/devapp/>

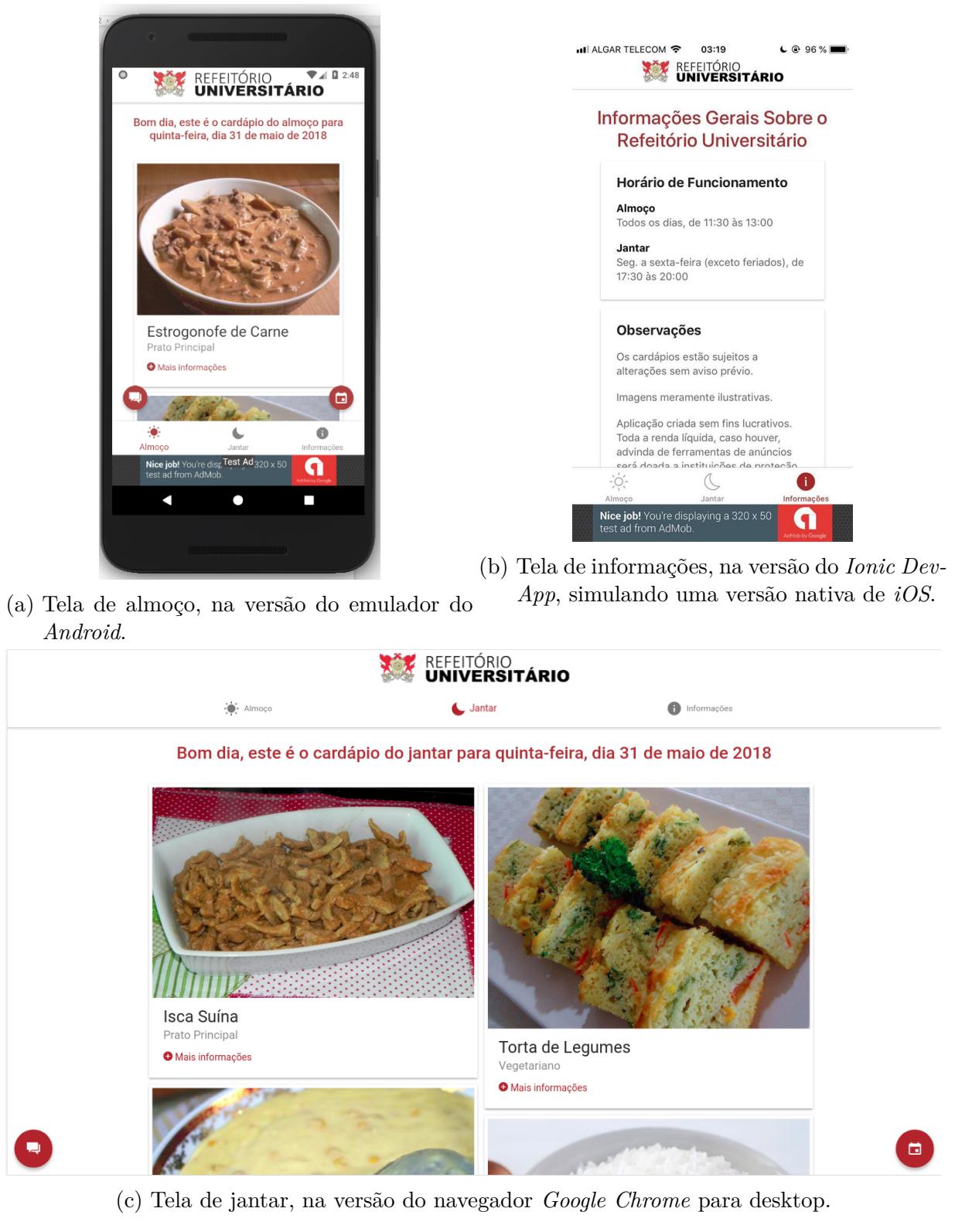


Figura 48 – Estrutura visual da aplicação pública/móvel mostrada em diferentes versões.

Fonte: Autoria própria.

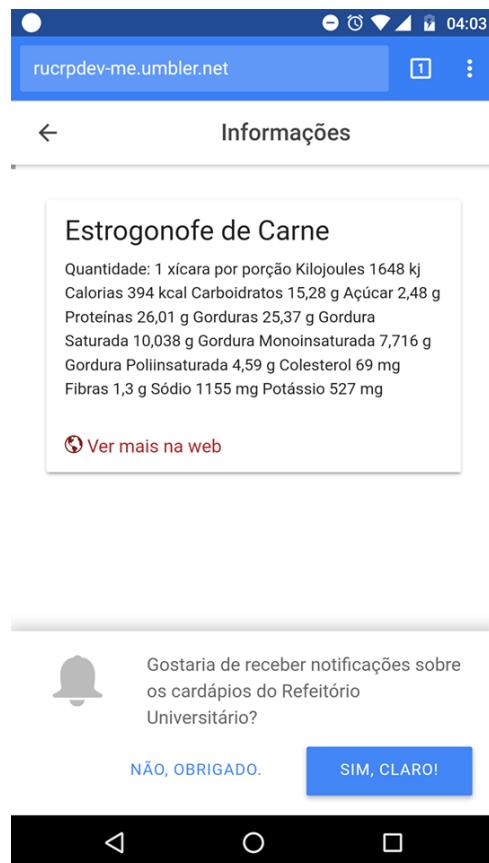


Figura 49 – Tela de informações sobre o alimento, na versão do navegador *Google Chrome* para a plataforma *Android*.

Fonte: Autoria própria.

O componente de comentários é acessado pelo botão *FAB* (*Floating Action Button*, Botão Flutuante de Ação) presente no canto superior acima da barra de abas na versão móvel e no canto inferior esquerdo da tela nas versões desktop. Esse componente permite que sejam adicionados comentários relativos aos cardápios apresentados, utilizando a plataforma do  *Disqus*, como mostra a Figura 50.

É importante ressaltar que a integração de comentários é realizada pela biblioteca *NGX Disqus*<sup>19</sup>, voltada para *Angular* e não especificamente para aplicações do *Ionic*. Este recurso quando acessado pela aplicação nativa do *Android* pode apresentar instabilidades no momento que o usuário tenta logar na plataforma do  *Disqus*, pois a ação de *login* não consegue retornar automaticamente para a aplicação. Sendo assim, na versão nativa o recurso de comentários fica limitado a visualizar os comentários, sendo a opção de comentar limitada às plataformas web móvel e desktop. Todos os comentários podem ser moderados e analisados pelo painel administrativo da plataforma do  *Disqus*.

Por fim, o componente de calendário pode acessado pelo *FAB* localizado no canto inferior direito acima das abas na versão móvel e no canto inferior direito da tela na

<sup>19</sup> <https://github.com/MurhafSousli/ngx-disqus>



Figura 50 – Tela de comentários sobre um cardápio ativo, na versão do *Apple Safari* para a plataforma *iOS*.

Fonte: Autoria própria.

versão desktop. Ao acessar essa tela é possível selecionar a data pela qual serão buscados os cardápios ativos caso esses existam, como mostra a Figura 51.

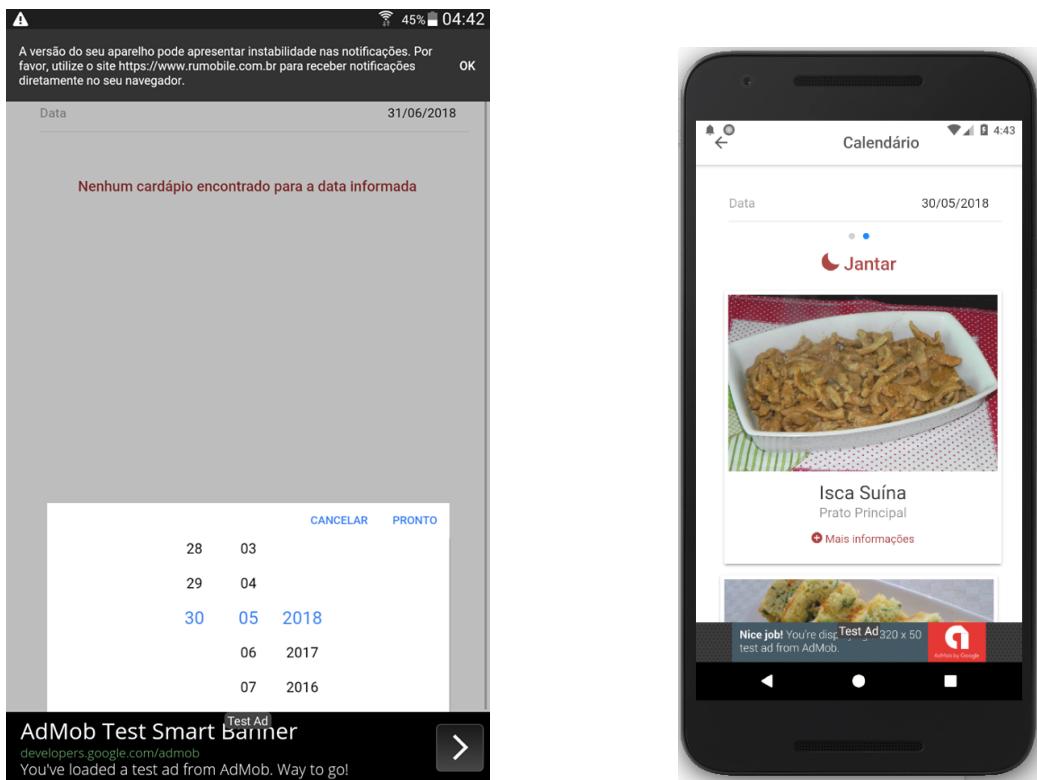
Ao buscar por cardápios ativos em outras datas é possível visualizar os cardápios de almoço e jantar daquela data em questão além das informações sobre os alimentos presentes em cada um, não sendo possível comentar ou visualizar comentários.

## 5.9 Automação

A fim de agilizar o processo de *build* da aplicação ou mesmo para poder acessá-la localmente, são utilizadas tarefas automatizadas.

Para a automatização são utilizados comandos de scripts do *NPM* executados na raiz da aplicação através de um terminal interpretador de comandos (terminal *bash*). Esses comandos são apresentados na Tabela 5.

Esses comandos são contidos na propriedade *scripts* do arquivo *package.json*. Por ser tratar de um arquivo *JSON*, os scripts são descritos em chave-valor, onde cada comando aponta para uma ação que serve como um atalho para uma tarefa no ciclo de



- (a) Seleção de data a ser buscada, na versão nativa em um tablet rodando uma versão mais antiga (4.4) do *Android*. (b) Exibição dos cardápios para a data buscada, na versão do emulador do *Android*.

Figura 51 – Tela de busca de alimentos ativos para uma determinada data.

Fonte: Autoria própria.

vida da aplicação. O comando `npm start`, por exemplo, executa `node ./server` para iniciar o servidor do *Node.js*. Neste caso não é necessário utilizar o `run` por se tratar de um comando padrão utilizado para iniciar a aplicação à partir de seu *entry point* (ponto de entrada).

O comando `npm run start:public` e `npm run start` servem como atalhos de execução dos comandos da *Angular CLI* e *Ionic CLI*, respectivamente, para iniciar a aplicação localmente em ambiente de desenvolvimento.

Já os demais comandos executam tarefas do *Gulp*<sup>20</sup>. Essas tarefas automatizam processos que incluem essencialmente a execução de comandos do terminal, limpeza de pastas e cópia de arquivos após o *build*. A Figura 52 ilustra a tarefa de *build* para o ambiente de Teste.

Ao executar o comando `npm run build:test`, o mesmo irá iniciar a tarefa do *Gulp build:test*, que chamará outras tarefas presentes no processo. O *Gulp* por definição realiza as tarefas de forma assíncrona, porém com a função `runSequence()`, proveniente

<sup>20</sup> <https://gulpjs.com/>

Tabela 5 – Lista de comandos e ações presentes na aplicação.

Comando	Ação
npm start	Responsável por iniciar o servidor, sendo a tarefa essencial utilizada pelo serviço de hospedagem para iniciar a aplicação após o deploy da mesma.
npm run start:dev	Inicia localmente e simultaneamente o servidor, a aplicação pública e a aplicação administrativa.
npm run start:public	Inicia localmente a aplicação pública.
npm run start:admin	Inicia localmente a aplicação administrativa.
npm run start:dev:dist	Inicia localmente e simultaneamente o servidor, a aplicação pública e a aplicação administrativa. Utiliza as versões de build da aplicação pública e da administrativa.
npm run build:test	Realiza o build da aplicação por completo e copia os arquivos para o diretório /dist/test, já pronto para ser enviado para o servidor de teste.
npm run build:staging	Realiza o build da aplicação por completo e copia os arquivos para o diretório /dist/staging, já pronto para ser enviado para o servidor de homologação.
npm run build:prod	Realiza o build da aplicação por completo e copia os arquivos para o diretório /dist/prod, já pronto para ser enviado para o servidor de produção.
npm run android:emulate	Executa a aplicação no emulador do Android. Ambiente de teste utilizado para acessar a API do servidor de teste.
npm run android:run	Executa a aplicação no dispositivo Android, caso esteja disponível. Ambiente de teste utilizado para acessar a API do servidor de teste.
npm build:android:test	Gera ou utiliza uma chave existente para assinar o APK. Realiza o build da aplicação Android e então gera o APK já assinado. Ambiente de teste utilizado para acessar a API do servidor de teste.
npm build:android:staging	Gera ou utiliza uma chave existente para assinar o APK. Realiza o build da aplicação Android e então gera o APK já assinado. Ambiente de homologação utilizado para acessar a API do servidor de homologação.
npm build:android:prod	Gera ou utiliza uma chave existente para assinar o APK. Realiza o build da aplicação Android e então gera o APK já assinado. Versão já para subir para a loja de aplicativos.

Fonte: Autoria própria.

da biblioteca *run-sequence*<sup>21</sup>, é possível garantir que as funções sejam executadas na ordem descrita nos argumentos, sendo tarefas simultâneas definidas em um *array*.

Isso se faz necessário pois algumas tarefas dependem da outra, como, por exemplo, antes de copiar os arquivos para o diretório de /dist, deve-se primeiro limpar os arquivos já existentes no diretório, realizar um novo *build* e só então copiar os arquivos. Com isso, tem-se a garantia de que houve sucesso ao atualizar os arquivos após a execução da tarefa.

<sup>21</sup> <https://www.npmjs.com/package/run-sequence>

```

gulp.task('build:test', cb => runSequence(
  'clean:dist:test',
  ['build:public:test', 'build:admin:test'],
  ['copy:public:test', 'copy:admin:test'],
  'transpile:server:test',
  'copy:server:test', cb));

gulp.task('clean:dist:test', () => del(`[${paths.dist.test}]/!(.git*|openshift|Procfile)**`), {dot: true}));

gulp.task('build:public:test', shell.task('cd client/public && npm run ionic:build:test'));

gulp.task('build:admin:test', shell.task('cd client/admin && npm run build:test'));

gulp.task('copy:public:test', () => {
  return gulp.src(`[${publicDistPath}/**`])
    .pipe(gulp.dest(`[${paths.dist.test}]/${publicPath}`));
});

gulp.task('copy:admin:test', () => {
  return gulp.src(`[${adminDistPath}/**`])
    .pipe(gulp.dest(`[${paths.dist.test}]/${adminPath}`));
});

gulp.task('transpile:server:test', () => {
  return gulp.src(_.union(paths.server.scripts, paths.server.json))
    .pipe(transpileServer())
    .pipe(gulp.dest(`[${paths.dist.test}]/${serverPath}`));
});

gulp.task('copy:server:test', () => {
  return gulp.src([
    'package.json',
  ], { cwdbase: true })
    .pipe(gulp.dest(paths.dist.test));
});

```

Figura 52 – Exemplo de tarefa do *Gulp* de *build* para o ambiente de Teste.

Fonte: Autoria própria.

## 5.10 Deploy

O último ponto do ciclo de vida da aplicação é a realização do *deploy* e distribuição da mesma. Realizado o *build* da aplicação, se fez necessário somente o *deploy* da aplicação para web e para móvel. Após gerar a aplicação para determinado ambiente por meio de um dos comandos do *NPM*, foi necessário iniciar um diretório do *Git*, como descrito pelo serviço da *Umbler*, para então ser enviado para o servidor de hospedagem e ficar disponível para acesso para outros usuários.

Para a aplicação móvel nativa foi gerado o arquivo *APK* para ser enviado para o serviço de distribuição de aplicativos *Google Play*. O serviço disponibiliza ambientes para testes, onde é possível definir os dispositivos que serão utilizadas para tal, e o de produção que é a disponibilização da aplicação para o público em geral.

Devido ao limite de cronograma, a aplicação não foi disponibilizada em produção para acesso ao público em geral e com isso não foi possível realizar a análise de receptividade e volume de uso pelos usuários do restaurante.

## 6 Conclusões

Este trabalho evidencia dois contextos muitos importantes que podem ser notados nos dias atuais, que são o acesso globalizado à informação de modo facilitado e a rápida evolução das tecnologias. Isso torna perceptível que a ênfase tecnológica se faz e fará cada dia mais em encontrar formas de automatizar, facilitar ou mesmo fornecer novos serviços englobados no paradigma presente no mundo contemporâneo. Serviços de informação, logística, alimentação, dentre outros, têm cada vez mais se adequado a essa nova realidade a fim de inovar, obter prospecção e satisfação de seus clientes.

Isso se aplica também a serviços gratuitos que têm o intuito de informar aos usuários ou facilitar o acesso às informações, sendo válido para casos onde há um novo recurso alternativo aos existentes. Sendo assim, foi possível perceber que tem-se englobado o objetivo deste trabalho, que permite o acesso aos cardápios de um refeitório universitário como maneira alternativa às oficiais.

Dentro do aspecto tecnológico abordado neste trabalho, foi possível observar o foco que se tem adotado no campo de desenvolvimento de aplicações web e móveis. Com o surgimento de novas tecnologias e evolução das antigas, o desenvolvimento de aplicações por completo se faz totalmente possível graças a essa variedade de recursos.

O desenvolvimento *full-stack* permite que equipes reduzidas de desenvolvedores, como as de *startups* e de empresas de pequeno e médio porte, possam entregar soluções robustas e de qualidade em um tempo ágil, perante às limitações que possam se fazer presentes.

Foi possível perceber que por meio de tecnologias como *Node.js*, *Express*, *MongoDB*, *Angular*, por exemplo, é plenamente factível a criação de aplicações web robustas e completas. Além disso, por meio do *Ionic* é possível criar uma aplicação híbrida que realiza a integração com aplicações do tipo, permitindo a entrega de uma aplicação móvel também em prazo reduzido. Isso se faz presente devido à grande evolução recente do *JavaScript*, com o surgimento constante de novas bibliotecas e *frameworks* visando facilitar a vida dos desenvolvedores com novos recursos e recursos melhorados.

Entretanto, um ponto desfavorável no que diz respeito ao desenvolvimento híbrido é que há uma queda de desempenho considerável em relação ao que seria esperado em uma aplicação nativa, especialmente em aparelhos e sistemas operacionais mais antigos. Sendo assim, é possível concluir que é mais sensato utilizar aplicações híbridas em casos de escassez de recursos e aplicações nativas para casos onde há uma grande quantidade de recursos e pessoal disponíveis.

## 6.1 Trabalhos Futuros

Vários aspectos da aplicação podem ser melhorados em alterações futuras. Seu estado pode ser definido como um software evolutivo, que poderá receber melhorias graduais ao longo do tempo.

A primeira questão que deverá ser realizada é a disponibilização da aplicação para o ambiente de produção, liberando o acesso ao público e permitindo o download da aplicação móvel. Feito isso, poderá ser realizada análises de receptividade dos usuários do restaurante tanto por meio de avaliações da aplicação em si, quanto por meio dos comentários sobre os cardápios, sendo esta última com o objetivo de como proceder com possíveis alterações nos cardápios com a finalidade de melhorar a receptividade por parte dos usuários.

Outro ponto que pode ser definido é a possibilidade de cadastro de usuários visitantes na aplicação pública. Isso possibilitaria a criação de mecanismos internos de avaliação, como comentários na aplicação na versão nativa ou outros tipos de avaliações, como estrelas, corações ou curtidas, por exemplo, ao invés de ser somente por comentários.

Um ponto que é possível alterar é a adição de integração com o *OneSignal* via *API REST*, de modo que o administrador possa enviar notificações diretamente dentro da aplicação administrativa, deixando somente a análise de receptividade das notificações para a plataforma do serviço em questão.

Uma outra possibilidade é criar uma integração com o site da UFV, onde o usuário possa informar seu login e senha e, via serviço, a aplicação acessar o site e trazer o saldo, permitindo que o usuário visualize o mesmo diretamente na aplicação.

Além disso, é possível criar um *job schedule*, ou uma trabalho programado, que seja executado uma vez por dia ou algum outro período definido, que acesse o site da UFV para coletar os cardápios disponibilizados oficialmente por lá. Isso permitiria a automação na atividade de cadastro de cardápios ativos no sistema, de modo que o administrador ou o usuário tivessem somente que conferir se está tudo correto com os cardápios programados.

## Referências

- ABDULLAH, H. M.; ZEKI, A. M. Frontend and backend web technologies in social networking sites: Facebook as an example. In: IEEE. **3rd International Conference on Advanced Computer Science Applications and Technologies (ACSAT), 2014.** [S.l.], 2014. p. 85–89.
- AMATYA, S.; KURTI, A. Cross-platform mobile development: challenges and opportunities. In: **ICT Innovations 2013.** [S.l.]: Springer, 2014. p. 219–229.
- ANDRADE, A. W.; AGRA, R.; MALHEIROS, V. Estudos de caso de aplicativos móveis no governo brasileiro. p. 780–791, 2012.
- ANGULO, E. **Case Study on Mobile Applications UX: Effect of the Usage of a Cross-Platform Development Framework.** Master Thesis, 2014.
- APACHE. **Apache Cordova Overview.** 2016. Disponível em: <<https://cordova.apache.org/docs/en/latest/guide/overview/index.html>>. Acesso em: 31/10/2016.
- AUTOMATTIC, I. **Mongoose Docs.** 2018. Disponível em: <<http://mongoosejs.com/docs>>. Acesso em: 01/06/2018.
- AZAR, K. M. J.; LESSER, L. I.; LAING, B. Y.; STEPHENS, J.; AURORA, M. S.; BURKE, L. E.; PALANIAPPAN, L. P. Mobile applications for weight management: theory-based content analysis. **American journal of preventive medicine**, Elsevier, v. 45, n. 5, p. 583–589, 2013.
- BACANCY, T. **MEAN.JS full stack development solution.** 2016. Disponível em: <<http://www.bacancytechnology.com/mean-js-full-stack-developmentsolution>>. Acesso em: 22/03/2016.
- BANKER, K.; BAKKUM, P.; VERCH, S.; GARRETT, D.; HAWKINS, T. **MongoDB in Action.** [S.l.]: Manning Publications Co., 2016.
- BEZERRA, I. N.; SICHIERI, R. Características e gastos com alimentação fora do domicílio no brasil. **Revista de Saúde Pública**, v. 44, n. 2, p. 221–229, 2010.
- BEZERRA, P. T.; SCHIMIGUEL, J. Desenvolvimento De Aplicações Mobile Cross-Platform Utilizando Phonegap. **Observatorio de la Economía Latinoamericana**, 2016.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide.** [S.l.]: Addison-Wesley, 1998.
- Boston Consulting Group. **The Mobile Revolution: How Mobile Technologies Drive a Trillion-Dollar Impact.** 2015. Disponível em: <[https://www.bcgperspectives.com/content/articles/telecommunications\\_technology\\_business\\_transformation\\_mobile\\_revolution/](https://www.bcgperspectives.com/content/articles/telecommunications_technology_business_transformation_mobile_revolution/)>. Acesso em: 31/10/2016.
- BUDIU, R. **Mobile: Native Apps, Web Apps, and Hybrid Apps.** 2013. Disponível em: <<https://www.ngroup.com/articles/mobile-native-apps/>>. Acesso em: 31/10/2016.

- CANTELON, M.; HARTER, M.; HOLOWAYCHUK, T. J.; RAJLICH, N. **Node.js in Action.** [S.I.]: Manning Publications Co., 2014.
- CHARLAND, A.; LEROUX, B. Mobile application Development : Web vs . Native. **Communications of the ACM**, v. 54, p. 0–5, 2011. ISSN 00010782.
- CORDOVA. **Apache Cordova Reference.** 2016. Disponível em: <<http://cordova.apache.org/docs/en/latest/guide/overview/index.html>>. Acesso em: 06/11/2016.
- COSTA, D. **Cross-platform mobile development using Web Technologies.** Master, 2013.
- CRUZ, D. L. d.; NACIF, M. Elaboração de aplicativo de avaliação nutricional para telefones celulares com sistema android. **Demetra: Food, Nutrition & Health/Alimentação, Nutrição & Saúde**, v. 10, n. 4, 2015.
- DANTAS, A. C. **Aplicativo RU-UFU, Google Play.** 2016. Disponível em: <<https://play.google.com/store/apps/details?id=com.fenix.rufu>>. Acesso em: 31/10/2016.
- DTI. **UFV mobile - Laboratório de Desenvolvimento para Dispositivos Móveis (LADBD2M).** 2018. Disponível em: <[https://play.google.com/store/apps/details?id=br.ufv.m&hl=pt\\_BR](https://play.google.com/store/apps/details?id=br.ufv.m&hl=pt_BR)>. Acesso em: 06/07/2018.
- EASON, J. **Android Developers Blog: An update on Eclipse Android Developer Tools.**, 2015. Disponível em: <<https://developer.android.com/studio/tools/sdk/eclipse-adt.html>>. Acesso em: 31/10/2016.
- EL-GAYAR, O.; TIMSINA, P.; NAWAR, N.; EID, W. Mobile applications for diabetes self-management: status and potential. **Journal of diabetes science and technology**, SAGE Publications, v. 7, n. 1, p. 247–262, 2013.
- ESTADÃO. **Smartphone: o verdadeiro computador pessoal.** 2015. Disponível em: <<http://economia.estadao.com.br/noticias/geral,smartphone-o-verdadeiro-computador-pessoal-imp-,1643233>>. Acesso em: 31/10/2016.
- FEKETE, G. **Being a Full Stack Developer.** 2014. Disponível em: <<https://www.sitepoint.com/full-stack-developer/>>. Acesso em: 05/11/2016.
- FERREIRA, R. **REST: Princípios e boas práticas.** 2017. Disponível em: <<http://blog.caelum.com.br/rest-principios-e-boas-praticas/>>. Acesso em: 01/06/2018.
- FERREIRA, S. R. G. Alimentação, nutrição e saúde: avanços e conflitos da modernidade. **Ciência e Cultura**, Sociedade Brasileira para o Progresso da Ciência, v. 62, n. 4, p. 31–33, 2010.
- FRAIN, B. **Responsive Web Design with HTML5 and CSS3.** [S.I.]: Packt Publishing Ltd, 2015.
- GIRDLEY, M. **Front-end vs Back-end.** 2014. Disponível em: <<http://codeup.com/different-types-of-programmers-front-end-vs-back-end/>>. Acesso em: 05/11/2016.
- GOOGLE, I. **Angular CLI Docs.** 2018. Disponível em: <<https://cli.angular.io/>>. Acesso em: 05/06/2018.

- GOOGLE, I. **Angular Docs**. 2018. Disponível em: <<https://angular.io/docs>>. Acesso em: 04/06/2018.
- GRAHAM, R. **Mobile First: What Does It Mean?** 2012. Disponível em: <<http://www.uxmatters.com/mt/archives/2012/03/mobile-first-what-does-it-mean.php>>. Acesso em: 31/10/2016.
- HERRON, D. **Node Web Development (2nd Edition)**. [S.l.]: Packt Publishing Ltd, 2013.
- HOLMES, S. **Getting MEAN with Mongo, Express, Angular, and Node**. [S.l.]: Manning Publications Co., 2016.
- IONIC. **Ionic Documentation - Core Concepts**. 2018. Disponível em: <<https://ionicframework.com/docs/intro/concepts/>>. Acesso em: 01/06/2018.
- IONIC. **Ionic Documentation - Ionic CLI**. 2018. Disponível em: <<https://ionicframework.com/docs/cli/>>. Acesso em: 01/06/2018.
- KRADIMCE, A.; BOGATINOSKA, D. C. Using hybrid mobile applications for adaptive multimedia content delivery. In: IEEE. **37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014**. [S.l.], 2014. p. 686–691.
- KLASNJA, P.; PRATT, W. Healthcare in the pocket: mapping the space of mobile-phone health interventions. **Journal of biomedical informatics**, Elsevier, v. 45, n. 1, p. 184–198, 2012.
- KORF, M.; OKSMAN, E. **Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options**. 2016. Disponível em: <[https://developer.salesforce.com/page/Native,\\_HTML5,\\_or\\_Hybrid:\\_Understanding\\_Your\\_Mobile\\_Application\\_Development\\_Options](https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options)>. Acesso em: 31/10/2016.
- KRANZ, M.; MÖLLER, A.; HAMMERLA, N.; DIEWALD, S.; PLÖTZ, T.; OLIVIER, P.; ROALTER, L. The mobile fitness coach: Towards individualized skill assessment using personalized mobile devices. **Pervasive and Mobile Computing**, Elsevier, v. 9, n. 2, p. 203–215, 2013.
- LESSEL, P.; BÖHMER, M.; KRÖNER, A.; KRÜGER, A. User requirements and design guidelines for digital restaurant menus. In: ACM. **Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design**. [S.l.], 2012. p. 524–533.
- LIU, C.; ZHU, Q.; HOLROYD, K. A.; SENG, E. K. Status and trends of mobile-health applications for ios devices: A developer's perspective. **Journal of Systems and Software**, Elsevier, v. 84, n. 11, p. 2022–2033, 2011.
- MADAUDO, R.; SCANDURRA, P. Native versus cross-platform frameworks for mobile application development. In: **VIII Workshop of the Italian Eclipse Community (September, 2013)**. [S.l.: s.n.], 2013.
- MARTÍN, I. S. M.; FERNÁNDEZ, M. G.; YURRITA, L. C. Aplicaciones móviles en nutrición, dietética y hábitos saludables; análisis y consecuencia de una tendencia a la alza. **Nutrición Hospitalaria**, SciELO Espana, v. 30, n. 1, p. 15–24, 2014.

- MONGODB, I. **The MongoDB 3.6 Manual**. 2018. Disponível em: <<https://docs.mongodb.com/manual/>>. Acesso em: 01/06/2018.
- MONGODB, I. **MongoDB Architecture**. 2018. Disponível em: <<https://www.mongodb.com/mongodb-architecture>>. Acesso em: 01/06/2018.
- MOUBRAY, A.; DANNER, T.; LITCHFIELD, J.; SELLARS, H.; LEE, R. Adapting Native Applications to Cross Platform Environments. **International Conference on Software Engineering Research and Practice | SERP'15** |, p. 210–216, 2015.
- NIXON, R. **Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites**. [S.l.]: "O'Reilly Media, Inc.", 2014.
- NPM, I. **NPM Docs**. 2018. Disponível em: <<https://docs.npmjs.com/>>. Acesso em: 03/06/2018.
- PALMIERI, M.; SINGH, I.; CICCHETTI, A. Comparison of cross-platform mobile development tools. In: IEEE. **Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on**. [S.l.], 2012. p. 179–186.
- PHAM, P. **Hybrid mobile application with Ionic and MEAN stack**. Bachelor Thesis, 2016.
- PIERRE, N. J.; XIUMEI, F.; DANIEL, G.; CLAUDE, G. J. Cross-platform mobile geolocation applications based on phonegap. **Lecture Notes on Software Engineering**, IACSIT Press, v. 3, n. 2, p. 78, 2015.
- PROENÇA, R. P. d. C. Alimentação e globalização: algumas reflexões. **Ciência e Cultura**, Sociedade Brasileira para o Progresso da Ciência, v. 62, n. 4, p. 43–47, 2010.
- SENADO. **Manual de Comunicação da Secom - Estrangeirismo**. 2015. Disponível em: <<https://www12.senado.leg.br/manualdecomunicacao/redacao-e-estilo/estilo/estrangeirismo>>. Acesso em: 05/11/2017.
- SENADO. **Manual de Comunicação da Secom - Estrangeirismos grafados sem itálico ou aspas**. 2016. Disponível em: <<https://www12.senado.leg.br/manualdecomunicacao/redacao-e-estilo/estilo/estrangeirismos-grafados-sem-italico>>. Acesso em: 05/11/2017.
- SOUZA, D. **Aplicativo BandejãoUFRJ, Google Play**. 2015. Disponível em: <<https://play.google.com/store/apps/details?id=com.github.diegofps.bandejaoufrj>>. Acesso em: 31/10/2016.
- SQLITE. **Features Of SQLite**. 2012. Disponível em: <<http://www.sqlite.org/features.html>>. Acesso em: 31/10/2016.
- STATISTA. **Number of apps available in leading app stores as of June 2016**. 2016a. Disponível em: <<https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>>. Acesso em: 29/10/2016.
- STATISTA. **Projected Google Play revenue from 2014 to 2019 (in billion U.S. dollars)**. 2016b. Disponível em: <<https://www.statista.com/statistics/444476/google-play-annual-revenue/>>. Acesso em: 30/10/2016.

- STATISTA. **Annual Apple App Store revenue in 2013 and 2015 (in billion U.S. dollars)**. 2016c. Disponível em: <<https://www.statista.com/statistics/296226/annual-apple-app-store-revenue/>>. Acesso em: 31/10/2016.
- STRONGLOOP, I. **Express Docs**. 2018. Disponível em: <<http://expressjs.com/>>. Acesso em: 04/06/2018.
- TESANOVIC, V. **Multi threading and multiple process in Node.js**. 2018. Disponível em: <<https://itnext.io/multi-threading-and-multi-process-in-node-js-ffa5bb5cde98>>. Acesso em: 03/06/2018.
- The Economist, N. L. **The truly personal computer**. 2015. Disponível em: <<http://www.economist.com/news/briefing/21645131-smartphone-defining-technology-age-truly-personal-computer>>. Acesso em: 31/10/2016.
- TILKOV, S. **Uma rápida Introdução ao REST**. Tradução André Faria Gomes. 2008. Disponível em: <<https://www.infoq.com.br/articles/rest-introduction>>. Acesso em: 01/06/2018.
- TRICE, A. **PhoneGap advice on dealing with Apple application rejections**. 2012. Disponível em: <<http://www.adobe.com/devnet/phonegap/articles/apple-application-rejections-and-phonegap-advice.html>>. Acesso em: 31/10/2016.
- UFV. **Divisão de Assuntos Comunitários (DAC)**. 2017. Disponível em: <[http://dac.crp.ufv.br/?page\\_id=161](http://dac.crp.ufv.br/?page_id=161)>. Acesso em: 06/07/2018.
- WESTBERG, H. **What is a Full-Stack Developer?** 2014. Disponível em: <<http://codeup.com/what-is-a-full-stack-developer/>>. Acesso em: 05/11/2016.
- WILKEN, J. **Ionic in Action**. [S.l.]: Manning Publications Co., 2016.
- XINOGLOS, S.; PSANNIS, K. E.; SIFALERAS, A. Recent advances delivered by html 5 in mobile cloud computing applications: a survey. In: ACM. **Proceedings of the Fifth Balkan Conference in Informatics**. [S.l.], 2012. p. 199–204.

# Apêndices

# APÊNDICE A – Documentação de requisitos

## **RF01 - Divisão de módulos**

*Objetivo:* Prover a divisão entre módulo administrativo e módulo público.

*Descrição:* Deverá haver a divisão entre dois módulos dentro da aplicação como um todo. O módulo administrativo deverá ser acessível mediante a login e utilizado para a manipulação dos registros presentes na aplicação. O módulo público deverá ser utilizado pelo acesso do público em geral.

## **RF02 - Aplicação pública multiplataforma**

*Objetivo:* Prover que a aplicação pública seja acessível de múltiplos dispositivos.

*Descrição:* A aplicação pública deverá ser multiplataforma e acessível por vários dispositivos. Deverá também ser instalável no sistema *Android* por meio de construção de um instalador para esta plataforma.

## **RF03 - Gerenciamento de perfis de acesso**

*Objetivo:* Prover opção de cadastro e remoção de administradores e usuários do sistema.

*Descrição:* O sistema deverá prover mecanismo de cadastro e remoção de usuários e administradores que irão fazer o gerenciamento do sistema. Deverá ser possível também a alteração da própria senha.

## **RF04 - Gerenciamento de alimentos**

*Objetivo:* Prover opção de cadastro, edição e remoção de alimentos a serem inseridos nos cardápios.

*Descrição:* O sistema deverá possuir uma funcionalidade de gerenciamento de alimentos, permitindo o cadastro e a remoção dos mesmos no banco de dados da aplicação. Cada alimento deverá possuir os seguintes dados: nome, tipo, descrição, foto e link para maiores informações (*FatSecret* ou similares).

## **RF05 - Gerenciamento de cardápios**

*Objetivo:* Prover mecanismo de cadastro, edição e remoção de cardápios.

*Descrição:* O sistema deverá possuir uma funcionalidade de gerenciamento de cardápios, permitindo o cadastro e a remoção dos mesmos no banco de dados da aplicação.

**RF06 - Gerenciamento de cardápios ativos**

*Objetivo:* Prover mecanismo de criação, visualização, edição e remoção de cardápios ativos.

*Descrição:* O sistema deverá possuir uma funcionalidade de gerenciamento de cardápios ativos, permitindo o cadastro e a remoção dos mesmos no banco de dados da aplicação. Os cardápios deverão conter uma data, o período que estará ativo, podendo ser almoço e jantar, e um cardápio previamente cadastrado. Os cardápios ativos no dia deverão ser visualizados na página inicial da aplicação e também serem buscados para apresentação de outras datas.

**RF07 - Manter dados de alteração de registros**

*Objetivo:* Registrar todas as manipulações de dados presentes na base da aplicação.

*Descrição:* Todas as entidades presentes na aplicação deverão manter o registro de usuário de criação, data de criação, usuário de atualização e data da última atualização.

**RF08 - Mecanismo de interação e feedback**

*Objetivo:* Permitir interação pelos usuários e analisar a receptividade do público usuário do Refeitório Universitário para com os cardápios ofertados pelo mesmo.

*Descrição:* Os usuários deverão ter a opção de comentar a respeito dos cardápios disponíveis na aplicação. Ao administrador do sistema deverá ser possível o levantamento de dados para futuras tomadas de decisões acerca dos cardápios.

**RF09 - Notificações aos usuários**

*Objetivo:* Enviar notificações para os usuários do aplicativo a fim de informar sobre os cardápios do dia ou da semana ou mesmo qualquer outra informação que seja considerada pertinente em ser divulgada.

*Descrição:* O sistema deverá prover serviço de notificações para a aplicação de modo que os usuários do mesmo possam receber informações sobre o cardápios ou quaisquer outras informações que o administrador julgar necessárias. Deverá ser possível ao administrador configurar as notificações, definindo texto e horários das mesmas.

**RF10 - Propagandas na aplicação**

*Objetivo:* Oferecer publicidade na aplicação com finalidade social.

*Descrição:* O sistema deverá conter mecanismo de publicidade, como o *Google AdSense* e *Google AdMob*, a fim de arrecadar recursos para a manutenção da aplicação e receita líquida destinada a ONGs escolhidas. Deverá ser criada uma conta do Google específica para esse fim.