

**UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"**

**FACULDADE DE CIÊNCIAS - CAMPUS BAURU**

**DEPARTAMENTO DE COMPUTAÇÃO**

**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**LUIZ FERNANDO DA SILVA CIESLAK**

**PROTÓTIPO OPEN-SOURCE DE SISTEMA INTEGRADO PARA  
O RESTAURANTE UNIVERSITÁRIO DA UNESP BAURU**

**BAURU**

**2017**

LUIZ FERNANDO DA SILVA CIESLAK

**PROTÓTIPO OPEN-SOURCE DE SISTEMA INTEGRADO PARA  
O RESTAURANTE UNIVERSITÁRIO DA UNESP BAURU**

Trabalho de Conclusão de Curso do Curso  
de Ciência da Computação da Universidade  
Estadual Paulista “Júlio de Mesquita Filho”,  
Faculdade de Ciências, Campus Bauru.  
Orientador: Prof. Dr. Sidnei Bergamaschi

BAURU  
2017

Luiz Fernando da Silva Cieslak

Protótipo open-source de sistema integrado para o Restaurante Universitário da UNESP Bauru/  
Luiz Fernando da Silva Cieslak. – Bauru, 2017-

49 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Sidnei Bergamaschi

Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”  
Faculdade de Ciências

Ciência da Computação, 2017.

1. JavaScript 2. Aplicativo móvel 3. Restaurante Universitário

Luiz Fernando da Silva Cieslak

## **Protótipo open-source de sistema integrado para o Restaurante Universitário da UNESP Bauru**

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Banca Examinadora

---

**Prof. Dr. Sidnei Bergamaschi**  
Orientador

---

**Simone Domingues Prado**  
Professora

---

**João Pedro Albino**  
Professor Convidado

Bauru, \_\_\_\_\_ de \_\_\_\_\_ de \_\_\_\_\_.

# Agradecimentos

Agradeço a minha mãe, minha namorada Bruna e aos meus amigos, especialmente da República Pão da Vida e da República Alabama, que me apoiaram e incentivaram no desenvolvimento deste projeto.

*Se as portas da percepção estivessem limpas, tudo apareceria para o homem tal como é:  
infinito.*

*Aldous Huxley*

# Resumo

O presente Trabalho de Conclusão de Curso trata sobre a criação de um protótipo de um sistema de gerenciamento do Restaurante Universitário da UNESP Bauru, contendo um aplicativo móvel feito em *Ionic* para a compra e venda de refeições, um *website* para os funcionários adicionarem o cardápio e realizarem outros tipos de operações, interligados por um servidor e banco de dados providos pelo *Firebase*. O sistema resolve problemas recorrentes presentes na implementação atual e tem objetivo de ser *open-source*, sendo mantido pela comunidade de estudantes.

**Palavras-chave:** aplicativo móvel, restaurante universitário, javascript, ionic, nodejs, angular

# Abstract

This project is about a management system of the State University of Sao Paulo's restaurant, which contains a mobile app for the end user buy and sell meals, a website for the managers to add the menu and to do other operations related to them and a server with database that integrates everything provided by *Firebase*. This management system resolves recurrent problems the actual system has and has the objective to be open-source, being sustained by the student community.

**Keywords:** mobile app, restaurant, javascript, ionic, nodejs, angular



# Lista de ilustrações

Figura 1 – Frequência de utilização do Restaurante Universitário. . . . .	21
Figura 2 – Avaliação geral do sistema do RU. . . . .	21
Figura 3 – Avaliação geral do sistema de transferências do RU. . . . .	22
Figura 4 – Avaliação do <i>website</i> de compras e do aplicativo bauRU. . . . .	22
Figura 5 – Avaliação da burocracia e dificuldade de uso do sistema do RU. . . . .	22
Figura 6 – Opinião dos alunos se há desistência do uso do RU devido aos problemas. . . .	22
Figura 7 – Reincidência do uso do RU caso os problemas fossem resolvidos. . . . .	23
Figura 8 – Tela de login. . . . .	28
Figura 9 – Tela do Sidemenu. . . . .	29
Figura 10 – Tela inicial do aplicativo. . . . .	30
Figura 11 – Tela da lista de refeições. . . . .	31
Figura 12 – Possíveis opções para a lista de refeições. . . . .	31
Figura 13 – Alerta de confirmação da compra de refeição. . . . .	31
Figura 14 – Tela dos detalhes da refeição. . . . .	32
Figura 15 – Seção de ajuda na página de login. . . . .	33
Figura 16 – Seção de ajuda acessível pelo sidemenu. . . . .	34
Figura 17 – Exemplos de notificações do aplicativo. . . . .	38
Figura 18 – Barra de Navegação após o login. . . . .	40
Figura 19 – Cadastro de novos usuários. . . . .	41
Figura 20 – Adicionar saldo para um determinado usuário. . . . .	41
Figura 21 – Cadastro de nova refeição no cardápio. . . . .	42
Figura 22 – Aviso de refeição existente na data escolhida. . . . .	42
Figura 23 – Informações da página de detalhes da refeição. . . . .	43
Figura 24 – Informações da página de detalhes da refeição. . . . .	44
Figura 25 – Aviso de tentativa de alteração de refeição inexistente. . . . .	44

# Lista de tabelas

Tabela 1 – Funções do AuthService . . . . .	25
Tabela 2 – Funções do UserService . . . . .	26
Tabela 3 – Funções do RefeicaoService . . . . .	27
Tabela 4 – Funções do TimeService . . . . .	27
Tabela 5 – Funções do ConnectionService . . . . .	28

# Lista de códigos

1	Modelo do banco de dados em JSON. . . . .	19
2	Injeção dos serviços no <i>constructor</i> . . . . .	25
3	Função de login em <i>auth-service</i> . . . . .	29
4	Exemplo de função do APScheduler. . . . .	36
5	Função sendNotification . . . . .	38

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>13</b>
<b>1.1</b>	<b>Mercado de aplicativos móveis</b>	<b>13</b>
<b>1.2</b>	<b>Problema</b>	<b>13</b>
<b>1.3</b>	<b>Objetivos</b>	<b>14</b>
1.3.1	Objetivo geral	14
1.3.2	Objetivos específicos	14
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>16</b>
<b>2.1</b>	<b>Frameworks</b>	<b>16</b>
2.1.1	Ionic	16
2.1.2	Cordova	16
2.1.3	Angular	16
<b>2.2</b>	<b>Javascript</b>	<b>17</b>
2.2.1	Typescript	17
2.2.2	AngularFire	17
2.2.3	RxJS	17
2.2.4	Moment	18
2.2.5	ECMAScript e Programação Assíncrona	18
<b>2.3</b>	<b>Banco de Dados</b>	<b>18</b>
2.3.1	Arquitetura	18
2.3.2	Regras	20
<b>3</b>	<b>APLICATIVO MÓVEL</b>	<b>21</b>
<b>3.1</b>	<b>Pré-Projeto</b>	<b>21</b>
3.1.1	Pesquisa	21
3.1.1.1	Respostas	21
3.1.2	Requerimentos	23
3.1.3	O problema do pagamento online	24
3.1.4	Divisão do trabalho	24
<b>3.2</b>	<b>Angular Services</b>	<b>25</b>
3.2.1	AuthService	25
3.2.2	UserService	25
3.2.3	RefeicaoService	25
3.2.4	TimeService	25
3.2.5	ConnectionService	26
<b>3.3</b>	<b>Manuseio dos Usuários</b>	<b>28</b>

3.3.1	Login . . . . .	28
<b>3.4</b>	<b>Sidemenu . . . . .</b>	<b>28</b>
<b>3.5</b>	<b>Homepage . . . . .</b>	<b>29</b>
<b>3.6</b>	<b>Compra de Refeições . . . . .</b>	<b>30</b>
<b>3.7</b>	<b>Ajuda e Sobre o projeto . . . . .</b>	<b>32</b>
3.7.1	Ajuda na página de Login . . . . .	32
3.7.2	Ajuda no sidemenu . . . . .	33
<b>4</b>	<b>SERVIDOR . . . . .</b>	<b>35</b>
<b>4.1</b>	<b>Motivos . . . . .</b>	<b>35</b>
<b>4.2</b>	<b>Cron Jobs . . . . .</b>	<b>35</b>
<b>4.3</b>	<b>Funções . . . . .</b>	<b>36</b>
4.3.1	Horário do servidor - utc() . . . . .	36
4.3.2	Reembolso do usuário - removeRefeicao() . . . . .	36
4.3.2.1	Alocar usuário da fila - assignNextUserFromQueue() . . . . .	36
4.3.3	Limpeza da fila - queueCleanup() . . . . .	37
<b>4.4</b>	<b>Notificações . . . . .</b>	<b>37</b>
4.4.1	FCM . . . . .	37
4.4.2	Funções . . . . .	37
4.4.3	Plugins do Cordova . . . . .	39
<b>5</b>	<b>WEBSITE PARA OS FUNCIONÁRIOS . . . . .</b>	<b>40</b>
<b>5.1</b>	<b>Planejamento . . . . .</b>	<b>40</b>
<b>5.2</b>	<b>Manuseio dos Usuários . . . . .</b>	<b>40</b>
5.2.1	Cadastro de novos usuários . . . . .	40
5.2.2	Adicionar Saldo . . . . .	41
<b>5.3</b>	<b>Cardápio . . . . .</b>	<b>41</b>
5.3.1	Adicionar Refeição . . . . .	41
5.3.2	Ver Refeições . . . . .	42
5.3.3	Alterar Refeição . . . . .	43
<b>6</b>	<b>TRABALHOS FUTUROS . . . . .</b>	<b>45</b>
<b>6.1</b>	<b>Adequação . . . . .</b>	<b>45</b>
<b>6.2</b>	<b>Implementação . . . . .</b>	<b>45</b>
<b>7</b>	<b>CONCLUSÃO . . . . .</b>	<b>47</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>48</b>

# 1 Introdução

## 1.1 Mercado de aplicativos móveis

Com a popularização da Internet no Século XXI, diversas soluções para problemas do cotidiano foram criadas utilizando-se tecnologias emergentes. O avanço geral da tecnologia e dos smartphones fez com que mais problemas fossem resolvidos e que as pessoas se tornassem mais próximas aos dispositivos eletrônicos. Quase todas as soluções e programas criados para o Computador Pessoal se tornaram *apps* que são instalados nos *smartphones* e possuem performance melhor ou igual aos programas de Desktop.

Uma pesquisa realizada pela ComScore ([COMSCORE, 2017](#)) mostra que o número de usuários de celulares no mundo ultrapassou o número de usuários de Desktop em 2014, o que implica em uma mudança de visão de mercado e de desenvolvimento pelas empresas e estudantes de tecnologia.

O seguinte Trabalho de Conclusão de Curso trata sobre a criação de protótipo de aplicativo móvel do Restaurante Universitário da UNESP Bauru para a compra de refeições e gerenciamento do saldo dos alunos.

O Restaurante Universitário da UNESP Bauru foi inaugurado em janeiro de 2015 e desde então apresenta problemas e processos muito burocráticos para os alunos em relação a compra e a transferência de refeições. Portanto, além de desenvolver o protótipo, será criado um novo modelo para a reserva e transferência de refeições, utilizando padrões atuais e tecnologias recentes.

O protótipo do aplicativo móvel será feito utilizando a plataforma Ionic, uma plataforma de criação open-source de aplicativos móveis renomada, com mais de 4 milhões de aplicativos já desenvolvidos e que é capaz de exportar para todos os Sistemas Operacionais do mercado.

## 1.2 Problema

Os estudantes que desejam comprar uma refeição no Restaurante Universitário enfrentam algumas dificuldades para isso. Em 2015, ano em que o Restaurante Universitário foi criado, os estudantes precisavam comparecer a Diretoria Técnica Administrativa (DTAd) para criar um cartão específico para o RU, mesmo já possuindo o cartão de estudante cedido pela universidade e tendo um cadastro ativo na mesma.

Então, se o estudante desejasse comprar uma refeição, ele deveria comparecer ao DTAd uma semana antes do dia desejado, em um horário específico, para realizar a compra. Isso

fez com que se criasse uma grande fila de estudantes no DTAd para a compra de refeições. O estudante que estava na fila não sabia se iria conseguir uma refeição devido ao número máximo de refeições que é servida diariamente. O estudante não podia transferir sua refeição para outra data, nem poder fazer a compra no dia desejado.

Hoje em dia, a compra de refeições pode ser feita pela internet, através do site [www.bauru.unesp.br](http://www.bauru.unesp.br), porém, ainda com as mesmas restrições: O sistema é aberto das 7h às 12h durante os dias da semana para a compra das refeições da próxima semana; e após a reserva da refeição, o aluno deve comparecer à Seção Técnica de Finanças até as 16h do penúltimo dia útil da semana corrente.

Algumas universidades públicas, como a USP e UNICAMP, possuem um sistema de Restaurante Universitário mais sofisticado, podendo o aluno realizar a compra no dia caso haja uma vaga disponível.

Portanto, este Trabalho de Conclusão de Curso pretende mostrar que pode-se construir um sistema de Restaurante Universitário que permita a compra e transferências de refeições, utilizando tecnologias recentes, contando com a ajuda da comunidade de softwares *open-source* e que facilite o uso das pessoas que necessitam deste serviço.

## 1.3 Objetivos

### 1.3.1 Objetivo geral

O presente Trabalho de Conclusão de Curso tem como objetivo desenvolver um protótipo *open-source* de Aplicativo Móvel para o Restaurante Universitário da UNESP Bauru, junto com a implementação de um novo sistema de gerenciamento, que permita a reserva e compra de refeições de maneira simples e eficaz, além de disponibilizar uma fila de espera para as vagas remanescentes de cada refeição.

### 1.3.2 Objetivos específicos

- a) Criar um aplicativo móvel que facilita a compra e transferência de refeições do restaurante universitário;
- b) Exportá-lo para o número máximo de Sistemas Operacionais presentes no mercado a fim de alcançar o maior número de pessoas;
- c) Tornar o aplicativo fácil de usar e navegar, utilizando-se conceitos de User Experience;
- d) Criar uma base de dados robusta e íntegra, para que os dados sejam livres de falhas e vazamentos;

- e) Tornar o aplicativo móvel open-source, para que a comunidade de alunos de Ciências da Computação nos câmpus da UNESP possam contribuir pela manutenção e evolução do aplicativo;
- f) Encaminhar o trabalho realizado à Diretoria Técnica Administrativa para que seja analisado e possivelmente implementado pelos responsáveis.
- g) Tornar o aplicativo móvel um modelo para os demais câmpus da UNESP para que seja implementado nestas unidades, podendo até ser implementado em outras universidades públicas.

O capítulo 2 trata sobre a fundamentação teórica do trabalho. Os capítulos 3, 4 e 5 tratam sobre o desenvolvimento do aplicativo móvel, servidor e *website*, respectivamente. Há uma discussão sobre as conclusões e trabalhos futuros nos capítulos 7 e 6.



## 2 Fundamentação Teórica

Este capítulo discute sobre a linguagem de programação utilizada para o desenvolvimento, junto com os *frameworks*, bibliotecas e arquitetura de banco de dados utilizados.

### 2.1 Frameworks

#### 2.1.1 Ionic

O aplicativo móvel deste TCC foi desenvolvido utilizando-se o *framework* Ionic, que é um plataforma *open-source* para desenvolver aplicativos *mobile* nativos utilizando linguagens de programação *web*. O Ionic faz uso principalmente do Javascript, utilizando o node.js ([NODE.JS, 2017](#)) para o desenvolvimento no *desktop*, [Angular](#) para o desenvolvimento da Interface do Usuário e *plugins* do [Cordova](#) para acessar funções do *hardware* dos dispositivos móveis e para exportar o aplicativo para qualquer Sistema Operacional Móvel em uso atualmente. O Ionic está na versão 3.x.x, sendo atualizado constantemente e contando com uma grande comunidade de colaboradores. É um dos *frameworks* mais utilizados para desenvolver aplicativos nativos, contando com mais de 4 milhões de aplicativos desenvolvidos e 5 milhões de desenvolvedores. Além de fornecer a estrutura para o desenvolvimento, o Ionic conta com uma documentação robusta ([DRIFTY, 2017b](#)) e elementos da Interface do Usuário pré-definidos. ([DRIFTY, 2017a](#))

#### 2.1.2 Cordova

Cordova é um outro *framework open-source* para o desenvolvimento de aplicativos móveis nativos utilizando-se Javascript ([APACHE, 2017](#)). Deste *framework*, serão utilizados somente os plugins de comunicação com o *hardware* dos dispositivos móveis, como por exemplo o acesso ao *Bluetooth* e *Wi-fi*.

#### 2.1.3 Angular

Angular é um *framework* para o desenvolvimento de aplicações em HTML ([MOZILLA, 2017](#)) e Javascript. Os aplicativos são escritos em *templates* HTML estilizados pela sintaxe do Angular, junto com classes de *componentes* que analisam e modificam os *templates* e adicionando lógica nos *serviços*, sendo estes serviços e componentes armazenados em *módulos* ([GOOGLE, 2017a](#)). O aplicativo é então apresentado em um *browser*, por meio de um *bootstrap* do módulo principal (*root*), respondendo as ações do usuário de acordo com as instruções passadas nos módulos. No Ionic, o Angular é um dos principais recursos utilizados, pois cuida de

toda a parte dos *templates* HTML e criação dos componentes, serviços e módulos. Atualmente está na versão 4.

## 2.2 Javascript

A seguir, serão elucidados os recursos e bibliotecas relacionadas a linguagem de programação Javascript.

### 2.2.1 Typescript

Typescript é uma linguagem de programação *open-source* criada e mantida pela Microsoft. É uma superclasse do Javascript, que adiciona características opcionais à linguagem que traz ferramentas para um melhor desenvolvimento em grandes projetos ([MICROSOFT, 2017](#)). Dentre essas características, pode-se citar:

- a) Tipagem estática, que permite uma checagem do tipo em tempo de compilação.
- b) Módulos e *namespaces*.
- c) Tipos genéricos
- d) Classes, interfaces e outras características de Programação Orientada a Objetos.
- e) Tuplas, *Mixins* e *Enums*.

Diversas aplicações e *frameworks* de Javascript dão suporte para o Typescript. O Angular, a partir da versão 2, permite a escrita de aplicações em Typescript.

### 2.2.2 AngularFire

AngularFire é uma biblioteca *open-source* para facilitar o uso do banco de dados Firebase ([GOOGLE, 2017f](#)) junto com o *framework* Angular. Permite uma melhor sincronização de dados e monitoração da autenticação do usuário em tempo real ([GOOGLE, 2017b](#)).

### 2.2.3 RxJS

ReactiveX (popularmente conhecida como RxJS) é uma biblioteca para programação assíncrona, trazendo um novo tipo de paradigma de programação, conhecido como "programação reativa". A biblioteca permite o envio assíncrono de dados através de *Observables* ([REACTIVEX, 2017](#)), que são *stream* de dados enviados em um intervalo de tempo. A partir disso, um Observador (*Observer*) pode escutar os dados dessa *stream* através da função *Subscribe*.

Além disso, a biblioteca permite diversas funções de manipulação e junção de *Observables* e também a integração com outras soluções de programação assíncrona. Os dados do banco de dados recebidos através do AngularFire utilizam o ReactiveX para uma melhor performance.

## 2.2.4 Moment

Moment.js é uma biblioteca *open-source* para tratar operações de tempo e calendário. É uma poderosa alternativa a Classe *Date* do Javascript, permitindo diversos tipos de manipulação, validação e amostragem de datas para aplicações Javascript ([MOMENT.JS, 2017](#)).

No sistema do Restaurante Universitário, o tempo é fator crucial para as operações do sistema. É necessário armazenar a data e hora das refeições de maneira que se possa validar as operações do usuário, armazenar de maneira eficiente e mostrar as datas em um formato legível. Portanto, a biblioteca Moment será de grande importância para o desenvolvimento do projeto.

## 2.2.5 ECMAScript e Programação Assíncrona

ECMAScript é uma linguagem de programação de *scripts*, o qual é base para o Javascript, que é uma das especificações da linguagem, mais especificamente o padrão ECMA-262. O padrão traz especificações e convenções para a linguagem, que é amplamente aceita e usada pela comunidade.

A última versão, ECMAScript 6 traz diversas especificações para a linguagem, as quais pode-se destacar algumas que serão amplamente utilizadas no projeto:

- a) Variáveis com escopo reduzido e constantes (*let* e *const*).
- b) "*Arrow functions*" que permitem uma melhor sintaxe na hora da criação de funções anônimas, imitando a função do paradigma de programação funcional.
- c) Interpolação de *strings*
- d) Manipulação e combinação de *Promises* ([FLANAGAN, 1998](#))

## 2.3 Banco de Dados

### 2.3.1 Arquitetura

A arquitetura de banco de dados escolhida para o projeto foi o banco de dados não-relacional (NoSQL). Os motivos para utilizar esta arquitetura são a falta de complexidade na criação e manutenção dos dados e uma melhor performance e escalabilidade quando comparado à arquiteturas SQL ([STRAUCH; KRIHA, 2017](#)).

O serviço online de banco de dados escolhido é o Firebase, um Banco de dados não-relacional de documentos, o qual todos os dados são armazenados em uma JSON, o que é facilmente acessado através de sua API ([GOOGLE, 2017f](#)), além de fornecer uma eficiente solução para cadastro de usuários e outras soluções para aplicações web que serão elucidadas nas próximas seções.

O código 1 mostra como a arquitetura do projeto ficou organizada no banco de dados Firebase.

Código 1 – Modelo do banco de dados em JSON.

```

1  {
2    "refeicoes" : {
3      "1" : {
4        "base1" : "Macarrao",
5        "base2" : "Arroz",
6        "guarnicao" : "Brocolis",
7        "principal" : "Carne de panela",
8        "salada1" : "Vinagrete",
9        "salada2" : "Alface",
10       "sobremesa" : "Bolo",
11       "suco" : "Laranja",
12       "timestamp" : 1497200458000,
13       "users" : {
14         "123" : true
15       },
16       "usersVeg" : {
17         "789" : true
18       },
19       "usersVeg_count" : 1,
20       "users_count" : 1,
21       "vagas" : 348,
22       "veg" : "Proteina da Soja"
23     }
24   },
25   "users" : {
26     "456" : {
27       "created_at" : 1494636721076,
28       "email" : "cieslakluiz@gmail.com",
29       "name" : "Luiz Fernando da Silva Cieslak",
30       "ra" : "131022776",
31       "refeicoes" : {
32         "123" : true
33       },
34       "saldo" : 12,
35       "updated_at" : 1494636721076,
36       "veg" : false

```

```

37     },
38     "789" : {
39         "created_at" : 149423314076,
40         "email" : "fulano@gmail.com",
41         "name" : "Fulano da Silva",
42         "ra" : "131302922",
43         "refeicoes" : {
44             "1" : true
45         },
46         "saldo" : 1,
47         "updated_at" : 149423314076,
48         "veg" : true
49     }
50 }
51 }

```

No caminho de */refeicoes/* serão armazenadas as refeições com as suas devidas informações e no caminho *users/* os usuários cadastrados, mostrados no código 1. Neste exemplo, "123" é a chave da refeição gerada automaticamente pelo Firebase, assim como a chave "456" e "789" são também geradas pelo serviço.

Note que, como o banco de dados não é relacional, é necessário ter uma duplicidade nos dados com uma lista de usuários no caminho de */refeicoes/* e uma lista de refeições no caminho de */users/*.

### 2.3.2 Regras

O *Firebase* disponibiliza a customização das regras de acesso e escrita no banco de dados. O sistema aceita um arquivo *.json* com as configurações desejadas ([GOOGLE, 2017g](#)). A configuração utilizada é permissão e escrita no banco de dados somente para usuários autenticados e a adição de três índices para a ordenação e consulta:

- a) campo *timestamp* dentro de */refeicoes/*
- b) campo *ra* dentro de */users/*
- c) campo *value* dentro de */refeicoes/queue/*

## 3 Aplicativo Móvel

### 3.1 Pré-Projeto

#### 3.1.1 Pesquisa

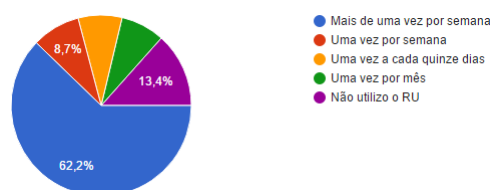
Para poder validar os problemas descritos na Introdução, foi feita uma pesquisa com os alunos da Universidade sobre as questões relativas ao sistema atual do Restaurante Universitário. A pesquisa foi feita através do *Google Forms* e contou com 127 respostas de alunos. A validação dos resultados foi feita através do Registro Acadêmico de cada aluno.

##### 3.1.1.1 Respostas

A seguir, serão mostrada as respostas dos alunos no formulário.

a) Você utiliza o Restaurante Universitário com qual frequência?

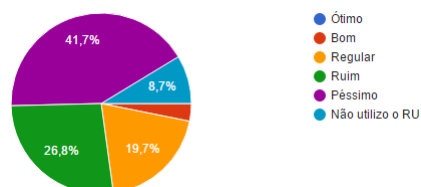
Figura 1 – Frequência de utilização do Restaurante Universitário.



Fonte: Elaborada pelo autor.

b) Como você avalia o sistema atual do RU em geral?

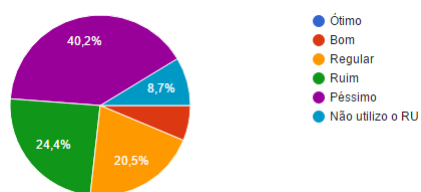
Figura 2 – Avaliação geral do sistema do RU.



Fonte: Elaborada pelo autor.

c) Como você avalia o sistema atual de transferência de refeições?

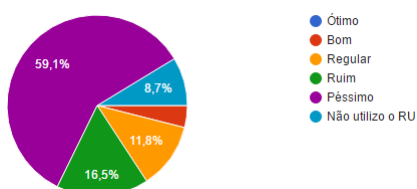
Figura 3 – Avaliação geral do sistema de transferências do RU.



Fonte: Elaborada pelo autor.

d) Como você avalia o sistema atual de compras pelo site ou pelo bauRU?

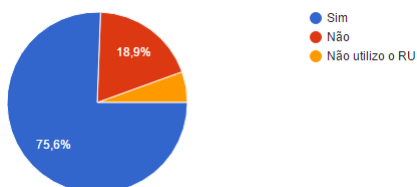
Figura 4 – Avaliação do *website* de compras e do aplicativo bauRU.



Fonte: Elaborada pelo autor.

e) Você acha que o sistema atual é difícil de usar e/ou muito burocrático?

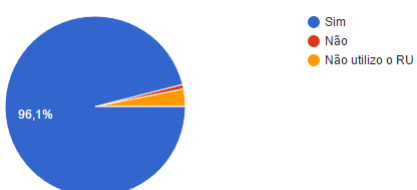
Figura 5 – Avaliação da burocracia e dificuldade de uso do sistema do RU.



Fonte: Elaborada pelo autor.

f) Você acredita que há pessoas que deixam de utilizar o restaurante devido aos problemas do sistema atual?

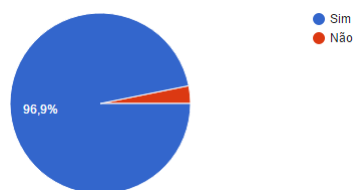
Figura 6 – Opinião dos alunos se há desistência do uso do RU devido aos problemas.



Fonte: Elaborada pelo autor.

g) Você utilizaria o RU com mais frequência caso esses problemas fossem resolvidos?

Figura 7 – Reincidência do uso do RU caso os problemas fossem resolvidos.



Fonte: Elaborada pelo autor.

Por fim, foi perguntado se os usuários do RU tinham alguma sugestão para a melhoria do sistema. Muitas pessoas frisaram o problema do servidor atual ficar sobrecarregado nas horas de início de compra de refeição, informando que o sistema é instável e sem capacidade para aguentar a demanda atual. Muitas pessoas também pediram por mais vagas e a abertura de refeições noturnas, uma vez que mais da metade dos cursos da universidade são noturnos. Já outros usuários pediram por ter um aplicativo móvel que facilite a compra de refeições.

Analisando as respostas obtidas, pode-se concluir que mais de dois terços dos usuários consideram que o sistema é ruim ou péssimo e o considera muito burocrático e quase a totalidade dos usuários que responderam acreditam que alunos deixam de utilizar o sistema devido aos problemas atuais, podendo utilizar mais deste benefício caso o sistema melhorasse. Todas as observações foram lidas e consideradas na confecção deste projeto, para que haja uma melhora na qualidade do serviço prestado pela UNESP.

### 3.1.2 Requerimentos

Antes do projeto começar, houve uma conversa com os funcionários responsáveis pelo funcionamento do Restaurante Universitário. Primeiramente, foi feita uma reunião com o Diretor da Divisão Técnica de Administração do Campus, o Sr. Claudio De Martino, o qual conseguiu elucidar algumas questões relacionadas ao funcionamento interno do sistema atual. De Martino nos informou que os criadores do sistema atual é a Seção Técnica de Informática do Campus e que ela não tem um controle de Qualidade de Software nem tem conhecimento da avaliação do sistema por parte dos alunos, porém sabe que o sistema recebe bastante reclamações. Este assunto é deveras preocupante, uma vez que não se tem o *feedback* do usuário alvo do sistema e não se tem uma comunicação com o mesmo para que haja melhorias.

De Martino também nos disse que o STI não possui um monitoramento das compras e reservas para a avaliação de uma expansão no número de vagas nas refeições, não obtendo um planejamento para tal, sendo que é evidente a necessidade de expansão devido a discrepância dos número de alunos da graduação (2000) para o número de vagas oferecidas (350).

Uma outra reunião foi realizada com a Sra. Thayane Carla Rodrigues Costa Caobianco, Supervisora da Seção Técnica de Nutrição, sobre as questões de preparação da refeição e do cardápio semanal. Caobianco nos informou que o cardápio é decidido em uma média de 40



dias antes do dia e que pode sofrer alterações durante esse período. As refeições começam a ser preparadas a partir das 7h do mesmo dia, porém com uma pré preparação às 16h do dia anterior, dependendo do tipo de refeição a ser servida.

Caobianco também informou que o sistema atual gera um relatório da refeição, informando quantas pessoas a compararam, diferenciando-as entre vegetarianos e não vegetarianos, para que haja uma preparação de acordo. Atualmente, a fila de espera é feita por ela em um caderno às 11h30, anotando os nomes das pessoas que desejam entrar na fila. Então, às 13h30, estas pessoas são anunciadas por ordem de chegada e é permitida a entrada delas no RU.

Em relação às estatísticas dos dados, ela informou que seria interessante guardar o número de desistências da refeição, já que na maioria dos dias o RU opera em lotação máxima. Além disso, alguns outros fatores podem ajudar na análise dos dados:

- a) Tipo da refeição servida
- b) Dia da semana
- c) Períodos de férias, feriados, greves ou outros eventos
- d) Clima

Após obter as informações, foi feita a análise de requisitos do sistema e aplicativo móvel, gerando assim as *User Stories* e o *Product Backlog* (SCHWABER; BEEDLE, 2001)

### 3.1.3 O problema do pagamento online

Assim que o projeto foi realizado, um dos objetivos era que houvesse um tipo de pagamento online embutido no aplicativo móvel, para que o usuário pudesse recarregar o seu saldo. Assim sendo, duas opções foram consideradas:

- a) Pagamento através de uma API, como o PagSeguro
- b) Pagamento através do cartão de crédito

Porém, nenhuma das duas opções se mostraram viável para o projeto. Na primeira opção, o dinheiro é mantido pela empresa criadora da API, o que poderia prejudicar a operação do RU. Já a segunda opção demandaria muito tempo para garantir uma operação eficiente e segura, além do fato que nem todas os alunos possuem esse tipo de pagamento.

### 3.1.4 Divisão do trabalho

Na fase de desenvolvimento, o *Product Backlog* foi dividido em *Sprints* de acordo com a prioridade e tamanho.

Todo o trabalho realizado foi *open-source* dentro da licença *Apache License 2.0* e está disponível no [GitHub](#).

## 3.2 Angular Services

O *framework Angular* permite a criação de serviços, que são injetados no construtor dos componentes para serem usados. O código 2 mostra um exemplo de como deve ser feita essa injeção.

Código 2 – Injeção dos serviços no *constructor*.

```
1 constructor(private db: AngularFireDatabase, private time:
    TimeService,
2 private _user: UserService, private _auth: AuthService) {
3 }
```

### 3.2.1 AuthService

AuthService é o serviço para autenticação do usuário. A tabela 1 mostra as funções presentes nesse serviço.

Tabela 1 – Funções do AuthService

Função	Descrição	Retorna
authenticated	Verifica se o authState do usuário é valido	boolean
uid	Retorna o uid do usuário cadastrado	string
signInWithEmail	Loga no sistema usando e-mail e senha.	Promise
signOut	Desloga o usuário do sistema.	Promise
signUp	Cria uma conta no sistema	Promise
resetPassword	Reseta a senha do usuário através do e-mail.	Promise

Fonte – Feito pelo autor

### 3.2.2 UserService

UserService é o serviço para realizar operações relacionadas com o caminho */users/* no banco de dados. A tabela 2 mostra as funções presentes nesse serviço.

### 3.2.3 RefeicaoService

UserService é o serviço para realizar operações relacionadas com o caminho */refeicoes/* no banco de dados. A tabela 3 mostra as funções presentes nesse serviço.

### 3.2.4 TimeService

TimeService é o serviço para operações relacionadas a tempo e data, utilizando a biblioteca *Moment*. A tabela 4 mostra as funções presentes nesse serviço.

Tabela 2 – Funções do UserService

Função	Descrição	Retorna
postSignup	Adiciona mais informações ao usuário (como nome e RA) dentro do caminho /users/	Promise
gravatarLink	Retorna a URL da imagem do usuário dentro do banco de dados Gravatar a partir de seu e-mail.	A URL da imagem
userObservable	Retorna a referência do usuário cadastrado no banco de dados.	Observable
history	Retorna a referência do histórico de transações do usuário cadastrado.	Observable
addHistory	Adiciona um novo item no histórico de transações do usuário	Promise
getSaldo	Retorna o saldo do usuário	Promise
isVeg	Retorna a informação se o usuário cadastrado é vegetariano ou não.	Promise
canBuy	Retorna a informação se o usuário pode comprar a refeição.	Observable
canQueue	Retorna a informação se o usuário pode entrar na fila da refeição.	Observable
debitSaldo	Retira uma unidade do saldo do usuário cadastrado.	Promise
incrementSaldo	Adiciona uma unidade do saldo do usuário cadastrado.	Promise
bought	Verifica se o usuário já comprou a refeição.	Promise
isQueued	Verifica se o usuário já está na fila de espera da refeição.	Promise
addToQueue	Adiciona a refeição no caminho de /users/uid/queue	Promise
removeRefeicao	Remove a refeição no caminho de /users/uid/queue	Promise
removeQueue	Remove a refeição no caminho de /users/uid/refeicoes	Promise

Fonte – Elaborado pelo autor

### 3.2.5 ConnectionService

ConnectionService é o serviço que verifica se o usuário possui conexão a internet. Esta operação é importante pois todas as operações realizadas no aplicativo dependem do resultado retornado do *Firebase*, que por sua vez precisa estar conectado. A tabela 5 mostra as funções presentes nesse serviço.

Tabela 3 – Funções do RefeicaoService

Função	Descrição	Retorna
postSignup	Adiciona as informações adicionais do usuário (como nome e RA) dentro do caminho /users/	Promise
gravatarLink	Retorna a URL da imagem do usuário dentro do banco de dados Gravatar a partir de seu e-mail.	A URL da imagem
userObservable	Retorna a referência do usuário cadastrado no banco de dados.	Observable
history	Retorna a referência do histórico de transações do usuário cadastrado.	Observable
addHistory	Adiciona um novo item no histórico de transações do usuário	Promise
getSaldo	Retorna o saldo do usuário	Promise
isVeg	Retorna a informação se o usuário cadastrado é vegetariano ou não.	Promise
canBuy	Retorna a informação se o usuário pode comprar a refeição.	Observable
canQueue	Retorna a informação se o usuário pode entrar na fila da refeição.	Observable
debitSaldo	Retira uma unidade do saldo do usuário cadastrado.	Promise
incrementSaldo	Adiciona uma unidade do saldo do usuário cadastrado.	Promise
bought	Verifica se o usuário já comprou a refeição.	Promise
isQueued	Verifica se o usuário já está na fila de espera da refeição.	Promise
addToQueue	Adiciona a refeição no caminho de /users/uid/queue	Promise
removeRefeicao	Remove a refeição no caminho de /users/uid/queue	Promise
removeQueue	Remove a refeição no caminho de /users/uid/refeicoes	Promise

Fonte – Elaborado pelo autor

Tabela 4 – Funções do TimeService

Função	Descrição	Retorna
isAllowed	Verifica se a operação desejada é possível de acordo com a timestamp da refeição. A regra atual é que as operações podem ser feitas até 1 dia antes da refeição.	boolean
serverTimestamp	Retorna o horário do servidor do Firebase Functions.	Observable
localTimestamp	Retorna o timestamp do dia e horário atual, provido pelo Moment.js	Number

Fonte – Elaborado pelo autor

Tabela 5 – Funções do ConnectionService

Função	Descrição	Retorna
isOnline	Verifica se o usuário possui conexão a internet.	Promise

Fonte – Elaborado pelo autor

### 3.3 Manuseio dos Usuários

#### 3.3.1 Login

Figura 8 – Tela de login.

Fonte: Elaborada pelo autor.

A figura 8 mostra a tela inicial do aplicativo, a qual o usuário deve digitar seu email e senha já cadastrados para autenticação. Assim que a autenticação é feita, o usuário é redirecionado para a *homepage*.

O usuário é autenticado através do serviço *auth-service*, o que utiliza os serviços do Firebase, mostrado no código 3. Também há a opção de redefinir a senha utilizando o email do usuário, como mostra o código 3.

### 3.4 Sidemenu

A navegação dentro do aplicativo móvel será feito através de um *Sidemenu*, no qual o template vem disponível no Ionic ([DRIFTY, 2017a](#)).

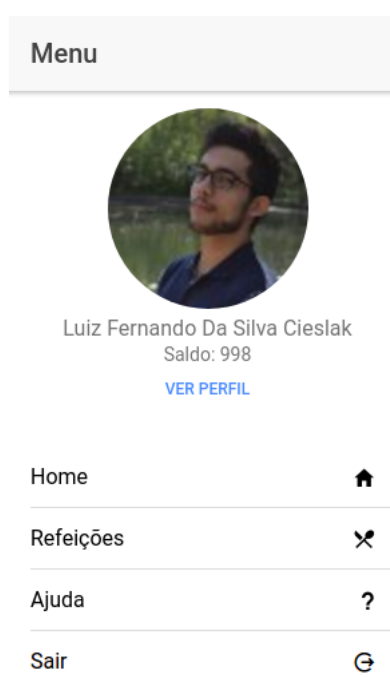
Código 3 – Função de login em *auth-service*.

```

1  /**
2   * Sign in into Firebase using Email.
3   * @returns Firebase Promise.
4   */
5  signInWithEmail(email: string, password: string): firebase.
    Promise<any>{
6      return this.afAuth.auth.signInWithEmailAndPassword(email,
        password);
7  }

```

Figura 9 – Tela do Sidemenu.



Fonte: Elaborada pelo autor.

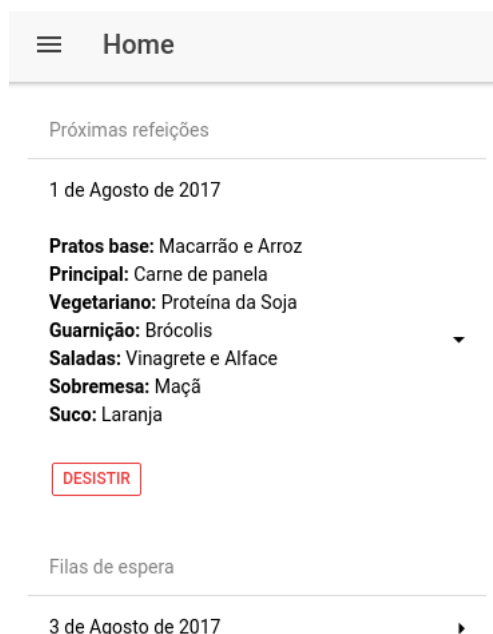
Como mostrado na figura 9, o cabeçalho do *sidemenu* possui um imagem do usuário disponível através do sistema Gravatar (GRAVATAR, 2017), além do seu nome e saldo disponível. Após este cabeçalho, aparecerá as opções de navegação

### 3.5 Homepage

A página inicial do aplicativo mostra um resumo das ações já feitas pelo usuário, como as refeições compradas e as refeições em que ele está na fila de espera.

É possível expandir as informações desta refeição, mostrando as suas informações, com a opção de desistir da mesma caso seja necessário, como mostra a figura 10

Figura 10 – Tela inicial do aplicativo.

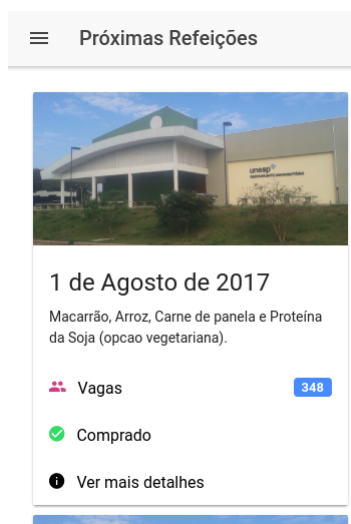


Fonte: Elaborada pelo autor.

### 3.6 Compra de Refeições

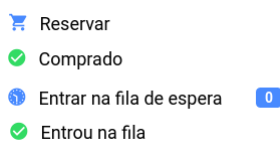
As refeições serão mostradas em uma lista como mostra a figura 11. Para cada elemento da lista, um *ion-card* (DRIFTY, 2017a) é criado contendo uma imagem da refeição, o número de vagas e ação a ser feita logo abaixo do número de vagas é determinado dinamicamente, de acordo com as informações fornecidas pelo banco de dados. A figura 12 mostra as opções que podem aparecer para o usuário, sendo que em cada opção é mostrada uma janela de confirmação (figura 13). Caso faça a reserva de uma refeição ou entre na lista de espera, o usuário é redirecionado para a [Homepage](#).

Figura 11 – Tela da lista de refeições.



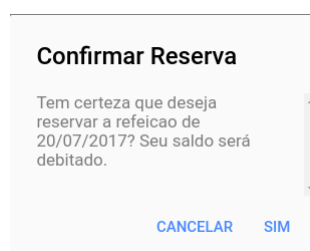
Fonte: Elaborada pelo autor.

Figura 12 – Possíveis opções para a lista de refeições.



Fonte: Elaborada pelo autor.

Figura 13 – Alerta de confirmação da compra de refeição.



Fonte: Elaborada pelo autor.

Caso a opção "ver detalhes" seja selecionada, a página de detalhes da refeição é aberta. O objetivo desta página é trazer detalhes adicionais da refeição que não foram colocados na página anterior, como saladas, guarnição, sobremesa e suco.



Figura 14 – Tela dos detalhes da refeição.



Fonte: Elaborada pelo autor.

A figura 14 mostra como a pagina está estruturada. A imagem da refeição possui um efeito "paralaxe", isto é, a imagem desce lentamente conforme a página desce com as informações.

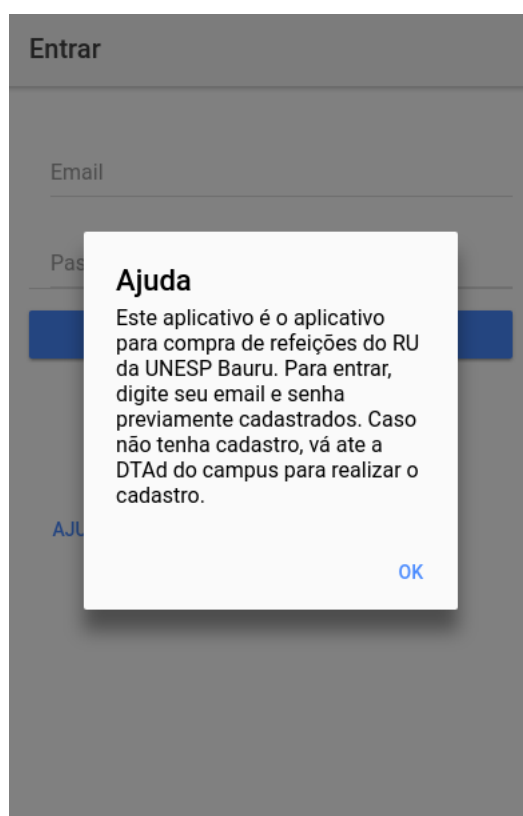
## 3.7 Ajuda e Sobre o projeto

Para ajudar na compreensão e na facilidade de uso, foram criadas duas seções de ajuda dentro do aplicativo móvel.

### 3.7.1 Ajuda na página de Login

Esta seção de ajuda auxilia o usuário que se depara com o aplicativo móvel sem saber como fazer para se cadastrar e utilizá-lo. A figura 15 ilustra como esta ajuda é mostrada para o usuário.

Figura 15 – Seção de ajuda na página de login.

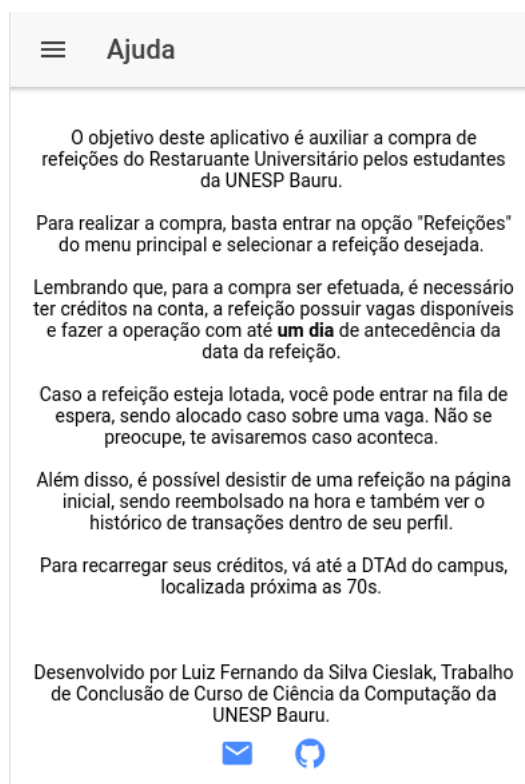


Fonte: Elaborada pelo autor.

### 3.7.2 Ajuda no sidemenu

A figura 16 mostra a página de ajuda criada para auxiliar o usuário sobre todo funcionamento do aplicativo móvel, desde a compra de refeições até a checagem do histórico de transações.

Figura 16 – Seção de ajuda acessível pelo sidemenu.



Fonte: Elaborada pelo autor.

## 4 Servidor

### 4.1 Motivos

Toda aplicação *web* ou *mobile* precisa ter uma parte de sua operação armazenada em um servidor, muitas vezes chamado de *back-end*, que mantém uma comunicação com o banco de dados e realiza operações automatizadas e longe do escopo do cliente. Este tipo de arquitetura evita vulnerabilidades de segurança, como a falsificação de dados e a engenharia reversa de funções que executam no lado do cliente (GOOGLE, 2017c).

Para o presente projeto de TCC, o tipo de serviço selecionado para realizar esta função é o *Cloud Functions for Firebase* (GOOGLE, 2017d). Os motivos para tal escolha serão mostrados a seguir:

- a) Integração com o banco de dados Firebase
- b) Escrita das funções em *node.js* sem a necessidade de criar um novo servidor
- c) Sem necessidade de manutenção de um servidor nem o planejamento de escalabilidade
- d) Possui gatilhos para escritas no banco de dados, bem como gatilhos externos, utilizando o protocolo HTTP
- e) Mantém os dados e funções longe do alcance do usuário, aumentando-se a segurança.

Nas seções a seguir, serão mostradas as funções que foram desenvolvidas para o projeto com as suas respectivas descrições e amostragem de código.

### 4.2 Cron Jobs

Uma das funções que o *back-end* tem de fazer é fechar a fila de espera das refeições após o término da mesma, reembolsando todos os usuários que não conseguiram uma vaga. Essa função deve ser executada todos os dias da semana em um mesmo horário. Portanto, para realizar tal função, será necessário o auxílio de um *software* chamado *Cron*, que planeja tarefas em intervalos pré-programados.

O *Firebase Functions* não possui o suporte nativo para planejar tarefas com o *cron*. Portanto, o que é recomendado por eles é que se faça o planejamento em um outro serviço, para que dispare requisições HTTP no intervalo desejado para a função desejada dentro do *back-end*.

Para o presente projeto, será criado um servidor no *Heroku* ([HEROKU, 2017](#)) utilizando a linguagem de programação *Python* e a biblioteca *APScheduler* ([GRÖNHOLM, 2017](#)). O código 4 mostra um exemplo de planejamento de tarefas utilizando este serviço.

Código 4 – Exemplo de função do APScheduler.

```

1 @sched.scheduled_job('cron', day_of_week='mon-fri', hour=13)
2 def scheduled_job():
3     print('This job is run every weekday at 1pm.')
4     # Run the http request
5     payload={'key': cron_key, 'rid': 0}
6     res = requests.get('https://us-central1-unespru.
7         cloudfunctions.net/queueCleanup', params=payload)
8     print(res.text)

```

Por questões de segurança, uma chave alfanumérica foi criada e depois inserida nas configurações do Firebase pelo comando *firebase functions:config:set cron.key="KEY"*, sendo necessária para a execução da função.

## 4.3 Funções

Segundo a documentação do Firebase Functions ([GOOGLE, 2017d](#)), as funções devem ser feitas seguindo o *framework node.js* e serem enviadas através dos comandos do terminal.

Neste projeto, foi feita uma adaptação para que as funções possam ser escritas com o *Typescript* devido ao seu melhoramento para o Javascript.

### 4.3.1 Horário do servidor - utc()

Essa função é um gatilho de requisição HTTP que tem como objetivo retornar o horário atual no formato *UTC* do servidor. Esta função será utilizada dentro da *TimeService* para realizar as operações de tempo utilizando o horário do servidor através da biblioteca *Moment*, evitando assim qualquer tipo de falsificação ou engenharia reversa do lado do usuário.

### 4.3.2 Reembolso do usuário - removeRefeicao()

Esta função é ativada através de um gatilho no banco de dados, que verifica quando um usuário desiste de uma refeição na *Homepage*, realizando o reembolso do mesmo.

#### 4.3.2.1 Alocar usuário da fila - assignNextUserFromQueue()

Seria necessário criar uma função para o servidor alocar o próximo usuário da fila para a refeição caso abrisse uma vaga. Porém, uma vaga sempre é aberta quando um usuário desiste

de uma refeição. Portanto, as operações realizadas para a alocação são feitas dentro da função *removeRefeicao()*.

### 4.3.3 Limpeza da fila - *queueCleanup()*

Como explicado na seção 4.2, esta função será executada após o fechamento do Restaurante Universitário, retirando os usuários da fila da refeição e reembolsando-os.

## 4.4 Notificações

As notificações de aplicativos móveis são um importante meio de comunicação entre o próprio aplicativo e o usuário final. A função delas é servir como um aviso de algo que aconteceu ou está para acontecer no aplicativo, se tornando assim uma maneira simples e eficaz de interação. (GOOGLE, 2017e)

### 4.4.1 FCM

Neste presente projeto, foi-se utilizado o serviço de mensagens do *Firebase Cloud Messaging* como a interface para o envio de notificações. O serviço já é integrado com [obanco de dados](#), tornando assim mais fácil a criação de notificações baseadas em eventos do *Firebase*.

Para enviar notificações aos usuários finais, é necessário uma configuração dentro do console do *Firebase* com os nomes dos pacotes dos aplicativos *Android* e *iOS*. O envio da notificação pode ser feita tanto no console ou por uma requisição *HTTP*, sendo que nesta última opção, deve-se informar a chave do servidor disponibilizada pelo *Firebase*.

O *Firebase* permite vários tipos de envio de notificações (GOOGLE, 2017e). Uma notificação pode ser enviada para somente um usuário, para um tópico de mensagens no qual os usuários já estarão inscritos ou para uma combinação de tópicos.

### 4.4.2 Funções

O aplicativo móvel do RU pode utilizar o serviço de notificações para diversos objetivos. Logo, é listado os que foram desenvolvidos neste projeto:

- a) Avisar que o horário da refeição está próximo para os usuários que a compraram.
- b) Avisar o usuário da fila de espera que conseguiu uma vaga.
- c) Avisar o usuário da fila de espera que não conseguiu uma vaga.
- d) Informar os demais usuários que a próxima refeição está com vagas sobrando.

Os itens *a, b* e *c* da lista acima são notificações de alerta para os usuários que já utilizaram o aplicativo e o último item *d* é uma notificação de retenção de usuários, para que

os mesmos utilizem o aplicativo. Este último item é passível de discussão pois pode ter um efeito negativo nos usuários.

Estes objetivos foram alcançados junto com as funções do *Firebase Functions*. O código 5 mostra como é enviada uma notificação para os usuários utilizando *Typescript* e a figura 17 mostra como essa notificação é recebida pelo usuário.

Código 5 – Função sendNotification

```

1
2 function sendNotification(payload): Promise<any> {
3   return rp({
4     url: 'https://fcm.googleapis.com/fcm/send',
5     method: 'POST',
6     headers: {
7       'Content-Type': 'application/json',
8       'Authorization': 'key=' + environment.fcmKey
9     },
10    body: JSON.stringify(payload)
11  });
12 }

```



Figura 17 – Exemplos de notificações do aplicativo.

Fonte: Elaborada pelo autor.

#### 4.4.3 Plugins do Cordova

Para receber as mensagens em um aplicativo feito em *Ionic* é necessário instalar um plugin do *Cordova* chamado *cordova-plugin-fcm* ([ECHANIQUE, 2017](#)), que permite a criação de notificações através do *FCM*.

Além do recebimento de mensagens, o plugin também permite a inscrição e o cancelamento de tópicos para receber mensagens direcionadas.



## 5 Website para os funcionários

### 5.1 Planejamento

Para que todo o sistema funcione de maneira síncrona com as tarefas e o funcionamento da administração do Restaurante Universitário, é necessário a implementação de uma aplicação para os funcionários realizarem operações relacionadas, como a adição e alteração do cardápio nos dias de refeições e o manuseio dos usuários.

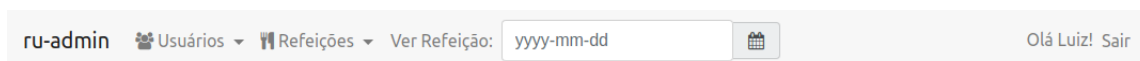
Sendo assim, foi feita a análise de requisitos desta aplicação, gerando assim as *User Stories* e o *Product Backlog* (SCHWABER; BEEDLE, 2001).

A aplicação será feita utilizando-se o [Angular](#) conectado com o banco de dados Firebase utilizando-se a biblioteca [AngularFire](#).

Os funcionários poderão entrar de acordo com as suas credenciais pessoais pré-estabelecidas pelo sistema, cadastradas no banco de dados.

Após o funcionário entrar no sistema, a barra de navegação superior muda, informando qual funcionário está autenticado no momento e as opções disponíveis, como mostra a figura 18

Figura 18 – Barra de Navegação após o login.



Fonte: Elaborada pelo autor.

### 5.2 Manuseio dos Usuários

#### 5.2.1 Cadastro de novos usuários

Um dos serviços disponíveis para o usuário será o cadastro de estudantes para o uso do aplicativo móvel. A figura 19 mostra como o cadastro será feito. A validação do estudante e de seu Registro Acadêmico devem ser feitos de maneira manual pelo funcionário presente no momento do cadastro, uma vez que não há um sistema ou *API* pública que faça esse tipo de validação. A figura 19 mostra o formulário para o cadastro.

Figura 19 – Cadastro de novos usuários.



O formulário, intitulado "Adicionar Usuário", contém os seguintes campos: "Nome", "RA", "Email", "Senha" e "Confirmar Senha", todos representados por retângulos vazios para entrada de texto. Abaixo dos campos de senha, há uma opção "Vegetariano?" precedida por uma caixa de seleção desativada. No canto inferior esquerdo do formulário, há um botão cinza com o texto "Registrar".

Fonte: Elaborada pelo autor.

Após o cadastro do usuário no sistema o mesmo deve confirmá-lo seguindo as instruções de confirmação de cadastro que serão enviadas para o seu email pelo *Firebase*.

Antes de adicionar o usuário, o sistema verifica se já existe um estudante cadastro com o RA fornecido. Caso já tenha, um aviso é mostrado para o funcionário.

### 5.2.2 Adicionar Saldo

O funcionário também poderá recarregar o saldo de um usuário mediante requisição, sendo que este deve realizar o pagamento na hora. A figura 20 mostra como este processo é feito. O recebimento e validação do pagamento deverá ser feito pelo funcionário presente.

Figura 20 – Adicionar saldo para um determinado usuário.



O formulário, intitulado "Adicionar Saldo", contém dois campos: "RA do usuário" e "Quantidade", ambos representados por retângulos vazios para entrada de texto. Abaixo do campo "Quantidade", há um botão cinza com o texto "Registrar".

Fonte: Elaborada pelo autor.

## 5.3 Cardápio

### 5.3.1 Adicionar Refeição

A figura 21 mostra o formulário para a adição de refeições no banco de dados. O banco de dados permite somente uma refeição em cada dia útil, excluindo os finais de semana.

Figura 21 – Cadastro de nova refeição no cardápio.

**Adicionar Refeição**

Data

Prato Base 1

Prato Base 2

Prato Principal

Vegetariano

Guarnição

Salada 1

Salada 2

Sobremesa

Suco

Fonte: Elaborada pelo autor.

É possível que o funcionário tente adicionar uma refeição em uma data na qual já tenha uma. Portanto, para evitar a sobreinscrição de dados, um aviso é mostrado caso ocorra este problema como mostra a figura 22.

Figura 22 – Aviso de refeição existente na data escolhida.

**Confirmar operação - Adicionar refeição** ✕

Erro: Já existe uma refeição para esta data.

Fonte: Elaborada pelo autor.

### 5.3.2 Ver Refeições

O funcionário tem a opção de ver o andamento de uma determinada refeição buscando-a através da data. A figura 23 mostra as informações que são mostradas dentro do *card*.

Figura 23 – Informações da página de detalhes da refeição.

The figure displays three sequential screenshots of a web application's meal details page for the date **19 de Outubro de 2017**.

**Screenshot 1 (Top):** Shows the meal name and ingredients. The tabs are **Cardápio**, **Usuários**, and **Outras Informações**. The meal details are:
 

- 19 de Outubro de 2017**
- Macarrão, Arroz, Carne de panela e Brócolis
- Opção vegetariana: Proteína da Soja
- Salada: Tomate e Alface
- Suco de Laranja e Maça de sobremesa

**Screenshot 2 (Middle):** Shows availability and user statistics. The tabs are **Cardápio**, **Usuários**, and **Outras Informações**. The statistics are:
 

- 19 de Outubro de 2017**
- 349 vagas restantes
- 1 usuários normais e 1 vegetarianos
- 0 pessoas na fila
- Tempo restante para compra acaba em 2 meses

**Screenshot 3 (Bottom):** Shows the creation and modification history. The tabs are **Cardápio**, **Usuários**, and **Outras Informações**. The history is:
 

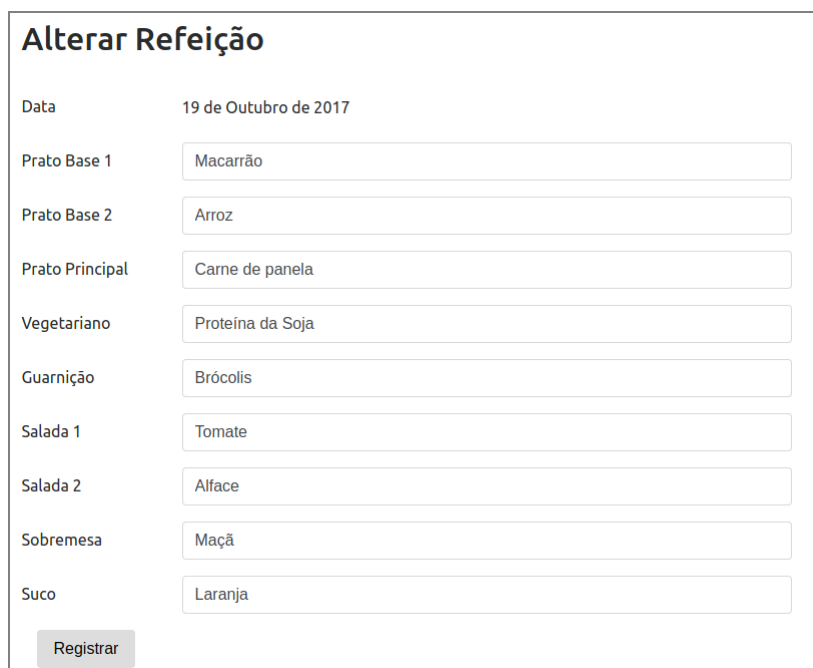
- 19 de Outubro de 2017**
- Adicionado em Qua, 12 de Jul de 2017 às 14:37 por admin@admin.com
- Última modificação em Qua, 12 de Jul de 2017 às 14:37 por admin@admin.com

Fonte: Elaborada pelo autor.

### 5.3.3 Alterar Refeição

É possível também o funcionário alterar algum item de uma refeição que já foi adicionada. A figura 24 mostra como isso pode ser feito. O funcionário terá o mesmo formulário da adição de refeição, podendo alterar os campos que desejar.

Este tipo de operação deve disparar notificações para os usuários que compraram uma das vagas ou estão na fila de espera desta refeição.



**Alterar Refeição**

Data: 19 de Outubro de 2017

Prato Base 1: Macarrão

Prato Base 2: Arroz

Prato Principal: Carne de panela

Vegetariano: Proteína da Soja

Guarnição: Brócolis

Salada 1: Tomate

Salada 2: Alface

Sobremesa: Maçã

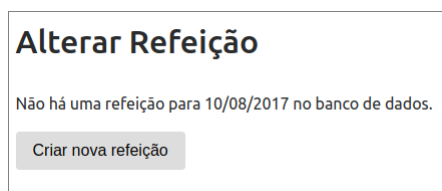
Suco: Laranja

[Registrar](#)

Figura 24 – Informações da página de detalhes da refeição.

Fonte: Elaborada pelo autor.

Caso o funcionário tente alterar uma refeição de uma data que não possua nenhuma, um aviso é mostrado com a opção de adicionar uma refeição para esta data, como mostra a figura 25



**Alterar Refeição**

Não há uma refeição para 10/08/2017 no banco de dados.

[Criar nova refeição](#)

Figura 25 – Aviso de tentativa de alteração de refeição inexistente.

Fonte: Elaborada pelo autor.

## 6 Trabalhos futuros

Como apresentado na seção [3.1.1](#), o sistema desenvolvido através deste Trabalho de Conclusão de Curso é de grande interesse dos usuários do restaurante, melhorando a qualidade do serviço.

O trabalho será apresentado aos responsáveis da Administração Geral do campus a fim de que seja a substituição do sistema atual. O trabalho em si ainda não está totalmente pronto, necessitando de algumas adequações e implementações apresentadas a seguir.

### 6.1 Adequação

O aplicativo móvel e o sistema para os funcionários não possui uma identidade visual alinhada. É necessária tal adequação para que haja uma melhor experiência de usuário e que se seja mais atrativo para a angariação de usuários. Além disso, é importante que seja criado um logotipo para a identificação do sistema.

### 6.2 Implementação

O sistema apresentado carece de uma unidade de testes automatizados, para identificar possíveis problemas que não foram detectados na sua implementação e para que não haja nenhuma falha quando o sistema for posto para o uso. Uma biblioteca que pode realizar esse tipo de operação é a Jasmine ([PIVOTAL, 2017](#)).

A fila de espera precisa ser mais robusta. O sistema atual implementa uma fila simples, na qual o usuário não consegue sair até que o horário da refeição acabe, segurando um crédito do usuário que poderia ser utilizado para realizar outras transações.

É de grande importância também gerar estatísticas dos dados e operações feitas pelo sistema a fim de contribuir com a criação de cardápios e a manutenção do número de vagas máximas para alunos. Os dados armazenados deverão ser utilizados para gerar gráficos aos usuários a fim de que haja uma melhor visualização. Tal implementação pode ser feita com o biblioteca D3.js ([BOSTOCK, 2017](#)).

Em um momento futuro, é preciso que se implemente um *website* para os usuários realizarem as mesmas operações realizadas pelo aplicativo móvel. Apesar de o uso de *smartphones* ser bastante difundido ([COMSCORE, 2017](#)), pode ser que não atinga a totalidade de usuários da instituição. Portanto, é interessante que haja uma outra alternativa para o usuário além do próprio aplicativo.

No momento atual, não existe nenhum canal de comunicação direto entre os usuários e os funcionários do restaurante, fazendo com que haja uma perda nos dois lados: O usuário não consegue avaliar o sistema nem sugerir modificações e os funcionários não obtêm o *feedback* necessário para a manutenção. Portanto, a criação de um meio de comunicação é também de grande importância.

## 7 Conclusão

O Trabalho de Conclusão de Curso apresentado teve como objetivo elucidar para os responsáveis do Restaurante Universitário que é possível ter um sistema implementado utilizando-se tecnologias emergentes, que seja de fácil utilização pelos usuários e que não possua a burocracia e os bloqueios do sistema presente.

O projeto consegue resolver os problemas com o Restaurante Universitário, que estão presentes desde a sua criação. Como apresentado pela seção [3.1.1](#) é evidente a insatisfação dos usuários e há um grande interesse para que estes problemas sejam resolvidos.

É claro que caso for implementado, o sistema estará sob responsabilidade dos administradores. Porém, o objetivo deste projeto é que ele continue sendo *open-source* e que seja mantido pela comunidade, mais especialmente pelos alunos do Bacharelado de Ciência da Computação, podendo até no futuro ser atrelado a programas de extensão da universidade.



# Referências

APACHE. *Documentation - Apache Cordova*. 2017. Disponível em <<https://cordova.apache.org/docs/en/latest/>>. Acesso em: 29 mar 2017.

BOSTOCK, M. *D3.js Data-Driven Documents*. 2017. Disponível em <<https://github.com/d3/d3/wiki>>. Acesso em: 03 nov 2017.

COMSCORE. *Mobile Marketing Statistics 2017*. 2017. Disponível em <<https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>>. Acesso em: 03 out 2017.

DRIFTY. *Ionic Components Documentation*. 2017. Disponível em <<http://ionicframework.com/docs/components>>. Acesso em: 29 mar 2017.

DRIFTY. *Ionic Documentation*. 2017. Disponível em <<https://ionicframework.com/docs>>. Acesso em: 29 mar 2017.

ECHANIQUE, F. *Github - Google FCM Push Notifications Cordova Plugin*. 2017. Disponível em <<https://github.com/fechanique/cordova-plugin-fcm>>. Acesso em: 03 set 2017.

FLANAGAN, D. *JavaScript: The Definitive Guide*. 3rd. ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1998. ISBN 1565923928.

GOOGLE. *Angular Architecture*. 2017. Disponível em <<https://angular.io/guide/architecture>>. Acesso em: 29 mar 2017.

GOOGLE. *angular/angularfire2: The official Angular library for Firebase*. 2017. Disponível em <<https://github.com/angular/angularfire2>>. Acesso em: 29 mar 2017.

GOOGLE. *Cloud Functions for Firebase*. 2017. Disponível em <<https://firebase.google.com/docs/functions>>. Acesso em: 29 mar 2017.

GOOGLE. *Documentation - Firebase Functions*. 2017. Disponível em <<https://firebase.google.com/docs/reference/functions/>>. Acesso em: 29 mar 2017.

GOOGLE. *Firebase Cloud Messaging*. 2017. Disponível em <<https://firebase.google.com/docs/cloud-messaging/>>. Acesso em: 03 set 2017.

GOOGLE. *Firebase Realtime Database*. 2017. Disponível em <<https://firebase.google.com/docs/database>>. Acesso em: 29 mar 2017.

GOOGLE. *Firebase Realtime Database Rules*. 2017. Disponível em <<https://firebase.google.com/docs/database/security/?hl=pt-br>>. Acesso em: 03 nov 2017.

GRAVATAR. *Gravatar - Globally Recognized Avatars*. 2017. Disponível em <<https://en.gravatar.com/site/implement/images/>>. Acesso em: 03 set 2017.

GRÖNHOLM, A. *Advanced Python Scheduler*. 2017. Disponível em <<https://github.com/agronholm/apscheduler>>. Acesso em: 04 nov 2017.

HEROKU. *Python - Heroku Dev Center*. 2017. Disponível em <<https://devcenter.heroku.com/categories/python>>. Acesso em: 03 set 2017.

MICROSOFT. *Documentation - Typescript*. 2017. Disponível em <<https://www.typescriptlang.org/docs/home.html>>. Acesso em: 29 mar 2017.

MOMENT.JS. *Momentjs - Docs*. 2017. Disponível em <<https://momentjs.com/docs>>. Acesso em: 29 mar 2017.

MOZILLA. *HTML5 - Web Developer Guides*. 2017. Disponível em <<https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5>>. Acesso em: 29 mar 2017.

NODE.JS. *Docs - Node.js*. 2017. Disponível em <<https://nodejs.org/en/docs>>. Acesso em: 29 mar 2017.

PIVOTAL. *Jasmine Docs*. 2017. Disponível em: <<https://jasmine.github.io/api/edge/global>>. Acesso em: 03 nov 2017.

REACTIVEX. *ReactiveX - Observable*. 2017. Disponível em <<http://reactivex.io/documentation/observable.html>>. Acesso em: 29 mar 2017.

SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130676349.

STRAUCH, C.; KRIHA, W. *NoSQL databases*. 2017. Disponível em <<http://www.christof-strauch.de/nosql dbs.pdf>>. Acesso em: 29 mar 2017.