

LILIANE RODRIGUES DOS SANTOS

DESENVOLVIMENTO DE UM APLICATIVO MÓVEL PARA
GERENCIAMENTO DE PEDIDOS E MONTAGEM DE MARMITAS POR
CLIENTES DE UM RESTAURANTE NA CIDADE DE TEÓFILO OTONI

FACULDADES UNIFICADAS DE TEÓFILO OTONI
TEÓFILO OTONI – MG
2018

LILIANE RODRIGUES DOS SANTOS

DESENVOLVIMENTO DE UM APLICATIVO MÓVEL PARA
GERENCIAMENTO DE PEDIDOS E MONTAGEM DE MARMITAS POR
CLIENTES DE UM RESTAURANTE NA CIDADE DE TEÓFILO OTONI

Monografia apresentada ao Curso de Sistemas de Informação das
Faculdades Unificadas de Teófilo Otoni, como requisito parcial à obtenção
do título de Bacharel em Sistemas de Informação.

Área de Concentração: Desenvolvimento de Software.

Orientador: Prof. Msc. Amaury Gonçalves Costa.

FACULDADES UNIFICADAS DE TEÓFILO OTONI
TEÓFILO OTONI – MG
2018



FACULDADES UNIFICADAS DE TEÓFILO OTONI
NÚCLEO DE TCC / SISTEMAS DE INFORMAÇÃO
Autorizado pela Portaria 4.012 de 06/123/2004 – MEC

FOLHA DE APROVAÇÃO


A monografia intitulada: *Desenvolvimento de um aplicativo móvel para gerenciamento de pedidos e montagem de marmitas por clientes de um restaurante na cidade de Teófilo Otoni – MG,*


elaborada pela aluna *Liliane Rodrigues dos Santos,*

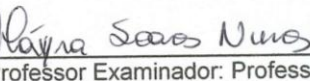
foi aprovada por todos os membros da Banca Examinadora e aceita pelo curso de Sistemas de Informação das Faculdades Unificadas de Teófilo Otoni, como requisito parcial da obtenção do título de

BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Teófilo Otoni, 5 de dezembro de 2018


Professor Orientador: Amaury Gonçalves Costa


Professor Examinador: Wilbert Viana Barbosa


Professor Examinador: Professor Convocado

“Comece fazendo o que é necessário, depois o que é possível, e de repente você
estará fazendo o impossível”.

São Francisco de Assis

LISTA DE FIGURAS

Figura 01: Camadas da engenharia de software	6
Figura 02: Evolução dos requisitos	11
Figura 03: Trello	30
Figura 04: Visual Studio Code	31
Figura 05: Genymotion	32
Figura 06: Astah	33
Figura 07: Node.js	34
Figura 08: NPM	34
Figura 09: Criação do projeto no firebase	41
Figura 10: Configurações do firebase no console	43
Figura 11: Diagrama de Casos de Uso	62
Figura 12: Tela de login	45
Figura 13: Tela de cadastro	46
Figura 14: Tela home e menu lateral	47
Figura 15: Tela de montar marmita	49
Figura 16: Tela de finalizar pedido	50
Figura 17: Tela de envio de sugestões	51
Figura 18: Tela Home administrador	52
Figura 19: Tela de cadastrar administrador	53
Figura 20: Tela de cadastrar categoria	54
Figura 21: Tela de cadastrar itens	55
Figura 22: Tela de editar itens	56
Figura 23: Tela de gerenciar cardápio	57
Figura 24: Tela de cadastrar cardápio	59
Figura 25: Tela de receber pedidos	60
Figura 26: Tela de visualizar feedbacks	61

Figura 27: Estrutura de armazenamento	63
Figura 28: Usuários da autenticação	68
Figura 29: Métodos de login	68
Figura 30: Modelos de e-mail	69
Figura 31: Teste de cadastro de categorias	70
Figura 32: Teste de cadastro e edição de itens	71
Figura 33: Teste cadastro de cardápio	72
Figura 34: Teste do processo de montagem de marmita	73
Figura 35: Teste do recebimento dos pedidos	74

RESUMO

O presente trabalho de conclusão de curso da graduação em Sistemas de Informação tem como título o “Desenvolvimento de um aplicativo móvel para gerenciamento de pedidos e montagem de marmita para clientes de um restaurante na cidade de Teófilo Otoni” e se concentra na área do Desenvolvimento de Software. O trabalho tem por objetivo aperfeiçoar o processo de pedido de marmitas de um restaurante na cidade de Teófilo Otoni, por meio de um aplicativo móvel, a fim de facilitar a oferta de marmitas, promovendo melhorias na organização dos pedidos recebidos e fornecendo aos clientes uma melhor alternativa para realização de pedidos. Para o desenvolvimento desta monografia, foram necessárias a realização de pesquisas de opinião e observações no ambiente do restaurante, e também a pesquisa bibliográfica referente aos assuntos abordados neste trabalho. A conclusão deste trabalho resultou no desenvolvimento de uma aplicação móvel para gerenciamento e realização de pedidos, capaz de atender as dificuldades enfrentadas por um restaurante e seus clientes. O aplicativo tornou-se um meio mais eficiente para o estabelecimento, sendo uma ferramenta propícia e vantajosa para o objetivo traçado.

Palavras-Chave: Aplicativo móvel; Marmitas; Restaurante; Clientes.

SUMÁRIO

INTRODUÇÃO	09
1 REFERENCIAL TEÓRICO	12
1.1 Engenharia de software	12
1.1.1 Modelos de processo de software	13
1.1.1.1 <i>Modelo cascata</i>	14
1.1.1.2 <i>Modelo incremental</i>	14
1.1.1.3 <i>Prototipação</i>	15
1.1.1.4 <i>Modelo espiral</i>	16
1.1.2 Especificação de requisitos	16
1.1.2.1 <i>Requisitos</i>	16
1.1.2.2 <i>Engenharia de requisitos</i>	19
1.1.2.3 <i>Prazos e custos</i>	19
1.1.3 Metodologias de desenvolvimento	19
1.1.3.1 <i>Metodologias tradicionais</i>	20
1.1.3.2 <i>Metodologias ágeis</i>	20
1.1.3.2.1 <i>Scrum Solo</i>	21
1.2 Banco de dados	22
1.2.1 Modelo de dados relacional.....	23
1.2.1.1 <i>SQL</i>	23
1.2.2 Modelo de dados não relacional.....	24
1.2.2.1 <i>Firestore</i>	26
1.2.2.1.1 <i>Firestore Realtime Database</i>	26
1.2.3 SGBD	27
1.3 Desenvolvimento móvel	28
1.3.1 Desenvolvimento nativo	28
1.3.2 Desenvolvimento híbrido	28

1.4 Ambiente web	29
1.4.1 Linguagens de Programação	29
1.4.2 HTML.....	30
1.4.3 CSS.....	30
1.4.4 JavaScript.....	31
1.4.5 TypeScript	32
1.4.6 Frameworks.....	32
1.4.6.1 Angular.....	32
1.4.6.2 Ionic.....	33
1.4.7 Serviço	34
1.4.7.1 Back-end as a Service.....	35
1.4.8 Ambiente de desenvolvimento	35
1.4.8.1 Visual Studio Code.....	35
2 METODOLOGIAS	36
2.1 Metodologia de desenvolvimento	36
2.2 Ambiente web	37
2.3 Ferramentas e aplicações	37
2.3.1 Visual Studio Code	38
2.3.2 Genymotion	38
2.3.3 Astah	39
2.3.4 Node.js	40
2.3.5 NPM	41
3 DESENVOLVIMENTO	42
3.1 Levantamento de requisitos	42
3.2 Preparação do ambiente	44
3.2.1 Instalação do Ionic	44
3.2.2 Criação do projeto	45
3.2.3 Instalação do firebase	45
3.2.4 Pacote e versões.....	46
3.2.5 Conexão com o Firebase	48
3.3 O aplicativo	50
3.3.1 Diagrama de Casos de uso	51
3.4 Construção da aplicação	52
3.4.1 Login.....	52

3.4.2 Cadastro de usuário	53
3.4.3 Tela inicial e de perfil.....	54
3.4.4 Montagem de marmita.....	56
3.4.5 Finalizar pedido	57
3.4.6 Tela de Enviar sugestões	58
3.4.7 Tela inicial administrador	59
3.4.8 Cadastrar administrador	60
3.4.9 Cadastrar categoria	61
3.4.10 Manipulação de itens.....	62
3.4.11 Gerenciar cardápio	64
3.4.12 Cadastrar cardápio	66
3.4.13 Receber pedidos	67
3.4.14 Visualizar feedbacks.....	68
3.4.15 Banco de dados	69
3.4.15.1 <i>CRUD</i>	70
3.4.15.2 <i>Autenticação</i>	71
4. TESTES E RESULTADOS	77
CONCLUSÃO	86
REFERÊNCIAS.....	89
APÊNDICE I	92

INTRODUÇÃO

O presente trabalho de conclusão de curso, concentrado na área do Desenvolvimento de Software, tem como tema o “Desenvolvimento de um aplicativo móvel para gerenciamento de pedidos e montagem de marmitas por clientes de um restaurante na cidade de Teófilo Otoni”.

O objetivo principal deste trabalho é aperfeiçoar o processo de pedido de marmitas em um restaurante na cidade de Teófilo Otoni, desde a solicitação do cliente até a sua entrega, através do desenvolvimento de um aplicativo móvel que facilite e proporcione melhorias no controle dos pedidos recebidos pelos clientes, oferecendo aos mesmos uma nova possibilidade para realização de pedidos.

Atualmente, a busca por soluções que possibilitam o aperfeiçoamento de rotinas e auxiliam na realização de tarefas é crescente, principalmente para empresas que procuram se atualizar em um mercado que está cada vez mais informatizado. Entre meio à estas soluções estão a utilização de aplicativos móveis, em que devido ao seu crescimento constante, acabam se tornando uma aposta de investimento para muitas empresas.

Na área da alimentação, observa-se a adoção de aplicações que auxiliam tanto clientes como restaurantes em suas tarefas. Além da otimização e da facilidade que proporciona, o fato de poder usar esse tipo de tecnologia é de grande importância, visto que a mesma pode estimular o interesse do cliente em uma compra, além de auxiliar na rotina da empresa com as funcionalidades que oferecem.

O restaurante Espaço Paladar trabalha hoje com todos os seus processos de forma manual. Sempre que os clientes entram em contato com o estabelecimento, é preciso que os funcionários informem a eles o cardápio do dia para que façam as escolhas na montagem da marmita. Nesse processo, é preciso transcrever o pedido

no papel para repassar a cozinha fazer o preparo. O processo de pedido de marmita pelos clientes é realizado tanto pessoalmente como por telefone e redes sociais.

Devido a este procedimento repetitivo e a alta demanda que o restaurante possui, o mesmo acaba passando por dificuldades para se organizar e manter o controle da quantidade de solicitações que recebe. Desse modo, a probabilidade de erros cresce e aumenta o tempo de preparo das marmitas e demais atendimentos que são realizados.

Por motivos assim surge a necessidade de investir em uma solução que possa automatizar o processo de pedido de marmitas, de modo que possibilite alcançar os clientes e os ofereça uma forma mais eficiente de fazer o pedido, e evite ao restaurante todos os problemas relacionados ao recebimento do pedido, ao atendimento e a entrega da marmita.

Com base neste cenário e tendo em vista o objetivo traçado para este projeto, foi levantada a seguinte pergunta problema a ser trabalhada nesta monografia: “De que maneira a utilização de um aplicativo móvel poderia aperfeiçoar o processo de montagem e pedido de marmitas pelos clientes de um restaurante?”.

Para responder à pergunta problema levantada, foram estabelecidas as seguintes hipóteses a serem validadas no trabalho:

H0: A utilização de um aplicativo móvel não aperfeiçoaria o processo de montagem e pedido de marmitas pois o restaurante consegue ter o controle dos pedidos recebidos, sem haver necessidade de uma ferramenta para essa função;

H1: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas, mas não seria aceita na rotina do restaurante e seus clientes;

H2: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas pois auxiliaria na organização e controle dos pedidos de marmitas recebidos pelos clientes;

H3: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas pois teria uma boa aceitação pelo restaurante e seus clientes, melhorando a oferta de marmitas.

O objetivo geral estipulado para este trabalho é: “Aperfeiçoar o processo de pedido de marmitas em um restaurante da cidade de Teófilo Otoni através do desenvolvimento de um aplicativo móvel, a fim de promover melhorias no gerenciamento dos pedidos recebidos”. Para atingi-lo, foram estabelecidos os

seguintes objetivos específicos: Identificar os problemas relacionados à montagem e pedido de marmitas que acarretam uma ineficiência na oferta desse serviço; Avaliar o processo de montagem e pedido de marmita, afim de analisar as melhores formas para que o pedido seja recebido corretamente; Estudar as melhores práticas para desenvolvimento do aplicativo, de modo que o mesmo seja simples, interativo e de fácil utilização pelos clientes e restaurante; Analisar aplicações semelhantes no mercado, afim de aprimorar as funcionalidades a serem desenvolvidas no aplicativo proposto.

Quanto aos fins esta pesquisa é descritiva e exploratória, pois tem por finalidade estudar um determinado problema de pesquisa através da coleta de dados e requisitos, a fim de identificar os problemas enfrentados e encontrar as principais causas das suas ocorrências. Quanto aos meios, foram utilizadas pesquisas de opinião e bibliográfica para analisar todos os procedimentos relacionados com o objetivo do projeto e elaborar a monografia. A pesquisa utilizou do método indutivo e os resultados obtidos foram através do método qualitativo com o uso de questionário.

A monografia está dividido em 4 capítulos: o primeiro é o referencial teórico, responsável por todo o embasamento conceitual das áreas envolvidas, como Engenharia de *Software*, Banco de Dados e Programação Web; o segundo apresenta a metodologia, que são todas as tecnologias escolhidas para o desenvolvimento da aplicação proposta e as ferramentas utilizadas; o terceiro capítulo traz o desenvolvimento, apresentando a preparação do ambiente, descrição do aplicativo e exibição das telas e funcionalidades de cada uma delas; o quarto capítulo, sendo o último, apresenta os testes realizados e resultados obtidos com o desenvolvimento da aplicação móvel.

1 REFERENCIAL TEÓRICO

1.1 Engenharia de software

A engenharia de *software* é fundamental para a construção de qualquer tipo de *software*. Ela é a base para garantir um desenvolvimento confiável, de forma organizada e que o mesmo tenha como garantia a sua qualidade. Sommerville (2011, p.5) ensina que a engenharia de *software* “é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de *software*, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado”.

Sommerville (2011, p.3) também explica que a “engenharia de *software* não se trata apenas do programa em si, mas de toda a documentação associada e os dados de configurações necessários para fazer esse programa operar corretamente”. E para atingir esse objetivo, a engenharia de *software* se baseia em quatro pilares, conhecidos como camadas, sendo elas o foco na qualidade, processo, métodos e ferramentas.

A principal abordagem da engenharia de *software* deve ser sustentada com o foco na qualidade (PRESSMAN, 2011, p.39). A qualidade de um *software* tem total importância no produto final desenvolvido. Sua gestão, conforme afirmado por Pressman (2011, p.39), “promovem uma cultura de aperfeiçoamento contínuo de processos, e é esta cultura que, no final das contas, leva ao desenvolvimento de abordagens cada vez mais efetivas na engenharia de *software*”.

Na mesma página, Pressman ensina que a camada de processo é a que sustenta a base da engenharia de *software*, pois é ela a responsável por controlar todas as outras camadas e também garantir que a entrega da aplicação seja dentro do prazo estipulado, através de metodologias que ele mesmo define.

O mesmo autor também afirma, mais adiante, sobre os métodos. Eles que são sustentados através de um conjunto de princípios básicos, sendo eles os responsáveis pelo gerenciamento das áreas da tecnologia, juntamente com modelos e técnicas para suas respectivas atividades.

A camada de ferramenta, que constitui o topo da pirâmide, de acordo com Pressman (2011, p.40), é a que “fornece suporte automatizado ou semi-automatizado para o processo e para os métodos.”

Todas essas abordagens podem ser melhor compreendidas na figura abaixo, na qual ilustra todas as tecnologias das camadas da engenharia.

Figura 01: Camadas da engenharia de software



Fonte: PRESSMAN, 2011, p.39

1.1.1 Modelos de processos de software

Conforme definido por Pádua Filho (2000, p.23), “um processo é um conjunto de passos parcialmente ordenados, constituídos por atividades, métodos, práticas e transformações, usado para atingir uma meta”. Ou seja, um processo é como uma receita de bolo, que possui uma coleção de passos sequenciais a serem seguidos, visando atingir o objetivo traçado inicialmente.

No contexto da engenharia de *software*, os “processos podem ser definidos para atividades como desenvolvimento, manutenção, aquisição e contratação de software” (IBIDEM), as quais auxiliam a equipe de desenvolvimento, por exemplo, e são o ponto de partida para a escolha de um modelo de ciclo de vida, também conhecido como modelos de processos de *software*. Esses modelos irão auxiliar a

equipe de desenvolvimento na documentação do *software* e orientar sobre as melhores formas de desenvolver o projeto, por exemplo.

1.1.1.1 Modelo cascata

O modelo cascata, segundo Pressman (2011, p.59), é

algumas vezes chamado de ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com o levantamento das necessidades por parte do cliente, avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do software concluído.

Por ser sequencial, o modelo cascata segue rigidamente pelas suas fases. O início do projeto se dá na fase de comunicação, em que são levantadas as necessidades do cliente. Em seguida vem a fase de planejamento, onde são estimuladas as estimativas, cronograma e acompanhamento. Na fase de modelagem é feita a análise do projeto. Logo após vem a fase de construção, sendo nela que é realizada toda a codificação e os testes. Para finalizar é a fase de emprego, que é a responsável pela entrega do projeto, suporte e feedback. (PRESSMAN, 2011, p.60).

Quando os requisitos estão bem definidos, compreendidos e estáveis, o modelo cascata pode se encaixar. Entretanto, essa situação geralmente acontece em alguma adaptação ou aperfeiçoamento de um sistema já existente, pois o modelo cascata não permite modificações durante o processo, não disponibiliza versões ao longo do desenvolvimento, e ambas as circunstâncias são sempre solicitadas pelos clientes (IBIDEM).

1.1.1.2 Modelo incremental

No modelo incremental, ao contrário do modelo cascata, geralmente os requisitos iniciais não estão bem definidos. O desenvolvimento de um *software* nesse modelo é feito através de incrementos, ou seja, pequenas funcionalidades

que vão sendo geradas à medida que o tempo vai avançando (PRESSMAN, 2011, p.62). Por exemplo, um cliente deseja um sistema de reservas de passagens. O primeiro incremento desse sistema pode ser o cadastro de ônibus, o segundo incremento consulta de poltronas disponíveis, e o terceiro incremento a própria reserva de uma poltrona. O modelo incremental também pode ser utilizado em casos de customizações de um sistema já existente.

Na mesma página, Pressman também afirma que,

Quando se utiliza um modelo incremental, frequentemente, o primeiro incremento é um produto essencial. [...]. Esse produto essencial é utilizado pelo cliente (ou passa por uma avaliação detalhada). Como resultado do uso e/ou avaliação, é desenvolvido um planejamento para o incremento seguinte. O planejamento já considera a modificação do produto essencial para melhor se adequar às necessidades do cliente e à entrega de recursos e funcionalidades adicionais.

Desse modo, o modelo incremental irá entregar uma versão operacional do sistema a cada incremento, de modo que o processo seja repetido a cada incremento até que no final ele possa atender ao que foi solicitado pelo cliente.

1.1.1.3 Prototipação

A prototipação é uma técnica utilizada quando o cliente não detalha especificamente os requisitos do *software*, quando está em processo de adaptação ou quando está validando a eficiência de um sistema. Ela pode ser implementada com os modelos de processos já citados, principalmente naqueles que não possuem os requisitos bem definidos, pois ela auxilia a compreender melhor as funcionalidades desejadas pelos clientes. (PRESSMAN, 2011, p.63).

Segundo Paula Filho (2000, p.26), a prototipação “é usada não para desenvolver o produto completo, mas para construir uma série de versões provisórias que são chamadas de protótipos. Os protótipos cobrem cada vez mais requisitos, até que se atinja o produto desejado”. Devido a isso a prototipagem permite que o levantamento de requisitos seja feito de forma progressiva.

1.1.1.4 Modelo espiral

O modelo espiral pode ser definido como

um modelo de processo de software evolucionário que acopla a natureza iterativa da prototipação com os aspectos sistemáticos e controlados do modelo cascata. Fornece potencial para o rápido desenvolvimento de versões cada vez mais completas do software. (BOEHM apud PRESSMAN, 2011, p.65).

Com esse modelo, o *software* que está sendo desenvolvido deve ser disponibilizado em uma série de versões evolucionárias através de iterações, sendo que cada uma delas correspondem a uma volta no espiral. O produto desenvolvido nas iterações é oferecido através de modelos ou protótipos que, ao decorrer do progresso do desenvolvimento, estão cada vez mais completos. (PRESSMAN, 2011, p.65). Em virtude das suas características, o modelo permite a atribuição de novos recursos à medida em que é desenvolvido, testado e validado em cada iteração.

1.1.2 Especificação de requisitos

A especificação de requisitos de *software*, conforme o próprio nome já diz, consiste em determinar ou classificar todos os requisitos definidos para o desenvolvimento de um *software*. A especificação geralmente é um documento que possui descritos todos os requisitos de um determinado produto (PAULA FILHO, 2000, p.14).

1.1.2.1 Requisitos

Um dos principais fatores capazes de levar ao fracasso no desenvolvimento de um *software* é a dificuldade de levantar, de forma organizada, todas as funcionalidades a serem trabalhadas, bem como suas características. O cliente na maioria das vezes não sabe precisamente o que quer, tem dificuldade em expressar

para os desenvolvedores quais serviços deseja, ou até mesmo solicita algo além do necessário para aquele projeto. Os desenvolvedores, por sua vez, devem ter definido exatamente o que fazer, para que consigam entregar exatamente o que foi solicitado pelo cliente.

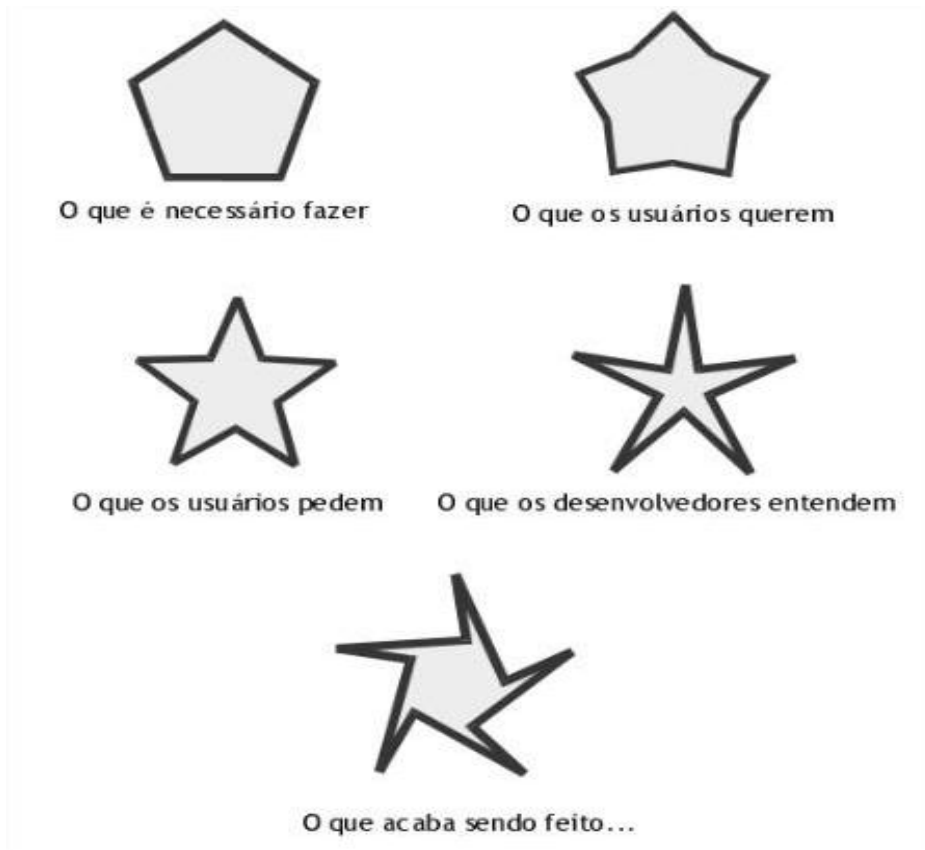
Essas funcionalidades, conforme mencionados anteriormente, são tratadas como requisitos na engenharia de *software*. Eles são conceituados, conforme Sommerville (2011, p.57), como

as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações.

Ou seja, os requisitos são todas as funções, exigências, condições, que são levantados para a construção de determinado *software*. Os mesmos se baseiam no que o cliente deseja que seja desenvolvido.

O levantamento de requisitos deve acontecer de forma adequada. Qualquer informação incorreta, ou falha de expressão e comunicação, pode resultar em uma solução totalmente diferente da esperada pelo cliente, conforme pode ser observado na Figura 02.

Figura 02: Evolução dos requisitos



Fonte: PAULA FILHO, 2000, p.15

Os requisitos de *software* por si só são classificados em duas modalidades, conhecidas como requisitos funcionais e requisitos não funcionais. Requisitos funcionais, segundo Sommerville (2011, p.59), são todas as funcionalidades, todos os serviços a serem fornecidos pelo *software* que atendam o comportamento do sistema em determinadas situações, ou até mesmo como o mesmo não deve se comportar em outras. Os requisitos não funcionais já estão relacionados a restrições daquelas funções ou serviços traçados pelos requisitos funcionais, abrangendo propriedades como confiabilidade e tempo de resposta.

Com base nessas definições, pode-se exemplificar um requisito funcional como: o sistema deverá realizar o login e cadastro do cliente, ou o sistema deverá gerar o cardápio diário para o cliente. Os requisitos não funcionais podem ser exemplificados como: o sistema deverá ser de fácil utilização, ou o sistema deverá ser intuitivo.

1.1.2.2 Engenharia de requisitos

Segundo Sommerville (2011, p.103), a engenharia de requisitos é o processo que tem como objetivo “o estudo da viabilidade do sistema, a obtenção e a análise de requisitos, a especificação de requisitos e sua documentação e, finalmente, a validação desses requisitos”. Através do estudo da viabilidade é que se determina se o projeto é viável ou não. Caso seja, é dado início às demais atividades.

Dessa forma, a engenharia de requisitos reduz problemas comuns no desenvolvimento de *software*, como a instabilidade dos requisitos, o que ajuda na coleta de requisitos corretos, embora as alterações nos requisitos muitas vezes são inevitáveis (PAULA FILHO, 2000, p.15).

1.1.2.3 Prazos e custos

Para que o produto final seja entregue sem atrasos e custos além do previsto, devem ser definidos também prazos e custos a serem seguidos. Os prazos estão relacionados ao número de requisitos, desenvolvedores e complexidade do *software*. Os custos são os gastos que podem acontecer ao longo do desenvolvimento e implantação do produto.

Paula Filho (2000, p.16) ensina que “requisitos, prazos e custos formam os vértices de um triângulo crítico. Aumentos de requisitos levam a aumentos de prazos ou custos, ou ambos. Reduções de requisitos podem levar a reduções de prazos ou custos (mas nem sempre)”.

1.1.3 Metodologias de desenvolvimento

O nome metodologia está relacionado a métodos, regras que “tem a função de formalizar a ordem de desenvolver um *software*, organizar a fase do ciclo de vida do mesmo e ter o equilíbrio entre os processos e os dados” (DENIS; WIXON apud

SILVA; SOUZA; CAMARGO, 2013, p.2). As metodologias podem ser de dois tipos, as tradicionais, que também são conhecidas como métodos clássicos, e as metodologias ágeis, que serão relatadas abaixo.

1.1.3.1 Metodologias tradicionais

As metodologias tradicionais de desenvolvimento surgiram em um contexto diferente do utilizado atualmente. Também conhecidas como metodologias práticas, as mesmas eram utilizadas em situações em que os requisitos do sistema estão totalmente definidos, já com previsão dos futuros.

Quando se desenvolvia um *software*, o custo para se fazer alguma correção ou manutenção nele era extremamente alto, de modo que na época não existiam ferramentas capazes de auxiliar no processo de desenvolvimento conforme existe hoje. Desse modo, o projeto de *software* tinha que ser todo planejado e documentado antes de iniciar sua produção (SOARES, 2004, p.2). Exemplos dessas metodologias foram citadas anteriormente, como o modelo de desenvolvimento cascata.

1.1.3.2 Metodologias ágeis

As metodologias ágeis de desenvolvimento surgiram em um contexto diferente. Devido aos dias atuais, em que as empresas trabalham cada dia mais buscando agilidade, o número de mudanças é alto e devem acontecer de forma rápida. Ao pesquisar um sistema, por exemplo, é comum as empresas buscarem soluções com foco no tempo de entrega do que na qualidade dele. Desse modo, o desenvolvimento e a entrega do *software* em tempo rápido acabam se tornando um requisito fundamental nesse tipo de metodologia (SOMMERVILLE, 2011, p.38).

Ao desenvolver um sistema, é impossível prever sua evolução ao longo do tempo, devido às mudanças que ocorrem frequentemente, assim como evoluções tecnológicas que acabam deixando um *software* descontinuado, ou até mesmo por

alterações das necessidades requeridas pelos clientes. Geralmente trabalha-se com requisitos que ainda estão instáveis, ou seja, que podem ser alterados pelo fato de ser impossível ter uma previsão de como o sistema afetará as rotinas esperadas. Desse modo, é preciso um método ágil que consiga atender a todos esses requisitos e as frequentes mudanças (PRESSMAN, 2011, p.83).

Os métodos ágeis surgiram para solucionar os problemas enfrentados com os métodos tradicionais, trazendo benefícios que não eram possíveis com os modelos citados anteriormente. Embora não sejam a solução para todos os projetos, existem vários modelos que são utilizados atualmente e que atendem a diversas empresas e desenvolvedores individuais. Exemplos deles são o *Extreme Programming*, conhecido como XP, e o Scrum, que será abordado na sua modalidade solo na pesquisa.

1.1.3.2.1 Scrum Solo

O Scrum, conforme explicado por Pressman (2011, p.95) é uma metodologia ágil de *software*, embora também seja definido como um framework, que foi criado por uma equipe de desenvolvimento na década de 90 com o intuito de guiar um projeto de desenvolvimento de *software* aplicando atividades incorporadas por ele.

O scrum é muito utilizado em projetos com equipes menores, pois nestes é possível obter melhores resultados do que sendo aplicados em grandes equipes. Por ser um modelo iterativo e incremental, ele se divide em ciclos de vida de desenvolvimento, conhecidos como *sprint*, de modo que a prioridade de cada uma será definida pelo dono do processo, ou seja, o *product owner* (PAGOTTO et al, 2016, p.4).

Embora sua utilização maior seja em pequenas e médias empresas, os mesmos autores ainda abordam que muitas organizações possuem apenas um desenvolvedor, ou até mesmo os programadores que trabalham de forma individual acabam sendo prejudicados pelo fato do scrum ser voltado a equipes. Desse modo, surgiu em 2012 uma nova metodologia denominada Scrum Solo, com a promessa de ser também um processo iterativo e incremental, porém utilizando além do próprio Scrum, também as boas práticas do *Personal Software Process* (PSP).

Conforme mencionado, o Scrum Solo carrega diversas características do próprio Scrum. Entretanto, a diferença está na duração de cada *sprint*, que são reduzidas a uma semana sem a necessidade de reuniões diárias para alinhamento, pelo fato de ser um desenvolvimento individual. Quando cada *sprint* é finalizada, é preciso que seja entregue uma versão do projeto que está sendo desenvolvido para que o mesmo possa ser testado e avaliado com suas novas funcionalidades de cada semana. (PAGOTTO et al, 2016, p.5).

1.2 Banco de dados

Banco de dados ou base de dados, segundo Elmasri e Navathe (2005, p.4), “é uma coleção de dados relacionados. Dados são fatos que podem ser gravados e que possuem significados implícitos”. Desse modo, um banco de dados representa um conjunto de dados armazenados de forma organizada, de modo a gerar algum sentido. Por exemplo, um banco de dados pode conter dados sobre uma determinada pessoa, como nome, endereço, telefones, e a partir desses dados é possível realizar um trabalho e obter as informações necessárias para algum fim.

Os bancos de dados são de grande importância para o dia a dia das pessoas, estando presente em diversas atividades como no pagamento de uma conta, conversas em redes sociais, vendas em crediário, reservas de hotéis, compras de lanches via aplicativo, entre outros.

Um banco de dados pode ter qualquer tamanho e complexidade variável, de modo que possa armazenar desde um simples registro até milhares deles. Conforme explicam Elmasri e Navathe (2005, p.3), “a construção de um banco de dados é o processo de armazenar os dados em alguma mídia apropriada controlada pelo SGBD”. Assim, a manipulação desses dados será através de funções capazes de inserir, pesquisar, recuperar, atualizar, excluir e gerar relatórios dos dados armazenados.

Observando a importância e necessidade de um banco de dados, é essencial a escolha da forma de persistência de dados na hora de desenvolver qualquer aplicação. Essa forma está relacionada ao modelo em que os dados estão organizados e as vantagens esperadas pelo mesmo.

Desse modo, um modelo de dados é entendido como um “conjunto de conceitos que podem ser usados para descrever a estrutura de um banco de dados” (ELMASRI; NAVATHE, 2005, p.19), sendo que essa estrutura pode ser dividida em dois tipos, o modelo de dados relacional e o modelo de dados não relacional.

1.2.1 Modelo de dados relacional

O modelo de dados relacional é classificado como representacional por utilizar uma ou um conjunto de tabelas para apresentar os dados e os relacionamentos entre eles. Essas tabelas possuem nomes únicos, conjuntos de atributos, de modo que cada um deles representa uma coluna e linhas, denominadas tuplas. Ele é totalmente estruturado e se destaca pela simplicidade. Suas tabelas possuem relacionamentos definidos com base no conteúdo de cada uma. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p.5).

O modelo de dados relacional é utilizado, por padrão, na maioria dos sistemas de gerenciamento de dados existentes, tais como o MySQL, SQL Server e Oracle. Em um banco de dados relacional é necessário projetar toda a sua estrutura para que seja possível fazer o armazenamento dos dados. Essas estruturas são através de tabelas, compostas por linhas e colunas e permitem uma melhor organização dos dados a serem guardados (BRITO, 2010, p.1).

Na mesma página, Brito levanta que “outra característica fundamental desse modelo é a utilização de restrições de integridade”, sendo as mais comuns as chaves primárias e as chaves estrangeiras, e também o processo de normalização, no qual se aplica certas regras sobre as tabelas do banco de dados.

O modelo relacional adota a linguagem de definição, manipulação e consulta de banco de dados, chamada SQL, que é considerada o fator de sucesso para os bancos de dados relacionais.

1.2.1.1 SQL

A *Structure Query Language* (SQL), que em tradução livre significa Linguagem Estruturada de Consulta, é a linguagem padrão de consultas utilizada para manipular banco de dados relacionais. Segundo Elmasri e Navathe (2005, p.149), “a SQL é uma linguagem de banco de dados abrangente: ela possui comandos para definição de dados, consultas e atualizações”.

Inicialmente denominada SEQUEL (*Structured English Query Language* - Linguagem de Pesquisa em Inglês Estruturado), a SQL foi criada com base em linguagens de Álgebra e Cálculo Relacional. A linguagem é padronizada pela ANSI (*American National Standards Institute* - Instituto Nacional Americano de Padrões) e pela ISO (*International Standards Organization* - Organização Internacional de Padrões). (ELMASRI; NAVATHE, 2005, p.148).

Mais adiante, os autores afirmam que “a SQL é uma linguagem de banco de dados abrangente: ela possui comandos para definição de dados, consultas e atualizações” possibilitando a manipulação dos registros.

1.2.2 Modelo de dados não relacional

O fluxo de dados que são gerados atualmente está cada vez mais crescente. Com isso, os modelos relacionais deixaram de ser a melhor alternativa para fazer o gerenciamento de tamanha quantidade de dados devido a sua estrutura pouco flexível. Um exemplo disso está nas redes sociais, pois, segundo Lóscio, Oliveira e Pontes (2011, p.1), elas “requerem o gerenciamento de grandes quantidades de dados não estruturados que são gerados diariamente por milhões de usuários”. Em consequência desse fluxo de dados e a busca por melhor desempenho, surgiu então um novo modelo de dados, conhecido como NoSQL.

Embora também tenha o termo SQL em seu nome, o NoSQL não se trata de uma linguagem para banco de dados. O termo significa “Not Only SQL”, que em tradução para o português quer dizer “Não apenas SQL”. O NoSQL é um modelo que permite armazenar, tratar e consultar dados não normalizados de várias formas, sendo que seu objetivo principal é oferecer maior flexibilidade na estruturação do banco. Em relação à forma de armazenamento, existem alguns tipos de banco de

dados não relacionais, sendo os principais o de chave-valor, coluna, de documento e de grafo (BRITO, 2010, p.2).

Um armazenamento de dados de chave-valor permite ao usuário armazenar dados em um tipo de esquema. Os dados consistem em duas partes, uma string que representa a chave e os dados reais que devem ser referidos como valor, criando assim um par de "chave-valor". Uma de suas fraquezas está na dificuldade em criar visualizações de dados personalizados, devido ao seu esquema (NAYAK; PORIYA; POOJARY, 2013, p.16).

Os armazenamentos de colunas são semelhantes ao modelo relacional por também utilizar linhas e colunas. Porém sua estrutura é definida por super colunas e família de colunas, de modo que nas colunas, cada chave está associada a um ou mais atributos. Os bancos de dados orientados a colunas são adequados para mineração de dados e aplicações analíticas, onde o método de armazenamento é ideal para as operações executadas nos dados (NAYAK; PORIYA; POOJARY, 2013, p.17).

Bancos de Dados de documentos referem-se a bancos que armazenam seus dados sob a forma de documentos, oferecendo ótimas opções de desempenho e escalabilidade. Os documentos são de formatos padrão, como XML, PDF, JSON, etc. Os documentos no banco de dados são endereçados usando uma chave exclusiva para sua representação (IBIDEM).

O armazenamento em grafos consiste em nós, arestas e propriedades, onde os nós são usados para modelar os objetos existentes, as arestas mantêm o relacionamento entre eles e as propriedades descrevem todos os atributos deles (NAYAK; PORIYA; POOJARY, 2013, p.18).

O NoSQL não possui uma linguagem de consulta padrão. A maioria dos provedores de banco de dados criaram sua própria linguagem de consulta, tornando-se necessário a criação de uma linguagem comum, assim como o SQL para os bancos de dados relacionais. O UnQL é uma linguagem de consulta não estruturada que está sendo desenvolvida objetivando a padronização na plataforma NoSQL.

Nayak, Poriya e Poojary levantaram algumas vantagens e desvantagens dos bancos de dados NoSQL em relação aos bancos de dados relacionais. Entre as vantagens, o NoSQL é facilmente escalável, rápido, flexível e está em constante crescimento. As desvantagens estão relacionadas ao fato de o mesmo não possuir

uma linguagem de consulta padrão, não ser compatível com ACID, não possuir interface padrão e ter uma manutenção difícil.

1.2.2.1 Firebase

O Firebase, segundo o site oficial¹, é uma tecnologia criada sobre a infraestrutura do Google para criar banco de dados e fazer buscas em tempo real com apenas algumas linhas de código. Ele fornece algumas funcionalidades como análises, bancos de dados, mensagens e relatórios de erros, de modo que os dados são armazenados com JSON e podem ser acessados de todas as plataformas.

O Firebase permite a criação de aplicativos da Web sem programação do lado do servidor, tornando o desenvolvimento mais rápido e fácil. O próprio Firebase faz a verificação de usuários, armazenamento de dados e implementa regras de acesso. Ele também suporta a criação de aplicativos iOS, Web e Android. Aplicativos usando o Firebase podem controlar e usar dados, sem a necessidade de pensar em como os dados serão armazenados e sincronizados (KUMAR, et al, 2016, p.2).

Os mesmos autores também falam que o Firebase trabalha com sincronização de dados, desse modo sempre que acontece alguma alteração, todos os dispositivos conectados recebem essa atualização em milissegundos. Ele também fornece suporte *offline* para os aplicativos, pois seus dados são mantidos em disco. Assim, quando é restabelecida a conexão, os dispositivos recebem as alterações e fazem a sincronização de acordo com o servidor.

Algumas vantagens do Firebase estão na segurança dos dados, pois ele requer criptografia SSL para proteção dos dados. Também pode-se citar as diversas opções que ele oferece, como o *Realtime Database*, *Analytics*, *Authentication*, *Hosting* e Monitor de desempenho são alguns exemplos (KUMAR et al, 2016, p.4)

1.2.2.1.1 Firebase Realtime Database

¹ Disponível em: <www.firebase.google.com/>

De acordo com o seu site oficial², o Firebase *Realtime Database* “é um banco de dados NoSQL hospedado na nuvem”, que permite o armazenamento e sincronização de dados em tempo real, tornando mais fácil aos usuários acessarem seus dados de qualquer lugar.

O *Realtime Database* segue os SDKs para web e dispositivos móveis. Desse modo, é possível desenvolver aplicativos sem precisar de servidores, e, também por conta dos SDKs, quando um usuário fica *offline*, o *Realtime Database* tem um recurso de cache local no dispositivo, no qual armazena as alterações até que sejam aplicadas quando o dispositivo ficar *online* novamente, de modo que são sincronizados automaticamente.

O *Realtime Database* fornece regras de segurança para definir a estrutura dos dados e quando os mesmos serão lidos e gravados. Ele possui uma integração com o Firebase *Authentication*, de modo que facilita a autenticação com redes sociais e a definição de acessos e permissões.

1.2.3 SGBD

Para que seja possível manipular os dados armazenados no banco de dados, utiliza-se um SGBD (Sistema de Gerenciamento de Banco de Dados) que disponibiliza uma interface para comunicação com a aplicação, permitindo ações como inserir, alterar ou consultar dados. De acordo com Elmasri e Navathe (2005, p.3),

um SGBD é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações.

Ou seja, o SGBD irá executar com mais facilidade todas as tarefas mencionadas de maneira robusta e eficiente, de forma que garanta algumas vantagens como a independência de dados, acesso eficiente aos dados, integridade e segurança dos dados, administração de dados, acesso concorrente e recuperação

² Disponível em: <www.firebaseio.com/>

de falha e tempo reduzido de desenvolvimento de aplicativo. (RAMAKRISHNAN, GEHRKE, 2008, p.7).

1.3 Desenvolvimento móvel

Antes de iniciar o desenvolvimento de uma aplicação móvel, é fundamental fazer a escolha da tecnologia que será utilizada para a mesma. Para essa decisão, existem algumas opções no mercado a serem consideradas, podendo-se citar o desenvolvimento de aplicativos nativos e os aplicativos híbridos.

1.3.1 Desenvolvimento nativo

O termo nativo refere-se ao desenvolvimento na própria linguagem do sistema operacional, podendo-se citar a plataforma Android e a IOS. Quando se escolhe desenvolver um aplicativo nativo, deve-se levar em consideração, conforme Lopes (2016, p.6), que seu diferencial, em relação aos aplicativos híbridos, está no desempenho da aplicação. Existem tarefas que exigem um processamento maior do dispositivo, pois os aplicativos nativos exploram os próprios recursos nativos do aparelho, como por exemplo nos jogos 3D que possuem complexidade maior comparado a outros aplicativos.

1.3.2 Desenvolvimento híbrido

O desenvolvimento híbrido está relacionado à criação de um aplicativo que, utilizando os recursos nativos do dispositivo, é desenvolvido utilizando tecnologias da web podendo ser executado em mais de uma plataforma, de forma que um único código seja desenvolvido e utilizado. Por esse motivo, os aplicativos híbridos

possuem um custo mais baixo que os nativos, embora suas funcionalidades sejam bem semelhantes (LOPES, 2016, p.6).

Existem soluções que possibilitam o desenvolvimento de aplicativos multiplataformas, podendo-se citar o Cordova, PhoneGap, Ionic e React Native. Essas soluções utilizam linguagens da web, mas garantem as mesmas funcionalidades das linguagens próprias dos sistemas operacionais mencionados anteriormente.

Como a aplicação a ser desenvolvida neste projeto não terá uma complexidade tão alta em relação ao desempenho, e a desenvolvedora deseja utilizar linguagem da web e desenvolver apenas um código para disponibilizar o aplicativo para diversas plataformas, o desenvolvimento híbrido será o ideal.

1.4 Ambiente web

1.4.1 Linguagens de Programação

As Linguagens de programação são conjuntos de instruções afim de representar ações ou resolver operações realizadas pelo computador. Elas são as ferramentas que possibilitam o desenvolvimento de softwares. Visto que os computadores são utilizados para infinitas tarefas, várias linguagens de programação foram criadas (e ainda são) para que possam atender essas necessidades (SEBESTA, 2018, p.31).

Existem linguagens de programação para diversos focos, como científico, empresarial, de inteligência artificial, para web, entre outros. Elas também são divididas em níveis, alto e baixo, que embora cada uma tenha suas particularidades, todas trabalham visando atingir o mesmo objetivo (SEBESTA, 2018, p.35).

As linguagens de programação são representadas através de códigos, podendo ser símbolos, números e palavras, que ao formarem um sentido lógico, produzem funções a serem realizadas pelo computador.

1.4.2 HTML

O HTML, acrônimo de *Hypertext Markup Language*, que em português significa Linguagem de Marcação de Hipertexto, é a principal linguagem de marcação da *World Wide Web*, segundo a W3C (principal organização de padronização da *World Wide Web*).

Caldeira (2015, p.1) define HTML como

um conjunto estruturado de instruções, conhecidas por etiquetas ou tags (em inglês), que dizem a um browser como publicar uma página web, ou seja, o browser interpreta essas etiquetas e desenha a página no ecrã. Estes conjuntos de instruções estão agrupados em ficheiros de tipo texto, i.e., sem qualquer tipo especial de formatação.

Sendo assim, o HTML é o que permitirá fazer a estruturação das telas da aplicação. Essa estruturação é por meio de tags, que definem cabeçalhos, corpo do texto, títulos, parágrafos, e diversos outros recursos que compõem uma página. O conjunto de tags são interpretados pelo navegador e exibidas na tela para o usuário.

Desde a invenção da web até hoje, o HTML evoluiu por oito versões, e atualmente se encontra em sua quinta versão, referenciado como HTML5. Sua nova versão aprimorou algumas questões que mudaram a forma de interagir com a web, exemplos disso estão relacionados a compatibilidade, carregamento de páginas mais fácil, interação com dados locais e com servidores, trouxe também novidades relacionadas a tags e atributos. (PRESCOTT, 2015, p.19)

Também segundo Prescott (2015, p.69-70), “o HTML é um bloco básico de construção de sítios web e todo o resto do conteúdo atua como tijolo de suporte”. Sendo assim, o HTML é uma linguagem exclusivamente de marcação, não cabendo a ele tratar de conteúdos relacionados a estilo (cores de fontes, tamanhos de textos), por exemplo. Tratamentos assim são de responsabilidade do CSS, o qual será abordado a seguir.

1.4.3 CSS

CSS é a abreviação para *Cascading Style Sheets*, que traduzindo para o português significa Folhas de Estilo em Cascata. O CSS é uma linguagem de estilo que, segundo a W3C, é definida como um mecanismo simples para adicionar estilo (por exemplo, fontes, cores, espaçamento) a documentos da Web, como por exemplo no HTML.

De acordo com Quierelli (2012, p.14), “uma das finalidades do CSS é deixar o código mais limpo para o desenvolvedor poder trabalhar com maior facilidade”, pois ele oferece várias funcionalidades destinadas a manipulação e posicionamento de elementos em uma página, definindo diferentes formas de distribuí-los visualmente na tela para o usuário.

1.4.4 JavaScript

O JavaScript, conforme definido pela ECMA (uma das criadoras da linguagem), é uma linguagem de script utilizado na parte cliente, ou seja, nos navegadores. É utilizado em páginas web juntamente com o HTML e CSS, sendo ele que proporciona a realização de eventos nas páginas e manipulação da estrutura do HTML.

O JavaScript foi criado pela Netscape juntamente com a Sun Microsystems em 1995, com a finalidade de proporcionar interatividade às páginas da web. Na época, a linguagem foi implementada no Netscape Navigator que era o navegador mais popular do momento (SILVA, 2010, p.23).

Segundo a ECMA, o JavaScript é a linguagem de script mais utilizada no mundo. Assim como todas as outras linguagens de script, elas são utilizadas para tornar páginas web mais interativas aos seus usuários, para controlar o navegador, fazer comunicação com servidor, desenvolver jogos, por exemplo. Embora seja uma linguagem que ofereça tantas possibilidades de implementação, a mesma depende de um navegador web para interpretar e executar os seus scripts.

Mesmo o JavaScript sendo uma linguagem do lado cliente, ela também pode ser utilizada do lado servidor devido aos interpretadores hospedados nos servidores. O JavaScript não pode ser estritamente considerado uma linguagem de programação, mas ao ser utilizado com frameworks como o Ionic, por exemplo, é

capaz de substituir uma linguagem originalmente de programação, como Java ou PHP (DIMES, 2015, p.23).

1.4.5 TypeScript

O TypeScript, assim como o JavaScript, é uma linguagem de script utilizada pelo lado cliente. De acordo com o site oficial da linguagem, o TypeScript é uma linguagem de código aberto que compila para o JavaScript puro. O TypeScript oferece suporte aos recursos do JavaScript mais recentes e em evolução para ajudar a construir componentes robustos.

A linguagem TypeScript é a JavaScript com anotações de tipo adicionadas a ele. O compilador typescript tem dois recursos principais: um transpilador e um verificador de tipos. Um transpilador é uma forma especial de compilador que gera o código fonte. No caso do compilador TypeScript, o código fonte dele é compilado para o código JavaScript. Um verificador de tipos procura por contradições em seu código. Por exemplo, se for atribuído uma *string* a uma variável e usá-la como um número, acarretará a um erro de tipo (WOLFF, 2016, p.7).

1.4.6 Frameworks

Conforme explicado por Alvim (2010, p.12), um framework pode ser definido como uma coleção de classes que buscam oferecer melhores práticas de desenvolvimento, de modo que também diminua o retrabalho, como a repetição de códigos, por exemplo. Para obter essas vantagens, é necessário que o desenvolvedor domine a linguagem de programação e saiba utilizar o framework no seu projeto.

1.4.6.1 Angular

Angular é um framework criado pela Google para o desenvolvimento *front-end* de aplicações múltiplas plataformas, através de linguagens da web (HTML, CSS e TypeScript). Ele fornece uma estrutura para a criação de aplicações na parte cliente, através das suas diversas bibliotecas e elementos que constituem o framework.

Os principais elementos do Angular são os componentes, *templates*, diretivas, serviços, roteamento e injeção de dependência, compondo toda a interface, comunicação, *back-end* e organização do código e aplicação. O Angular é uma tecnologia de código livre, e facilita o desenvolvimento tanto para páginas web para sistemas e aplicações *mobiles* com o Ionic, por exemplo (GUEDES, 2017, p.1).

O Angular utiliza um conceito chamado *Single Page Application* – SPA, que são “aplicações desenvolvidas em JavaScript que rodam quase inteiras no lado do cliente (*browser*). Assim que o usuário acessa o site, a aplicação fica armazenada do lado do cliente em forma de templates (pedaços de HTML)”. Essas aplicações buscam requisições no servidor apenas quando necessitam de dados mais brutos. Fora dessa situação, as aplicações fazem uma transição entre templates no próprio navegador, diminuindo o trabalho do servidor (GUEDES, 2017, p.2-3).

1.4.6.2 Ionic

Com o aumento da demanda por aplicativos móveis, buscou-se também formas que pudessem agilizar o processo de desenvolvimento sem perder a qualidade da aplicação. Esse foi o ponto de partida para o surgimento de frameworks como o Ionic para o desenvolvimento híbrido.

O Ionic é um framework de código aberto criado pela empresa Drifty, em 2012, que oferece toda uma estrutura de interface de usuário para o desenvolvimento de aplicativos móveis híbridos de maneira rápida, fácil e acessível. O Ionic é focado na experiência de usuário, ou na interação da interface do usuário de um aplicativo (controles, interações, gestos, animações). É de fácil aprendizado e se integra muito bem com outras bibliotecas ou estruturas, como o Angular, ou pode ser usado como autônomo sem uma estrutura de *front-end* usando uma simples inclusão de *script*.

Ele é uma combinação de várias tecnologias, sendo que sua camada superior é o próprio framework, fornecendo a camada de interface do usuário do aplicativo. Logo abaixo está o Angular, uma estrutura de aplicação web. Ambas as estruturas ficam no topo do Apache Cordova, que permite que o aplicativo da Web utilize os recursos nativos do dispositivo e se torne um aplicativo nativo (GRIFFITH, 2017, p.2).

De acordo com o site oficial³ do framework, o Ionic é uma maneira simples para programadores web desenvolverem aplicativos móveis. Ele é gratuito, de código aberto, com plataforma em nuvem e possibilita o desenvolvimento para todas as lojas de aplicativos. O Ionic foi desenvolvido para funcionar e se comportar de maneira excelente nos dispositivos móveis mais recentes, com práticas recomendadas, como transições aceleradas por hardware eficientes e gestos otimizados para toque. Hoje, o framework é considerado uma das tecnologias de desenvolvimento móvel multiplataforma mais populares do mundo.

Além dos seus próprios componentes para interface do usuário, o Ionic Framework possui uma robusta interface de linha de comando (CLI). Essa ferramenta cria rapidamente os aplicativos e fornece vários comandos úteis para o desenvolvimento. Ele auxilia a instalação, atualização, compilação, depuração e vários outros serviços.

O framework possui um conjunto de serviços adicionais para auxiliar no desenvolvimento, tais como o Ionic View, Ionic Creator, Ionic Market e Ionic Devapp.

Atualmente o framework se encontra na quarta versão, a qual trouxe melhorias no desempenho, aprimoramento de tempo, temas, compatibilidade com múltiplos frameworks e documentação nova são algumas delas. Ele tem integração oficial com o Angular, mas o suporte para Vue e React está em desenvolvimento. Sua nova versão também trouxe independência para o framework, pois agora o mesmo tem uma própria biblioteca para funcionar, não dependendo mais dos componentes da web acoplados ao Angular.

1.4.7 Serviço

³ Disponível em: <www.ionicframework.com/>

1.4.7.1 *Back-end as a Service*

Back-end as a Service – BaaS, é uma abordagem que oferece aos desenvolvedores uma maneira de conectar sua aplicação ao *back-end* em nuvem, tanto para aplicativos da web quanto *mobiles*. O BaaS também disponibiliza recursos de gerenciamento de usuários, notificações por *push* e integração a redes sociais, por exemplo. Esse serviço é fornecido através de kits de desenvolvimento de software – SDK, e aplicativos de interfaces de programação – API, podendo-se citar o exemplo do Firebase que foi abordado anteriormente. O BaaS possui alguns benefícios relacionados a eficiência, tempo de desenvolvimento menor, otimização e segurança (LANE, 2013, p.1).

1.4.8 Ambiente de desenvolvimento

1.4.8.1 *Visual Studio Code*

Para codificação do aplicativo proposto neste trabalho, será utilizado o Visual Studio Code que, segundo o seu site oficial, é um editor de código redefinido e otimizado para criar e depurar aplicativos modernos da Web e da nuvem. O Visual Studio Code foi desenvolvido pela Microsoft, é gratuito e possui diversos recursos que auxiliam no desenvolvimento de uma aplicação. A depuração do código é feita diretamente do editor, ele é personalizável, possui preenchimento automático, extensões, plugins que facilitam o trabalho, e um próprio terminal integrado, muito útil para os comandos do Ionic CLI.

2 METODOLOGIAS

O desenvolvimento de uma aplicação envolve linguagens, tecnologias e ferramentas que tornam possível toda a criação e manutenção do *software* ao decorrer do projeto. As metodologias escolhidas foram analisadas levando em consideração o proposto pelo aplicativo, de modo que cada uma delas, em sua totalidade, pudessem atingir o proposto e esperado pelo *software*.

2.1 Metodologia de desenvolvimento

O modelo de processo de software escolhido e utilizado neste projeto foi o incremental, através do Scrum Solo. Desse modo, foram estipulados alguns incrementos para serem realizados em determinados períodos de tempo. Assim, a medida em que as funcionalidades eram desenvolvidas, a parte de banco de dados era realizada gradativamente.

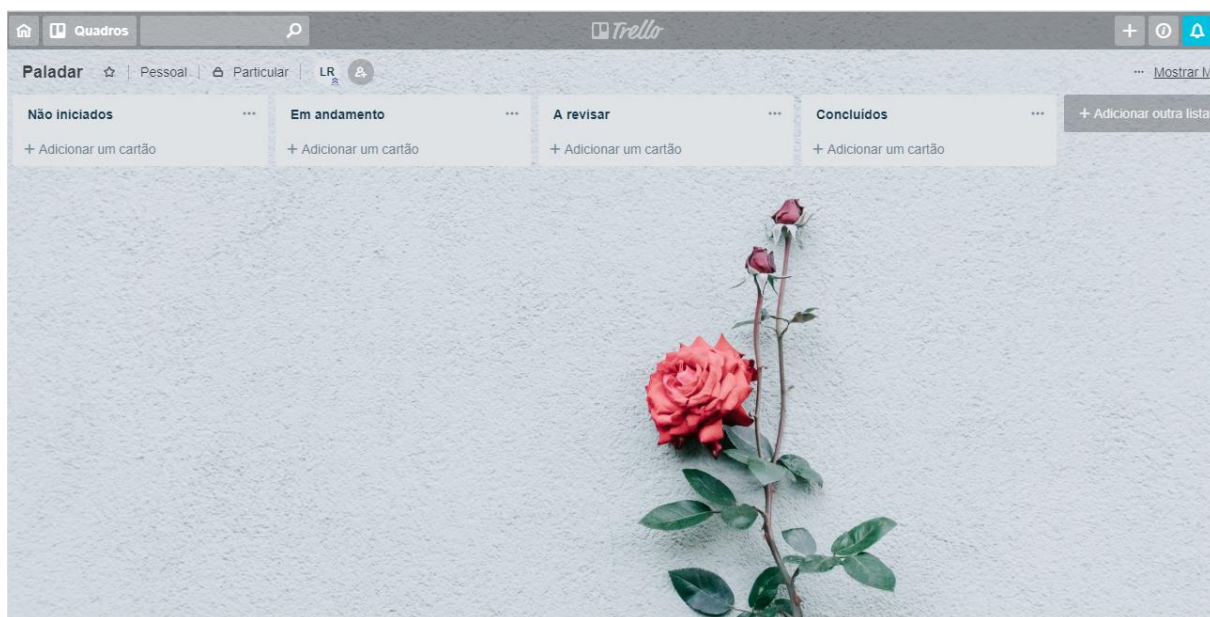
Para auxiliar nessa fase, foi utilizada uma ferramenta de chamada Trello⁴, de modo que auxiliava na organização das funcionalidades estipuladas. O Trello é uma ferramenta exibida através de um quadro, em que o usuário pode fazer a criação de listas de acompanhamento e adicionar cartões representando as funções a serem feitas. Ele pode ser utilizado individualmente ou por um grupo.

Foram criadas quatro listas, identificadas como “Não iniciados”, “Em andamento”, “A revisar”, e “Concluídos”, de acordo com a Figura 03. Todas as tarefas foram inseridas nos cartões e definidos o tempo de conclusão para a mesma. Os incrementos eram identificados pelo calendário, e caso não fossem concluídos, era realizado outro planejamento quanto a sua finalização.

⁴ Disponível em: <<https://trello.com/>>

A ordem de desenvolvimento das modalidades do aplicativo foram especificadas de acordo com a prioridade dos requisitos e pelos pré-requisitos que alguns exigiam.

Figura 03: Trello



Fonte: da própria autora

2.2 Ambiente web

Para o ambiente web, foram escolhidas as linguagens HTML5, CSS3 e o TypeScript, que possibilitam a codificação da parte *front-end* e *back-end* da aplicação. Quanto ao framework, o Ionic3 foi escolhido para o desenvolvimento híbrido, utilizando também o Angular4 para fornecer a base para a aplicação. O banco de dados utilizado é o Firebase *Realtime Database*, ele que faz uso de um serviço de *back-end* para acelerar o processo de desenvolvimento, conhecido como BaaS.

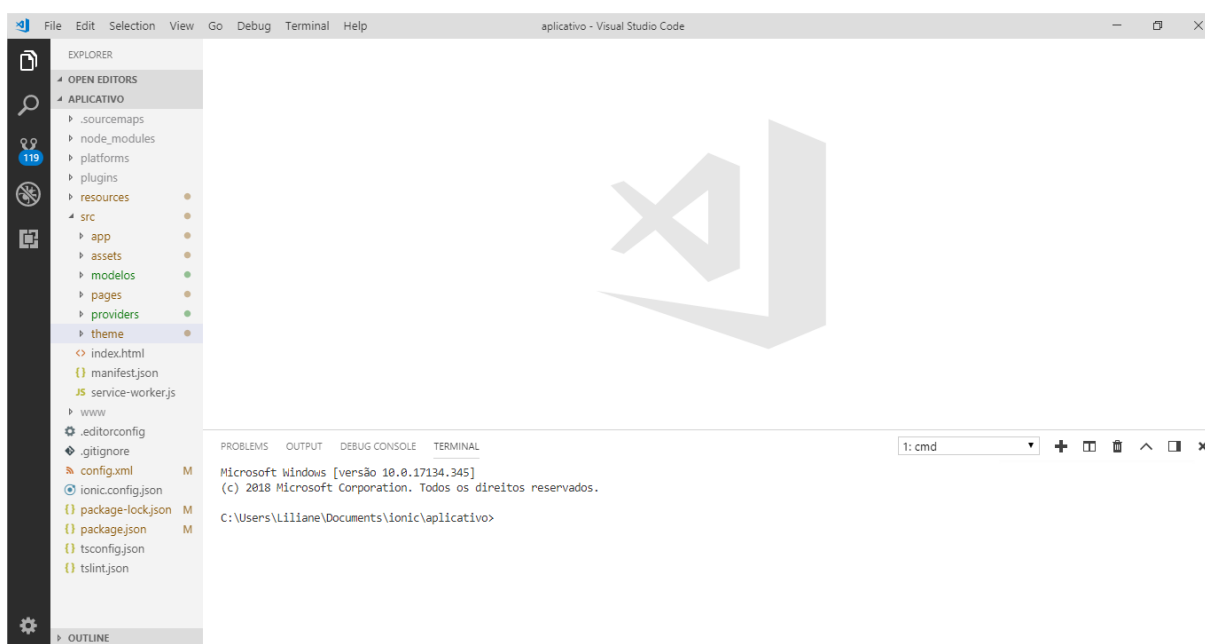
2.3 Ferramentas e aplicações

Foram selecionadas algumas ferramentas e aplicações essenciais para o processo de desenvolvimento e testes do aplicativo móvel, sendo elas o Visual Studio Code, Genymotion, Astah, Node.js e NPM.

2.3.1 Visual Studio Code

O aplicativo proposto será codificado no editor de texto Visual Studio Code versão 1.28. O editor é grátis, de código aberto, e oferece uma variedade de plugins e extensões a serem incorporadas na aplicação para facilitar e acelerar o processo de desenvolvimento. A instalação é simples, através do seu site oficial⁵ é feito o *download* do arquivo executável e seguido suas próprias orientações.

Figura 04: Visual Studio Code



Fonte: da própria autora

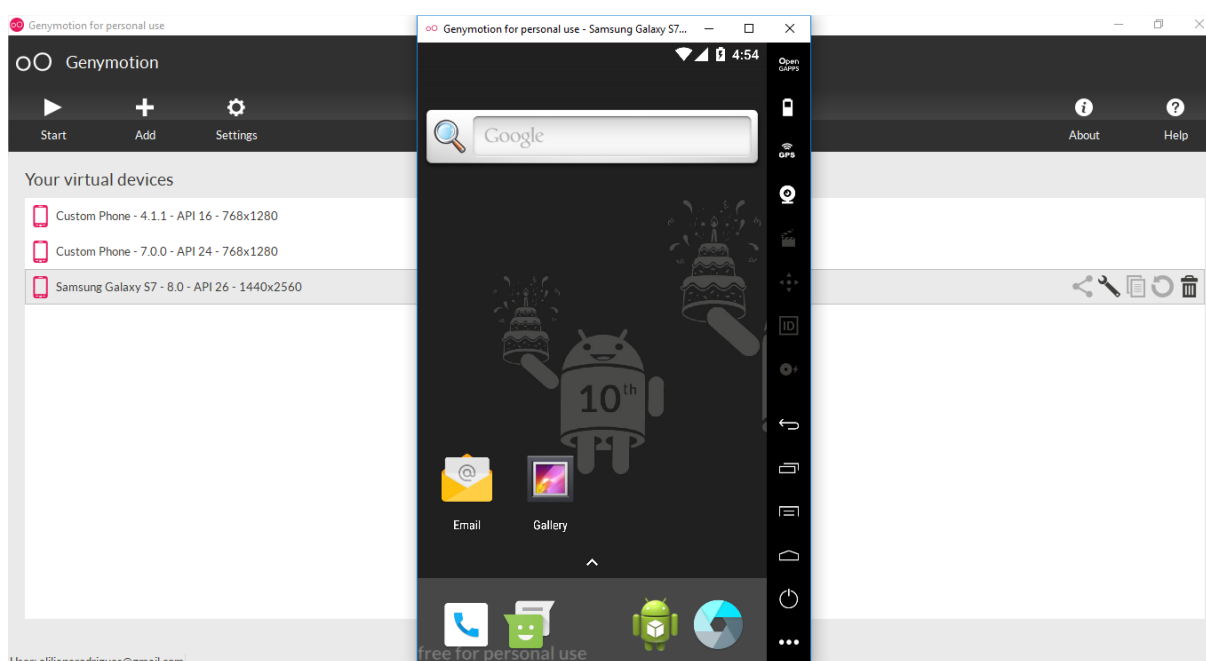
2.3.2 Genymotion

⁵ Disponível em: <<https://code.visualstudio.com/>>

Para realização de testes relacionados ao layout da aplicação, foi escolhido o Genymotion na versão 2.12.2 para emular um dispositivo virtual android. A licença utilizada foi a de uso pessoal que é gratuita. Para utilizá-lo é preciso fazer a inscrição no site e autenticação na aplicação quando já estiver instalada no computador.

O *download* do Genymotion é realizado através do seu site oficial⁶. Sua instalação é simples e tem como pré-requisito a instalação do VirtualBox, *software* que permite a virtualização dos dispositivos virtuais. Após todo esse processo, é preciso fazer o download de um dispositivo virtual. Esse dispositivo é escolhido através da versão do android e do modelo pretendido, podendo ser adicionados a quantidade desejada de dispositivos. Feita a instalação do dispositivo, o mesmo já pode ser iniciado e utilizado para emulação.

Figura 05: Genymotion



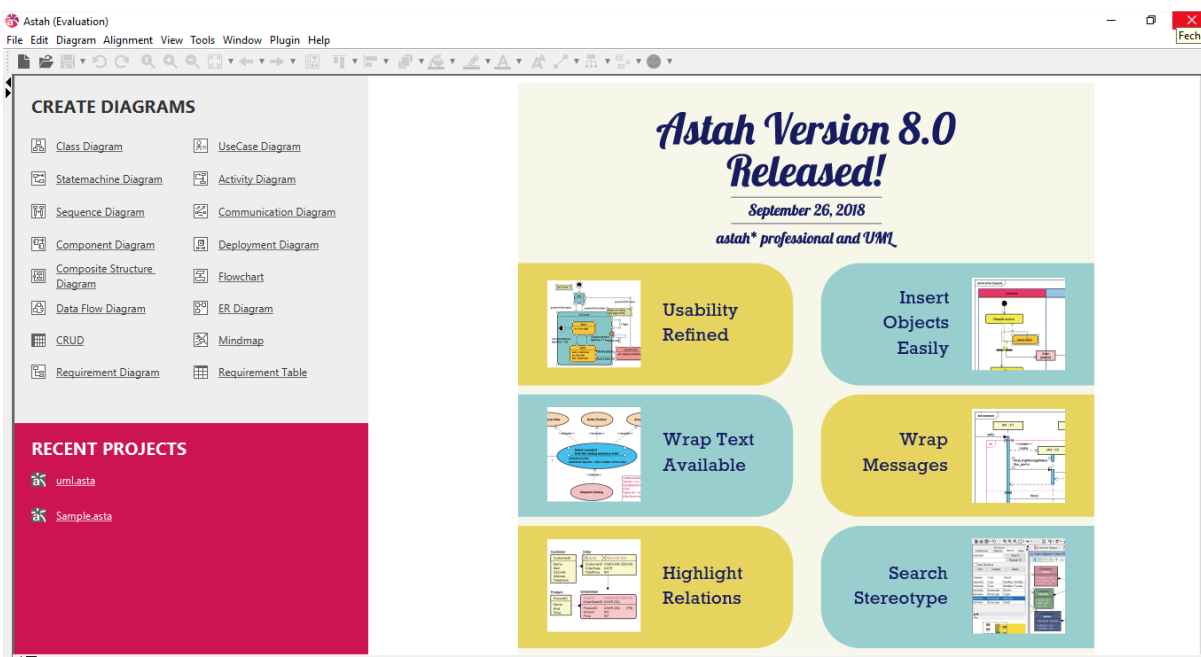
Fonte: da própria autora

2.3.3 Astah

⁶ Disponível em: <<https://www.genymotion.com/>>

O Astah é um programa que permite a criação de diagramadas de vários tipos, sendo, entre eles, o diagrama de casos de uso. O astah foi utilizado em sua versão 8.0 na versão gratuita para estudantes.

Figura 06: Astah



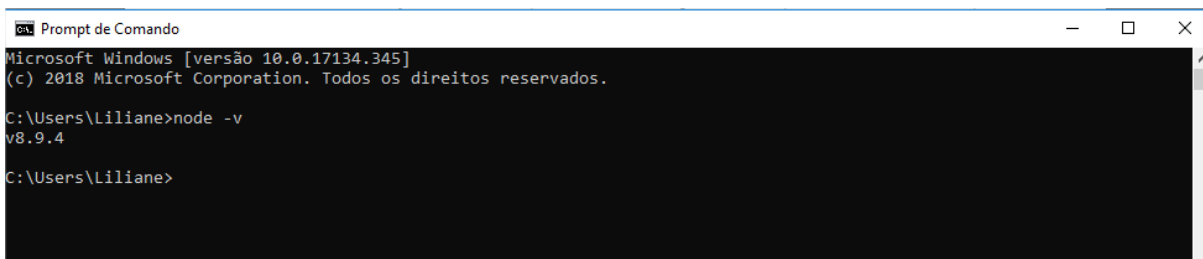
Fonte: da própria autora

2.3.4 Node.js

O Node.js⁷ é um interpretador de código JavaScript de código aberto, capaz de executar diversas instruções simultaneamente. Ele foi utilizado na versão 8.9.4, conforme mostra a Figura 07.

⁷ Disponível em: <<https://nodejs.org/>>

Figura 07: Node.js



```
Prompt de Comando
Microsoft Windows [versão 10.0.17134.345]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Liliane>node -v
v8.9.4

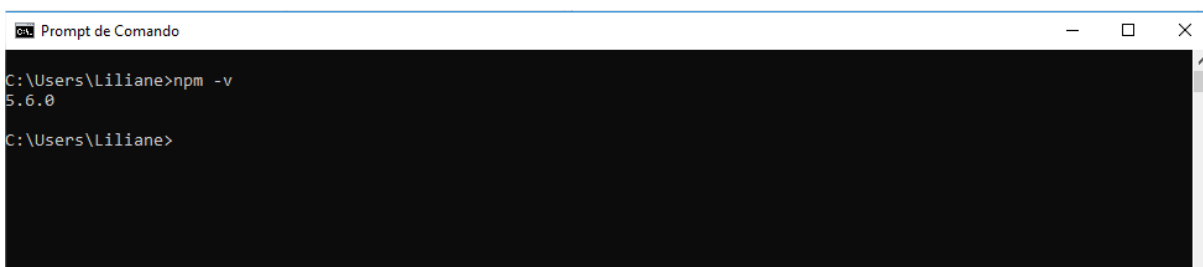
C:\Users\Liliane>
```

Fonte: da própria autora

2.3.5 NPM

O NPM – Node *Package Manager*, é um gerenciador de pacotes que funciona junto com o Node.js. O NPM consiste em um kit de ferramentas para desenvolvimento que permite a instalação de vários pacotes, como o próprio ionic. Ele foi utilizado na versão 5.6.0, conforme demonstra a Figura 08.

Figura 08: NPM



```
Prompt de Comando

C:\Users\Liliane>npm -v
5.6.0

C:\Users\Liliane>
```

Fonte: da própria autora

3 DESENVOLVIMENTO

Este capítulo apresenta o passo a passo para desenvolvimento do software proposto na pesquisa. Este passo a passo vai desde o processo de levantamento e especificação de requisitos até a conclusão do desenvolvimento da aplicação. O desenvolvimento foi realizado em ambiente Windows, utilizando um editor de texto, terminal, navegador e emulador para a produção e realização de testes.

3.1 Levantamento de requisitos

O primeiro passo para o desenvolvimento de um *software* é a fase de levantamento de requisitos. Essa fase é de extrema importância para um projeto, pois é através dela que se compreende o problema enfrentado pelo cliente e a solução que ele deseja.

O *software* desenvolvido deve ser de qualidade, obedecendo e satisfazendo ao que foi solicitado pelo cliente. Desse modo, foi realizado o processo de levantamento de requisitos através de observações realizadas no ambiente do restaurante e também através de pesquisas de opiniões que foram abordadas com clientes do estabelecimento.

Após o levantamento dos dados, foram identificados dois tipos de requisitos a serem seguidos, os requisitos funcionais, que serão representados pela sigla RF, e os requisitos não funcionais, representados pela sigla RNF. Os RF representam as funcionalidades que a aplicação deverá realizar. Os RNF apresentam as características a serem obedecidas pelo aplicativo.

A aplicação atenderá dois lados, cliente e restaurante. Primeiramente, foram estabelecidos os requisitos da parte cliente, e depois os requisitos do lado do

restaurante, mas também existem requisitos que atendem ambos, conforme é exibido abaixo.

Quadro 01: Requisitos funcionais

Número	Descrição
RF01	O aplicativo deverá realizar o cadastro do cliente.
RF02	O aplicativo deverá solicitar <i>login</i> e senha para autenticação de todos os usuários.
RF03	O aplicativo deverá permitir a consulta do cardápio do dia.
RF04	O aplicativo possibilitará ao cliente montar o seu prato, de acordo com as opções do cardápio.
RF05	O aplicativo deverá conter um campo de observação na montagem da marmita.
RF06	O aplicativo deverá mostrar o valor final da marmita.
RF07	O aplicativo deverá conter uma opção de <i>feedback</i> para clientes darem sugestões.
RF08	O aplicativo permitirá acrescentar itens adicionais a marmita.
RF09	O aplicativo exibirá as formas de entrega ou busca da marmita.
RF10	O aplicativo deverá conter um perfil para o usuário.
RF11	O aplicativo deverá ter um menu com as opções disponíveis.
RF12	O aplicativo deverá permitir o cadastro de usuário administrador pelo perfil do restaurante.
RF13	O aplicativo deverá realizar o cadastro de categorias.
RF14	O aplicativo deverá realizar o cadastro e listagem de itens, e a edição do preço adicional.

RF15	O aplicativo deverá ter a opção de consultar os pedidos realizados no dia.
RF16	O aplicativo deverá ter a opção de consultar as sugestões enviadas pelos clientes.
RF17	O aplicativo deverá registrar todos os pedidos atendidos.
RF18	O aplicativo deverá definir o cardápio do dia e fazer a sua edição.

Fonte: da própria autora

Quadro 02: Requisitos não funcionais

Número	Descrição
RNF01	O cliente só utilizará o aplicativo caso tenha feito <i>login</i> .
RNF02	O cliente só poderá fazer pedido em horários estabelecidos pelo restaurante. Nos demais horários, ele poderá consultar o cardápio do dia.
RNF03	O cliente não poderá solicitar pratos que não estejam no cardápio do dia.
RNF04	A aplicação deverá ser intuitiva e com ótimo desempenho para o cliente e restaurante.
RNF05	A aplicação funcionará com acesso a internet, mas não perderá os dados caso a conexão for perdida em algum momento.
RNF06	O campo de sugestão ficará disponível para o cliente.

Fonte: da própria autora

3.2 Preparação do ambiente

3.2.1 Instalação do Ionic

Conforme apresentado na metodologia, o framework utilizado neste software é o Ionic. Feita a instalação das ferramentas de pré-requisito (node.js e npm), o ionic foi instalado via terminal do windows (Prompt de comando), executando o seguinte comando:

```
npm install -g ionic
```

3.2.2 Criação do projeto

Com o ionic instalado, o próximo passo é a criação do projeto. O framework oferece sugestões de *templates* iniciais, com menu lateral, inferior ou vazio. Para o aplicativo desta pesquisa, foi escolhido um template com menu lateral, conhecido como *sidemenu*. A criação do projeto segue a seguinte instrução:

```
ionic start aplicativo sidemenu
```

O *start* se refere ao início do aplicativo, em seguida é definido um nome, e após isso o template (podendo ser *sidemenu*, *tabs* ou *blank*). Depois do projeto criado, para que o mesmo seja executado, é preciso estar na sua pasta origem e seguir o comando:

```
ionic serve
```

Após isso será aberta uma guia no navegador executando a aplicação. Vale ressaltar que existem outras ferramentas para fazer a execução, como emuladores, Ionic lab, Ionic DevApp e algum dispositivo conectado.

3.2.3 Instalação do firebase

Através do Prompt de Comando ou do próprio terminal incluso no Visual Studio Code, é realizado a instalação do Firebase e Angular Fire (módulo que simplifica a utilização dos serviços do banco de dados) no projeto com a seguinte instrução:

```
npm install angularfire2 firebase --save
```

Com esse comando, todos os pacotes e dependências necessárias para conexão com o banco de dados são instalados.

3.2.4 Pacote e versões

O arquivo *package.json* é o que gerencia todas as dependências e versões utilizadas no projeto. Nele é possível visualizar as versões de cada pacote utilizado e os requerimentos desses pacotes. O *package.json* é obtido automaticamente no ionic quando o projeto é criado utilizando a CLI.

Abaixo está o *package* do aplicativo desenvolvido neste trabalho.

Trecho de código 01: Arquivo package.json

```
{
  "name": "paladar",
  "version": "0.0.1",
  "author": "Ionic Framework",
  "homepage": "http://ionicframework.com/",
  "private": true,
  "scripts": {
    "start": "ionic-app-scripts serve",
    "clean": "ionic-app-scripts clean",
    "build": "ionic-app-scripts build",
    "lint": "ionic-app-scripts lint"
  },
  "dependencies": {
    "@angular/animations": "5.2.11",
    "@angular/common": "5.2.11",
    "@angular/compiler": "5.2.11",
    "@angular/compiler-cli": "5.2.11",
    "@angular/core": "5.2.11",
```



```

"@angular/forms": "5.2.11",
"@angular/http": "5.2.11",
"@angular/platform-browser": "5.2.11",
"@angular/platform-browser-dynamic": "5.2.11",
"@firebase/app": "^0.1.6",
"@ionic-native/core": "~4.11.0",
"@ionic-native/splash-screen": "~4.11.0",
"@ionic-native/status-bar": "~4.11.0",
"@ionic/storage": "2.1.3",
"angularfire2": "^5.0.0-rc.6",
"cordova-android": "7.0.0",
"cordova-plugin-device": "^2.0.2",
"cordova-plugin-ionic-keyboard": "^2.1.3",
"cordova-plugin-ionic-webview": "^2.1.4",
"cordova-plugin-splashscreen": "^5.0.2",
"cordova-plugin-whitelist": "^1.3.3",
"firebase": "^5.4.0",
"ionic-angular": "3.9.2",
"ionicons": "3.0.0",
"rxjs": "5.5.11",
"sw-toolbox": "3.6.0",
"zone.js": "0.8.26"
},
"devDependencies": {
  "@ionic/app-scripts": "3.1.11",
  "@ionic/lab": "1.0.8",
  "typescript": "~2.6.2"
},
"description": "An Ionic project",
"cordova": {
  "plugins": {
    "cordova-plugin-whitelist": {},
    "cordova-plugin-device": {},
    "cordova-plugin-splashscreen": {},
    "cordova-plugin-ionic-webview": {
      "ANDROID_SUPPORT_ANNOTATIONS_VERSION": "27.+"
    },
    "cordova-plugin-ionic-keyboard": {}
  },
  "platforms": [
    "android"
  ]
}
}

```

O trecho de código acima representa o arquivo package.json.

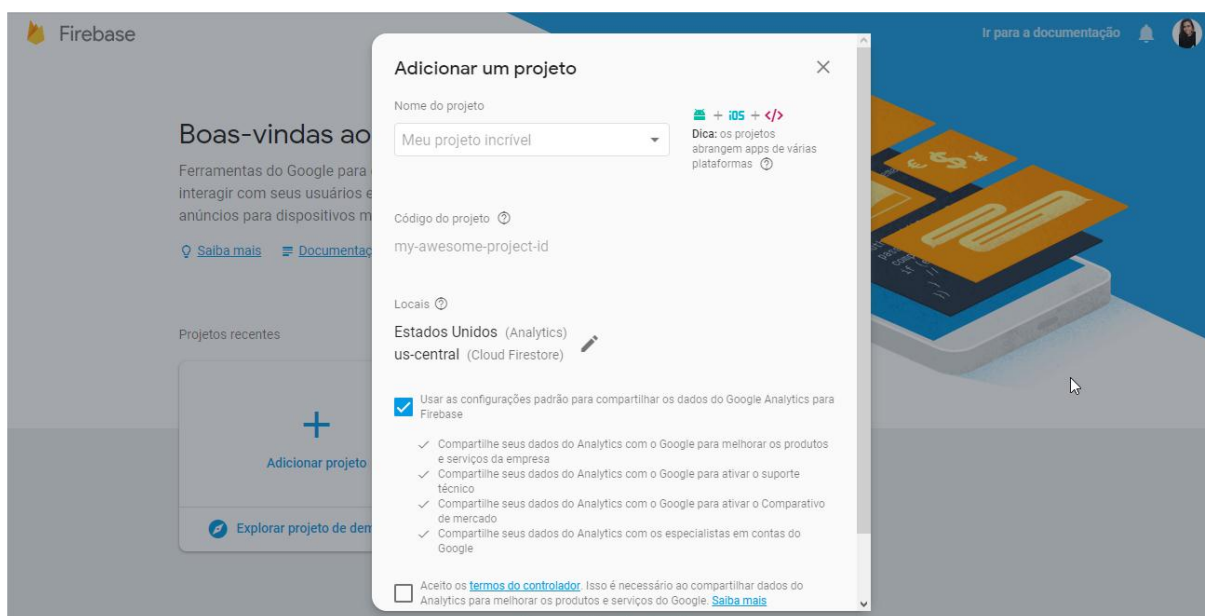
Fonte: da própria autora

Logo no início do arquivo, o mesmo apresenta o nome do projeto, autor, o endereço eletrônico e o status da aplicação, que ainda é privada. Abaixo disso estão os *scripts*, dependências e suas respectivas versões. Nota-se que nas dependências está informando a versão do ionic e do firebase que foram usados, sendo a 3.9.2 e 5.4.0 respectivamente. No final do arquivo também é apresentado a plataforma instalada no aplicativo, que no caso é a android.

3.2.5 Conexão com o Firebase

Para conexão com o firebase, o primeiro passo é a criação do projeto no site oficial⁸. Na Figura 09 é exibida a tela em que é necessário o preenchimento do nome do projeto e marcar a opção de aceitação dos termos de uso.

Figura 09: Criação do projeto no firebase



Fonte: da própria autora

Feito isso, o próximo passo a ser estabelecido é a conexão com o banco de dados na aplicação. No ionic, para configurar essa conexão é preciso acrescentar

⁸ Disponível em: <<https://console.firebase.google.com/u/0/>>

algumas linhas de código, fornecidos pelo próprio Firebase, no arquivo *app.module.ts*.

Nas configurações do Firebase é exibida uma opção para “Adicionar o firebase ao seu aplicativo da web”. Nessa opção, é apresentado as configurações do projeto criado, contendo a chave, domínio de autenticação, endereço da base de dados, id do projeto, local de armazenamento e id do remetente de mensagens.

Essas mesmas configurações, denominadas *snippet*, são coladas no arquivo *app.module.ts* no Ionic. É necessário fazer a importação de alguns módulos para que o Firebase funcione corretamente. Abaixo está um trecho do arquivo apenas com as linhas referentes ao Firebase e a Figura 10 do console na web.

Trecho de Código 02: Arquivo *app.module.ts*

```
import { AngularFireModule } from 'angularfire2';
import { AngularFireDatabase, AngularFireDatabaseModule } from
'angularfire2/database';
import { AngularFireAuthModule } from 'angularfire2/auth';

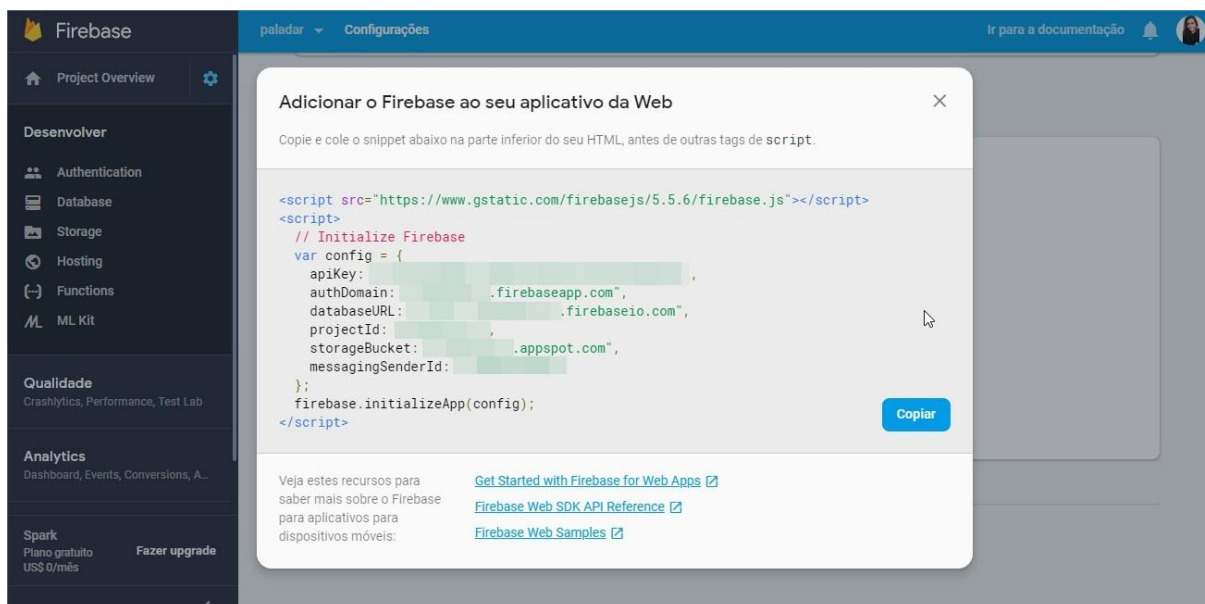
export const firebaseAppConfig = {
  apiKey: "████████████████████████████████████████",
  authDomain: "████████████████████.firebaseapp.com",
  databaseURL: "████████████████████",
  projectId: "████████████████████",
  storageBucket: "████████████████████.appspot.com",
  messagingSenderId: "████████████████████"
};

@NgModule({
  declarations: [
  ],
  imports: [
    AngularFireModule.initializeApp(firebaseAppConfig),
    AngularFireDatabaseModule,
    AngularFireAuthModule,
  ],
  bootstrap: [IonicApp],
  entryComponents: [
  ],
  providers: [
    AngularFireDatabase,
  ]
})
export class AppModule {}
```

O trecho de código acima representa o arquivo `app.module.ts`

Fonte: da própria autora

Figura 10: Configurações do firebase no console



Fonte: da própria autora

3.3 O aplicativo

O aplicativo produzido neste trabalho recebeu o nome de Paladar. Ele foi desenvolvido seguindo os requisitos, metodologias e pesquisas realizadas ao decorrer do projeto. Como a aplicação deve atender dois públicos, cliente e restaurante, foi realizado um controle de usuários para determinar os acessos e restrições entre os dois perfis.

O perfil cliente é o usuário comum. Ele terá acesso a página inicial, montagem de marmita, envio de sugestões para o restaurante e visualização das informações pessoais. O perfil restaurante é o administrador terá acesso a realização de diversos cadastros, sendo de itens, categorias, cardápio e o de usuário administrador, podendo também fazer a manipulação de cada um deles.

Para melhor controle e organização nos processos, foram definidas algumas regras. Essas regras foram implementadas na aplicação seguindo como base os procedimentos que já eram realizados pelo restaurante.

Os pratos produzidos pelo restaurante são identificados como itens no aplicativo. Esses itens podem ser arroz, feijão, macarrão, entre outros. Cada um dos itens pertence a uma categoria, sendo ela definida com base no cardápio físico que o restaurante já seguia para se organizar na exibição dos pratos. Exemplos dessas categorias são carnes, saladas e legumes.

O cardápio do restaurante é diário. Para cada dia da semana existe determinados pratos a serem feitos. Desse modo, a aplicação também seguirá essa regra. O administrador fará o cadastro do cardápio para todos os dias da semana, e fará suas alterações caso necessário.

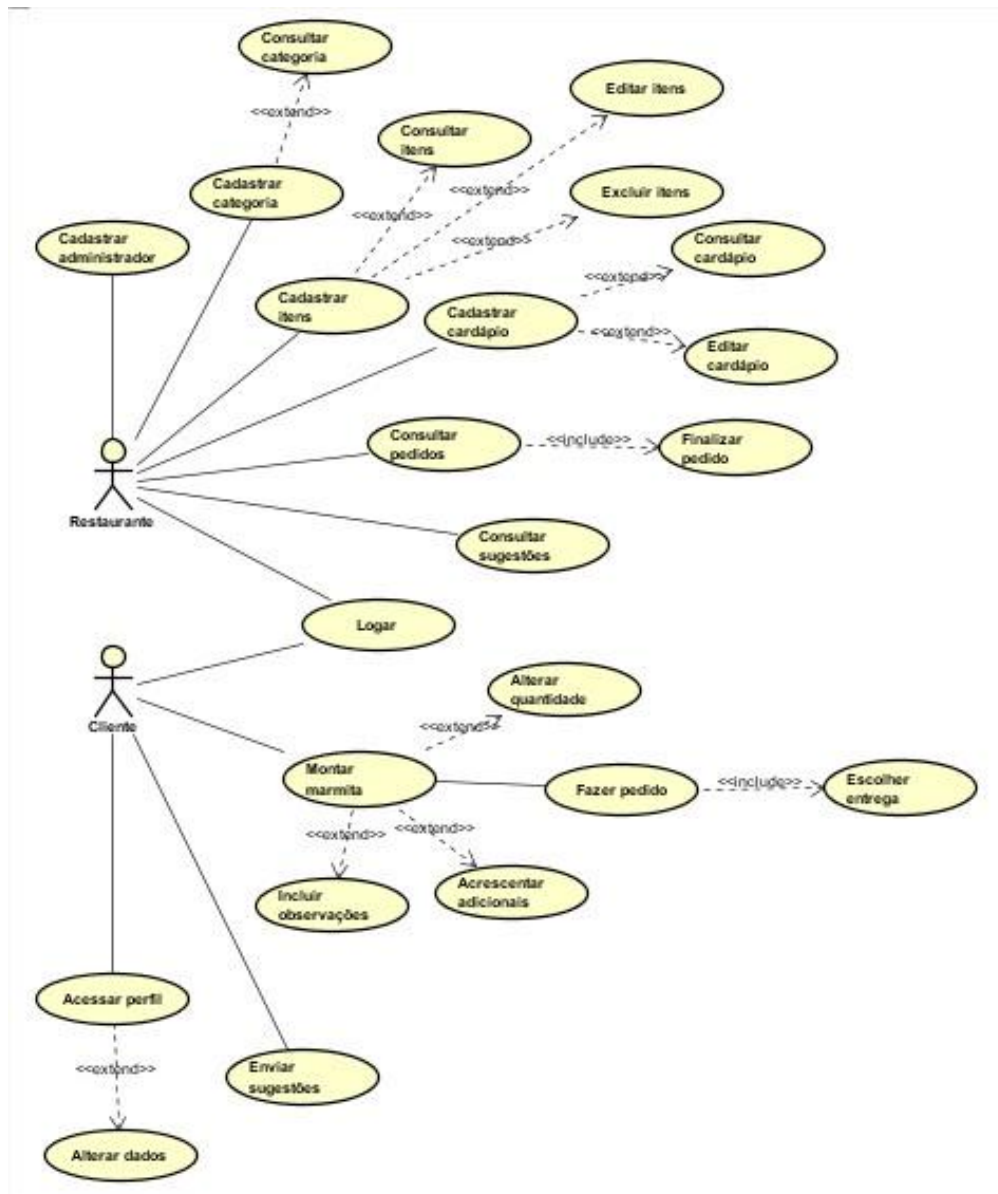
A marmitta possui um valor fixo estipulado. Para seguir esse valor o cliente deve escolher um item de cada categoria. Caso o cliente deseje adicionar mais itens a sua marmitta, serão exibidas as opções adicionais disponíveis naquele dia. Se o cliente escolher adicionais, o valor de cada item será somado ao valor da marmitta e exibido ao cliente.

Os valores dos itens adicionais são definidos pelo administrador. Ele poderá editar dados dos itens, categorias e cardápio, e fazer exclusão caso necessário.

3.3.1 Diagrama de Casos de uso

Abaixo está o diagrama de caso de uso da aplicação. Nele é exibido todas as funções do aplicativo, ondem podem ser visualizadas tanto individualmente, restaurante e cliente, como por um todo.

Figura 11: Diagrama de Casos de Uso



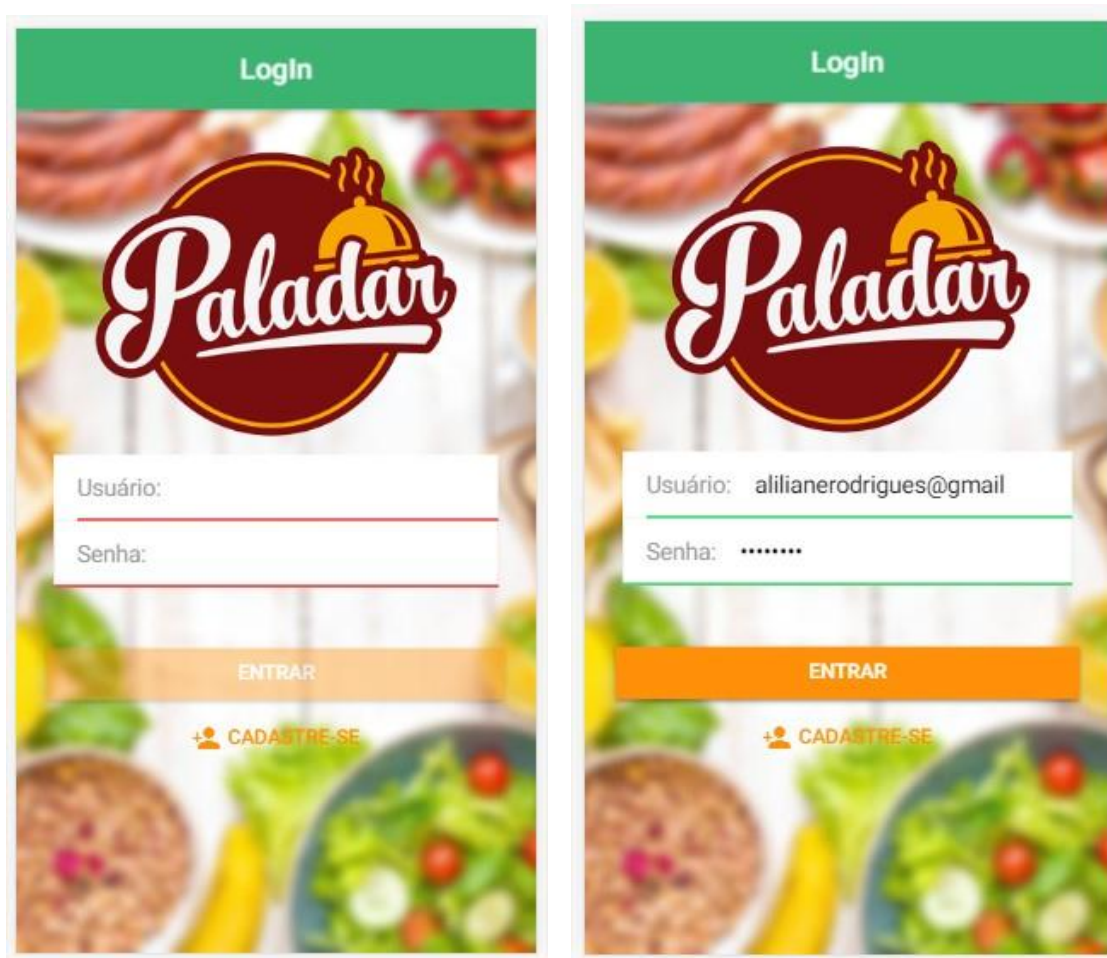
Fonte: da própria autora

3.4 Construção da aplicação

3.4.1 Login

O *login* é a tela inicial do aplicativo. É nesta tela que o usuário, caso seja cadastrado, informa as suas credenciais para ter acesso a aplicação. Caso o usuário digite credenciais incorretas ou não correspondentes aos caracteres exigidos, o aplicativo fará as validações e impedirá o acesso. Se o usuário não for cadastrado, o mesmo poderá se inscrever na opção "Cadastre-se". Quando o usuário faz o primeiro *login*, ele ficará *logado* para acessos posteriores, podendo fazer *logout* a qualquer momento. A aplicação inicialmente faz o *login* apenas com *e-mail* e senha, mas posteriormente serão implementadas o *login* pelo Facebook e Gmail.

Figura 12: Tela de login

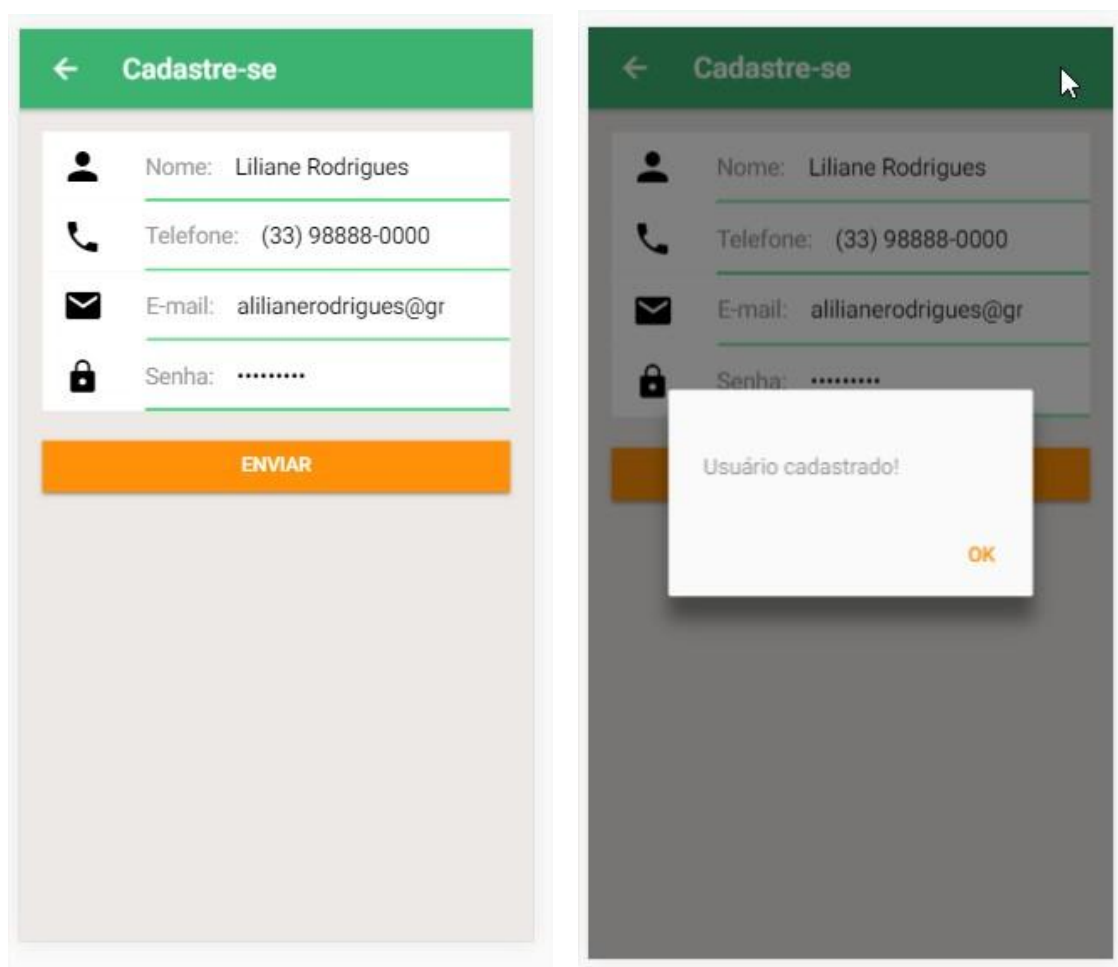


Fonte: da própria autora

3.4.2 Cadastro de usuário

Quando o usuário, na tela de login, escolhe a opção de “Cadastre-se”, ele é direcionado para a tela de Cadastro. Nela, o cliente irá preencher com algumas informações básicas, sendo, sequencialmente, o nome, telefone, e-mail e a senha escolhida para posterior acesso a aplicação. Essa tela é apenas de cadastro de clientes. Assim, quando o mesmo salva suas informações, automaticamente já faz a autenticação e entra no sistema.

Figura 13: Tela de cadastro

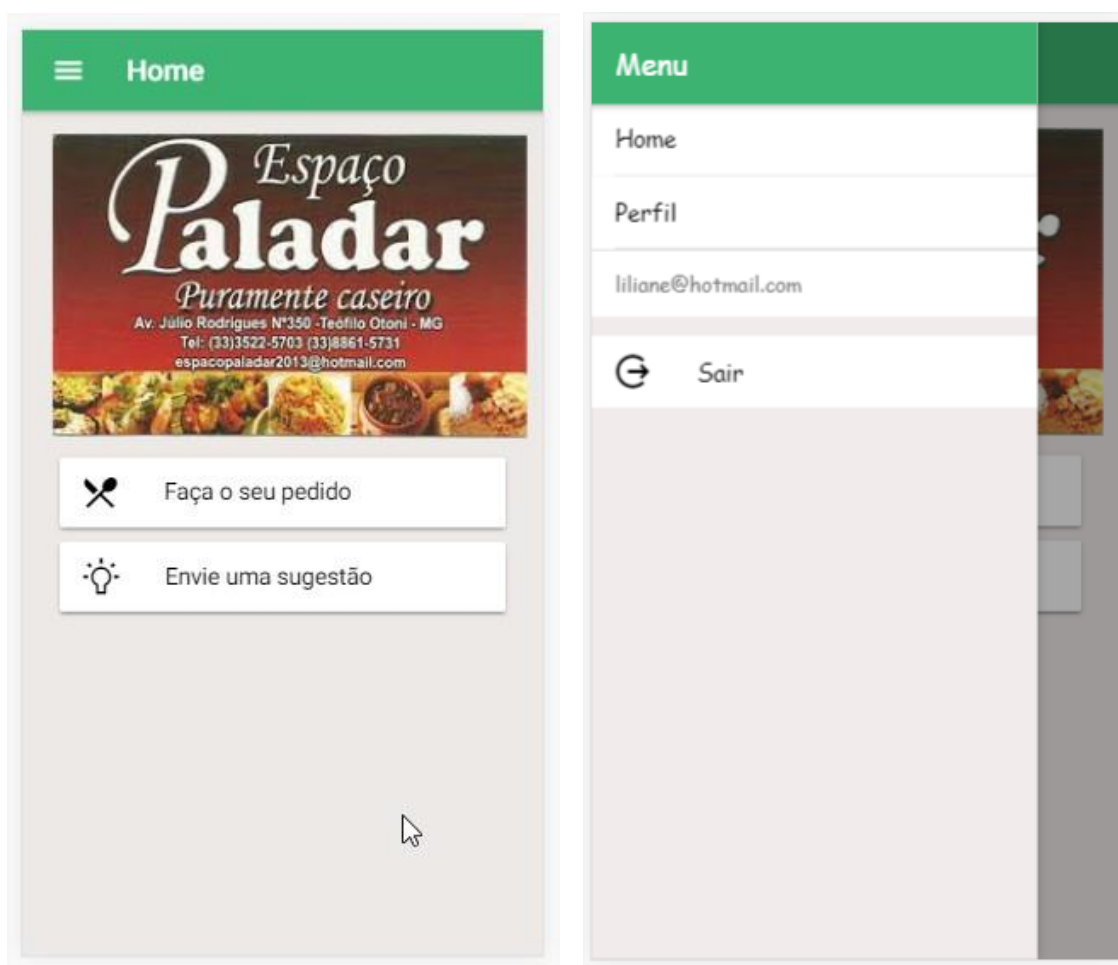


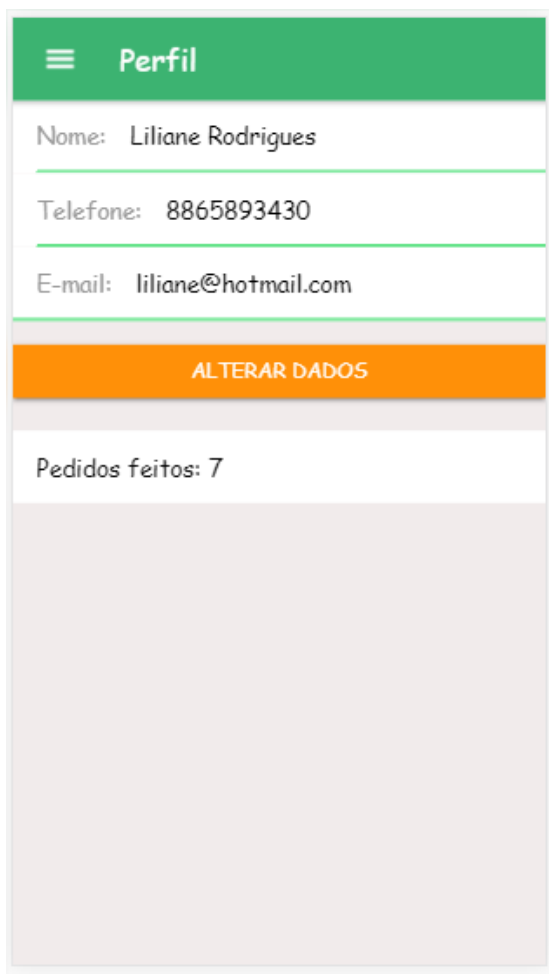
Fonte: da própria autora

3.4.3 Tela inicial e de perfil

Ao fazer login ou se cadastrar no sistema, o cliente entra na tela Home da aplicação. Nesta tela é exibido a ele algumas informações do restaurante e logo abaixo as opções de Montar a marmita e Enviar sugestões. Nela também possui um menu lateral, em que o usuário inicialmente poderá visualizar suas informações básicas que foram preenchidas no cadastro, visualizar a quantidade de pedidos já realizados, voltar a página home e fazer *logoff*.

Figura 14: Tela home e menu lateral



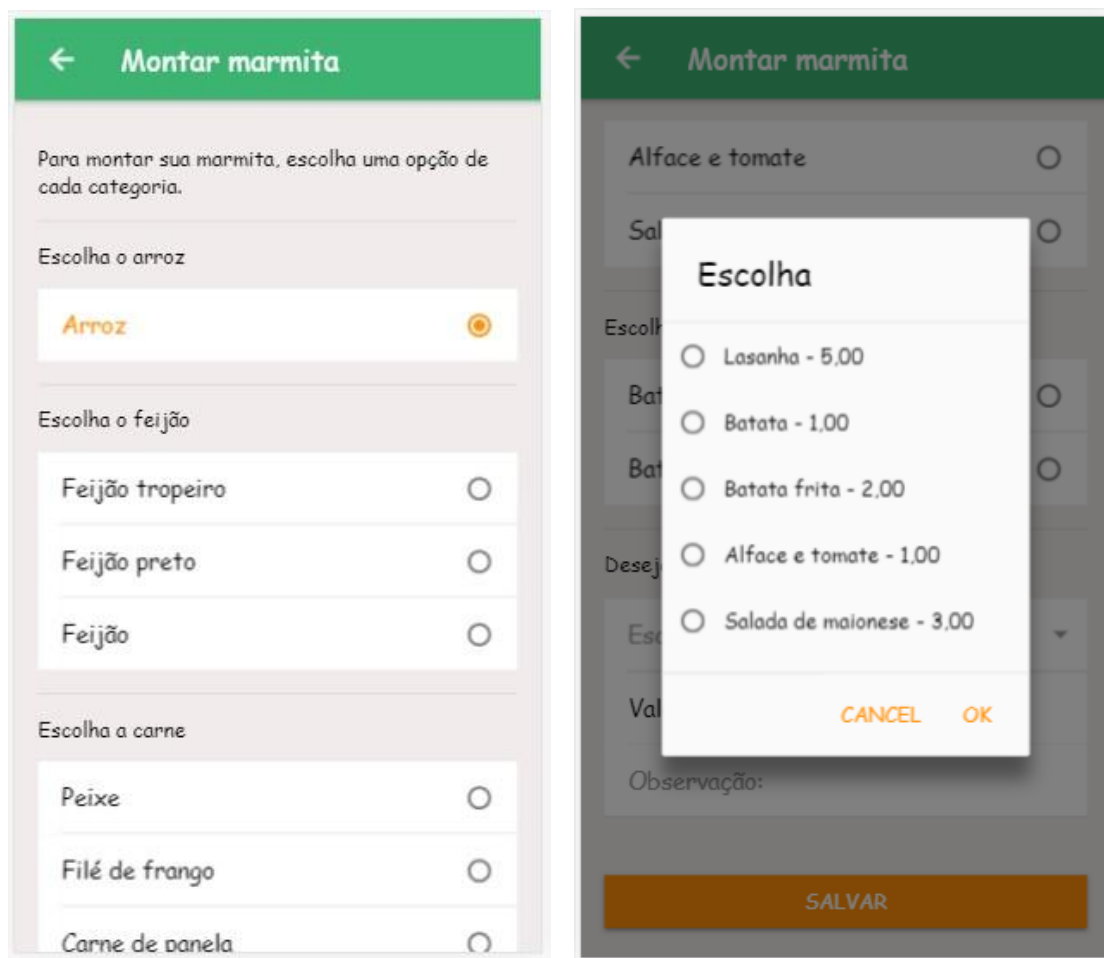


Fonte: da própria autora

3.4.4 Montagem de marmita

Na tela de montagem de marmita, é exibido para o cliente o cardápio do dia correspondente. Sendo exibidas as categorias e os itens referentes a elas. O cliente fará a escolha de um item por categoria (caso queira) e também poderá escolher itens adicionais. No final é exibido o valor final da marmita, sendo a soma do valor da marmita e dos seus adicionais (caso tenha). O cliente também poderá inserir alguma observação, como algum tempero que não queira em seu prato por exemplo. Após isso ele irá salvar a sua montagem para finalizar o pedido em sequência.

Figura 15: Tela de montar marmitta



Fonte: da própria autora

3.4.5 Finalizar pedido

Ao fazer a escolha dos itens do cardápio, o cliente irá para tela de finalização do pedido. Nesta tela é exibido a ele os itens escolhidos juntamente com os adicionais e os seus preços. Também é apresentado o valor final da marmitta e as observações. Logo abaixo é exibido ao cliente as duas opções de entrega, que é a de buscar no restaurante, que já vem selecionada por padrão, e a opção de entregar em casa. O cliente poderá voltar a tela de montagem de marmitta caso queira alterar alguma informação, ou poderá salvar o pedido, sendo essa opção a que finaliza e envia para o restaurante.

Figura 16: Tela de finalizar pedido

The image shows two side-by-side screenshots of a mobile application interface for finalizing a meal order. Both screens have a green header with the text 'finalizar-marmita'.

The left screenshot shows a list of items under the heading 'Pedido:'. The items are: Arroz, Feijão preto, Filé de frango, Salada de maionese, Batata frita, and Batata frita - 2,00. Below the list, it shows 'Valor total: 12' and 'Observações: Sem cebolinha'. At the bottom, there is a section for 'Tipo de entrega' with the option 'Retirar no restaurante' selected (indicated by a radio button).

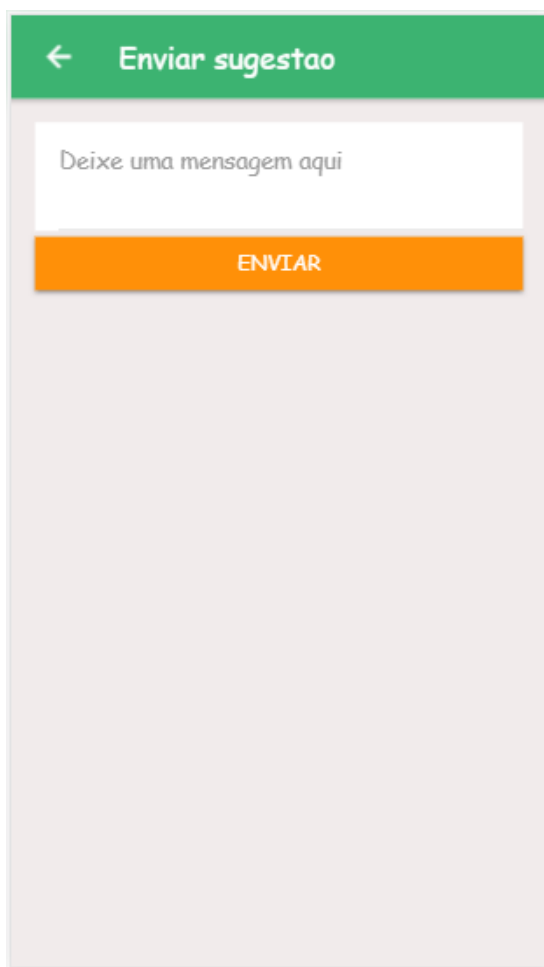
The right screenshot shows the same list of items, but with 'Batata frita' and 'Batata frita - 2,00' highlighted. Below the list, it shows 'Valor total: 12' and 'Observações: Sem cebolinha'. At the bottom, there is a section for 'Tipo de entrega' with the option 'Entrega em casa' selected (indicated by a radio button). At the very bottom, there are two buttons: 'VOLTAR' (red) and 'FINALIZAR' (orange).

Fonte: da própria autora

3.4.6 Tela de Enviar sugestões

Outra função disponível ao cliente é a de envio de sugestões. Nesta tela ele poderá enviar ao restaurante qualquer sugestão referente a pratos, ou até mesmo alguma crítica ou elogio. Essa é uma das formas em que o restaurante conseguirá acompanhar a satisfação do cliente e saberá onde deve investir mais. A tela de envio contém apenas um campo de texto e um botão de envio.

Figura 17: Tela de envio de sugestões

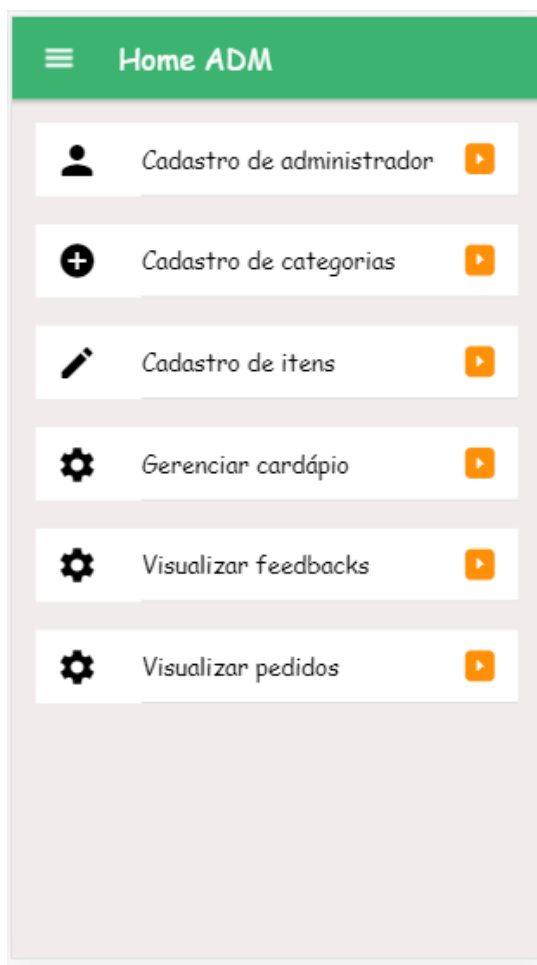


Fonte: da própria autora

3.4.7 Tela inicial administrador

A tela de Home para o administrador mostra as opções disponíveis para o restaurante, sendo o cadastro de administrador, categorias, itens e cardápio.

Figura 18: Tela Home administrador

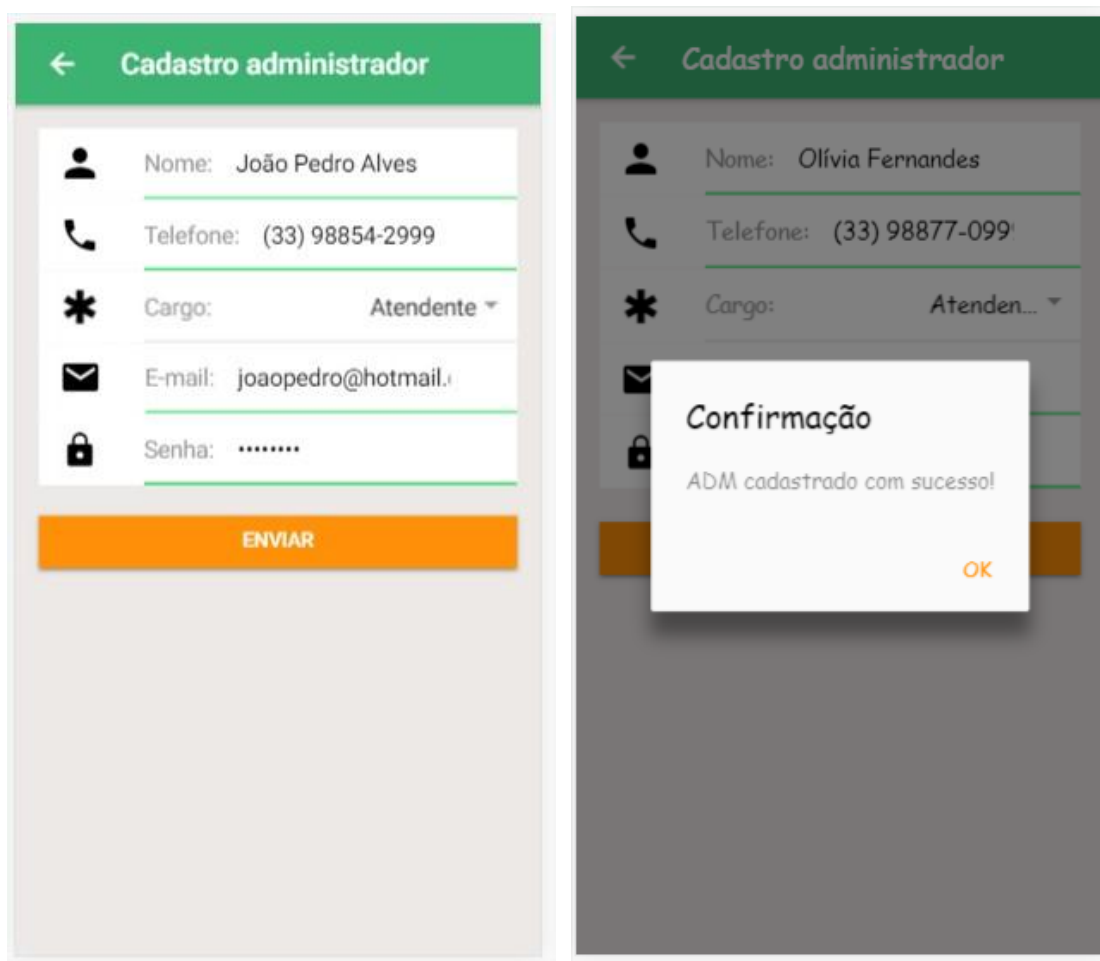


Fonte: da própria autora

3.4.8 Cadastrar administrador

A tela de cadastro de administrador obedece aos mesmos princípios do cadastro de usuário. O único diferencial seguido é que para o administrador é definido um cargo, podendo ser gerente ou atendente.

Figura 19: Tela de cadastrar administrador

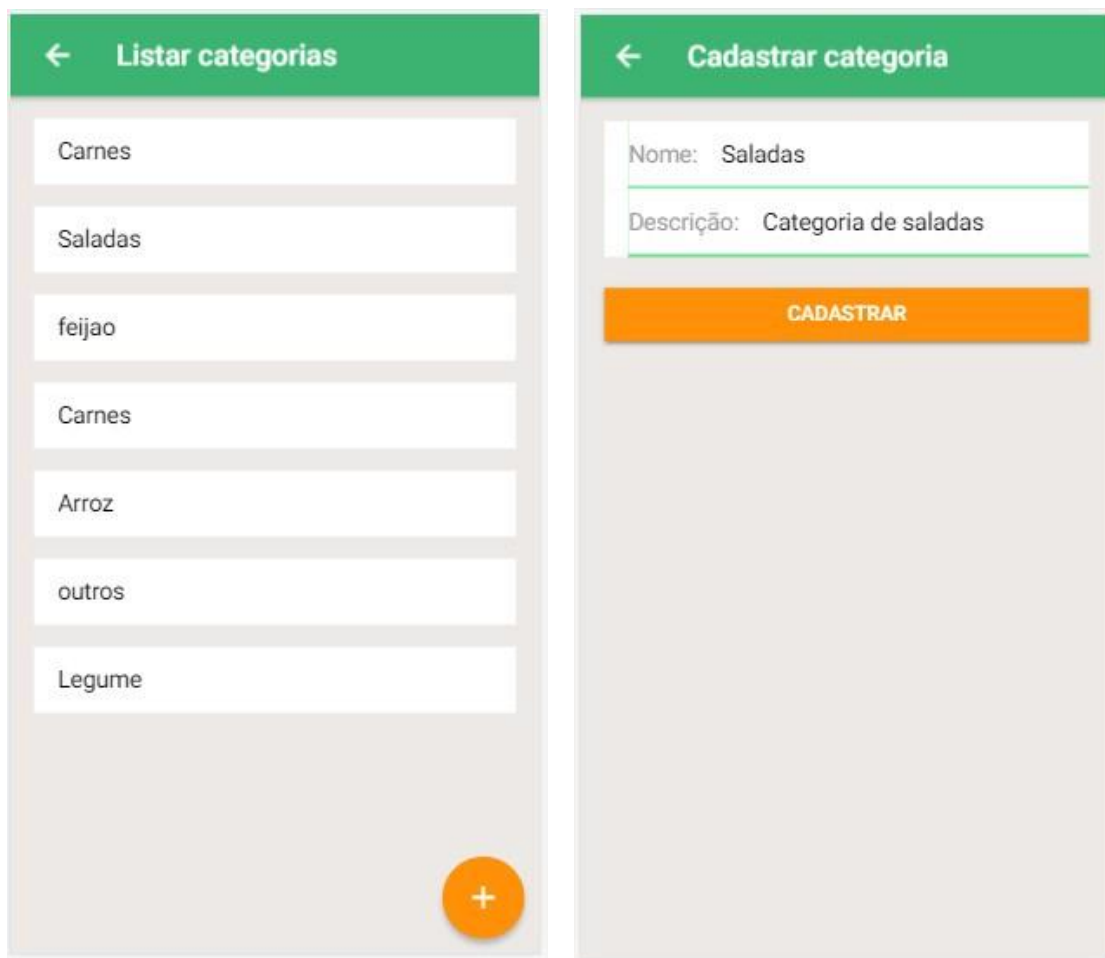


Fonte: da própria autora

3.4.9 Cadastrar categoria

A tela de cadastro de categoria exibe todas as categorias cadastradas (caso tenha). A tela possui um botão flutuante com o símbolo “mais”, sendo ela a opção de fazer o cadastro da categoria através do nome e descrição.

Figura 20: Tela de cadastrar categoria

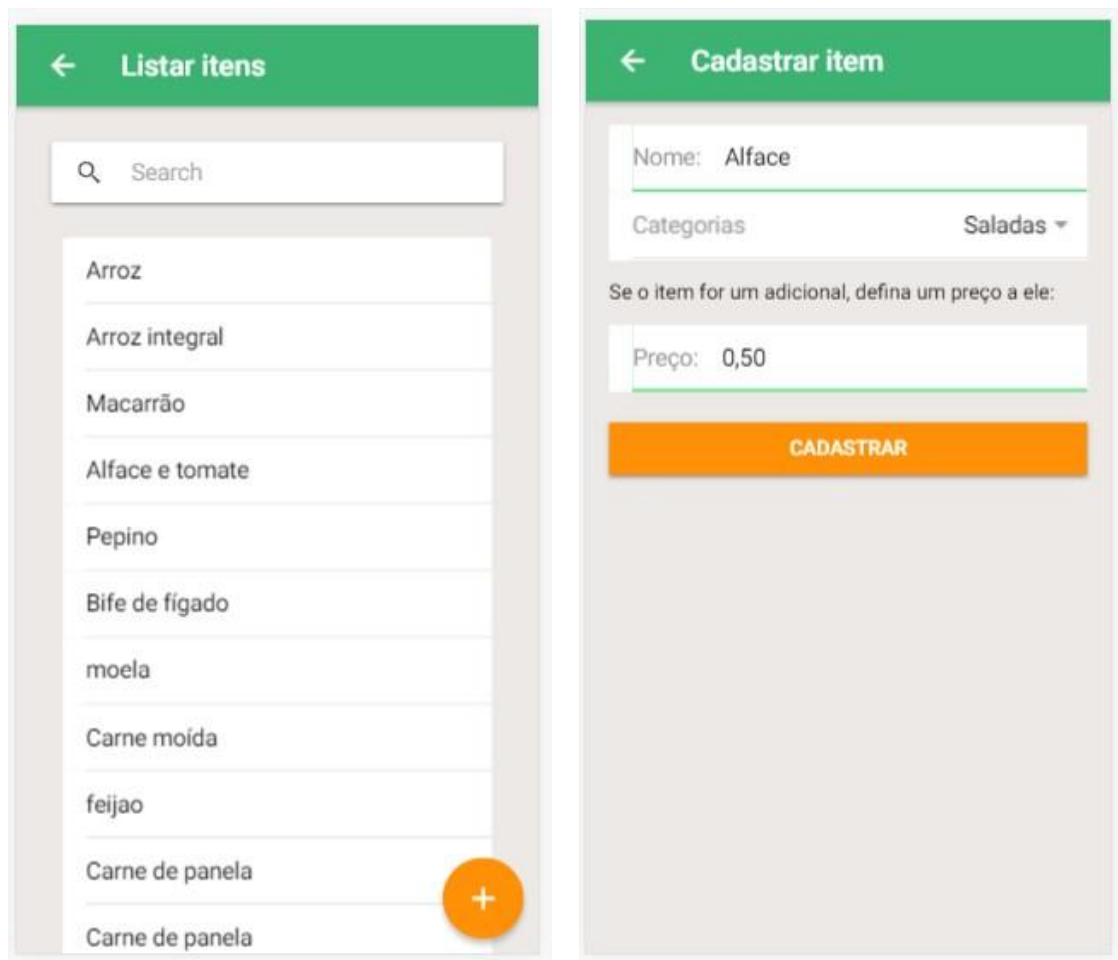


Fonte: da própria autora

3.4.10 Manipulação de itens

A tela de cadastro de itens segue os mesmos princípios da categoria. Seu único diferencial é o campo de pesquisa localizado acima da listagem. O campo de pesquisa permite ao usuário pesquisar algum item pelo nome. Essa tela também possui um botão flutuante para o cadastro de itens.

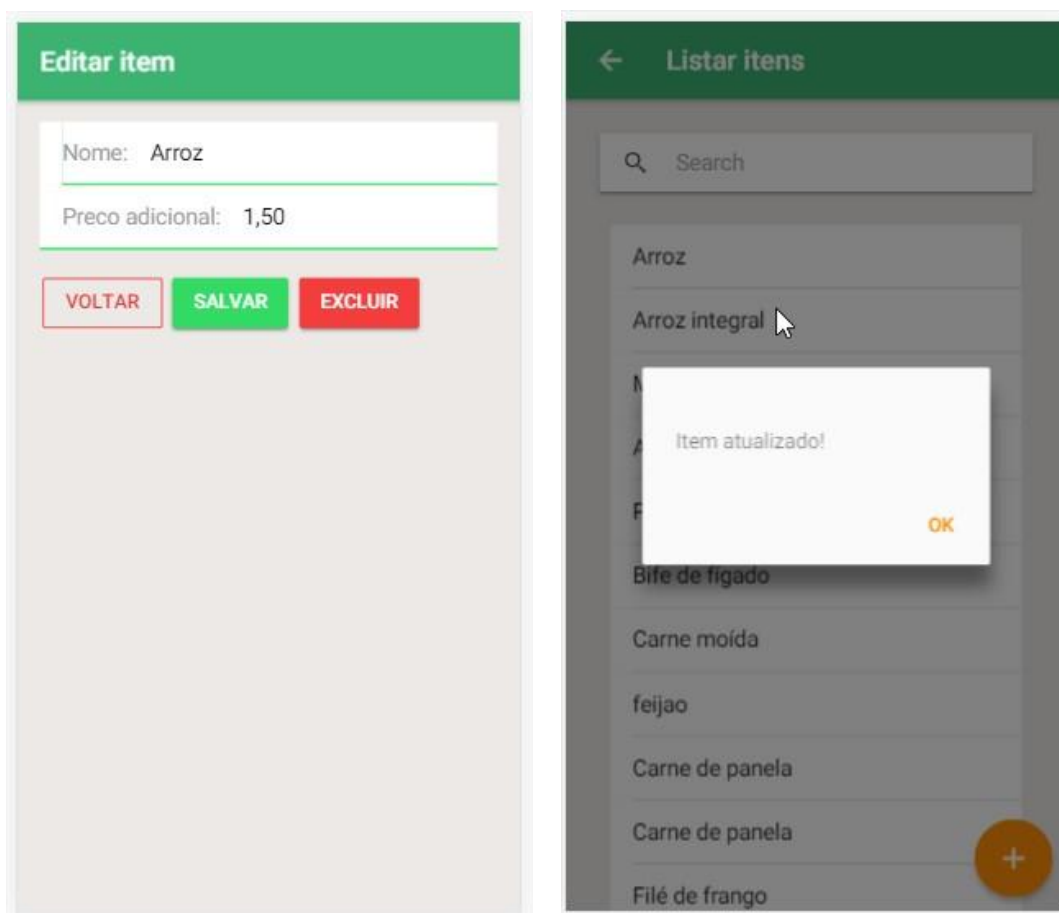
Figura 21: Tela de cadastrar itens



Fonte: da própria autora

Ao clicar em algum item na tela de listar itens, o usuário será direcionado para a tela de edição de itens. Nesta tela ele poderá fazer alteração do nome ou do preço adicional de um item, e também excluir aquele item do banco de dados.

Figura 22: Tela de editar itens



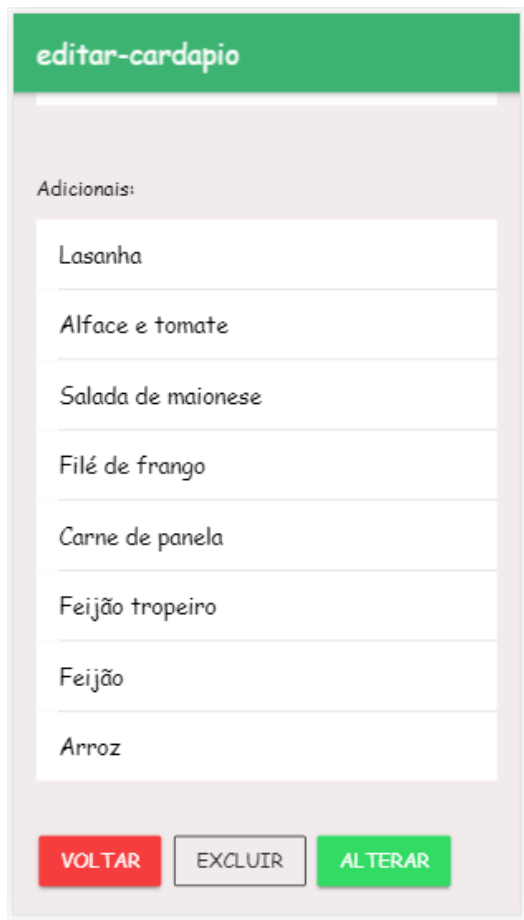
Fonte: da própria autora

3.4.11 Gerenciar cardápio

A tela de gerenciamento do cardápio exibe através de lista os cardápios da semana e um botão flutuante referenciando a tela de cadastro do cardápio. Ao clicar em algum dia da semana, são exibidos o cardápio referente e as opções de edição ou exclusão. Ao clicar no botão flutuante, é aberta a tela de cadastro, em que são escolhidos o dia da semana, e os itens para cada cardápio.

Figura 23: Tela de gerenciar cardápio





editar-cardapio

Adicionais:

- Lasanha
- Alface e tomate
- Salada de maionese
- Filé de frango
- Carne de panela
- Feijão tropeiro
- Feijão
- Arroz

VOLTAR EXCLUIR ALTERAR

Fonte: da própria autora

3.4.12 Cadastrar cardápio

No cadastro do cardápio, o administrador escolhe o dia da semana em que pretende fazer o cadastro, e sequencialmente vai escolhendo por categorias os itens que irão disponibilizar naquele dia. Após isso, a aplicação exibe o preço padrão da marmitta, que poderá ser alterado apenas para o dia em que está realizando o cadastro. Finalizando a escolha, é exibida a opção para salvar o cardápio.

Figura 24: Tela de cadastrar cardápio

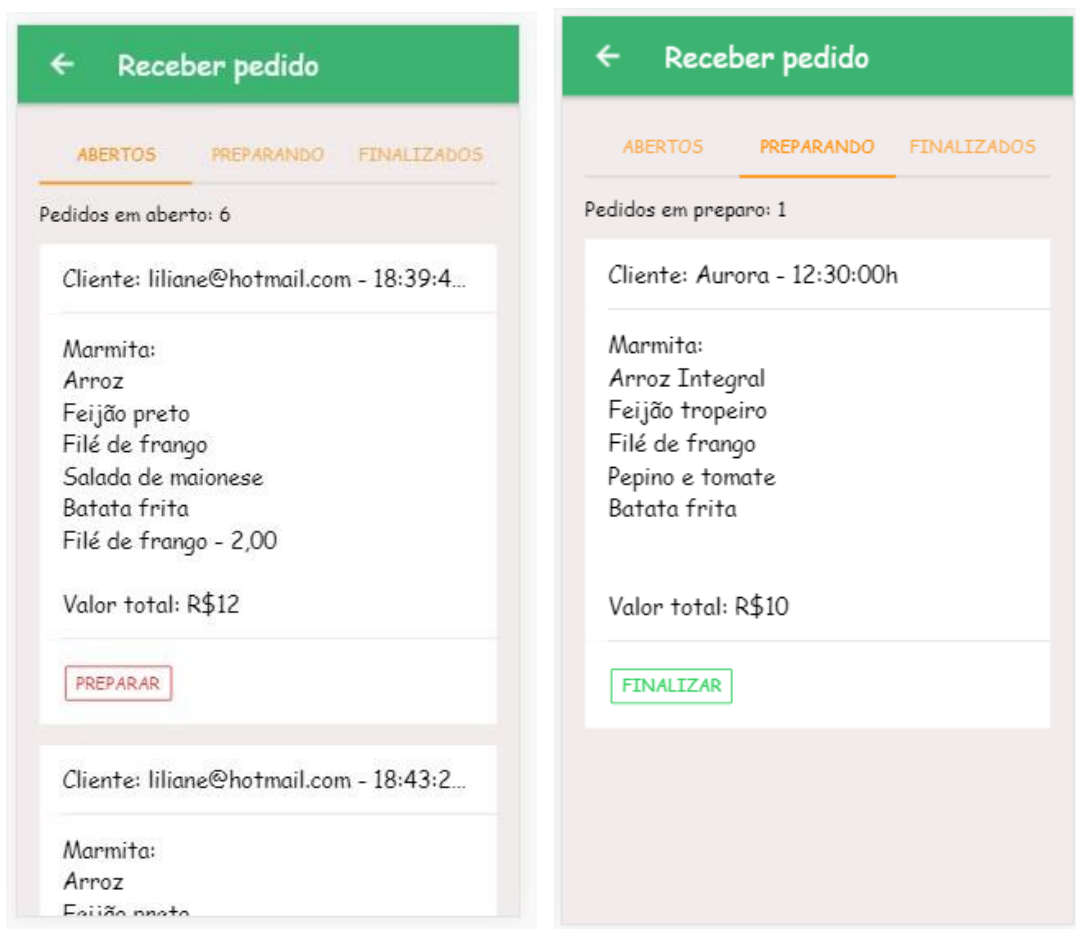
The image displays two side-by-side screenshots of a mobile application interface for registering a menu. Both screens have a green header with a back arrow and the text 'Cadastrar cardápio'. The left screenshot shows a form with the following fields: 'Dia da semana' (Domingo), 'Arroz', 'Feijão', 'Carnes', 'Saladas', 'Legumes', and 'Outros', each with a dropdown arrow. Below these is a text field containing 'Valor da marmita: R\$ 10'. The right screenshot shows the same form, but the dropdown menus are populated with specific items: 'Arroz', 'Feijão, Fe...', 'Carne de ...', 'Alface e t...', 'Batata, B...', and 'Strogonoff'. At the bottom of the right screenshot is an orange button labeled 'SALVAR CARDÁPIO'.

Fonte: da própria autora

3.4.13 Receber pedidos

Na tela de recebimento dos pedidos, é exibida ao administrador três opções de pedidos: Abertos, Preparando e Finalizados. O administrador irá visualizar todos os pedidos que estão sendo realizados na aba de prontos, sendo que este pedido contém o cliente, a marmita montada, valor total, horário do pedido e as observações caso tenha. Quando o pedido é lido, o administrador clica na opção de “Preparar” para direcionar o pedido ao preparo. Na aba de preparar, o administrador terá a opção de “Finalizar” o pedido.

Figura 25: Tela de receber pedidos



Fonte: da própria autora

3.4.14 Visualizar feedbacks

A tela de visualizar feedbacks é onde o administrador poderá ler todas as sugestões que são enviadas pelo cliente, sendo que será implementado posteriormente opção de dar retornos às mensagens.

Figura 26: Tela de visualizar feedbacks

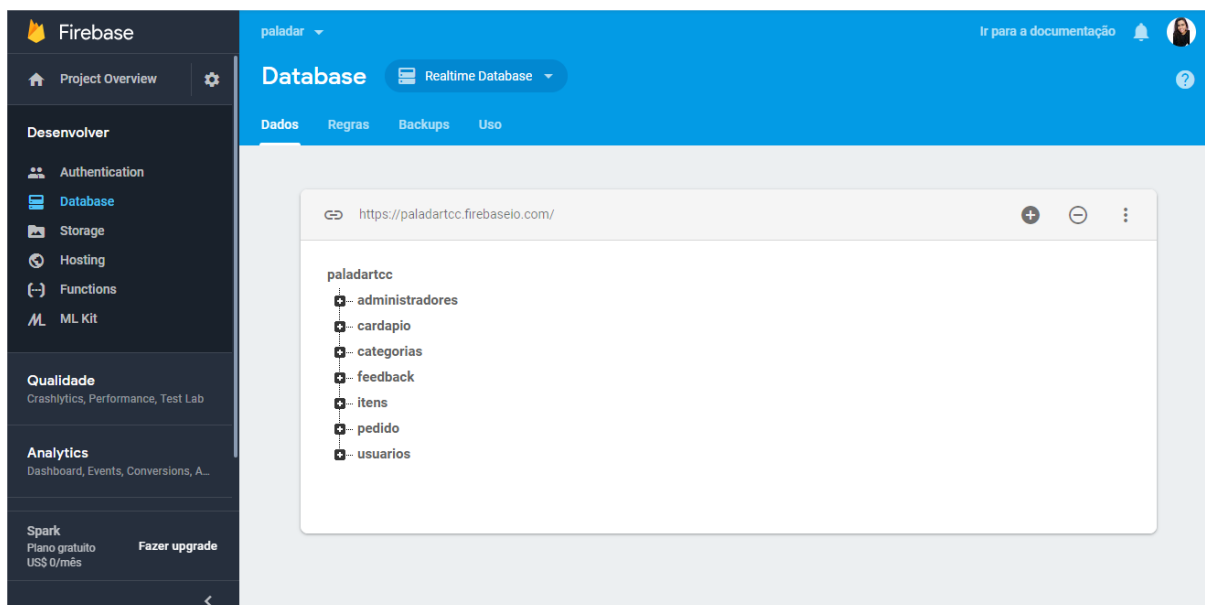


Fonte: da própria autora

3.4.15 Banco de dados

O Firebase *Realtime Database* utiliza a estrutura de documentos JSON para armazenamento de dados. Com isso, foram definidos os documentos a seguir, sendo: administradores, cardápio, categorias, itens, pedido e usuário, conforme é exibido pelo console na Figura 27.

Figura 27: Estrutura de armazenamento



Fonte: da própria autora

3.4.15.1 CRUD

O CRUD (*Create, Read, Update e Delete*) no banco de dados, corresponde às operações de inserir, selecionar, atualizar e excluir registros. Nos trechos de código abaixo mostram como essas operações são realizadas com o ionic e firebase.

Trecho de código 03: Criação de registros

```
import { Injectable } from '@angular/core';
import { AngularFireDatabase } from
'../../../../../node_modules/angularfire2/database';
import { Categoria } from '../../modelos/categoria;

@Injectable()
export class ItensProvider {

  constructor(private db: AngularFireDatabase) {
```



```

    }

    private cat = this.db.list<Categoria>('/categorias');

    createCategoria(categoria: Categoria){
        return this.cat.push(categoria);
    }
}

```

O trecho de código acima representa a inserção de um registro no banco de dados.

Fonte: da própria autora

Trecho de código 04: Seleção de registros

```

import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-
angular';
import { AngularFireDatabase } from 'angularfire2/database';
import { Observable } from '../node_modules/rxjs';
import { CadastroCategoriaPage } from '../cadastro-
categoria/cadastro-categoria';

@IonicPage()
@Component({
    selector: 'page-listar-categoria',
    templateUrl: 'listar-categoria.html',
})
export class ListarCategoriaPage {

    categorias: Observable<any[]>;

    constructor(
        public navCtrl: NavController,
        public NavParams: NavParams,
        private db: AngularFireDatabase,) {

```

```

    this.categorias = db.list("categorias").valueChanges();

}
}

```

O trecho de código acima representa a seleção de um registro no banco de dados.

Fonte: da própria autora

Trecho de código 05: Atualização de registros

```

import { Injectable } from '@angular/core';
import { AngularFireDatabase } from
'../../../../../node_modules/angularfire2/database';
import { Categoria } from '../../modelos/categoria;

@Injectable()
export class ItensProvider {

    constructor(private db: AngularFireDatabase) {

    }

    private cat = this.db.list<Categoria>('/categorias');

    update(key, categoria) {
        return this.cat.update(key, categoria);
    }
}
}

```

O trecho de código acima representa a atualização de um registro no banco de dados.

Fonte: da própria autora

Trecho de código 06: Exclusão de registros

```

import { Injectable } from '@angular/core';
import { AngularFireDatabase } from
'../../../../../node_modules/angularfire2/database';

```

```
import { Categoria } from '../../modelos/categoria;

@Injectable()
export class ItensProvider {

  constructor(private db: AngularFireDatabase) {
  }

  private cat = this.db.list<Categoria>('/categorias');

  remove(key) {
    return this.cat.remove(key);
  }
}
```

O trecho de código acima representa a exclusão de um registro no banco de dados.

Fonte: da própria autora

3.4.15.2 Autenticação

O próprio firebase oferece recursos de autenticação. Assim, foi desenvolvido um *provider* responsável por essa função e pela comunicação com a aplicação. A seguir está o trecho de código correspondente a função de autenticação.

Trecho de código 07: Arquivo auth.ts

```
import { Injectable } from '@angular/core';
import { AngularFireAuth } from 'angularfire2/auth';
import firebase from 'firebase/app';

@Injectable()
export class AuthProvider {

  private user: firebase.User;
```

```
constructor(public afAuth: AngularFireAuth) {
  afAuth.authState.subscribe(user => {
    this.user = user;
  });
}

signin(credenciais) {
  console.log('Sign in');
  return
this.afAuth.auth.signInWithEmailAndPassword(credenciais.email,
  credenciais.senha);
}
signup(credenciais) {
  return
this.afAuth.auth.createUserWithEmailAndPassword(credenciais.email,
  credenciais.senha);
}
get authenticated(): boolean {
  return this.user !== null;
}
public getEmail() {
  return this.user && this.user.email;
}
signOut(): Promise<void> {
  return this.afAuth.auth.signOut();
}
}
```

O trecho de código acima representa o arquivo auth.ts.

Fonte: da própria autora

Pelo console do firebase, é possível visualizar todos os usuários que estão cadastrados para autenticação na aplicação, sendo tanto cliente como administrador. Nesta tela é possível verificar o identificador, que é o e-mail cadastrado, data de criação do usuário, última data de conexão no aplicativo e o id do usuário. A autenticação oferece as funções para redefinição de senha, desativação e exclusão da conta do usuário.

Figura 28: Usuários da autenticação

The screenshot shows the Firebase Authentication console. The left sidebar contains navigation options: Desenvolver (Authentication, Database, Storage, Hosting, Functions, ML Kit), Qualidade, Analytics, Ampliar, and Spark. The main content area is titled 'Authentication' and has tabs for 'Usuários', 'Método de login', 'Modelos', and 'Uso'. The 'Usuários' tab is active, displaying a table of users. A search bar is at the top of the table with the text 'Pesquise por endereço de e-mail, número de telefone ou UID do usuário'. A blue button 'Adicionar usuário' is to the right of the search bar. The table has the following data:

Identificador	Provedores	Criado em	Conectado	UID do usuário ↑
liliane@hotmail.com	✉	17 de ago de ...	28 de out de 2...	2fIHpv9qpkPaFAtqir3lwzguJ...
katia@katia.com	✉	16 de out de 2...	16 de out de 2...	8YPLflu9MRh2wDXsOXdz...
luiza@hotmail.com	✉	26 de set de 2...	26 de set de 2...	Dc3frZKch6MxbFChPYEYyyqHQ8G3...
joaopedro@hotmail.com	✉	13 de out de 2...	13 de out de 2...	WJJJaaEkgHvVkJx1706eUvmHR...
adm@paladar.com	✉	16 de out de 2...	28 de out de 2...	oLHEsXmuFIOj1Qj0JsoQ408UM1y2

A context menu is open over the first user, showing options: Redefinir senha, Desativar conta, and Excluir conta. At the bottom right of the table, it says 'Linhas por página: 50' and '1-5 de 5'.

Fonte: da própria autora

Como já informado do tópico do login, o mesmo está sendo feito apenas pelo *e-mail* e senha, pois foi a opção ativada e implementada. Na Figura 29 está a aba de Método de login, da autenticação, onde são disponíveis todos os modos de login fornecidos pela ferramenta.

Figura 29: Métodos de login

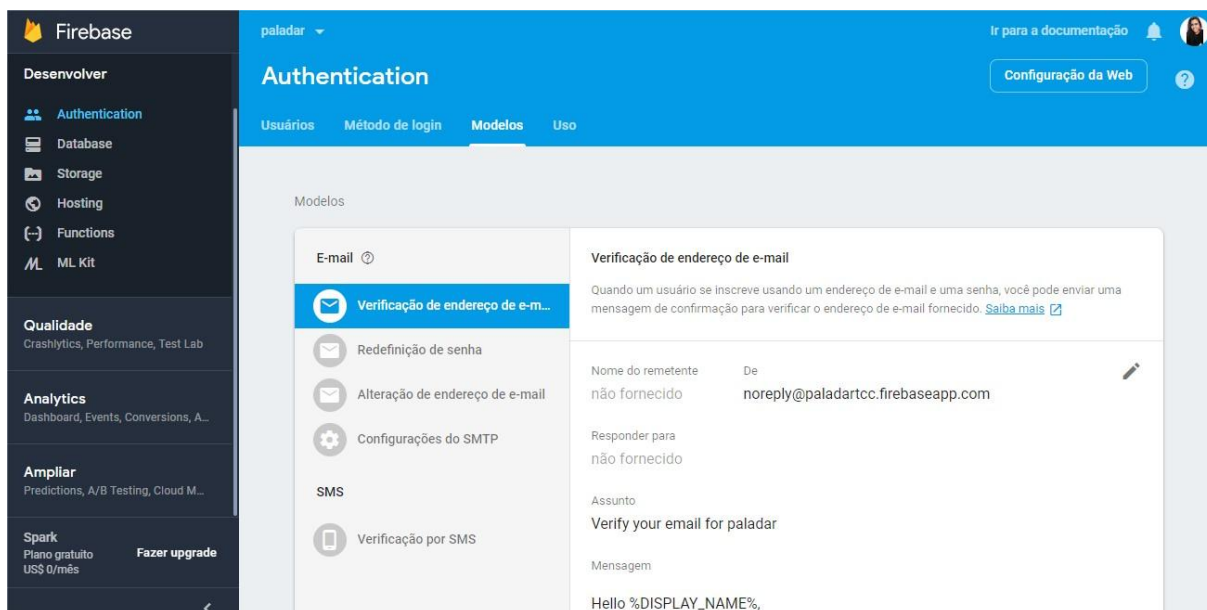
The screenshot shows the Firebase Authentication console with the 'Método de login' tab selected. The 'Provedores de login' section is visible, listing various login providers and their status:

Provedor	Status
✉ Email/senha	Ativado
☎ Smartphone	Desativado
🌐 Google	Desativado
🎮 Play Games	Desativado
📘 Facebook	Desativado
🐦 Twitter	Desativado
🐙 GitHub	Desativado
👤 Anônimo	Desativado

Fonte: da própria autora

Outro recurso da autenticação são os modelos de *e-mail*. Entre eles estão o de verificação do endereço de *e-mail*, redefinição de senha e alteração do endereço de *e-mail*. Esses modelos são fornecidos pelo firebase para facilitar a implementação no aplicativo.

Figura 30: Modelos de e-mail



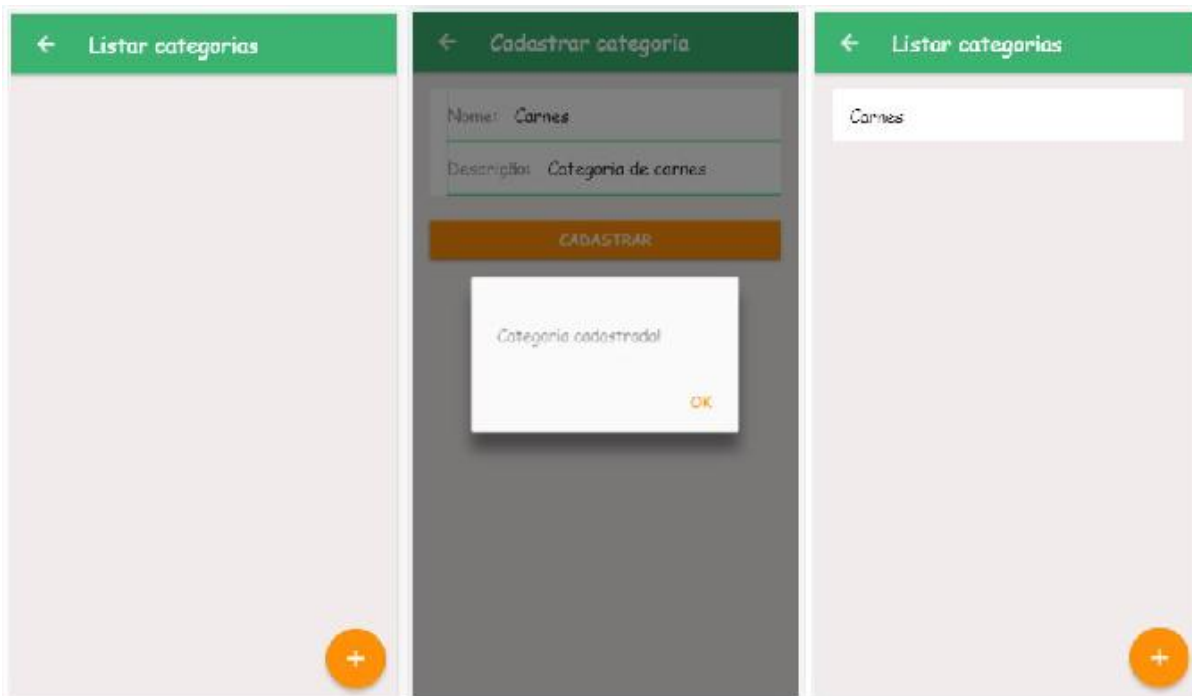
Fonte: da própria autora

4. TESTES E RESULTADOS

O objetivo deste capítulo é mostrar o funcionamento da aplicação através dos testes e a análise dos resultados do desenvolvimento do aplicativo móvel. Todos os dados utilizados na base de dados são fictícios, apenas para a exibição do funcionamento e para auxiliar na coleta dos resultados.

O primeiro teste a ser realizado é o de cadastro de categorias. Nele será cadastrado a categoria com nome “Carnes”, conforme mostra a Figura 31.

Figura 31: Teste de cadastro de categorias

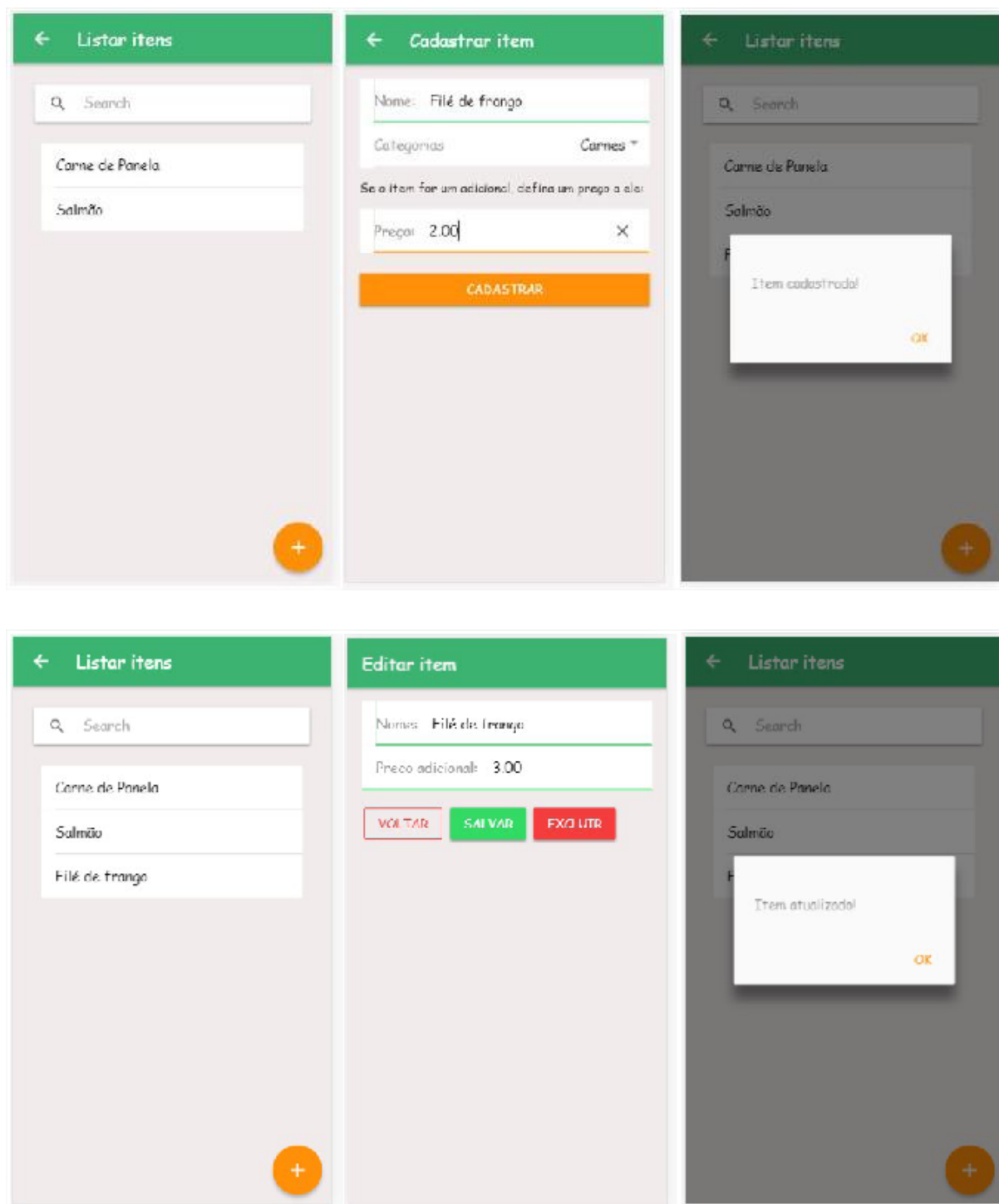


Fonte: da própria autora

O segundo teste é o de cadastro e edição de itens. Neste, já haviam alguns pratos cadastrados conforme são exibidos na listagem. No teste será cadastrado o

item “Filé de Frango” com valor adicional de “R\$2,00”. Posteriormente, na edição, o valor do item será alterado para “R\$3,00”.

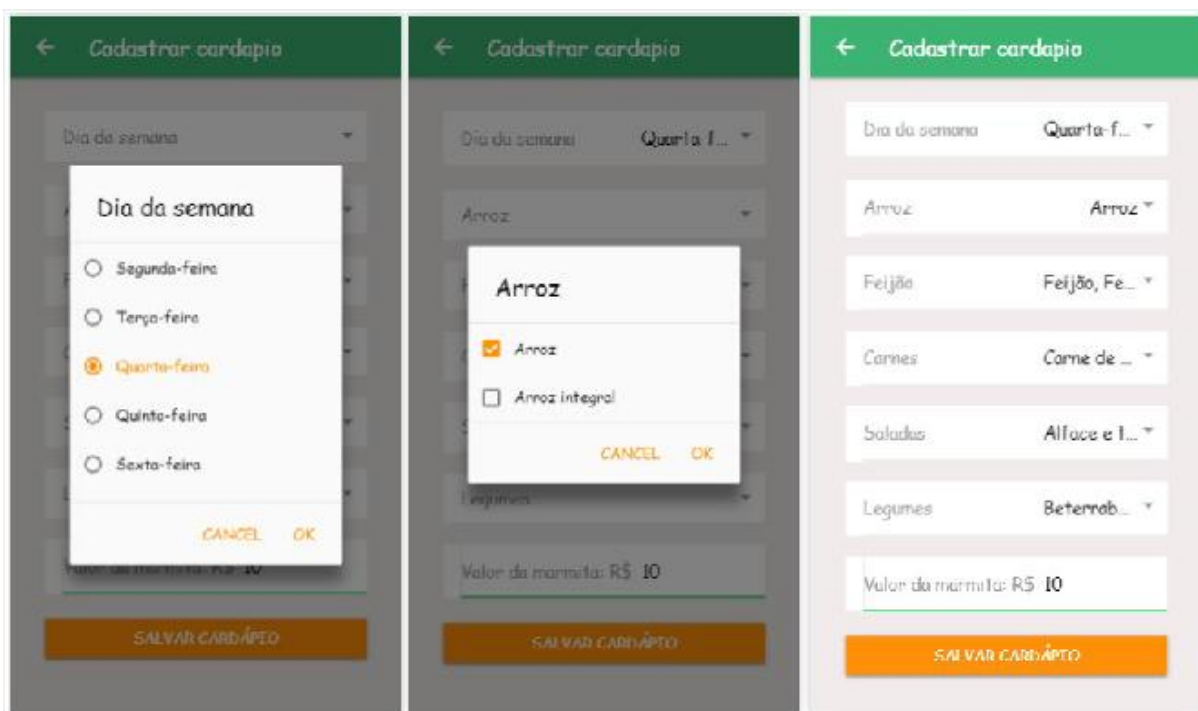
Figura 32: Teste de cadastro e edição de itens



Fonte: da própria autora

O terceiro teste é o cadastro de cardápio. O administrador escolherá dia e os itens que deseja. Ao finalizar e clicar em salvar cardápio, é exibida uma mensagem de confirmação caso todos os campos tenham sido preenchidos. Caso contrário, a aplicação retornará erro.

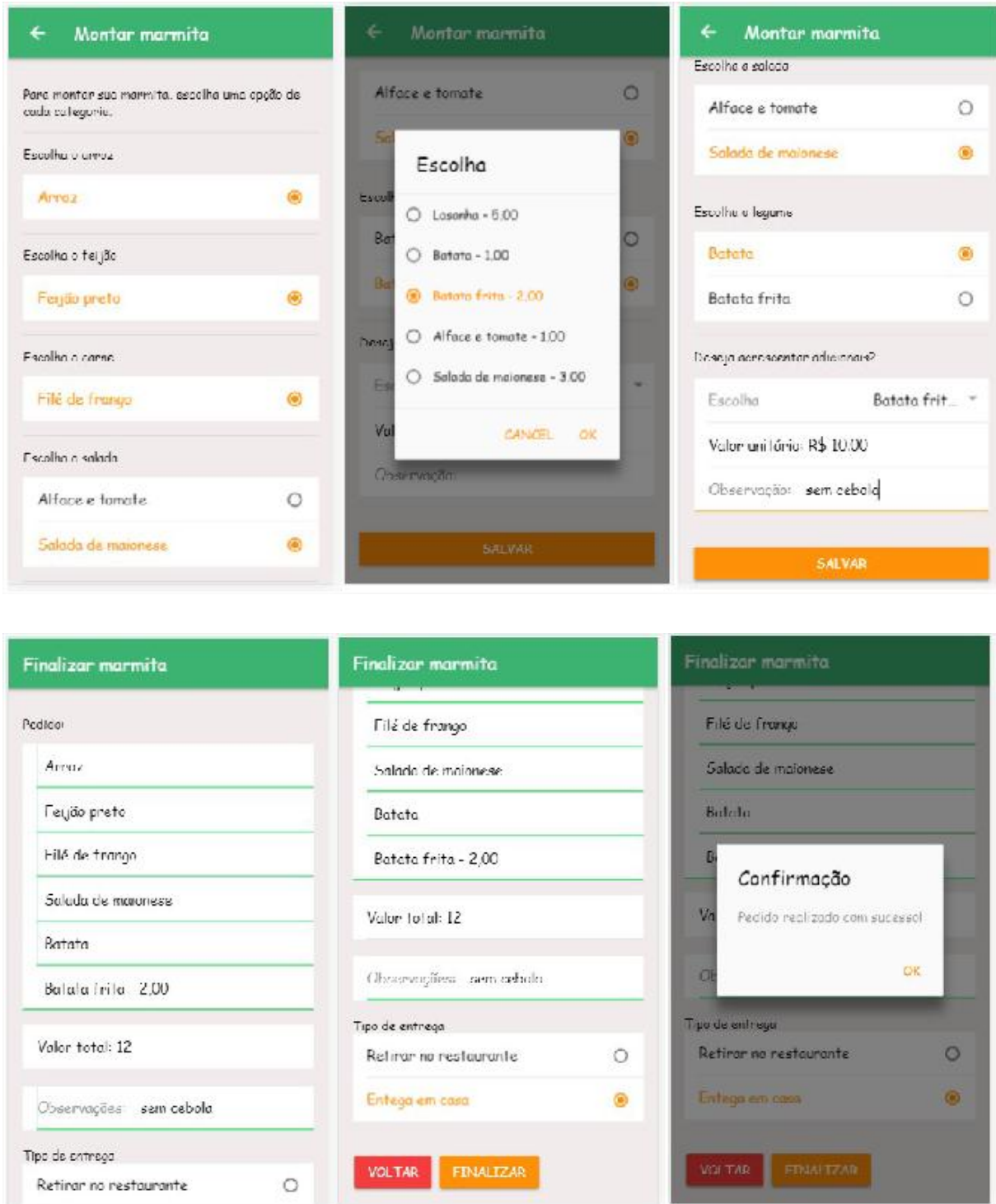
Figura 33: Teste cadastro de cardápio



Fonte: da própria autora

O quarto teste consiste na montagem da marmitta. Nesta tela, o cliente poderá escolher um item por categoria e também um item adicional, caso queira acrescentar na marmitta. Após isso é exibido na tela tudo o que foi pedido e o valor total com as observações para finalização do pedido. Caso o cliente não marque alguma opção da categoria, a mesma será desconsiderada.

Figura 34: Teste do processo de montagem de marmitta

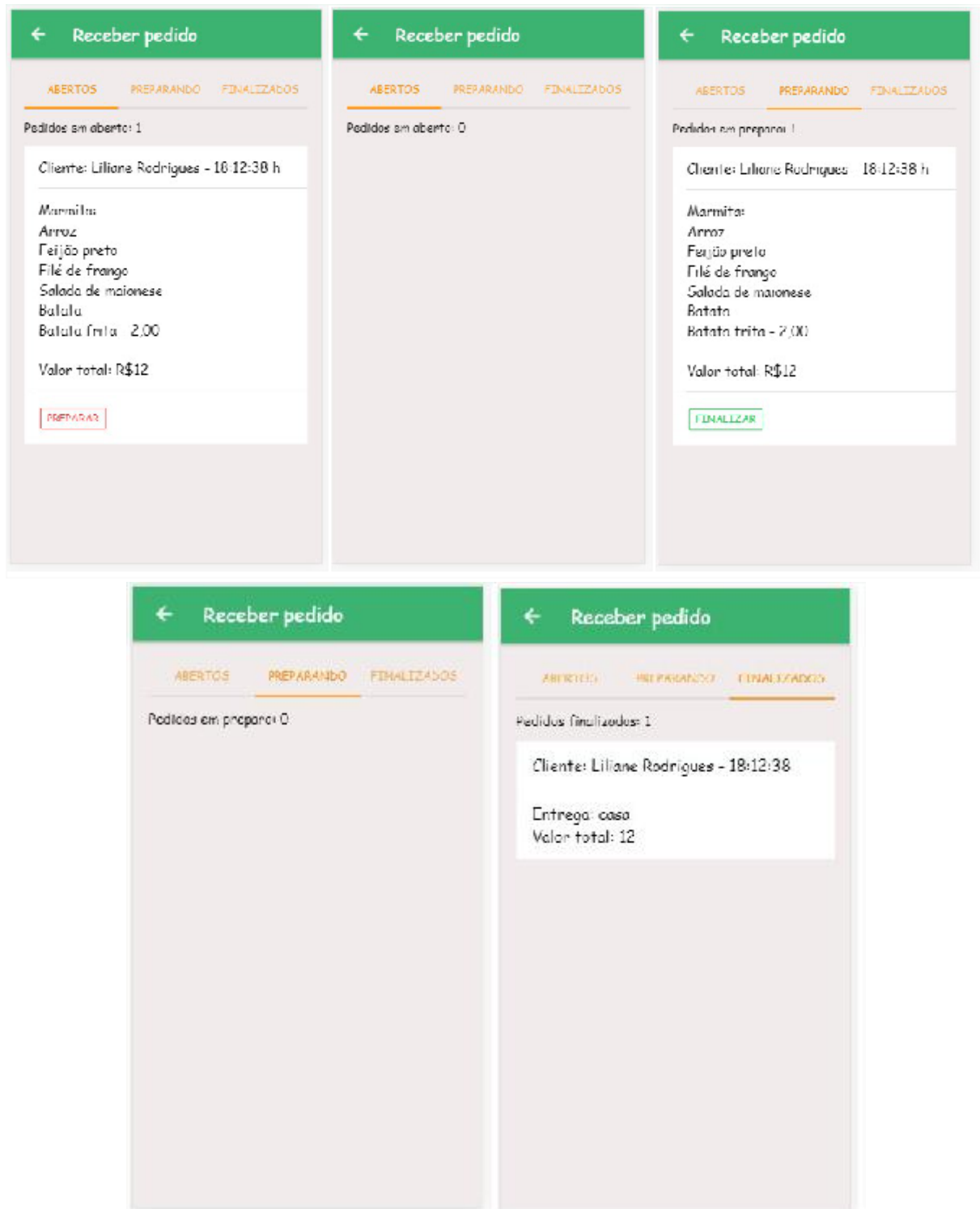


Fonte: da própria autora

O quinto e último teste é o de recebimento dos pedidos. Sempre que um pedido é realizado, o mesmo já é cadastrado com status aberto. Quando o restaurante for prepará-lo, deverá alterar seu status através das opções que são

exibidas, para que assim o pedido seja direcionado ao preparo e posteriormente a finalização. Os pedidos são exibidos por fila, obedecendo o horário em que foram realizados.

Figura 35: Teste do recebimento dos pedidos

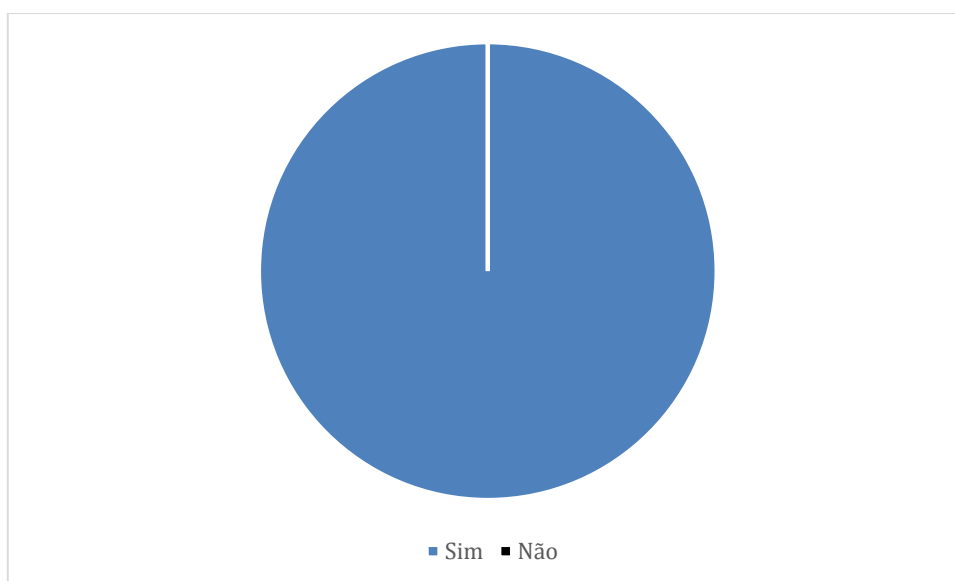


Fonte: da própria autora

Os resultados foram obtidos com base no questionário aplicado (apêndice I), onde as perguntas foram realizadas após o restaurante testar e validar a aplicação, a fim de medir o nível de satisfação deles com o *software*. Com isso, foram formuladas seis questões, em que todas elas possuem como alternativas de resposta as opções Sim ou Não. A aplicação foi testada pelos donos e funcionários do restaurante. Os resultados coletados pelo questionário serão apresentados com o apoio de gráficos, conforme mostra a seguir.

A primeira questão abordada foi em relação a simplicidade e facilidade de utilização do aplicativo. O Gráfico 01 exibe o resultado, onde 100% dos entrevistados responderam Sim, concluindo-se então que o aplicativo é uma ferramenta simples e de fácil utilização para eles.

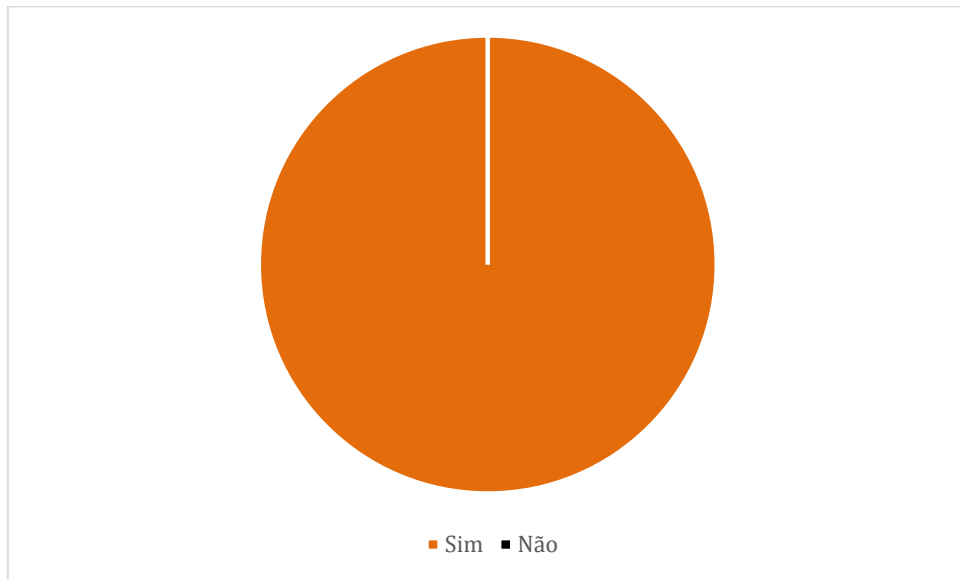
Gráfico 01: Utilização do aplicativo



Fonte: da própria autora

A segunda questão foi sobre as funcionalidades contempladas na aplicação. Para isso, 100% dos entrevistados responderam Sim, conforme apresentado no Gráfico 02. Com isso, foi validado que o aplicativo contemplou todas as funcionalidades que foram traçadas no início do projeto.

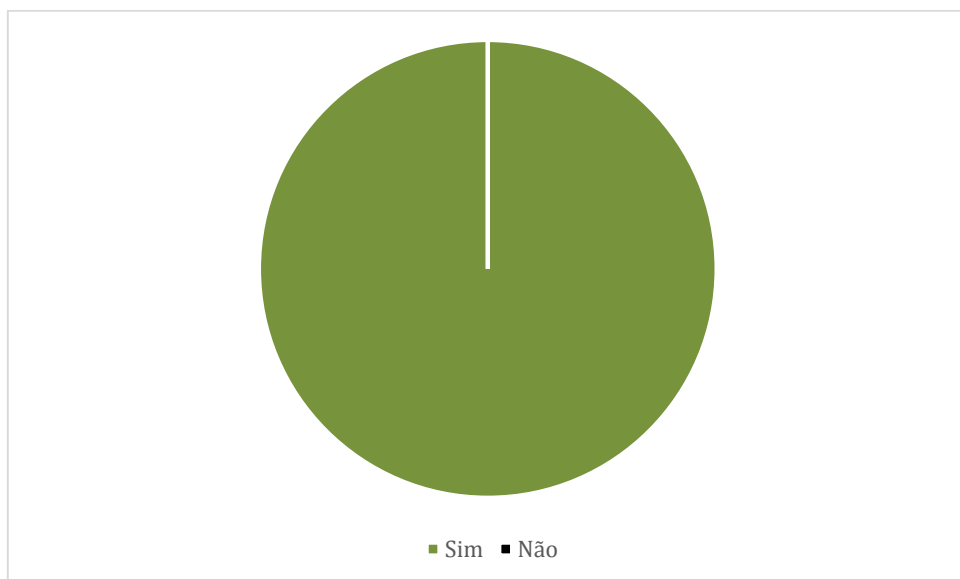
Gráfico 02: Funcionalidades do aplicativo



Fonte: da própria autora

A terceira questão buscou saber a satisfação dos usuários a respeito da eficiência para a organização e controle dos pedidos de marmitas recebidos. O Gráfico 03 exibe o resultado, onde 100% das respostas foram Sim, concluindo-se que o aplicativo é útil para a organização e controle dos pedidos.

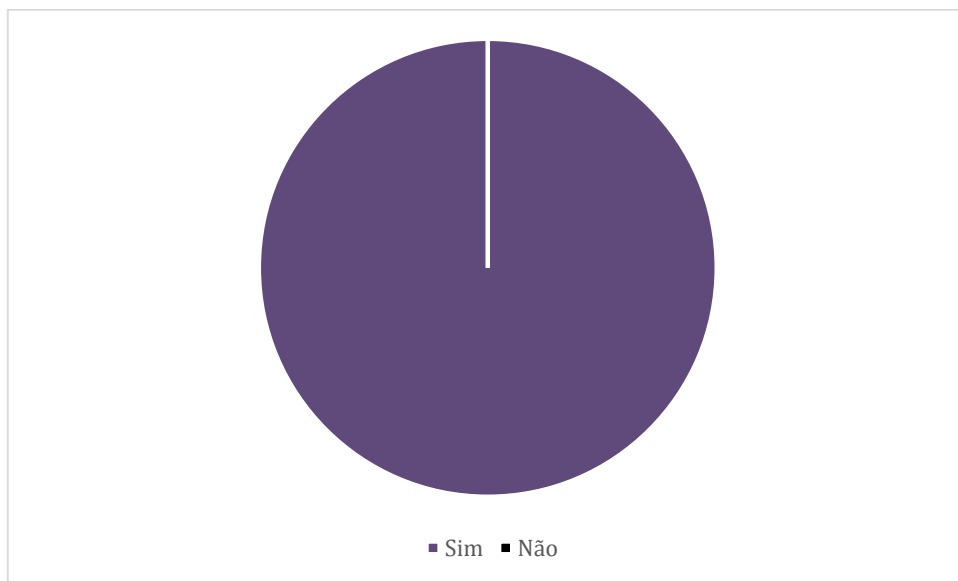
Gráfico 03: Organização e controle dos pedidos



Fonte: da própria autora

A quarta questão foi sobre a competência do aplicativo para atender a rotina do restaurante. Para isso, 100% dos entrevistados responderam Sim, conforme apresentado no Gráfico 04. Com isso, foi validado que o aplicativo é uma ferramenta eficiente para atender a rotina do estabelecimento.

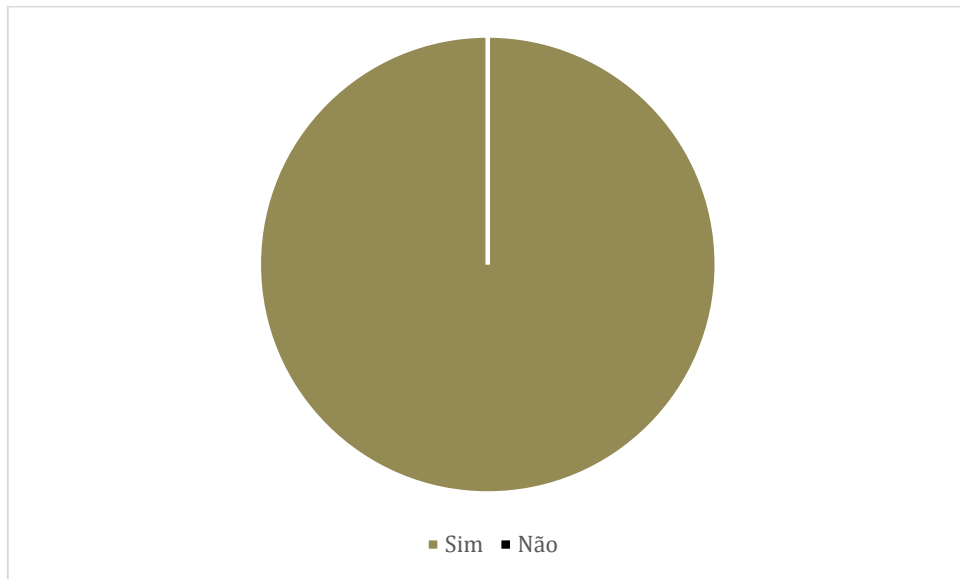
Gráfico 04: Eficiência da rotina



Fonte: da própria autora

A quinta questão buscou saber a opinião da aplicação como um todo. O Gráfico 05 exibe o resultado, onde 100% das respostas foram Sim, concluindo-se que o aplicativo é uma ferramenta que promete aperfeiçoar o processo de montagem e pedido de marmidas no restaurante.

Gráfico 05: Montagem e pedido de marmita



Fonte: da própria autora

Com base nos resultados coletados pelo questionário, o aplicativo tem grande potencial para auxiliar o restaurante nos problemas enfrentados. As funcionalidades oferecidas por ele possuem capacidade para que o restaurante consiga fazer o controle desejado, além de possuir suporte para que consiga crescer e ser desenvolvido novos recursos desejados, além de aperfeiçoar os existentes.

CONCLUSÃO

Com o desenvolvimento do presente trabalho foi possível apresentar a capacidade com que uma aplicação móvel aperfeiçoará as rotinas de um restaurante e seus clientes no processo de pedidos de marmitas. Através deste aplicativo, os clientes podem realizar pedidos de marmita de um jeito fácil, e o restaurante consegue fazer todo o controle dos pedidos e demais atividades que envolvem este processo.

Para alcançar o objetivo geral estipulado, foram seguidos alguns objetivos específicos, como: Identificar os problemas relacionados à montagem e pedido de marmitas que acarretam uma ineficiência na oferta desse serviço; avaliar o processo de montagem e pedido de marmita, a fim de analisar as melhores formas para que o pedido seja recebido corretamente; estudar as melhores práticas para desenvolvimento do aplicativo, de modo que o mesmo seja simples, interativo e de fácil utilização pelos clientes e restaurante; analisar aplicações semelhantes no mercado, a fim de aprimorar as funcionalidades a serem desenvolvidas no aplicativo proposto.

Para o desenvolvimento deste trabalho, foi preciso a realização de estudos, análises e pesquisas, a fim de fazer a coleta de dados e o levantamento de requisitos da aplicação. Com a evolução do projeto, também foram realizados testes com o objetivo de avaliar os avanços obtidos e identificar as falhas e dificuldades.

O projeto utilizou a pesquisa de opinião para coletar, junto aos clientes do restaurante, tudo o que era necessário para a construção da aplicação. Também foi realizada pesquisa bibliográfica para auxiliar na elaboração da monografia.

Os resultados obtidos neste projeto foram através da aplicação de um questionário, onde através de perguntas com respostas de sim ou não, foram analisadas o nível de satisfação do restaurante com o aplicativo. Foi utilizada abordagem quali-quantitativa e o método indutivo.

Observando esse contexto e visando atingir os objetivos propostos, foram estabelecidas as seguintes hipóteses:

H0: A utilização de um aplicativo móvel não aperfeiçoaria o processo de montagem e pedido de marmitas pois o restaurante consegue ter o controle dos pedidos recebidos, sem haver necessidade de uma ferramenta para esta função;

Esta hipótese foi invalidada. Com o desenvolvimento desta pesquisa e pelo fato de o restaurante buscar uma nova solução para os problemas enfrentados, é visível que o mesmo não possui o controle ideal dos pedidos que são realizados pelos clientes. Foi identificado, tanto pelo restaurante quanto pelas observações realizadas, que o estabelecimento passa por dificuldades e precisa de um meio de solução para os seus problemas. O fato de uma aplicação auxiliar neste processo é de grande benefício, pois atenderá as funções já executadas e proporcionará as melhorias esperadas pelo restaurante.

H1: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas, mas não seria aceita na rotina do restaurante e seus clientes;

Esta hipótese foi invalidada. Como o presente projeto utilizou dados de observações e pesquisas de opinião, foi possível validar com os funcionários e clientes do restaurante a viabilidade da utilização do aplicativo desenvolvido. Pelos testes realizados no final do projeto e os resultados do questionário, também foi possível observar essa aceitação. Conforme já abordado na H0, a aplicação móvel poderá sim aperfeiçoar todo o processo. Pelo fato de ambas as partes esperarem melhorias em suas rotinas e acreditarem nos benefícios desta ferramenta, o aplicativo móvel será aceito por elas.

H2: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas pois auxiliaria na organização e controle dos pedidos de marmitas recebidos pelos clientes;

Esta hipótese foi validada. Através das análises, testes realizados ao decorrer do desenvolvimento e os resultados, foi possível identificar os meios que a aplicação oferece para realizar todo o controle e organização dos pedidos. O comprometimento do restaurante em utilizar a aplicação também foi um ponto crucial para esta validação.

H3: A utilização de um aplicativo móvel aperfeiçoaria o processo de montagem e pedido de marmitas pois teria uma boa aceitação pelo restaurante e seus clientes, melhorando a oferta de marmitas.

Não foi possível validar esta hipótese. A garantia de melhorar a oferta de marmitas não pode ser validada pelo fato de a aplicação não ser disponibilizada a tempo suficiente de ser utilizada pelos clientes e restaurante e serem colhidos os dados necessários para essa comprovação. Por este motivo não se pode afirmar que ela irá melhorar a oferta, embora os resultados colhidos pelo questionário tenha sido positivos.

Devido ao tempo não ter sido suficiente, infelizmente não foi possível a finalização de tudo o que era desejado no aplicativo. Entretanto, o mesmo conseguiu atingir os objetivos traçados e pretende-se continuar trabalhando nas melhorias e futuras implementações adicionais para ele.

A conclusão deste trabalho foi bastante significativa. Desenvolver uma aplicação que atenda às necessidades de pessoas de um ambiente e contribua positivamente em seus processos é de grande satisfação. É impossível calcular a relevância desta pesquisa, mas pode-se afirmar que ela enriqueceu a desenvolvedora com todo o conhecimento e ganhos profissionais e acadêmicos adquiridos neste tempo.

REFERÊNCIAS

ALVIM, Paulo. *Tirando o Máximo do Java EE 6 Open Source com jCompany Developer Suite*. 3ed. Belo Horizonte: Powerlogic Publishing, 2010.

BRITO, Ricardo W. *Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa*. Fortaleza, 2010. Disponível em: <<http://www.infobrasil.inf.br/>>. Acesso em: 18 mai 2018.

CALDEIRA, Carlos Pampulim. *Introdução ao HTML* (HyperText Markup Language). Évora, 2015. Disponível em: <<https://dspace.uevora.pt/>>. Acesso em: 12 mai 2018.

DIMES, Troy. *JavaScript: Um guia par aprender a linguagem de programação JavaScript*. 1ed. São Paulo: Babelcube, 2015.

ELMASRI, Ramez; NAVATHE, Shamkant B. *Sistemas de Banco de Dados*. 4ed. São Paulo: Pearson Addison-Wesley, 2005.

GRIFFITH, Chris. *Mobile app development with ionic*. 1ed. Sebastopol: O'Reilly Media, 2017.

GUEDES, Thiago. *Crie aplicações com Angular: O novo framework do Google*. 1ed. São Paulo: Casa do Código, 2017.

KUMAR, K.N.Manoj et al. *Implementing smart home using Firebase*. 2016. Disponível em: <<http://euroasiapub.org/wp-content/uploads/2016/11/16EASOct-4186-1.pdf/>>. Acesso em: 01 set 2018.

LANE, Kin. *Overview Of The Backend as a Service (BaaS) Space Prepared*. 2013. Disponível em: <<http://www.integrove.com/>>. Acesso em: 21 mai 2018.

LOPES, Sérgio. *Aplicações mobile híbridas com Cordova e PhoneGap*. 1ed. São Paulo: Casa do Código, 2016.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues; PONTES, Jonas César de Souza. *NoSQL no desenvolvimento de aplicações Web colaborativas*. 2011. Disponível em: <<http://www.addlabs.uff.br/>>. Acesso em: 18 mai 2018.

NAYAK, Ameya. PORIYA, Anil. POOJARY, Dikshay. *Type of NOSQL Databases and its Comparison with Relational Databases*. Nova Iorque, 2013. Disponível em: <<https://research.ijais.org/volume5/number4/ijais12-450888.pdf/>>. Acesso em: 30 ago 2018.

PAGOTTO, Tiago et al. *Scrum Solo: Processo de software para desenvolvimento individual*. São Paulo, 2016. Disponível em: <<https://ieeexplore.ieee.org/iel7/7511893/7521364/07521555.pdf/>>. Acesso em: 25 ago 2018.

PAULA FILHO, Wilson de Pádua. *Engenharia de Software: Fundamentos, Métodos e Padrões*. 1ed. Rio de Janeiro: LTC, 2001.

PRESCOTT, Preston. *HTML5*. 1ed. São Paulo: Babelcube INC, 2015.

PRESSMANN, Roger S. *Engenharia de software*. 7ed. São Paulo: AMGH Editora, 2011.

QUIERELLI, Davi Antonio. *Criando Sites Com Html Css Php*. 1ed. Leme: Clube dos Autores, 2012.

RAMAKRISHNAN, Raghu; JOHANNES, Gehrke. *Sistemas de Gerenciamento de Banco de Dados*. 3ed. São Paulo: McGraw-Hill, 2008.

SEBESTA, Robert W. *Conceitos de linguagens de programação*. 11ed. Porto Alegre: Bookman, 2018.

SILBERSCHATZ, Abraham; KORTH, Henry; SUDARSHAN, S. *Sistema de banco de dados*. 5ed. Rio de Janeiro: Elsevier, 2006

SILVA, Daisy Eliana dos Santos; SOUZA, Ingredy Thaís; CAMARGO, Talita. *Metodologias Ágeis para o desenvolvimento de software: Aplicação e o uso da*

metodologia Scrum em contraste ao modelo tradicional de gerenciamento de projetos. Guarulhos, 2013. Disponível em: <<http://revistas.ung.br/index.php/computacaoaplicada/article/>>. Acesso em: 29 ago 2018.

SOARES, Michel dos Santos. *Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.* Conselheiro Lafaiete, 2004. Disponível em: <<http://www.dcc.ufla.br/infocomp/index.php/INFOCOMP/article/view/68/>>. Acesso em: 30 Ago 2018.

SOMMERVILLE, Ian. *Engenharia de software.* 9ed. São Paulo: Pearson Prentice Hall, 2011.

WOLFF, Ivo Gabe. *TypeScript Blueprints.* 1ed. Birmingham: Packt Publishing, 2016.

Disponível em: <<http://www.sqlite.org/about.htm>>. Acesso em: 13 mai 2018.

Disponível em: <<http://www.typescriptlang.org/index.html>>. Acesso em: 6 mai 2018.

Disponível em: <<https://angular.io/>>. Acesso em: 20 mai 2018.

Disponível em> <<https://code.visualstudio.com/>>. Acesso em: 25 mai 2018.

Disponível em: <<https://firebase.google.com/>>. Acesso em: 20 mai 2018.

Disponível em: <<https://www.ecma-international.org>>. Acesso em: 6 mai 2018.

Disponível em: <<https://www.ionicframework.com>>. Acesso em: 10 mai 2018.

Disponível em: <<https://www.w3.org/TR/CSS/>>. Acesso em: 5 mai 2018.

Disponível em: <<https://www.w3.org/TR/html5/>>. Acesso em: 5 mai 2018.

APÊNDICE I

Questionário

- 1 - Você classifica o aplicativo como sendo simples e de fácil utilização?
- 2 - O aplicativo desenvolvido contempla todas as funcionalidades prometidas?
- 3 - O aplicativo é uma ferramenta útil para organização e controle dos pedidos de marmitas recebidos?
- 4 - O aplicativo consegue atender a rotina do restaurante com eficiência?
- 5 - O aplicativo promete aperfeiçoar o processo de montagem e pedido de marmita?