UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE COMPUTAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL FELIPE VARGAS FERREIRA

GUILHERME DUARTE FRANCO

REBECA BATISTA MEDEIROS DA FONSECA

RANGO: facilitando o almoço universitário

RIO DE JANEIRO

2021

GABRIEL FELIPE VARGAS FERREIRA

GUILHERME DUARTE FRANCO

REBECA BATISTA MEDEIROS DA FONSECA

RANGO: facilitando o almoço universitário

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profa. Silvana Rossetto

RIO DE JANEIRO

2021

Ferreira, Gabriel Felipe Vargas
F383r RANGO: facilitando o almoço universitário / Gabriel Felipe Vargas Ferreira, Guilherme Duarte Franco, Rebeca Batista Medeiros da Fonseca. – Rio de Janeiro, 2021.

135 f.

Orientadora: Silvana Rossetto.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2021.

1. Aplicativos móveis. 2. Quentinhas. 3. Ambiente universitário. I. Franco, Guilherme Duarte. II. Fonseca, Rebeca Batista Medeiros da. III. Rossetto, Silvana (Orient.). IV. Universidade Federal do Rio de Janeiro, Instituto de Computação. V. Título.

GABRIEL FELIPE VARGAS FERREIRA

GUILHERME DUARTE FRANCO

REBECA BATISTA MEDEIROS DA FONSECA

RANGO: facilitando o almoço universitário

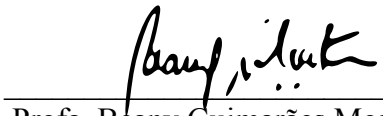
Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 15 de Outubro de 2021.

BANCA EXAMINADORA:

Profa. Silvana Rossetto, D.Sc.
(Instituto de Computação - UFRJ)

Profa. Maria Luiza Machado Campos, Ph.D.
(Instituto de Computação - UFRJ)



Profa. Beany Guimarães Monteiro, D.Sc.
(Escola de Belas Artes - UFRJ)

Por Gabriel:

Dedico este trabalho à Melissa, por ter me apoiado, motivado e ter sido minha companheira em momentos importantes da minha graduação e deste trabalho e também aos meus pais, que me deram tudo que eu precisei para chegar até este momento.

Por Guilherme:

Dedico este trabalho aos meus pais, que me deram oportunidade e incentivo, à minha companheira Ester pelo apoio contínuo e ao meu amigo Murillo, que sempre acreditou em mim.

Por Rebeca:

Dedico este trabalho a Deus que me deu sabedoria para chegar até aqui, à minha família que me deu suporte nessa parte importante da minha vida e ao meu companheiro João Paulo que sempre me motivou a continuar.

AGRADECIMENTOS

Gostaríamos de agradecer à UFRJ e aos professores e professoras do DCC pela formação de qualidade; em especial à professora Silvana Rossetto que nos acolheu e de forma atenciosa mostrou o caminho para a realização deste trabalho; também a Fernanda Arnaut, Júlia Lopes e Luísa Forain, alunas do curso Comunicação Visual Design, que conceberam a ideia e o protótipo original do aplicativo; e por fim aos alunos membros do grupo de extensão Devmob, com os quais compartilhamos conhecimento e que contribuíram com este trabalho nas fases de pesquisa e testes nos oferecendo críticas e sugestões.

RESUMO

No ambiente universitário da UFRJ, existem diversas opções de refeições na hora do almoço, uma delas é a compra de quentinhas, que além de oferecer várias opções pela quantidade de vendedores espalhados pelo campus, é uma forma barata e rápida de almoçar. Existem alguns aplicativos móveis de venda de refeições, mas devido a escala deles, não suprem as necessidades dos vendedores de quentinha, que possuem uma operação mais caseira e local. Há então uma demanda por alternativas que atendam às necessidades dos clientes e vendedores de quentinhas. Este trabalho apresenta a proposta de uma aplicação, chamada Rango, que visa suprir esta carência, facilitando a comunicação entre clientes e vendedores e o comércio de quentinhas, dentro e fora do contexto do ambiente universitário da UFRJ. Para entender o que uma aplicação deste tipo precisaria ter para atender as necessidades de clientes e vendedores, foram realizadas pesquisas de demanda que ajudaram a definir o escopo inicial da proposta. Foram desenvolvidos dois aplicativos que interagem entre si: o do cliente, que apresenta vendedores de quentinhas e seus respectivos cardápios e possibilita a realização de reservas destas quentinhas; e o aplicativo do vendedor, que permite um fácil e intuitivo gerenciamento e criação de um cardápio de quentinhas para reservas. Ambos aplicativos permitem que clientes e vendedores conversem diretamente entre si. Por fim, os dois aplicativos foram experimentados e avaliados por usuários finais. Os relatos recebidos contribuíram para aprimorar a versão final dos aplicativos e propor extensões futuras.

Palavras-chave: aplicativos móveis; quentinhas; ambiente universitário.

ABSTRACT

On the UFRJ campus, there are several meal options at lunchtime, one of which is through the purchase of packaged meals, that besides offering several options due to the number of vendors spread across the campus, is a cheap and quick way to have lunch. There are some mobile applications for food sale, but due to their scale, they do not meet the needs of packaged meals vendors, who have a more homemade and local operation. There is then a demand for alternatives that meet the needs of customers and vendors. This work presents the proposal of a mobile application, called “Rango”, that aims to overcome this shortfall, facilitating communication between customers and vendors and the commerce of packaged meals, inside and outside the context of the university. To understand what an app of this type would need to meet the needs of customers and vendors, demand surveys were conducted, that helped to define the initial scope of the proposal. Two apps that interact with each other were developed: that of the customer, which presents vendors of packaged meals and their respective menus and allows making reservations of these meals; and the vendor's app, that enables easy and intuitive management and creation of a menu for reservations. Both apps allow customers and vendors to talk directly to each other. Finally, the two apps were tried and evaluated by end users. The reports received contributed to improve the final version of the apps and to propose future extensions.

Keywords: mobile apps; packaged meals; university space.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

CPU – *Central Process Unit*

HTTP – *Hypertext Transfer Protocol*

MVP – *Minimum Viable Product*

NoSQL – *Non Structured Query Language*

PWA – *Progressive Web App*

RAM – *Random Access Memory*

SMS – *Short Message Service*

URL – *Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

No campus Cidade Universitária da UFRJ existem diferentes opções para alimentação, como restaurantes, lanchonetes, restaurantes universitários e quentinhas, das quais alunos, professores e técnicos se beneficiam. Cada opção possui seus prós e contras, atendendo às necessidades e desejos de cada cliente. Os critérios destes costumam ser: preço, sabor, localização, quantidade de alimento, variedade, tempo de espera e restrições alimentares.

O serviço de quentinhas no campus é protagonizado por trabalhadores autônomos que vendem os alimentos, previamente preparados e embalados, em quiosques ou utilizando a estrutura de automóveis posicionados em locais habituais. Alguns permitem a realização de reservas e geralmente interagem com seus clientes por meio de aplicativos de mensagens instantâneas. A divulgação costuma ser feita por boca a boca, o que dificulta a transmissão de informações e a descoberta pelos clientes de novos vendedores. Contudo, esse serviço chama atenção dos estudantes e servidores pelo seu preço, geralmente mais baixo que os preços praticados nos restaurantes, rapidez na compra e praticidade do cliente poder consumir a quentinha no ambiente que desejar.

Segundo a 31ª edição da Pesquisa Anual do Uso de TI, realizada em 2020 pelo Centro de Tecnologia de Informação Aplicada da FGV EASP, há 234 milhões de *smartphones* em uso no Brasil, o que significa mais de um *smartphone* por habitante (MEIRELLES, 2020, p. 64). Os aplicativos móveis disponibilizados atualmente auxiliam e potencializam uma grande quantidade de atividades relacionadas à, entre outras, comunicação, transporte e alimentação.

Os aplicativos iFood[1], Uber Eats[2] e Rappi[3], além daqueles próprios de cada restaurante, detêm grande porção dos usuários do mercado de *delivery*, ou entrega, de comida. Apesar disso, estes aplicativos não atendem com qualidade as pessoas que buscam se alimentar no campus. O preço, tempo de entrega e eventual imprevisibilidade da entrega, devida à atrasos ou problemas não previstos, através de aplicativos como esses inviabilizam seu uso no dia a dia do estudante ou servidor. Estes costumam ter um curto intervalo para almoço e janta, fazendo com que opções mais rápidas como quentinhas ou mais baratas como o restaurante universitário se destaquem entre os clientes, essa última sendo muito utilizada pelos alunos. Ademais, o uso dos aplicativos de *delivery* já firmados no mercado não é

[1] iFood. Disponível em: https://www.ifood.com.br. Acesso em: 25 ago. 2020.
[2] Uber Eats. Disponível em: https://www.ubereats.com/br. Acesso em: 25 ago. 2020.
[3] Rappi. Disponível em: https://www.rappi.com.br. Acesso em: 25 ago. 2020.

favorável para os vendedores de quentinhas, por envolverem um entregador externo e empregarem estratégias de negócios agressivas (G1, 2020).

1.1 OBJETIVOS DO TRABALHO

A nossa proposta é o desenvolvimento de um aplicativo para celular que tem como objetivo facilitar e centralizar a procura por quentinhas e a comunicação entre compradores e vendedores, chamado Rango. O aplicativo possui duas versões: uma voltada para o comprador, onde este poderá ver uma lista personalizada de vendedores e pratos à venda, além de permitir a busca, reserva de quentinhas e possuir um mapa exibindo a localização de vendedores; a outra voltada para o vendedor, concedendo-o a facilidade de criar um cardápio com imagens, nomes e preços, gerenciar reservas, divulgar sua localização e enviar notificações para seus clientes.

1.2 ORGANIZAÇÃO DO TEXTO

No Capítulo 2, detalhamos os aplicativos de *delivery* de comida utilizados comumente e as razões pelas quais estes não atendem o serviço de quentinhas. No Capítulo 3, apresentamos os resultados obtidos na pesquisa sobre o serviço de quentinhas na Cidade Universitária e os requisitos definidos. Logo, no Capítulo 4, descrevemos o trabalho de idealização, projeto e prototipagem inicial do aplicativo. Além disso, relatamos o desenvolvimento do aplicativo e dos sistemas auxiliares, detalhando e justificando as tecnologias usadas na implementação. No Capítulo 5, apresentamos os resultados alcançados e os testes de usabilidade realizados. Finalmente, no Capítulo 6, concluímos com a visão geral da trajetória do projeto e trabalhos futuros. A descrição do gerenciamento e metodologia de realização deste trabalho está presente no Apêndice J.

2 SERVIÇO DE QUENTINHAS NO CAMPUS E APLICATIVOS DE REFEIÇÕES

Neste capítulo, detalhamos o serviço de quentinhas no campus Cidade Universitária da UFRJ e em seguida discutimos os aplicativos de refeições comumente usados pelo público em geral. Expomos os motivos desses aplicativos não serem utilizados no auxílio desse serviço no campus, tanto por parte dos clientes quanto por parte dos vendedores. Finalmente, explicamos como o projeto Rango contribuiria para melhorar esse serviço.

2.1 SERVIÇO NA UFRJ

Com o objetivo de contextualização, nesta seção será detalhado o funcionamento do serviço de quentinhas no campus. Existem locais que vendem refeições preparadas no momento, levadas pelos clientes em embalagens de quentinha, feitas de isopor ou alumínio, porém, focaremos nos vendedores que fazem a preparação e embalamento previamente, seja em casa ou em algum estabelecimento.

Primeiramente, um número limitado de quentinhas é produzido pelo vendedor, preferencialmente no início do dia. Essas refeições são transportadas em automóveis para o local de venda durante a manhã, onde uma estrutura geralmente é montada próxima ao veículo. Esta estrutura pode ser formada por uma mesa ou suporte, cartaz com o nome e arte do vendedor, recipientes organizadores para as quentinhas, entre outros itens. A localização escolhida costuma ser estratégica, próxima às entradas e saídas dos edifícios, onde há mais pessoas estudando ou trabalhando. Dependendo dos vendedores, os clientes podem reservar suas refeições previamente, no início do dia ou no dia anterior, ou comprar no momento. O horário de pico é no almoço. Os principais meios de contato entre clientes e vendedores são aplicativos de mensagens instantâneas, como o WhatsApp, e telefone. Grande parte da divulgação é feita por boca a boca ou por panfletos, que contêm o nome, arte e informações de contato. Já o pagamento costuma ser feito no momento da compra, com dinheiro, cartão ou até mesmo Pix. Além das quentinhas, podem ser oferecidas também bebidas, complementos, sobremesas e talheres descartáveis. Finalmente, os preços das refeições costumam ser acessíveis, geralmente a partir de R$ 10,00.

2.2 APLICATIVOS

Foram selecionados cinco aplicativos de refeições conhecidos no Brasil: iFood, Uber Eats, Rappi, Delivery App e GoomerGo. Os três primeiros possuem um funcionamento semelhante: os clientes podem ver listas de restaurantes, próximos de sua localização, e ao selecionar um são direcionados a um cardápio. Depois, escolhem as refeições, complementos e bebidas, adicionando em uma "sacola". Por fim, seguem para as configurações de entrega e pagamento, finalizando o pedido. Os últimos dois, Delivery App e GoomerGo, não possuem uma lista de restaurantes, exibindo diretamente o cardápio de um restaurante.

Usando o iFood, Uber Eats ou Rappi, os restaurantes podem também habilitar a opção de retirada, na qual o próprio cliente vai até o local para buscar seu pedido. Além disso, é possível definir pagamento *online* e/ou presencial, seja entrega ou retirada. O serviço de entregas pode ser coordenado pelo próprio restaurante ou por entregadores parceiros do aplicativo, afetando a taxa cobrada pela empresa. Apesar de serem muito usados por clientes e restaurantes, estes três aplicativos possivelmente são caros demais para pequenos empreendedores. Além de taxas por pedido, há aquelas de adesão, mensalidades e adicionais por pagamentos *online*.

Buscando um público alvo diferente de restaurantes, a empresa Neemo[4] oferece o *Delivery App*. Os diferenciais são que cada vendedor recebe um site e aplicativo personalizados, e as entregas são gerenciadas pelo próprio restaurante, não havendo entregadores parceiros. Um ponto negativo para os clientes é que não há uma lista central de vendedores próximos, portanto cada restaurante possui seu cardápio em um site e aplicativo específicos. É necessário então que o usuário conheça previamente que tal estabelecimento usa o Delivery App. Com essas mudanças, o custo para o restaurante é mais baixo, ainda possuindo opções de pagamento *online* e *offline*, e de retirada e entrega (caso tenha uma equipe de entregadores).

A empresa Goomer[5] oferece uma solução parecida com o Delivery App, o *GoomerGo*. Além das funcionalidades do último, este oferece um plano gratuito de pedidos via WhatsApp. O cliente escolhe sua refeição pelo cardápio do restaurante e ao finalizar o pedido é redirecionado para o aplicativo de mensagens. Então, basta enviar a mensagem criada automaticamente para confirmar o pedido. De novo, cada restaurante possui um site personalizado com seu cardápio, porém no caso do GoomerGo o aplicativo é um *progressive*

[4] Neemo. Disponível em: https://www.neemo.com.br . Acesso em: 18 set. 2021.
[5] Goomer. Disponível em: https://goomer.com.br. Acesso em: 18 set. 2021.

web app (PWA, aplicativo *web* progressivo, em tradução livre). Esse tipo de aplicativo aparece na lista de aplicativos do *smartphone*, porém não requer instalação e funciona como um atalho para o site, que deve ser responsivo e otimizado para telas de celular. Isto é importante para o cliente, pois ele pode ter diversos *web apps* de restaurantes que usam o GoomerGo sem se preocupar com espaço de armazenamento do seu dispositivo.

Analisando todas essas soluções, concluímos o seguinte: o iFood, Uber Eats e Rappi são direcionados para restaurantes melhor consolidados no mercado, que possuem sua equipe de entrega ou desejam usar a equipe da empresa para alcançar mais clientes. Já o Delivery App e GoomerGo focam na venda em áreas menores, atingindo um menor número de clientes porém oferecendo taxas mais acessíveis. No Quadro 1, são apresentadas comparações entre os aplicativos discutidos.

Quadro 1 - Relação de aplicativos de refeições

EMPRESA	PLATAFORMAS	TAXAS
iFood	Site e aplicativo	"Plano Básico" (entregas feitas pelo restaurante): 12% sobre o valor do pedido; "Plano Entrega" (entregas feitas por entregadores parceiros): 23% sobre o valor do pedido.[6]
Uber Eats	Site e aplicativo	Entregas feitas pelo restaurante: 17% sobre o valor do pedido; Entregas feitas por parceiros ou retirados pelo cliente: 30% sobre o valor do pedido.[7]
Rappi	Site e aplicativo	"Plano *Marketplace*" (entregas feitas pelo restaurante): 10% sobre o valor do pedido; "Plano *Full-service*" (entregas feitas por entregadores parceiros): 23% sobre o valor do pedido.[8]
Delivery App	Site e aplicativo	Plano "*Start*": a partir de R$ 189,00 mensais.[9]
GoomerGo	Site e *web app*	Pedidos via WhatsApp: grátis;

[6] iFood para Restaurantes. Disponível em: https://parceiros.ifood.com.br/restaurante. Acesso em: 8 set. 2021.
[7] Uber Eats for Restaurants. Disponível em: https://www.ubereats.com/restaurant/pt-BR/signup. Acesso em: 8 set. 2021.
[8] Partners Rappi. Disponível em: https://partners.rappi.com/self-onboarding. Acesso em: 8 set. 2021.
[9] Planos | Delivery App. Disponível em: https://www.neemo.com.br/planos. Acesso em: 8 set. 2021.

		Plano "*Go Essencial*" (gestor de pedidos): - Até 100 pedidos por mês: R$ 29,90 mensais; - Até 200 pedidos por mês: R$ 59,90 mensais; - Até 400 pedidos por mês: R$ 89,90 mensais; - Até 800 pedidos por mês: R$ 149,90 mensais; - Número ilimitado de pedidos: R$ 269,90 mensais.[10]

Apesar de existirem essas soluções, foi constatado na pesquisa de demanda, detalhada no capítulo seguinte, que o principal meio de comunicação entre clientes e vendedores de quentinhas é o WhatsApp. Buscando entender esse fato, foi feita uma pesquisa com o "Maná Caseiro Quentinhas", localizado na Praça Mestre Monir Salomão - Jacarepaguá, Rio de Janeiro - RJ, no dia 3 de setembro de 2021. O formulário usado nesta pesquisa está disponível no Apêndice A. Este vendedor foi escolhido pois possui um modelo de negócio semelhante àquele dos vendedores situados no campus Cidade Universitária da UFRJ, atendendo a clientes que trabalham por perto, principalmente no horário de almoço.

Sobre o uso de aplicativos de *delivery,* como o iFood, o Maná Caseiro não os adota pois não possui estrutura de logística. Entregas são feitas, porém a pé por um funcionário contratado ou em casos mais raros, com carro. Contudo, a entrega com carro é realizada somente para locais próximos e por caminhos que não sejam contramão. Logo, usar um aplicativo do porte do iFood não é viável, pois seria impraticável entregar as quentinhas para clientes distantes. Além disso, esses vendedores possuem uma quantidade limitada de quentinhas para serem vendidas a cada dia. Então, um número maior de clientes possivelmente não seria um benefício, pois não haveria oferta suficiente de quentinhas para atender à demanda. Finalmente, as taxas cobradas pelos aplicativos comuns poderiam gerar despesas críticas para os vendedores, já que trabalham com margens pequenas de lucro, quando comparados a grandes restaurantes.

Na perspectiva dos clientes, esses aplicativos não se adequam ao serviço de quentinhas pois não oferecem um mapa informando a localização dos vendedores. Desta forma, dificultam a procura e descoberta de novas quentinhas. Ademais, os aplicativos iFood, Uber Eats e Rappi não possuem um canal de comunicação direta entre clientes e restaurantes, necessitando de um funcionário intermediário da empresa, o que prejudica a interatividade. Já

[10] Goomer | Preços. Disponível em: https://goomer.com.br/precos. Acesso em: 8 set. 2021.

os serviços Delivery App e GoomerGo não possuem uma interface que centralize todos os vendedores, impondo a necessidade de múltiplos aplicativos. Por fim, devido às taxas cobradas — no caso do GoomerGo, com o plano “Go Essencial” — possivelmente o preço cobrado pelas quentinhas seria aumentado.

Conhecendo o modelo de negócio do serviço de quentinhas no campus — que também existe em outros locais como o Centro do Rio de Janeiro, onde muitas pessoas trabalham — e estudando as opções de aplicativos de refeição disponíveis, foi observado que estes não contemplam as necessidades específicas dos clientes e vendedores de quentinhas. Nosso projeto, o aplicativo Rango, tem então o objetivo de ocupar esse espaço carecido.

3 LEVANTAMENTO DE REQUISITOS

Para entender o que uma aplicação que atende às necessidades dos compradores e também dos vendedores de quentinhas precisaria ter, foi realizado um levantamento de requisitos, que foi dividido em duas partes: uma pesquisa de demanda com um grupo de possíveis usuários dos aplicativos e, a partir dessa pesquisa, a definição dos requisitos necessários para o desenvolvimento do projeto. Os formulários usados para o aplicativo do cliente e do vendedor estão, respectivamente, disponíveis nos Apêndices B e D e as respostas nos Apêndices C e E.

3.1 PESQUISA DE DEMANDA

Durante o período de 21 de março de 2020 a 2 de junho de 2020, realizamos a pesquisa com pessoas que frequentam em seu dia a dia o campus Cidade Universitária da UFRJ, com o propósito de captar dados sobre o uso dos serviços de quentinhas espalhados nela. O recurso utilizado foi o Formulários Google (GOOGLE, 2021h). Dois formulários foram elaborados, um visando os clientes, consumidores de quentinhas, e outro direcionado a vendedores. Devido à pandemia do novo coronavírus e à suspensão de aulas na UFRJ, não foi possível realizar a pesquisa com os vendedores presencialmente, como pretendíamos inicialmente, já que esta ação necessitaria de um contato direto para afirmar a qualidade do questionário. A pesquisa também não pôde ser feita *online*, por não possuirmos os contatos dos vendedores.

Neste período, o formulário para clientes foi enviado para oito grupos distintos do Facebook e Telegram, compostos por estudantes e servidores associados à UFRJ. No total, 194 respostas foram recebidas.

A seguir, apresentamos uma análise das respostas recebidas na pesquisa de demanda, separada em quatro subseções: a primeira comentando sobre o perfil do participantes; a segunda sobre quem respondeu que não faz uso do serviço de quentinhas na Cidade Universitária, a fim de entender os motivos de não usarem; na terceira comentamos sobre quem respondeu que já compra quentinhas e os motivos que os levam a escolher determinado vendedor; e por fim, uma subseção que analisa se os participantes da pesquisa usariam o aplicativo proposto.

3.1.2 Perfil do participante

A partir das três primeiras perguntas, procuramos entender o perfil do participante na UFRJ. Observando a Figura 1, vemos que a grande maioria se identificou como estudante. Como exposto pela Figura 2, 95% do total estudam ou trabalham nos seguintes locais: Centro de Tecnologia (CT), Centro de Ciências Matemáticas e da Natureza (CCMN), Centro de Letras e Artes (Letras), Centro de Ciências da Saúde (CSS) e Reitoria. Apresentado na Figura 3, mais da metade respondeu que costuma comprar quentinhas, o que indica que os aplicativos do nosso projeto possuem um público alvo estabelecido de bom número. Ademais, aqueles que não costumam comprar também seriam favorecidos pelo projeto, como observamos na próxima seção.

Figura 1 - “Qual é o seu vínculo com a UFRJ?”

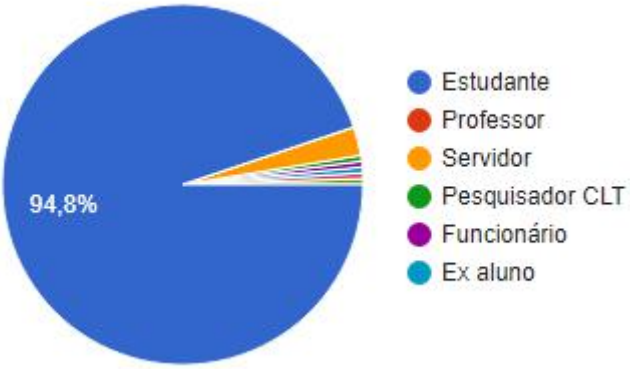


Figura 2 - “Onde trabalha/estuda na UFRJ?”

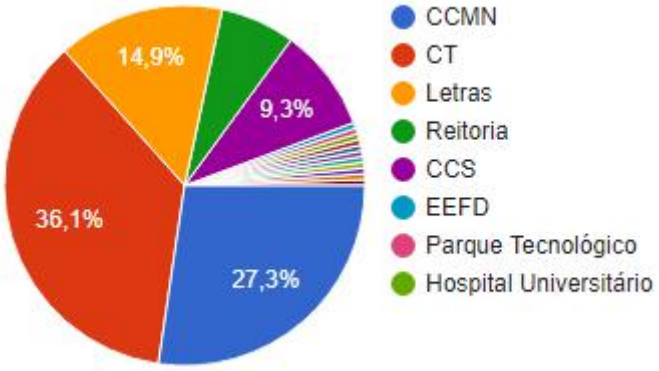
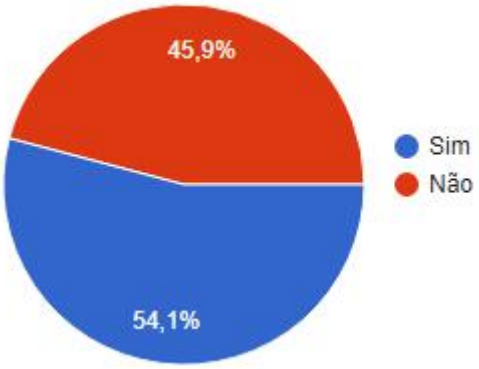


Figura 3 - “Costuma comprar quentinhas?”



3.1.3 Não costuma comprar quentinhas

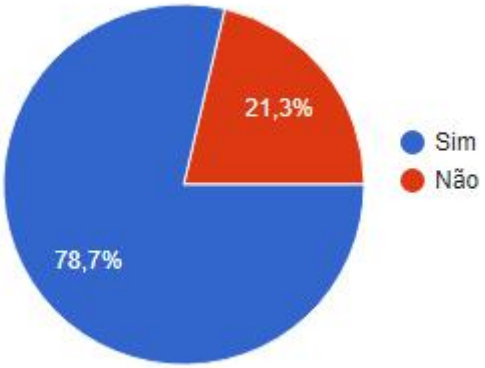
Para os participantes nesta segmentação foi perguntado qual o motivo para não terem o hábito de comprar quentinhas. Estas foram as justificativas dadas, em ordem de frequência:

- Alto custo;
- Alimentam-se em um restaurante universitário;
- Alimentam-se em casa ou levam quentinha de casa;
- Alimentam-se em restaurantes;
- Não sabem onde as quentinhas são vendidas;
- Medo de contaminação, comidas gordurosas ou com muito tempero;
- Falta de opções vegetarianas.

Dessas pessoas, onze possuem alguma restrição alimentar entre vegetarianismo, veganismo, intolerância à lactose e intolerância ao glúten. Foi observado que há demanda para quentinhas que atendam às restrições alimentares, porém pouca oferta. O Rango possivelmente seria uma boa ferramenta de comunicação entre cliente e vendedor para que essa demanda seja atendida.

Como podemos observar na Figura 4, 78,7% dos participantes deste grupo usariam um aplicativo que facilitaria a procura de quentinhas. Novamente, isto indica que é possível expandir a clientela desse serviço através do nosso projeto.

Figura 4 - “Você usaria um aplicativo que facilitaria a procura de quentinhas?”



3.1.4 Costuma comprar quentinhas

As perguntas a seguir buscaram elucidar o comportamento de clientes do serviço de quentinhas. Na Figura 5, observamos que 35,2% dos participantes compram ocasionalmente, 33,3% compram de duas a três vezes por semana, 25,7% compram uma vez por semana e 5,7% compram todos os dias. Além disso, foi relatado que algumas pessoas dividem a sua opção de alimentação entre o restaurante universitário e quentinhas, dependendo do cronograma, situação econômica ou querer.

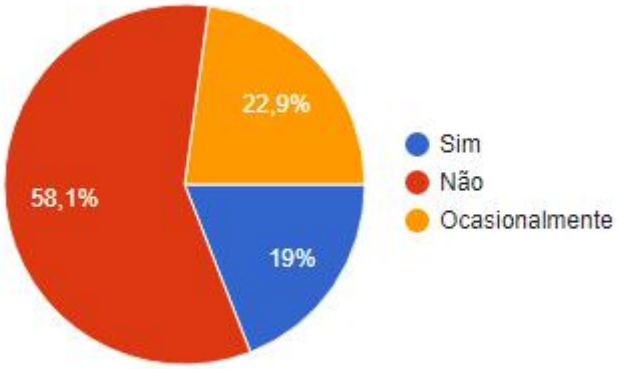
Figura 5 - “Com que frequência você compra quentinhas?”

Sobre a pergunta “Você costuma comprar sempre no mesmo local/vendedor?”, 81% disseram que sim. Apesar disso, 93,3% responderam que gostariam de conhecer mais opções de vendedores. Com um aplicativo seria possível enriquecer a fidelidade dos clientes, permitindo que definam seus vendedores favoritos mostrando-os com preferência e

notificando o usuário de novidades, assim como incentivar os clientes a diversificar suas opções.

Em seguida, na Figura 6 observamos que 58,1% dos participantes não reservam suas quentinhas, enquanto 19% reservam e 22,9% o fazem ocasionalmente. Dado isso, é importante que um cliente seja capaz de realizar reservas ou simplesmente consultar cardápios, fazendo o pedido somente no momento da compra. Conforme relatado pelos participantes, as reservas são feitas através de mensagens trocadas no aplicativo WhatsApp.

Figura 6 - “Você reserva suas quentinhas?”



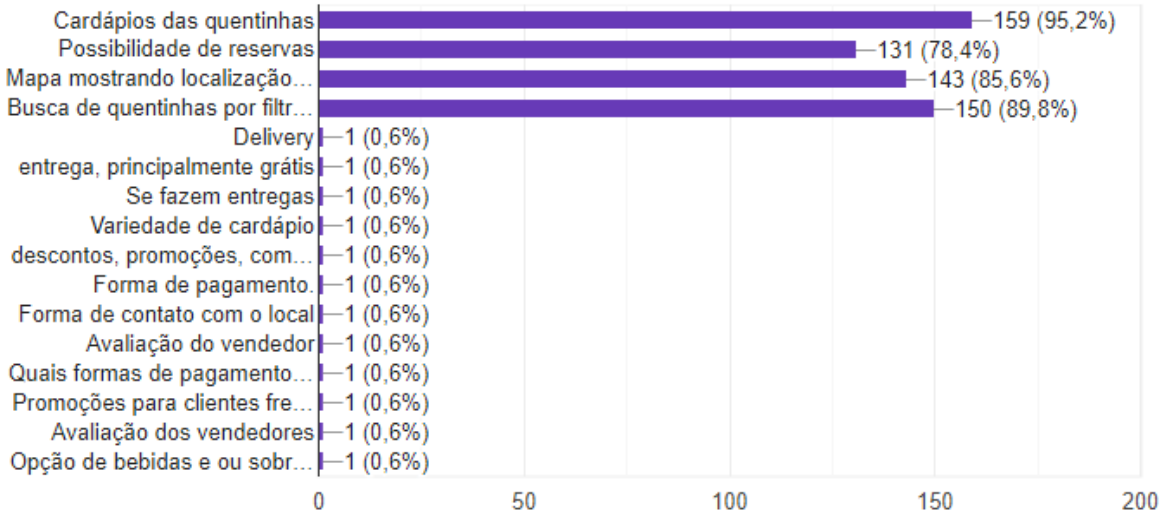
Por fim, 92,4% responderam que se interessam por um aplicativo que visa facilitar o uso do serviço de quentinhas.

3.1.5 Sobre o aplicativo

Esta seção abrange todos os participantes da pesquisa, que compram ou não quentinhas. Complementando o resultado anterior, 95,8% não conhecem um aplicativo voltado para quentinhas. Os que responderam que conhecem, mencionaram aplicativos como iFood, Uber Eats e similares. Porém, como apontamos no capítulo anterior, estes aplicativos não possuem funcionalidades focadas na realidade do serviço de quentinhas como este que se apresenta na Cidade Universitária.

Finalmente, através da pergunta “Quais serviços você espera que um aplicativo desse tipo deve fornecer?”, identificamos as funcionalidades mais importantes do aplicativo: cardápios de quentinhas, reservas, mapa com localização dos vendedores e busca de quentinhas por filtros. Essas e outras respostas podem ser observadas na Figura 7.

Figura 7 - “Quais serviços você espera que um aplicativo desse tipo deve fornecer?”



3.2 REQUISITOS

A partir da pesquisa realizada, debates prévios e estudo dos aplicativos de *delivery* mais conhecidos, concluímos que o nosso aplicativo precisaria realizar os seguintes requisitos:

A. Ser gratuito, permitindo o seu uso por qualquer pessoa, mediante cadastro;
B. Oferecer ao cliente opções de quentinhas de vendedores próximos geograficamente;
C. Disponibilizar ao cliente uma maneira de visualizar os vendedores próximos à sua localização;
D. Possibilitar a reserva de quentinhas, proporcionando conforto;
E. Fortalecer a fidelidade do cliente ao vendedor;
F. Promover uma comunicação direta entre cliente e vendedor;
G. Proporcionar a um vendedor de quentinhas o alcance de um número maior de clientes;
H. Disponibilizar ao vendedor um meio de gerenciar sua loja, através de uma interface simples e intuitiva, para editar as informações exibidas aos clientes;
I. Disponibilizar ao vendedor meios que auxiliem o acompanhamento de seu rendimento.

Estes requisitos são referenciados nos Quadros 3, 4 e 5 das funcionalidades do aplicativo.

4 PROJETO E IMPLEMENTAÇÃO

Neste capítulo será detalhado o projeto do aplicativo, começando pelo protótipo original, pela definição da arquitetura da solução e das funcionalidades que serão oferecidas; e indo até a implementação, onde serão apresentadas as tecnologias e ferramentas usadas e trechos de código de algumas das principais características do aplicativo.

4.1 PROJETO

Nesta seção serão apresentados o protótipo original e as diferenças dele para a versão implementada, a arquitetura lógica definida e por fim as funcionalidades escolhidas para compor a primeira versão do aplicativo.

4.1.1 Protótipo original

A ideia do aplicativo de quentinhas surgiu de um protótipo[11] feito pelas alunas Fernanda Arnaut, Júlia Lopes e Luísa Forain do curso de graduação em Comunicação Visual Design da UFRJ, na disciplina Projeto de Design de Interação. Mantivemos a maior parte das funcionalidades, mas também adicionamos novas ou alteramos a forma como algumas funcionam. Toda a identidade gráfica se manteve igual ou seguiu as ideias propostas originalmente. Alguns aspectos importantes que foram alterados são mostrados no Quadro 2:

Quadro 2 - Diferenças entre o protótipo e versão final

PROTÓTIPO ORIGINAL	VERSÃO DESENVOLVIDA
Apenas um aplicativo, onde o usuário informaria no cadastro o tipo de conta a ser criada.	Dois aplicativos distintos, um para cada tipo de usuário, com o objetivo de diminuir a complexidade e o tamanho individual de cada aplicativo.
Três abas na versão do aplicativo para cliente: principal, pesquisas e perfil.	Foi adicionada a aba de histórico de pedidos e a de busca foi movida para a tela de mapa com vendedores próximos.
Nas telas onde podem ser vistas informações sobre outros usuários, haviam	Foram adicionadas nas telas de detalhes de usuários informações extras, principalmente

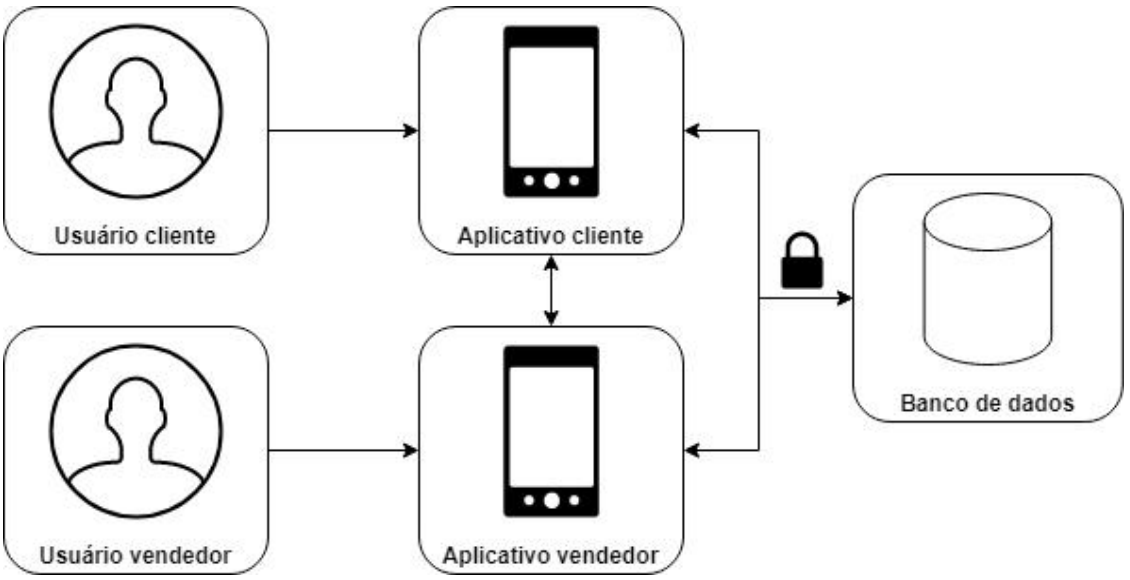
[11] Protótipo original. Disponível em: https://tinyurl.com/fkdzbkc2. Acesso em: 21 set. 2021.

poucas informações.	nas de vendedor, tais como descrição do vendedor, pagamentos aceitos e horários de funcionamento. Também foi adicionada a funcionalidade de abrir o contato do usuário, se existente, diretamente no WhatsApp.
Clientes recebiam quatro tipos diferentes de notificações: atualização de vendedores favoritos, confirmação de reservas, novas mensagens e promoções.	Foram retiradas as notificações de promoções e atualizações de vendedores favoritos, por não entrarem no escopo do projeto. Notificações de confirmação de reservas e novas mensagens foram mantidas.
O aplicativo do vendedor continha três abas: principal, gerenciamento de quentinhas e perfil.	Foi adicionada uma aba de conversas recentes com clientes, facilitando o acesso dessa área para os vendedores.
Havia poucas funcionalidades relacionadas a gerenciamento e funcionamento da loja.	Vendedores conseguem escolher horários de funcionamento, localização física da loja no mapa, visualizar histórico de pedidos assim como um relatório de vendas.
O gerenciamento de quentinhas não possuía um *layout* agradável e carecia de funcionalidades. Não existiam formas intuitivas e rápidas de gerenciamento, criação e edição de pratos.	Toda a aba de gerenciamento de quentinhas foi reformulada, de forma a apresentar uma interface mais convidativa e intuitiva para o vendedor. Nela é mostrado o cardápio diário — contendo imagem, nome e preço dos pratos — em uma lista horizontal, além de uma lista vertical com todos os pratos, incluindo botões de acesso rápido para adição ou remoção do cardápio do dia e edição do prato.

4.1.2 Arquitetura lógica

A arquitetura projetada para o sistema possui três elementos principais. Primeiro, o **usuário** — cliente ou vendedor — que interage com uma interface gráfica. Esta será um **aplicativo** móvel — um para cada tipo de usuário — que se comunica de forma segura com um **banco de dados**. Os aplicativos também farão o processamento dos dados, enquanto o banco de dados fará somente a persistência. O diagrama pode ser observado na Figura 8. Na seção 4.2.2 detalharemos a arquitetura, definindo os componentes que a compõem.

Figura 8 - Estrutura inicial da arquitetura lógica da aplicação



4.1.3 Funcionalidades

A seguir, estão listadas as funcionalidades do aplicativo, separadas nos Quadros 3, 4 e 5, de forma a apontar se a funcionalidade é somente para o usuário do tipo cliente ou do tipo vendedor, ou se é referente aos dois tipos de usuário.

Quadro 3 - Funcionalidades de todos os usuários

TODOS OS USUÁRIOS		
FUNCIONALIDADE	**ATENDE AO REQUISITO**	**DESCRIÇÃO**
Criar conta	A	Deve ser possível criar uma conta de usuário no aplicativo, incluindo nome, email e senha obrigatórios; foto de perfil e celular são opcionais.
Recuperar Senha	A	Deve ser possível recuperar a senha utilizando o email cadastrado.
Autenticação	A	Deve ser possível autenticar o usuário no aplicativo com email e senha.
Editar dados pessoais	F, H	Deve ser possível editar dados pessoais da conta de usuário, como nome e telefone. No aplicativo do vendedor, também será possível alterar a

		descrição da loja e os tipos de pagamentos aceitos.
Editar dados cadastrais	A	Deve ser possível editar os dados de cadastro, podendo alterar o email e senha de cadastro.
Editar foto de perfil	F	Deve ser possível alterar a foto de perfil do usuário.
Ver perfil de outro tipo de usuário	F	Deve ser possível, para um usuário cliente, ver os perfis dos vendedores, que contêm: foto de perfil, nome, descrição, formas de pagamento, horários de funcionamento, telefone e as quentinhas disponíveis, além de um botão de acesso ao chat e outro para ver a localização do vendedor no mapa. Deve ser possível, para um usuário vendedor, ver os perfis de clientes que fizeram reservas com ele. As informações contidas no perfil do cliente são: foto de perfil, nome, telefone e um resumo de reservas feitas, além de um botão de acesso ao *chat*.
Receber notificações	F	Deve ser possível receber notificações que avisem o usuário de eventos importantes, como uma nova reserva para um vendedor.
Configurar notificações	F	Deve ser possível para o usuário escolher se receberá notificações e quais tipos de notificações irá receber. Sendo elas do tipo alterações de reserva e novas mensagens do *chat*.
Enviar mensagem	E, F	Deve ser possível para o usuário cliente enviar mensagens para os vendedores. Deve ser possível para o usuário vendedor enviar mensagens para os clientes que fizeram pedidos com ele.

No Quadro 3 temos as funcionalidades pertinentes a ambos os tipos de usuário, referenciando os requisitos atendidos e retratando as funções básicas de uso dos aplicativos e de comunicação entre cliente e vendedor.

Quadro 4 - Funcionalidades de usuários clientes

USUÁRIO CLIENTE		
FUNCIONALIDADE	**ATENDE AO REQUISITO**	**DESCRIÇÃO**
Listar quentinhas	B	Deve ser possível visualizar uma lista de sugestões de quentinhas de vendedores próximos que estejam abertos, exibindo as fotos dos pratos, nomes e preços.
Listar vendedores	C	Deve ser possível visualizar uma lista de vendedores próximos que estejam abertos, mostrando a foto do vendedor, assim como o nome e a distância.
Listar quentinhas já compradas	B, E	Deve ser possível listar quentinhas já reservadas anteriormente, caso estejam disponíveis no momento, para que possam ser facilmente requisitadas novamente.
Ver quentinha	B	Deve ser possível ver a página de uma quentinha, com detalhes como foto, nome, descrição, valor e a loja que a vende.
Fazer reserva	D	Deve ser possível fazer a reserva de uma quentinha que esteja disponível.
Cancelar reserva	D	Deve ser possível cancelar uma reserva feita, que ainda não foi concluída.
Ver histórico	D	Deve ser possível ver o histórico de pedidos feitos, onde se pode visualizar o estado atual do pedido, o nome do vendedor, o nome e a quantidade da quentinha, o valor total do pedido e a data em que foi feito.
Ver o mapa	C	Deve ser possível visualizar um mapa, onde é mostrado a localização atual do cliente, além de marcadores indicando vendedores próximos e uma lista de cartões, exibindo foto do vendedor, nome, se está aberto ou o próximo horário de funcionamento, e um botão “Ver mais” que leva ao perfil dele.
Filtrar mapa	C	Deve ser possível filtrar os vendedores que serão mostrados no mapa, podendo filtrar pelo nome do vendedor e também pela distância, medida em quilômetros.

Marcar vendedor como favorito	E	Deve ser possível marcar vendedores como favoritos. Estes serão mostrados na aba de perfil do cliente e também com prioridade na listagem de vendedores na tela principal.
Alterar raio de busca	C	Deve ser possível alterar o raio de busca de vendedores, através da aba de perfil do cliente e também na aba de mapa.

O Quadro 4 mostra todas as funcionalidades do aplicativo do cliente, referenciando os requisitos aos quais atendem. Essas funcionalidades foram consideradas a fim de possibilitar e facilitar a realização de reservas, além de fortalecer os laços do cliente com os vendedores, conhecidos ou desconhecidos.

Quadro 5 - Funcionalidades de usuários vendedores

USUÁRIO VENDEDOR		
FUNCIONALIDADE	**ATENDE AO REQUISITO**	**DESCRIÇÃO**
Listar pedidos do dia	I	Deve ser possível visualizar uma listagem de pedidos não completos do dia, assim como visualizar uma listagem de pedidos já finalizados do dia.
Alterar estado do pedido	I	Deve ser possível alterar o estado de um pedido entre Reservado, Vendido e Cancelado.
Gerenciar quentinhas	G, H	Deve ser possível adicionar, editar e excluir quentinhas. Assim como marcar elas como disponíveis ou não, definir a quantidade disponível e também colocar até cinco quentinhas em destaque.
Escolher horários de funcionamento	H	Deve ser possível escolher quais dias da semana ele ficará aberto e também os horários de funcionamento.
Escolher localização	G, H	Deve ser possível escolher a localização em que a loja irá aparecer no mapa para os clientes.
Fechar/abrir loja	H	Deve ser possível abrir ou fechar a loja manualmente, independente do horário de funcionamento.

Ver relatório	I	Deve ser possível ver um relatório dos pedidos recebidos, onde contém as informações de valor total recebido, número de clientes, número de quentinhas vendidas, o nome da quentinha mais vendida e um gráfico com o total recebido por dia.
Ver Conversas Recentes	E, F	Deve ser possível visualizar uma lista com todas as conversas, via chat, ordenada pelos mais recentes.

O Quadro 5 mostra todas as funcionalidades do aplicativo do vendedor, referenciando os requisitos aos quais atendem. São retratadas as funções que permitem aos vendedores gerenciar sua loja, suas quentinhas e seus pedidos de reserva, além de facilitarem a comunicação com os clientes.

4.2 IMPLEMENTAÇÃO

Nesta seção serão descritas as tecnologias utilizadas e a motivação para a escolha delas, a consolidação da arquitetura do sistema, a solução de persistência dos dados e a metodologia de desenvolvimento.

4.2.1 Tecnologias usadas

Os aplicativos foram desenvolvidos através do *framework* híbrido Flutter (GOOGLE, 2021g), que usa a linguagem de programação Dart (GOOGLE, 2021b). Além disso, utilizamos o Firebase (GOOGLE, 2021f) para soluções de banco de dados, armazenamento, autenticação e notificação. Todas essas ferramentas são desenvolvidas pela Google, o que nos proporciona um ambiente harmonioso para a implementação do projeto.

4.2.1.1 Flutter

O Flutter foi escolhido para o desenvolvimento por ser uma ferramenta versátil — visto que o código escrito é compilado tanto para Android (GOOGLE, 2021a) quanto para iOS (APPLE, 2021a), excluindo a necessidade de desenvolver separadamente aplicativos

distintos para cada plataforma — pelo nosso conhecimento prévio e pela maior facilidade de implementação de componentes, principalmente gráficos, que o Flutter oferece.

A estrutura básica do Flutter são os *widgets*, que são todos os componentes que fazem parte da interface de usuário do aplicativo: textos, botões, imagens e até o próprio aplicativo. Quase todos os *widgets* possuem a capacidade de englobar outro dentro dele, sendo chamado de widget filho, de forma que podemos dizer que um aplicativo em Flutter é um *widget* composto de vários outros *widgets* menores. Na Figura 9, podemos ver um exemplo desta estrutura. Nela, o *widget* principal é do tipo *Center*, que centraliza todos os componentes contidos nele, possuindo um *Column* como filho, que por sua vez tem dois *widgets* filhos: um *Container* e um *ElevatedButton,* que é um botão com o texto "Tentar novamente".

Figura 9 - Exemplo da estrutura de um *widget*

```
Widget build(BuildContext context) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      mainAxisSize: MainAxisSize.max,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        Container(
          margin: EdgeInsets.symmetric(horizontal: 10, vertical: 5),
          child: AutoSizeText(
            noInternetMessage,
            textAlign: TextAlign.center,
          ), // AutoSizeText
        ), // Container
        ElevatedButton(
          onPressed: () async {
            await Repository.instance.checkInternetConnection(context);
          },
          child: Text('Tentar novamente'),
        ), // ElevatedButton
      ],
    ), // Column
  ); // Center
}
```

4.2.1.2 Firebase

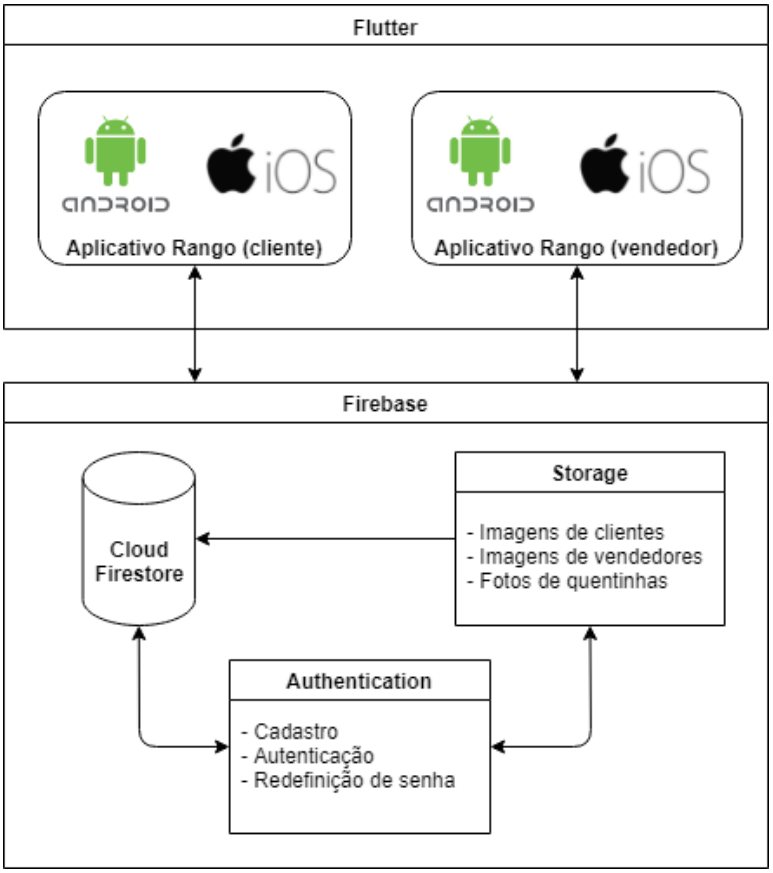
O Firebase é uma plataforma *backend-as-a-service* — ou "parte de suporte como um serviço", em tradução livre — que oferece serviços de autenticação, banco de dados, armazenamento, análise de erros, notificação, entre outros, para aplicações *web* e *mobile*. O Firebase foi escolhido para esse projeto pois oferece todos os tipos de funcionalidades que

desejamos para o desenvolvimento dos aplicativos, além de ser de fácil uso, baixo custo — de fato, com o número esperado de usuários inicialmente não haverá custo algum, dado que a precificação da plataforma oferece um plano gratuito generoso — e ter alta compatibilidade e integração com o Flutter, visto que ambos são mantidos pela Google. Também é importante notar que, usando o Firebase, não é necessário desenvolver uma API própria nem hospedar algum serviço remotamente, já que o mesmo provê um grande leque de soluções prontas e de fácil utilização, reduzindo o trabalho de implementação da comunicação da plataforma com o código do aplicativo. Além disso, a integração entre Flutter e Firebase é feita através de uma simples configuração de credenciais. Para a persistência de dados, escolhemos uma de suas soluções: o *Cloud Firestore* (GOOGLE, 2021c), por ser completo, robusto, possuir gerência de dados em forma de pastas e ter suporte a transações, o que atende melhor os requisitos do projeto. Também foram utilizadas outras ferramentas oferecidas pelo Firebase dentro da aplicação, as quais serão comentadas com mais detalhes posteriormente.

4.2.2 Arquitetura de sistema

Como pode ser observado na Figura 10, a arquitetura de sistema da solução proposta está centrada em dois componentes principais: (i) A interface gráfica utilizada pelos usuários, e os aplicativos do cliente e do vendedor — ambos com versões para Android e iOS — implementado com o *framework* Flutter; e (ii) O banco de dados (serviço *Cloud Firestore*), que foi desanexado da solução de armazenamento de arquivos (serviço *Storage*), implementados com a plataforma Firebase. Ambos são interligados com a solução *Authentication*, de autenticação, que implementa requisitos básicos de segurança e privacidade do sistema como um todo, a partir da criação de conta e realização de *login* do usuário.

Figura 10 - Detalhamento da arquitetura



4.2.3 Componentes principais da arquitetura

Nesta seção abordaremos o uso da plataforma Firebase — sobre a qual o sistema de requisições e armazenamento de dados do aplicativo foi construído — e sua implementação no projeto.

4.2.3.1 Banco de dados

Toda a parte do banco de dados, como explicitado anteriormente, foi implementada utilizando o *Cloud Firestore*. O Firestore é um banco de dados NoSQL, flexível e escalável, organizado em documentos, que são aninhados em coleções. Em cada documento são armazenados os campos com os dados, onde tais campos podem ser dos tipos:

- ***String*:** um campo de texto**;**
- ***Number*:** números, sejam eles inteiros ou reais;
- ***Boolean*:** um campo do tipo verdadeiro ou falso;

- ***Map*:** uma estrutura chave-valor, onde a chave é um identificador de texto e o valor é um campo de qualquer um dos tipos referenciados aqui, inclusive um *map*;
- ***Array*:** uma lista, ou coleção, de campos, estes podendo ser de qualquer tipo aqui referenciado, inclusive um *array*;
- ***Null*:** um campo nulo, indicando um valor inexistente;
- ***Timestamp*:** denota um instante de tempo, contendo o dia, mês, ano e o horário completo;
- ***Geopoint*:** um ponto geográfico, contendo latitude e longitude;
- ***Reference*:** uma referência a outro documento.

Além desses campos, dentro de um documento ainda é possível armazenar coleções. Cada documento possui um identificador (*id*) único que é gerado automaticamente no momento de sua criação. Coleções são conjuntos de vários documentos. O banco de dados foi dividido em quatro coleções principais:

- ***sellers*:** armazena os usuários do tipo vendedor;
- ***clients*:** armazena os usuários do tipo cliente;
- ***orders*:** armazena as reservas (pedidos) criadas;
- ***chat*:** armazena as conversas entre clientes e vendedores;

Cada documento dentro da coleção *chat* é referente a uma troca de mensagens entre dois usuários: um cliente e um vendedor, não permitindo que dois clientes ou dois vendedores troquem mensagens nem que mais de dois usuários participem da mesma conversa. Ao contrário dos outros documentos, o *id* de cada documento dentro das coleções de *chat* não é criado automaticamente pelo Firebase, mas sim pelo aplicativo e possui o seguinte formato: *idDoCliente_idDoVendedor*.

A seguir, os Quadros 6, 7, 8, 9, 10 e 11 descrevem os campos possíveis dentro de cada um desses documentos.

Quadro 6 - Campos de um documento *client*

NOME	**TIPO**	**DESCRIÇÃO**
deviceToken	String	Identificador único de dispositivo, referente ao último dispositivo que o usuário realizou um *login*. É usado no envio de notificações.

email	String	O email de cadastro do usuário.
favoriteSellers	Array	Uma lista contendo os identificadores de cada um dos vendedores marcados como favoritos pelo cliente.
notifications	Map	Um *map* com duas chaves do tipo booleanas: *messages* e *reservations*, indicado se o usuário deseja receber notificações sobre mensagens novas e atualizações de seus pedidos.
phone	String	Telefone celular do cliente.
name	String	O nome do cliente.
picture	String	Uma URL para a foto do perfil do cliente. Essa URL aponta para uma foto armazenada no Storage do Firebase.

Quadro 7 - Campos de um documento *seller*

NOME	**TIPO**	**DESCRIÇÃO**
meals	Coleção	Uma coleção de documentos *meals*, que são as opções de quentinhas criadas pelo vendedor.
deviceToken	String	Identificador único de dispositivo, referente ao último dispositivo que o usuário realizou um *login*. É usado no envio de notificações.
email	String	O email de cadastro do usuário.
active	Boolean	Indica se a loja está aberta ou não.
canReservate	Boolean	Indica se a loja aceita reservas pelo aplicativo.

currentMeals	Map	Um map que contém como chaves outros *maps*. Esses *maps* internos possuem como chaves, os identificadores de cada quentinha que está sendo vendida no momento pelo vendedor e o campo *featured* que indica se aquela quentinha está marcada como destaque, sendo que cada vendedor só pode definir até cinco quentinhas em destaque.
description	String	Um campo de descrição de vendedor, que será mostrado na tela de perfil dele para os clientes.
paymentMethods	String	Campo para o vendedor indicar quais formas de pagamento ele aceita.
shift	Map	Um *map* de *maps*, com os *maps* internos possuindo chaves referentes a cada dia da semana e contendo 3 campos: open, que indica se a loja funciona naquele dia, *openingTime* e *closingTime*, com os horários de abertura e fechamento da loja, caso ela funcione naquele dia.
notificationSettings	Map	Um *map* com duas chaves do tipo booleanas: *messages* e *orders*, indicado se o usuário deseja receber notificações sobre mensagens novas e novos pedidos.
phone	String	Telefone celular do vendedor.
name	String	O nome do vendedor.
logo	String	O logotipo da loja do vendedor.
location	Geopoint	Longitude e latitude da localização atual da loja.

Quadro 8 - Campos de um documento *meal*

NOME	**TIPO**	**DESCRIÇÃO**
name	String	Nome da quentinha.
description	String	Descrição/detalhes da quentinha.
price	Number	Preço da quentinha.
quantity	Number	Número de unidades disponíveis da quentinha.
createdAt	Timestamp	Data e horário em que a quentinha foi criada.
updatedAt	Timestamp	Data e horário em que a quentinha foi atualizada.

Quadro 9 - Campos de um documento *order*

NOME	**TIPO**	**DESCRIÇÃO**
clientId	String	Identificador do cliente que fez o pedido.
clientName	String	O nome do cliente que fez o pedido.
sellerId	String	O identificador do vendedor daquele pedido.
sellerName	String	O nome do vendedor daquele pedido.
mealId	String	Identificador da quentinha pedida.
mealName	String	O nome da quentinha pedida.
price	Number	O valor da quentinha pedida.
quantity	Number	A quantidade de quentinhas naquele pedido.
requestedAt	Timestamp	O instante de tempo em que a reserva foi feita.
reservedAt	Timestamp	O instante de tempo em que a reserva foi confirmada pelo vendedor.
soldAt	Timestamp	O instante de tempo em que a reserva foi marcada como vendida.

canceledAt	Timestamp	O instante de tempo em que a reserva foi cancelada.
status	String	O estado atual da quentinha, indicando se ela está reservada, confirmada, vendida ou cancelada.

Quadro 10 - Campos de um documento *chat*

NOME	**TIPO**	**DESCRIÇÃO**
messages	Coleção	Uma coleção de documentos *message*.
clientId	String	Identificador do cliente participando no *chat*.
clientName	String	O nome do cliente participando no *chat*.
sellerId	String	O identificador do vendedor participando no *chat*.
sellerName	String	O nome do vendedor participando no *chat*.
lastMessageSent	String	Guarda o texto da mensagem mais recente. É usado para mostrar como pré-visualização na lista de conversas recentes.
lastMessageSentAt	Timestamp	Data e horário em que a mensagem mais recente foi enviada. É usado na lista de conversas recentes.

Quadro 11 - Campos de um documento *message*

NOME	**TIPO**	**DESCRIÇÃO**
message	String	O conteúdo da mensagem.
sentAt	Timestamp	Data e horário em que a mensagem foi enviada.
sentBy	String	Indica quem enviou o texto, tendo o valor "*seller*" se quem enviou foi o vendedor ou "*client*" se quem enviou foi o cliente.

4.2.3.2 Autenticação

O serviço de autenticação do Firebase, chamado *Authentication*, nos permite gerenciar a criação, edição, leitura e *login* de usuários no aplicativo. Diferentes provedores de autenticação podem ser utilizados, como email e senha, celular (recebendo um código de verificação por SMS), credenciais da Google, Facebook, Twitter e outros. Neste projeto, utilizamos somente email e senha. Cada usuário cadastrado recebe um código identificador, que é utilizado para reconhecê-lo nas diferentes operações do aplicativo. Mais importante, esse serviço utiliza o protocolo para autorização OAuth 2.0 (INTERNET ENGINEERING TASK FORCE, 2012), padrão de indústria, para possibilitar operações autenticadas e seguras. Além disso, utilizando as Regras de Segurança dos serviços do Firebase é possível assegurar que, por exemplo, somente um usuário autenticado possa fazer alterações no seu cadastro.

4.2.3.3 Armazenamento de arquivos

Para o armazenamento das imagens de perfil dos usuários e também das quentinhas, utilizamos o serviço *Storage*, já incluído na plataforma do Firebase. Escolhemos o *Storage* por atender nossas necessidades, ser uma ferramenta que já utilizamos no desenvolvimento de outros projetos e também por possuir uma utilização gratuita, com o máximo de uso de 5 GB de arquivos armazenados. Nele, temos uma única pasta chamada *users*, que contém pastas, que possuem como nome o identificador (*id*) de cada usuário, com os arquivos de imagens de cada usuário. Se o usuário for do tipo cliente, ele possuirá apenas um arquivo, chamado *picture.png*, correspondente à foto de perfil do usuário. Se o usuário for vendedor, ele possuirá um arquivo chamado *logo.png*, que é a imagem de logo do vendedor, e uma pasta *meals*, contendo as fotos de cada uma das quentinhas, que são nomeadas com o *id* de cada prato.

4.2.4 Regras de segurança

O Firebase permite utilizar Regras de Segurança (GOOGLE, 2021m) para gerenciar o acesso ao banco de dados e ao armazenamento. Escrevendo códigos simples na sua interface, conseguimos relacionar a autenticação ao controle de dados para garantir a propriedade dos usuários sobre suas informações. Esses códigos, retirados da interface do Firebase,

encontram-se no Apêndice M. A seguir, detalhamos como usamos essa funcionalidade no nosso projeto.

Em ambas as coleções de clientes e vendedores, é necessário estar autenticado para realizar operações de leitura. Para operações de escrita, o usuário precisa estar autenticado e, mais importante, ser proprietário do documento. Nas coleções de quentinhas (*meals*) que existem dentro do documento de cada vendedor, também é exigido estar autenticado para operações de leitura. Já para operações de escrita — que nesse caso podem ser divididas em criação, exclusão e atualização — é necessário estar autenticado e, somente para criação e exclusão, ser proprietário do documento de vendedor. Isto é necessário pois o cliente pode gerar uma atualização no campo *quantity* da quentinha ao cancelar uma reserva confirmada. Para leitura e escrita na coleção de pedidos, é exigido do usuário estar autenticado e ser ou o cliente ou o vendedor registrado no documento. Finalmente, na coleção *chat* é necessário estar autenticado e ser ou o cliente ou o vendedor registrado no identificador do documento, que possui o formato *idDoCliente_idDoVendedor*.

As regras de segurança para o armazenamento de arquivos são mais simples, já que cada usuário possui uma pasta nomeada com o seu código identificador. Desse modo, é necessário estar autenticado para acessar qualquer pasta, porém somente o usuário proprietário de uma pasta tem permissão de realizar operações de escrita nesta.

4.2.5 Questões de concorrência

Tanto no aplicativo do cliente quanto do vendedor, tivemos que lidar com questões de concorrência. As operações que lidam com a quantidade de quentinhas disponíveis exigem um cuidado especial para que os dados sejam consistentes para um vendedor e todos os clientes. No aplicativo do cliente, as ações de solicitar e cancelar uma reserva requerem que a aplicação trabalhe com o valor mais atualizado da quantidade de quentinhas disponíveis, assim como no aplicativo do vendedor, nas operações de confirmar, desconfirmar e cancelar uma reserva. Para garantir isto, é utilizada a funcionalidade de Transação (GOOGLE, 2021o) que define as seguintes regras, pertinentes para a questão:

- “as operações de leitura têm que vir antes das operações de gravação”;
- “uma função que chama uma transação (função de transação) pode ser executada mais de uma vez se uma edição simultânea afetar um documento lido pela transação”;
- “todas as operações são bem-sucedidas ou nenhuma delas é aplicada”;

- “as transações apresentarão falha quando o cliente estiver desconectado”.

Seguindo a primeira regra: no aplicativo do vendedor, no momento em que este confirma uma reserva, é necessário verificar se existem quentinhas disponíveis para atender o pedido, que pode ser de uma ou mais unidades, antes de alterar o estado do pedido e decrementar o número de quentinhas disponíveis. Observando a segunda regra, se dois clientes cancelam pedidos de reserva da mesma quentinha ao mesmo tempo, uma transação aguardará a finalização da outra para que o número de quentinhas disponíveis seja atualizado corretamente. Pela terceira regra, qualquer erro gerado dentro de uma transação cancelará as operações seguintes e reverterá as operações de escrita. Finalmente, um usuário que não está conectado à Internet não poderá realizar essas ações. Fazendo o uso das transações, garantimos que as ações sensíveis à concorrência sejam executadas sem comprometer a consistência dos dados entre os usuários.

4.2.6 Telas do aplicativo

As telas do aplicativo foram organizadas em abas. Em cada um dos aplicativos existem quatro abas principais, separando as funcionalidades. As telas do aplicativo do cliente estão disponíveis no Apêndice K e as do aplicativo do vendedor no Apêndice L. A descrição das mesmas vem a seguir:

Para o aplicativo do cliente:

- **Quentinhas:** nesta tela são mostradas para o cliente no máximo três listas: “Peça novamente”, uma lista de quentinhas disponíveis no momento já reservadas anteriormente, “Sugestões”, mostrando uma listagem de sugestões de quentinhas disponíveis e por último “Vendedores”, com a lista dos vendedores abertos próximos;
- **Mapa:** um mapa com a localização e cartões de informação dos vendedores dentro do raio de busca do cliente. Os vendedores mostrados podem ser filtrados pelo nome ou distância;
- **Histórico:** histórico de pedidos feitos;
- **Perfil:** aqui o usuário pode ver as suas informações, podendo as editar, ver os vendedores favoritos, configurar o recebimento de notificações e também sair do aplicativo.

Para o aplicativo do vendedor:

- **Pedidos do dia:** nesta tela são mostrados os pedidos do dia em andamento. O vendedor poderá marcá-los como reservados e em seguida marcá-los como vendidos, ou então cancelá-los. Além disso, há um botão que ao ser pressionado mostra outra tela, listando os pedidos que foram concluídos no dia;
- **Gerenciar cardápio:** nesta tela, o vendedor poderá cadastrar as suas quentinhas e disponibilizá-las para venda, alterar a foto, preço e quantidade de cada uma delas.
- **Conversas recentes:** esta tela mostra as conversas recentes do vendedor com seus clientes, o permitindo entrar em contato diretamente pelo aplicativo. Quando um cliente faz uma reserva, uma mensagem com uma breve descrição do pedido é automaticamente enviada para o vendedor para que este seja notificado sobre as quentinhas reservadas enquanto troca mensagens.
- **Perfil:** nesta tela o vendedor consegue definir e alterar suas informações, como horários de funcionamento, localização da loja, nome, descrição, telefone, formas de pagamento aceitas, email, senha e suas configurações de notificações. É possível também fechar e abrir a loja nessa área. No final desta tela existe um botão, que ao ser pressionado exibe um relatório com algumas métricas de vendas, como número de quentinhas vendidas, qual é a mais vendida e outros dados.

4.2.7 Trechos de código

Nesta seção, apresentamos partes do código do aplicativo, explicando alguns dos algoritmos realizados e como foram implementadas algumas das principais funcionalidades do projeto.

4.2.7.1 Pacotes usados

Na implementação dos dois aplicativos, fizemos o uso de **pacotes Dart**, que são bibliotecas de código que acrescentam ao projeto funcionalidades específicas já implementadas pela comunidade. Esses pacotes são de código fonte aberto e estão

disponibilizados no site pub.dev (GOOGLE, 2021k), o repositório oficial de pacotes para aplicações Dart e Flutter. Estes são os principais pacotes que utilizamos na implementação dos aplicativos:

- **geoflutterfire**: complementa o funcionamento do Firebase Firestore, facilitando consultas no banco de dados usando geolocalização. No aplicativo do cliente, potencializa a busca por vendedores próximos a ele;
- **paginate_firestore**: também funciona de forma complementar ao Firestore, possibilitando a implementação trivial de paginação nas consultas ao banco de dados. Utilizamos, por exemplo, na tela de mensagens entre cliente e vendedor, para que as mensagens antigas sejam carregadas dinamicamente, em demanda;
- **geolocator**: provê uso facilitado de serviços de geolocalização, como consulta da localização do dispositivo e cálculo da distância entre coordenadas geográficas. O Flutter desenvolve aplicativos para Android e iOS, porém a implementação nativa desse e outros serviços nesses sistemas operacionais são distintas. Logo, o uso desse pacote (neste caso, também chamado de *plugin*), é vital para o desenvolvimento híbrido com Flutter;
- **firebase_messaging e flutter_local_notifications**: bibliotecas usadas para a implementação das notificações no aplicativo. A primeira é utilizada para o aplicativo ficar à escuta de atualizações de certos eventos, como mensagens de *chat* ou atualizações de pedidos e receber as informações correspondentes. A segunda é a utilizada para mostrar de fato a notificação no celular do usuário, usando as informações recebidas com o uso da primeira biblioteca.
- **persistent_bottom_nav_bar**: fornece uma barra de navegação persistente altamente customizável, que fica posicionada na borda inferior do aplicativo. Nela, ficam posicionados os ícones que levam às quatro telas principais dos aplicativos, que transicionam entre si com uma animação de *slide*. A aparência foi personalizada para seguir o protótipo inicial, mantendo a identidade visual consistente.

4.2.7.2 Streams

Os dados do Cloud Firestore precisam ser solicitados de forma assíncrona e para isso utilizamos em algumas funcionalidades o *widget* ***stream***.

Um *stream* é uma sequência de eventos síncronos que diferentemente do *Future* — que retorna o próximo evento uma única vez quando solicitado — informa sempre quando há novos eventos. Ou seja, quando se faz uma busca com o *stream* é retornado um observador que notifica caso tenha mudanças no banco de dados, proporcionando à aplicação um reflexo em tempo real do banco de dados.

Os *streams* possuem maneiras fáceis de responder a erros e isso facilita na tratativa dos mesmos. Na Figura 11 observamos um exemplo dessas tratativas, onde é possível identificar se a conexão ainda está sendo feita e mostrar ao usuário o indicativo de que está carregando, e caso ocorra algum erro é possível também mostrar uma mensagem informativa.

É possível integrar *streams* diretamente com o *layout* do Flutter, sem a necessidade de um elemento intermediário para renderizar componentes no aplicativo, através do *widget StreamBuilder*. Esse *widget* possui dois parâmetros importantes: ***stream***, que no caso da Figura 11 retorna a lista de vendedores localizados no raio de busca do cliente, e ***builder***, que trata o valor retornado pelo *stream* através de um *AsyncSnapshot* para exibir na tela do aplicativo uma indicação de carregamento, uma mensagem de erro ou, não exposto na Figura 11, a lista de vendedores. O código observado na figura é responsável pela lógica por trás da tela inicial do aplicativo do cliente, ilustrada na Figura K-3 (esquerda) do Apêndice K.

Figura 11 - Exemplo de uso do Stream

```
return StreamBuilder(
  stream: Repository.instance.getNearbySellersStream(
    locationSnapshot.data, rangeSnapshot.data,
    queryByActive: true, queryByTime: true,
  ),
  builder: (ctxNearbySellers, AsyncSnapshot<List<DocumentSnapshot>> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return _buildLoadingSpinner();
    }
    if (snapshot.hasError) {
      return Container(
        height: 0.6.hp - 56,
        alignment: Alignment.center,
        child: AutoSizeText(
          snapshot.error.toString(),
          style: GoogleFonts.montserrat(
            fontSize: 45.nsp,
            color: Theme.of(ctxNearbySellers).accentColor,
          ),
        ),
      );
    }

    ....

);
```

4.2.7.3 Consulta de vendedores próximos

A seguir, foram separados dois trechos de código utilizados no aplicativo do cliente para consultar os vendedores próximos, localizados dentro do raio de busca. A primeira função observada na Figura 12, *getNearbySellersStream*, foi escrita de forma a escolher se ocorre um filtro por vendedores ativos e abertos no momento ou não. Na primeira tela do aplicativo, onde são exibidas as quentinhas sugeridas e os vendedores, é feito esse filtro. Já na tela do mapa o *stream* de vendedores não é filtrado, permitindo o cliente ver também os vendedores que estão fechados e indicando quando estes estarão abertos novamente.

A função *getNearbySellersStream* recebe quatro parâmetros: a localização do usuário, o raio de busca, um booleano indicando a filtragem por vendedores ativos (opcional) e um booleano indicando a filtragem por vendedores abertos no momento (opcional). É criada uma referência para a coleção de vendedores, armazenada na variável *queryRef*. Vale notar que na definição dessa variável, o código *.where("active")* somente retorna um objeto de tipo *Query*, sem fazer qualquer filtro, que será usado em seguida. Dependendo se os filtros foram solicitados, a consulta adiciona uma checagem pelo campo *active* do vendedor como *true* e/ou uma checagem pelo campo *shift.$weekday.open* como *true*, que indica que o vendedor está aberto naquele dia da semana. A seguir, o *stream* de vendedores é armazenado na variável *sellersStream*, utilizando o pacote *geoflutterfire* mencionado anteriormente. Finalmente, é feito um último filtro no *stream*, caso necessário.

Figura 12 - Função que retorna um stream de vendedores no raio de busca

```
Stream<List<DocumentSnapshot>> getNearbySellersStream(
    Position userLocation,
    double radius, {
    bool queryByActive = false,
    bool queryByTime = false,
  }) {
    GeoFirePoint center = geo.point(
      latitude: userLocation.latitude,
      longitude: userLocation.longitude
    );

    DateTime currentTime = DateTime.now();
    String weekday = weekdayMap[currentTime.weekday];

    var queryRef = FirebaseFirestore.instance.collection('sellers').where("active");

    if (queryByActive) queryRef = queryRef.where("active", isEqualTo: true);
    if (queryByTime) queryRef = queryRef.where("shift.$weekday.open", isEqualTo: true);

    var sellersStream = geo.collection(collectionRef: queryRef).within(
      center: center,
      radius: radius,
      field: "location",
      strictMode: true
    );

    if (queryByTime) {
      return filterTimeRange(sellersStream);
    } else {
      return sellersStream;
    }
}
```

Da maneira como o banco de dados do projeto foi estruturado, os horários de funcionamento de um vendedor são armazenados em dois campos: *openingTime*, que indica o horário de abertura e *closingTime*, que indica o horário de fechamento. Esses horários são armazenados como campos de tipo *number* no formato **hora** concatenado de **minuto** — 12:30 seria **1230**, 08:05 seria **805** — em cada dia da semana. Para consultar os vendedores abertos no momento, é necessário checar se o horário atual, formatado da mesma maneira, é maior que ou igual a *openingTime* e menor que ou igual a *closingTime*. Devemos mencionar agora uma limitação do Firestore. De acordo com a documentação, “Em uma consulta composta, as comparações de intervalos (<, <=, > e >=) e de valores diferentes (!= e not-in) precisam filtrar no mesmo campo” (GOOGLE, 2021c). Como utilizamos dois campos diferentes, *openingTime* e *closingTime*, isto significa que não é possível fazer esse filtro diretamente em uma consulta ao banco de dados. Ao invés disso, a função *filterTimeRange* e a auxiliar *isInTimeRange* mostradas na Figura 13 são utilizadas para filtrar o *stream* após a consulta. Esta limitação possivelmente gera processamento além do estritamente necessário, e é algo que no futuro se beneficiaria de uma atualização desta funcionalidade do Firestore.

Figura 13 - Funções utilizadas para filtrar os vendedores

```
Stream<List<DocumentSnapshot>> filterTimeRange(Stream<List<DocumentSnapshot>> stream) {
  DateTime currentTime = DateTime.now();
  int formattedTime = int.parse(DateFormat("HHmm").format(currentTime));
  String weekday = weekdayMap[currentTime.weekday];

  return stream.map((documents) => documents
        .where((seller) => isInTimeRange(seller, weekday, formattedTime))
        .map((i) => i)
        .toList());
}

bool isInTimeRange(DocumentSnapshot seller, String weekday, int time) {
  var sellerData = seller.data() as Map<String, dynamic>;

  return sellerData["shift"][weekday]["openingTime"] <= time &&
         sellerData["shift"][weekday]["closingTime"] >= time;
}
```

As Figuras 12 e 13 mostram a busca feita no banco de dados para que apareçam na tela os vendedores mais próximos que estejam abertos. Esse código é executado no aplicativo do cliente ao acessar a tela inicial, ilustrada na Figura K-3 (esquerda) do Apêndice K.

4.2.7.4 Confirmação de reserva

Na Figura 14, iremos analisar um exemplo de transação do Firestore, mencionado na seção 4.4. A função em questão é chamada quando o vendedor confirma a reserva de um pedido. Nesta operação, é importante que o número de unidades disponíveis da quentinha esteja atualizado com o valor mais recente do banco de dados. Primeiramente, as leituras e escritas estão envolvidas em uma função *runTransaction* do Firestore, definindo uma transação com as propriedades expostas anteriormente. Duas variáveis são definidas: *mealRef*, que armazena uma referência para o documento da quentinha, e *orderRef*, uma referência para o documento do pedido. Ambos documentos poderão ser atualizados. Depois, é feita a operação *transaction.get<Meal>(mealRef)* que retorna o documento mais atualizado da quentinha, do qual é extraída a quantidade. Com este valor, é verificado se o vendedor possui unidades suficientes para atender a esse pedido. Caso sim, a quantidade é decrementada adequadamente e o pedido é atualizado com o novo estado *reserved*. Ambas operações são feitas através da função *transaction.update*. Caso contrário, uma mensagem de erro é retornada, instruindo o usuário.

Figura 14 - Função que realiza a confirmação de uma reserva

```
Future<void> reserveOrderTransaction(Order order) async {
  try {
    return await FirebaseFirestore.instance.runTransaction((transaction) async {
      DocumentReference<Meal> mealRef = mealsRef(order.sellerId).doc(order.mealId);
      DocumentReference<Order> orderRef = ordersRef.doc(order.id);

      DocumentSnapshot<Meal> snapshot = await transaction.get<Meal>(mealRef);
      int quantity = snapshot.data().quantity;

      if (quantity - order.quantity >= 0) {
        transaction.update(mealRef, {'quantity': quantity - order.quantity});
        transaction.update(orderRef, {'status': 'reserved', 'reservedAt': Timestamp.now()});
      } else {
        switch (quantity) {
          case 0:
            throw ("Você não possui mais quentinhas dessa. Cancele o pedido ou aumente o número de
quentinhas, caso tenha mais.");
            break;
          case 1:
            throw ("Você só possui $quantity unidade dessa quentinha disponível. Cancele o pedido ou
aumente o número de quentinhas, caso tenha mais.");
            break;
          default:
            throw ("Você só possui $quantity unidades dessa quentinha disponíveis. Cancele o pedido ou
aumente o número de quentinhas, caso tenha mais.");
        }
      }
    });
  } catch (e) {
    throw e;
  }
}
```

Este código é executado após a tela de confirmação de reserva, ilustrada na Figura K-4 (esquerda) do Apêndice K.

4.2.7.5 Notificações

O Firebase possui uma ferramenta para recebimento e envio de mensagens para aplicações, o *Firebase Cloud Messaging* (GOOGLE, 2021d). Com ele, é possível enviar mensagens para os dispositivos de usuários, utilizando um identificador único para cada dispositivo, chamado aqui de *token*, tornando possível enviar notificações únicas para cada usuário.

Nos aplicativos, no momento que o usuário faz o login — seja ele o primeiro login ou não — é atribuído um *token* de dispositivo para aquele usuário. Se o *token* for igual ao já registrado, ou seja, o usuário está entrando com o mesmo dispositivo, este *token* não será atualizado no banco de dados. Caso seja diferente, o mesmo será sobrescrito.

A principal forma de realizar o envio dessas mensagens é desenvolvendo um sistema *backend*, que trabalharia em conjunto ao Cloud Messaging. Como desenvolver um sistema desse tipo estava fora do escopo do projeto, abordamos o envio de notificações de outra forma.

Utilizando os *tokens* salvos, conseguimos enviar as notificações para outros usuários dentro do próprio aplicativo através de requisições HTTP, através da url *https://fcm.googleapis.com/fcm/send* fornecida pelo Firebase. Ao enviar uma requisição *POST* para esse endereço, conseguimos mandar uma mensagem para um dispositivo específico. Na Figura 15, é mostrado o caso onde é enviada uma notificação após o envio de uma nova mensagem em uma conversa com outro usuário. O corpo da requisição contém os campos ***priority,*** referente a prioridade da notificação, ***data***, contendo os dados da notificação e ***to,*** que é o token do dispositivo do destinatário da mensagem.

Dentro dos dados da mensagem, os seguintes campos são definidos:

- ***id***: identificador da notificação. Não é utilizado pelo aplicativo, mas é obrigatório, então usamos o valor padrão *2*;
- ***channelId, channelName e channelDescription****:* identificador, nome e descrição do canal de notificações, respectivamente. Usado para separar as notificações em categorias que podem ser reconhecidas pelo sistema operacional do celular do usuário, para que o mesmo possa desativar notificações de um canal específico;
- ***status****:* o estado daquela mensagem. Como utilizamos apenas mensagens do tipo de notificação, que não esperam por respostas, o estado de todas sempre será *done*, ou seja, finalizada;
- ***title****:* o título da notificação que será mostrada no aparelho do usuário;
- ***description****:* o texto que será mostrado no corpo da notificação no aparelho do usuário;
- ***payload****:* dados extras que serão usados no dispositivo destino para quando o usuário clicar na notificação recebida. Esse *payload* varia de acordo com o tipo de notificação. Se esta for referente a atualizações de pedidos, ele será apenas um texto escrito "*orders*". Caso seja sobre uma nova mensagem de uma conversa, ele seguirá o formato "*chat/identificadorDoDestinatario/nomeDoDestinatario"*.

Figura 15 - Código mostrando o envio de notificação após nova mensagem

```
Future<void> _sendNewMessageNotification() async {
    var name = FirebaseAuth.instance.currentUser.displayName;
    if (name == '') {
      name = 'Um vendedor';
    }
    try {
      var data = (<String, String>{
        'id': '2',
        'channelId': '2',
        'channelName': 'Chat',
        'channelDescription': 'Canal usado para notificações do chat',
        'status': 'done',
        'description': '$name te enviou uma mensagem no chat.',
        'payload': 'chat/$userId/$name',
        'title': 'Você recebeu uma nova mensagem!',
      });
      await http.post(
        Uri.parse('https://fcm.googleapis.com/fcm/send'),
        headers: <String, String>{
          'Authorization':
              'key=${const String.fromEnvironment('MESSAGING_KEY')}',
          'content-type': 'application/json; charset=UTF-8'
        },
        body: jsonEncode(<String, dynamic>{
          'priority': 'high',
          'data': data,
          'to': _client.deviceToken,
        }),
      );
    } catch (error) {
      print(error);
    }
  }
```

O envio dessas mensagens, no aplicativo do vendedor, também é feito quando o vendedor altera o estado de um pedido. No aplicativo do cliente, é também feito quando um cliente faz uma nova reserva ou cancela uma já existente.

Após o envio da mensagem, é necessário recebê-la no dispositivo de destino. Para isso, utilizamos o pacote *firebase_messaging.* Ao implementá-lo, o dispositivo do usuário fica à espera do envio de mensagens. Para que isso seja ativado, é necessário uma configuração inicial no arquivo *main* do aplicativo, como pode ser visto na Figura 16. Usamos duas funções da biblioteca *firebase_messaging:* a ***FirebaseMessaging.instance.subscribeToTopic('all')*** para que o dispositivo receba qualquer tipo de notificação que seja enviada a ele e a ***FirebaseMessaging.onMessage.listen,*** onde é definido o que será feito ao receber uma nova mensagem. Neste caso, é verificado se a mensagem recebida é do tipo *chat* e se a tela onde o usuário está atualmente é a de conversas. Isto é feito para que não sejam mostradas notificações desnecessárias de novas mensagens enquanto o usuário estiver na tela de

conversas. Em qualquer outro caso, será chamada a função ***_showNotification(message.data)***, onde passamos os dados da notificação a ser mostrada para o usuário.

Figura 16 - Código mostrando configuração inicial da escuta de notificação

```
FirebaseMessaging.instance.subscribeToTopic('all');
  FirebaseMessaging.onMessage.listen((RemoteMessage message) {
    if ((message.data['payload'].toString().contains('chat') &&
            chatScreenKey.currentState == null) ||
        !message.data['payload'].toString().contains('chat')) {
      _showNotification(message.data);
    }
  });
```

A biblioteca do Firebase que utilizamos para receber as notificações também dispõe da visualização delas, mas não atendeu aos nossos requisitos. Logo, decidimos usar outro pacote para auxiliar esta tarefa: a *flutter_local_notifications*. Na configuração inicial, é possível definir uma ação disparada quando o aplicativo abrir, após o usuário clicar na notificação.

Na Figura 17, mostramos o caso do aplicativo do vendedor, onde existem três possibilidades de ação a serem feitas quando uma notificação é selecionada:

- Se refere à uma atualização de estado de um pedido, o aplicativo troca para a aba Pedidos do Dia;
- Se refere ao recebimento de uma nova mensagem em uma conversa, é utilizado o identificador e o nome do usuário que enviou a mensagem e o aplicativo mostra automaticamente a tela referente.

No aplicativo do cliente, a diferença é que o mesmo não possui quentinhas, ou seja, não recebe notificações deste tipo, apenas notificações de novas mensagens e atualizações de estado de pedidos realizados.

Figura 17 - Ações realizadas após uma notificação ser selecionada

```
await flutterLocalNotificationsPlugin.initialize(
    initializationSettings,
    onSelectNotification: (String payload) async {
      if (payload != null) {
        if (payload == 'orders') {
          _controller.jumpToTab(0);
        } else if (payload == 'meals') {
          _controller.jumpToTab(1);
        } else if (payload.contains('chat')) {
          String clientId = payload.split('/')[1];
          String clientName = payload.split('/')[2];
          pushNewScreen(
            currentKey.currentState.context,
            withNavBar: false,
            screen: ChatScreen(clientId, clientName, key: chatScreenKey),
          );
        }
      }
    },
  );
```

Finalmente, na Figura 18, mostramos a chamada da função utilizada para mostrar a notificação. Nela são passados os parâmetros *id*, título, descrição e *payload* da notificação, além de informações do canal daquela notificação.

Figura 18 - Função para visualização de uma notificação

```
await flutterLocalNotificationsPlugin.show(
    int.parse(message['id']),
    message['title'],
    message['description'],
    platformChannelSpecifics,
    payload: message['payload'],
  );
```

4.2.7.6 Armazenamento local

Além do Cloud Firestore, banco de dados do Firebase, utilizamos também outra forma de armazenamento, porém de forma local através de *Shared Preferences* (GOOGLE, 2021n) e *NSUserDefaults* (APPLE, 2021b).

O pacote do Dart *shared_preferences* é usado para armazenar pequenas informações, no formato chave-valor, diretamente no dispositivo em que o aplicativo foi instalado. Este

plugin faz a abstração desta implementação de armazenamento local para dispositivos Android, com *Shared Preferences*, e iOS, com *NSUserDefaults*.

A Figura 19 mostra a forma de guardar e recuperar informações do armazenamento local. Nesse exemplo a informação guardada é o raio de busca dos vendedores, que pode ser alterado na aba de perfil do aplicativo do cliente. Foi utilizado o armazenamento local pois se trata de uma preferência do usuário que não será utilizada em mais nenhuma outra área além do próprio celular do mesmo.

A função *getSellerRange* aguarda o retorno da instância do *SharedPreferences*. Com esta, verifica se o campo chave *seller_range* existe. Caso sim, retorna o valor correspondente, que é do tipo *double*. Senão, retorna um valor inicial padrão que é de 10 km.

A função *setSellerRange* também aguarda a instância do *SharedPreferences*, porém com esta, simplesmente define o valor da chave *seller_range* como o parâmetro *range*, de tipo *double*, que é passado para a função.

Figura 19 - Exemplo de utilização do *Shared Preferences*

```
Future<double> getSellerRange() async {
  var prefs = await SharedPreferences.getInstance();
  if (prefs.getDouble('seller_range') == null) {
    return Future.value(10.0);
  }
  return Future.value(prefs.getDouble('seller_range'));
}

setSellerRange(double range) async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  prefs.setDouble('seller_range', range);
}
```

Este código é executado em diversos momentos na aplicação, tal como no perfil do cliente, ilustrado na Figura K-9 (direita) do Apêndice K.

5 AVALIAÇÕES

Este capítulo apresenta as metodologias e resultados das avaliações realizadas neste trabalho. Duas avaliações distintas foram realizadas, uma voltada para os testes com os usuários, que teve o objetivo de analisar as funcionalidades e usabilidades com usuários reais, e outra voltada para o desempenho, que teve o objetivo de analisar o desempenho do aplicativo em diferentes dispositivos.

5.1 AVALIAÇÃO DE USUÁRIOS

Nesta seção, será apresentada a metodologia utilizada nos testes com usuários, seguida das duas rodadas de testes, a primeira com usuário técnicos e a segunda com um público maior, onde são comentados os principais pontos levantados em cada rodada. Por fim, comentamos os resultados gerais das avaliações de ambos aplicativos.

5.1.1 Metodologia

Como forma de avaliar a execução da aplicação, de modo a entender se o projeto possui uma interface amigável e intuitiva e se as funcionalidades essenciais para a proposta foram implementadas, realizamos testes de ambos aplicativos com alguns usuários, onde a primeira leva de testes foram com usuários técnicos e a segunda com todos os tipos de usuários. Em cada aplicativo, criamos contas nossas para podermos interagir com os usuários que estivessem participando dos testes.

Para estes testes, enviamos às pessoas que se propuseram a experimentar os aplicativos um formulário de cada aplicativo para ser preenchido após a finalização do experimento. Os formulários dos aplicativos do cliente e do vendedor estão, respectivamente, disponíveis nos Apêndices F e H, enquanto que as respostas estão nos Apêndices G e I. Além dos campos a serem preenchidos, cada formulário continha uma breve descrição do aplicativo e do próprio teste, nossos meios de contato, uma explicação de como instalar o aplicativo no dispositivo e um roteiro do que deveria ser testado em cada aplicação.

Nesse pequeno roteiro, colocamos uma série de ações para que cada usuário testasse as funcionalidades principais do aplicativo. Foi decidido que essas ações não seriam explicadas

em detalhe, para que o usuário pudesse realizá-las sozinho, de forma a testar se a interface dos aplicativos é realmente intuitiva.

Todos os testes foram realizados apenas em dispositivos Android. Não foi possível fazer testes com celulares iOS por existirem certos entraves que não poderíamos superar na hora de realizar testes com usuários.

5.1.2 Testes com grupo técnico

A primeira rodada de testes foi realizada com um pequeno público de usuários que possuem alguma experiência prévia com o desenvolvimento de aplicativos *mobile*. Isto foi decidido para que se houvesse algum *bug*, problema crítico no aplicativo, o mesmo fosse reproduzido por esse grupo de usuários e então resolvido antes do início dos testes com um grupo mais abrangente.

Nesta rodada, todos que participaram dos testes testaram ambos os aplicativos, onde atuamos como vendedores no aplicativo do cliente e como clientes no aplicativo do vendedor, para que fosse possível que os participantes conseguissem completar todos os testes requisitados.

Algo que foi comentado pela maior parte dos usuários que testaram essa versão do aplicativo foi a falta de um local de fácil acesso às conversas recentes no aplicativo do vendedor. Até essa versão, o vendedor só poderia acessar a conversa com algum cliente acessando o perfil do mesmo, sendo que o perfil de um cliente só poderia ser acessado através da lista de pedidos ou do histórico, de forma que não existia um acesso fácil e intuitivo.

Em razão disso, foi decidido pela adição de uma nova aba no aplicativo do vendedor, onde é mostrada uma lista de cartões referentes às conversas com os clientes, ordenados pela data de envio da última mensagem. Nestes cartões são exibidos o nome do cliente, a última mensagem enviada na conversa e o horário de envio dessa mensagem. Também foram relatados alguns *bugs*, na sua maioria pequenos e não críticos, que foram prontamente solucionados, sendo o principal o que impossibilitava a criação de contas novas.

5.1.3 Testes com grupo abrangente

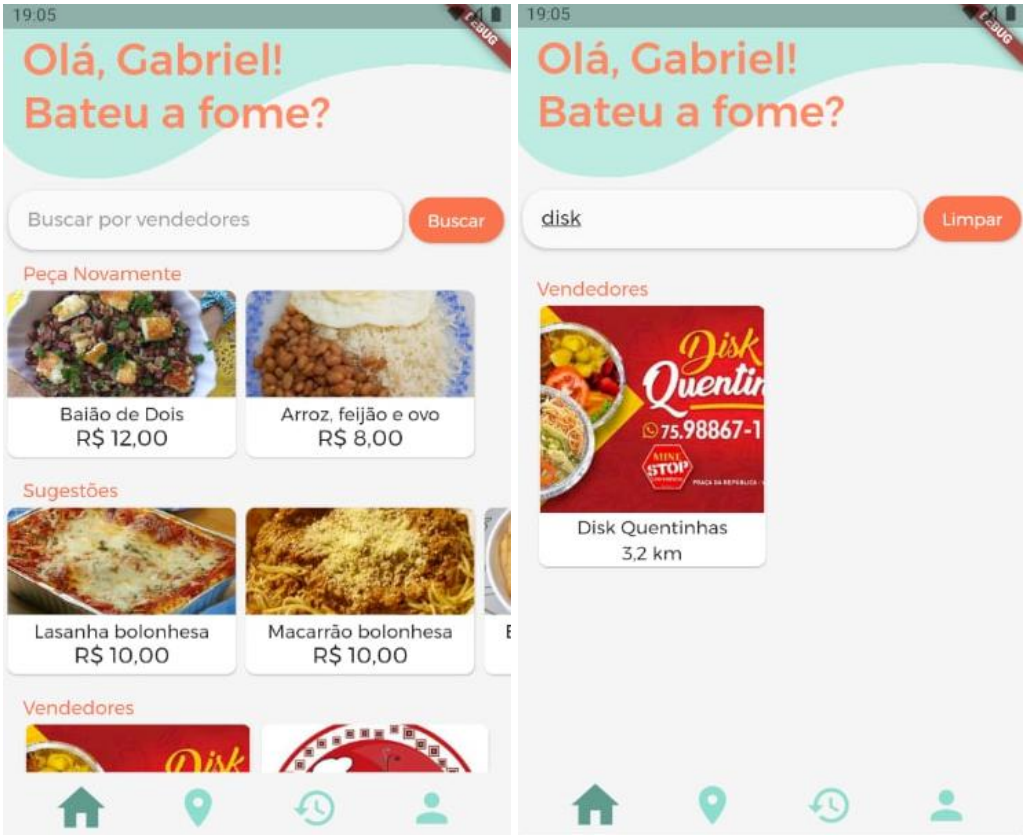
Nesta segunda rodada de testes, enviamos os aplicativos para um grupo maior de usuários, onde a versão do aplicativo vendedor enviada continha a nova aba de conversas recentes, enquanto que o aplicativo do cliente não teve nenhuma grande mudança.

Nesta rodada, a maior parte dos usuários testaram apenas o aplicativo do cliente, enquanto que apenas alguns testaram ambos os aplicativos. Isto foi feito para termos uma maior possibilidade de mais usuários dispostos a ajudar com os testes. Novamente, atuamos como vendedores no aplicativo do cliente e como clientes no aplicativo do vendedor.

Não foi possível realizar testes do aplicativo com vendedores de quentinhas, pois no momento em que as avaliações foram realizadas, as aulas presenciais na UFRJ ainda estavam suspensas devido à pandemia do novo coronavírus, o que impossibilitou o contato com os vendedores de quentinha da Cidade Universitária.

Uma questão comentada por diversos usuários foi a falta de uma busca por vendedores na tela inicial do aplicativo do cliente. Usuários relataram que não conseguiram achar o filtro de vendedores, que está contido na aba de mapas, ou que o mesmo não é intuitivo. Também foi percebido que para o usuário pesquisar algum vendedor, ele é obrigado a abrir a aba do mapa, fazendo com que o usuário tenha um trabalho desnecessário. Considerando isso, resolvemos implementar, após esses testes, uma busca por vendedores na tela principal do aplicativo, como pode ser observado nas Figura 20. Foram adicionados um campo de texto, onde o usuário pode digitar o nome de um vendedor, e um botão de busca, onde o usuário irá clicar após terminar de escrever, para realizar a busca.

Figura 20 - Campo de busca na tela inicial do aplicativo



Ao ser realizada a busca, as listas da tela são limpas, sendo mantida apenas a lista de vendedores, mostrando todos os vendedores com o nome compatível com o texto buscado. A busca não diferencia maiúsculas de minúsculas e ignora acentuação.

5.1.4 Resumo das avaliações

Nas subseções a seguir, será apresentado um resumo dos resultados das avaliações de cada um dos aplicativos, comentando a quantidade de respostas, as notas médias recebidas nos quesitos apresentados e alguns exemplos de comentários recebidos.

5.1.4.1 Aplicativo do cliente

Obtivemos 28 respostas neste formulário, onde algumas delas podem ser vistas no Quadro 12. Apenas dois responderam que não conseguiram completar todos os testes, um informou que ao clicar no botão de WhatsApp do vendedor, nada aconteceu e o outro que não

conseguiu realizar uma reserva devido à distância dele com os vendedores. Também obtivemos apenas uma resposta “Não” onde perguntamos se o aplicativo possui todas as funcionalidades necessárias para atender a demanda de um comprador.

O aplicativo teve uma média de 4,5 no quesito usabilidade, com 100% dos usuários achando que o aplicativo ajudaria clientes a conhecerem novos vendedores e apenas duas respostas dizendo que teriam uma chance menor a 4, em uma escala de 1 a 5, de utilizar o aplicativo em seu lançamento.

Quadro 12 - Exemplos de respostas no formulário do cliente

Resposta
“Faltou a busca dos restaurantes na home.”
“Senti falta de um sistema de nota, para saber quais pratos são mais comprados e tals.”
“Sugestão: criar um campo "observações" na hora de fazer o pedido para caso o cliente queira fazer uma solicitação específica. Pelo chat é um pouco menos prático.”
“Na página de editar perfil, trocaria o texto de ‘Editar’ para ‘Trocar foto’ (caso tenha) ou ‘Adicionar foto’ (caso não tenha), ou até melhor, colocaria um botão com ícone que tenha uma indicação para a edição da foto. Eu achei que esse botão/texto de "editar" fosse pra editar o formulário inteiro.“
“Agrupar pedidos em um "carrinho", que desse pra comprar mais de 1 pedido por vez.”
“Ótimo para quem quer agilidade na compra de uma refeição. Aplicativo de uso prático e intuitivo.”
“Em algum momento quando fui reservar na minha loja apareceu uma mensagem de erro do firebase falando que estava indisponível. Não sei como aconteceu e quando reiniciei o app não tive mais problema”

Os pontos mais comentados nos formulários foram a necessidade de uma busca na tela principal, que implementamos após os testes, e um tutorial das principais funcionalidades do aplicativo em seu primeiro uso pelo usuário.

Em relação aos *bugs*, poucos usuários tiveram e a maioria não ocorreu mais ao reiniciar o aplicativo.

5.1.4.2 Aplicativo do vendedor

Nessa avaliação, obtivemos 16 respostas, onde todos responderam que conseguiram realizar todos os passos do roteiro. Algumas respostas recebidas neste formulário podem ser visualizadas no Quadro 13.

A média de usabilidade do aplicativo ficou aproximadamente 4,4, onde apenas uma pessoa respondeu com nota menor que 4.

Duas pessoas responderam informando que não acham que o aplicativo possui todas as funcionalidades necessárias para atender a demanda de um vendedor, onde uma disse não saber por não ser vendedor e a outra comentou sobre a possibilidade de selecionar de uma vez todos os horários de funcionamento, ao invés de individualmente.

Na questão sobre se o aplicativo ajudaria vendedores a se conectarem com clientes, todas as respostas foram de nota 4 ou 5. Das respostas sobre se usariam o aplicativo quando fosse lançado, apenas duas respostas foram abaixo de 4.

Quadro 13 - Exemplos de respostas no formulário do vendedor

Respostas
“Aplicativo bem simples, rápido e intuitivo. Seria ótimo, principalmente, para pequenos e médios empreendedores.”
“O aplicativo deixa o usuário colocar uma senha fácil.As telas precisam de um tutorial quando for a primeira vez que o usuário acessar as sessões.”
“O botão de sair, recomendo colocar também home e dentro da sessão do perfil do usuário colocar também o link de sair, sem precisar clicar na engrenagem.”
“Muito bonito e fluido a navegação, estão de parabéns"
“Está excelente! Parabéns!”

Assim como no aplicativo do cliente, tivemos bastante elogios, principalmente em relação à interface, também não tivemos relatos de *bugs* críticos, mas sim sugestões pontuais que não necessitarão de muito trabalho para serem implementadas.

5.1.5 Conclusão

A partir das respostas dos formulários, foi possível concluir que ambas aplicações obtiveram resultados satisfatórios e com possibilidade de ajustes e melhorias futuras, apesar de não ter sido possível realizar as entrevistas inicialmente planejadas com os vendedores.

Os relatos de *bugs* foram resolvidos e todas as sugestões de melhorias foram anotadas para serem implementadas futuramente nos aplicativos.

Por terem sido implementadas as funcionalidades básicas necessárias para o uso planejado dos aplicativos, junto das boas avaliações recebidas pelos usuários nos testes, podemos considerar que os aplicativos são ambos MVP, Mínimo Produto Viável, e que poderão ser lançados para o público.

5.2 AVALIAÇÃO DE DESEMPENHO

Esta seção descreve a metodologia e a forma de execução usadas na análise do desempenho de alguns modelos de celulares pré-selecionados. O objetivo é examinar a viabilidade do uso do aplicativo do vendedor nos modelos de celulares mais comuns hoje em dia.

5.2.1 Metodologia

Foi usada a ferramenta *Test Lab* do Firebase (GOOGLE, 2021e) para avaliar o uso de memória e CPU do aplicativo do vendedor, usando o pacote *integration_test*[12] — assim como os pacotes mencionados na seção 4.2.7.1, disponível no site pub.dev — do Flutter. A metodologia usada consiste em criar um robô que simula as tarefas feitas por um vendedor, usuário real, no aplicativo e com isso avaliar o desempenho do celular usado.

A simulação é feita por comandos de clicar, preencher texto e identificar *widgets* dentro do aplicativo. Com isso conseguimos simular o caminho que o usuário realiza para ir nas principais funcionalidades.

Por restrições do pacote do *integration_test*, não foi possível realizar testes no aplicativo do cliente. O aplicativo do cliente exige o uso da geolocalização para iniciar o aplicativo, porém esse pacote não tem suporte para liberar permissões em tempo real e por

[12] Pacote *integration_test*. Disponível em: https://pub.dev/packages/integration_test. Acesso em: 27 set. 2021.

isso não foi possível o teste. Prosseguimos apenas com o fluxo principal do aplicativo do vendedor, que são tarefas realizadas de forma frequente pelo vendedor.

Os testes realizados no aplicativo do vendedor foram:

- Login;
- Confirmar uma reserva de quentinha;
- Adicionar novo prato;
- Fechar a loja.

5.2.2 Execução

Os testes foram realizados em três dispositivos das marcas: LG, Xiaomi e Samsung. As Figuras 21 a 26 mostram os gráficos de uso da CPU e da memória nos três dispositivos avaliados.

O gráfico de CPU de cada modelo mostra a porcentagem de CPU utilizada, ou seja, o quanto de processamento foi usado ao longo da execução do teste. Já o gráfico de memória mostra a demanda de armazenamento de dados durante o experimento.

De acordo com o gráfico da Figuras 21, o uso da CPU no modelo LG variou de 0 até pouco mais que 30% durante todo o uso do aplicativo nos testes. O gráfico de memória, exibido na Figura 22, mostra que o aplicativo usou cerca de 200 MB de memória ao longo dos testes. Esse dispositivo tem as seguintes especificações:

- Modelo: LG-8932
- Android 8.0
- Processador: Qualcomm Snapdragon 835 2.45 GHz x 4 + 1.9 GHz x 4 Octa-Core MSM8998
- Memória: 4 GB RAM

Figura 21 - Gráfico de CPU do modelo LG

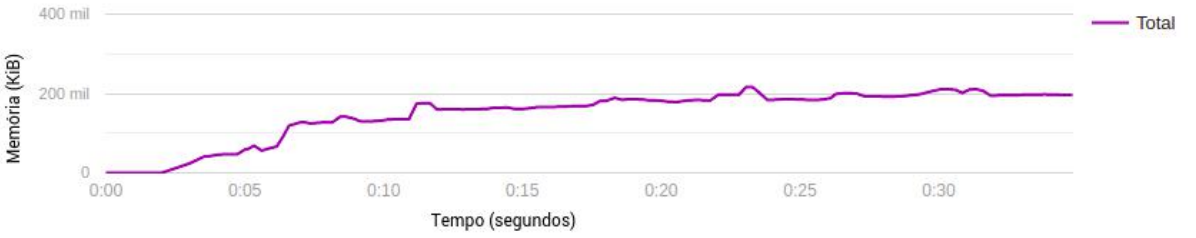
Figura 22 - Gráfico de Memória do modelo LG

De acordo com o gráfico da Figuras 23, o uso da CPU no modelo Xiaomi variou de 0 até pouco menos que 50% durante todo o uso do aplicativo nos testes. Temos também o gráfico de memória, exibido na Figura 25, que mostra que o aplicativo usou cerca de 200 MB de memória ao longo dos testes. Esse dispositivo tem as seguintes especificações:

- Modelo: Redmi Note 6 Pro
- Android 9
- Processador: 4 x 1.8 GHz Kryo 260 + 4 x 1.6 GHz Kryo 260
- Memória: 3 GB RAM

Figura 23 - Gráfico de CPU do modelo Xiaomi

Figura 24 - Gráfico de Memória do modelo Xiaomi



De acordo com o gráfico da Figura 25, o uso da CPU no modelo Samsung variou de 0 até pouco mais que 50% durante todo o uso do aplicativo nos testes. Temos também o gráfico de memória, exibido na Figura 26, que mostra que o aplicativo usou cerca de 250 MB de memória ao longo dos testes. Esse dispositivo tem as seguintes configurações:

- Modelo: Samsung SM-G930AZ
- Android 8.0
- Processador: 2.15 GHz Dual-Core Kryo + 1.6 GHz Dual-Core Kryo
- Memória: 4 GB RAM

Figura 25 - Gráfico de CPU do modelo Samsung

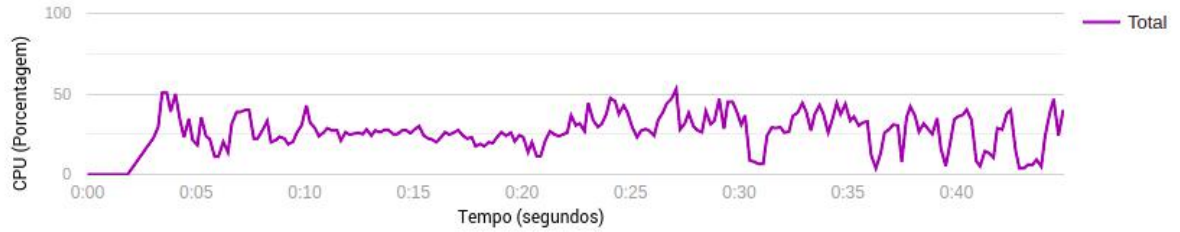


Figura 26 - Gráfico de Memória do modelo Samsung

5.2.2 Conclusão

Os dispositivos usados nos experimentos descritos acima foram escolhidos por serem de marcas renomadas e relativamente comuns no mercado, por essa razão, as chances de usuários reais utilizarem esses modelos são maiores.

De acordo com os resultados acima, a média de memória utilizada foi de 230 MB. Celulares mais populares do mercado possuem, pelo menos, 1 ou 2 GB de memória, portanto a aplicação não teria problemas nesse quesito.

Os gráficos mostraram que na média o uso da CPU foi de 30% durante o uso do aplicativo e isso é relativamente baixo no uso de aplicativos do dia a dia. Por isso, conforme os resultados, concluímos que é viável o uso do aplicativo nos celulares modernos.

6 CONCLUSÃO

Este trabalho propôs duas aplicações para dispositivos móveis de venda e compra de quentinhas, visando facilitar a descoberta de novos vendedores pela parte do cliente e aumentar e facilitar a operação dos vendedores. Inicialmente o projeto foi pensado para uso dentro da Cidade Universitária da UFRJ, mas o mesmo foi projetado sem nenhuma restrição para isso, de forma que pode ser usado em qualquer lugar que possua vendedores de quentinhas e clientes.

Por meio da utilização das ferramentas de desenvolvimento Flutter e Firebase, transformamos o projeto em um produto funcional: dois aplicativos que se conectam a um mesmo banco de dados, que puderam ser testados por usuários reais, a fim de avaliar os principais aspectos da execução proposta.

Os produtos finais desenvolvidos neste trabalho obtiveram ótimos resultados em suas avaliações, principalmente em seu *design* e *layout*, que foram um dos principais objetivos: desenvolver uma aplicação que apresentasse um visual agradável, intuitivo e convidativo para seus usuários.

Os códigos do projeto são abertos e estão disponíveis no GitHub[13], de forma a encorajar a colaboração para futuras melhorias. Os aplicativos do cliente[14] e do vendedor[15] podem ser acessados e instalados por qualquer pessoa, porém apenas em dispositivos Android. A versão para dispositivos iOS não foi publicada devido aos requerimentos impostos, onde é necessário possuir iPhone e um computador Mac para a publicação do aplicativo. As ferramentas usadas no gerenciamento e controle de versões do projeto estão disponíveis no Apêndice J.

Os aplicativos serão disponibilizados também na Google Play Store, loja virtual de aplicativos Android, para download por qualquer pessoa. Após o adquirimento das ferramentas para publicação de aplicativos iOS, poderão ser disponibilizados também na loja virtual Apple Store. Tendo em vista que o Rango estará disponível para download facilmente e que os aplicativos não se restringem ao uso na UFRJ, poderão ser aproveitados em diversos outros cenários além do campus.

Ao desenvolver este trabalho, colocamos em prática conhecimentos adquiridos durante o curso de Ciência da Computação, em disciplinas como Computação 1 e 2, Fundamentos da

---

[13] Repositório no GitHub. Disponível em: https://github.com/DevMobUFRJ/rango. Acesso em: 20 set. 2021.
[14] Rango cliente. Disponível em: https://tinyurl.com/rangoCliente. Acesso em: 20 set. 2021.
[15] Rango vendedor. Disponível em: https://tinyurl.com/fmxf63et. Acesso em 20 set. 2021.

Engenharia de *Software* e Banco de Dados, nas quais foi possível aprender conceitos importantes como estruturação, modelagem, desenvolvimento e gerenciamento de *software*. Outras disciplinas também contribuíram de forma direta para a nossa formação e consequentemente para este trabalho.

Através da universidade conhecemos o grupo de extensão Devmob, que por meio da troca de conhecimento entre os membros e elaboração de projetos, aprendemos de forma prática sobre desenvolvimento *mobile*. Por meio do Devmob, também tivemos a oportunidade de conhecer a ideia e o protótipo do aplicativo que tornou-se a ideia central deste trabalho de conclusão de curso.

6.1 FUNCIONALIDADES FUTURAS

Para que fosse possível manter o escopo deste trabalho, algumas funcionalidades dos aplicativos não foram desenvolvidas. Apesar disso, a seguir apresentamos as principais implementações que melhorariam o projeto, o tornando mais adequado para uso pelo público em geral:

- ***Onboarding*:** para facilitar o entendimento do usuário sobre as funcionalidades dos aplicativos, na primeira utilização dos mesmos, logo após a autenticação, será apresentado ao usuário um tutorial do aplicativo, que mostrará as principais funções do mesmo, ensinando a ele como desfrutar ao máximo do que o aplicativo tem a oferecer. Um pacote do Flutter poderá ser usado para desenvolver o *onboarding*.
- **Marcações nas quentinhas:** para facilitar a busca por quentinhas pelos clientes, os vendedores poderão colocar marcações em seus pratos. Um exemplo seria em um prato vegetariano, o vendedor poderia adicionar a marcação *vegetariano*, de forma que o cliente, ao buscar por essa marcação, conseguirá encontrar o prato.
- **Envio de imagens no chat:** para melhorar a comunicação entre usuário e vendedor, imagens poderão ser enviadas em conversas. Para evitar a necessidade de utilizar um plano pago do Firebase, já que utilizamos um gratuito que possui limite de utilização de armazenamento de imagens, ao implementar o envio de imagens, elas poderiam ser deletadas automaticamente após um determinado tempo.
- **Preferências por tipos de comidas:** para atender às necessidades de usuários que possuem algum tipo de restrição alimentar, os clientes poderão configurar se possuem algum tipo de restrição ou preferência, fazendo com que seja mostrado apenas pratos

que respeitem esses requisitos. Essas opções poderão ser definidas na configuração de perfil, considerando diversas restrições alimentares e preferências.

- **Avaliação de vendedores**: para que o cliente possa avaliar a sua experiência com um vendedor, atendimento e qualidade da comida, será implementado um sistema de nota, no formato de cinco estrelas. Além disso, poderá registrar um elogio ou reclamação. Dessa forma, os clientes terão mais confiança na escolha da quentinha desejada. Estas informações ficarão disponíveis na página do vendedor para consultas futuras. Não será possível avaliar vendedores com quem o cliente não fez negócio ou quentinhas que não foram compradas.
- **Conversas recentes agrupadas no aplicativo do cliente:** para tornar mais fácil o acesso às conversas do cliente entre vendedores, uma tela listando as conversas recentes será adicionada no aplicativo do cliente, assim como foi adicionada no do vendedor. Porém, para não adicionar outra aba na barra de navegação inferior, que já contém quatro, o botão para esta tela ficará localizado na aba de perfil do cliente.
- **Extensão da busca na tela principal:** como foi constatado após a rodada final de testes com usuários, a busca por vendedores na tela principal foi implementada de forma simples, buscando apenas por vendedores. Posteriormente, esta procura será melhorada, de forma que possibilite a busca pelo nome de uma quentinha. No resultado dessa busca, além de ser exibida a lista de vendedores, será mostrada a lista de quentinhas correspondentes.
- **Ilustrações em listas vazias:** existem situações nos aplicativos em que a tela não tem informações a serem mostradas para o usuário. Nesses casos, para não exibir um campo vazio, são mostrados textos para o usuário, informando que nada foi encontrado e o incentivando a utilizar as funcionalidades do aplicativo. Futuramente, além desses textos, serão acrescentadas ilustrações relacionadas ao contexto, de modo que a interface seja mais agradável e intuitiva para o usuário.
- **Tema escuro:** atualmente existe uma demanda para as aplicações, *mobile* ou *desktop*, principalmente aquelas que possuem muito uso da cor branca, de oferecerem ao usuário a opção de usar um tema escuro, de forma a não prejudicar tanto a própria visão. Visto isso, o aplicativo possuirá uma paleta de cores escura que o usuário poderá usar ou não, ao ativar as configurações do aplicativo.

Como mencionado na seção 2.2, uma pesquisa foi realizada com um vendedor do “Maná Caseiro Quentinhas”, localizado na Praça Mestre Monir Salomão - Jacarepaguá, Rio

de Janeiro - RJ, no dia 3 de setembro de 2021. Após obter as respostas ao formulário, que se encontra no Apêndice A, os aplicativos foram mostrados ao vendedor e suas funcionalidades foram apresentadas. Durante este processo, diversos comentários e sugestões foram feitos pelo vendedor, que foram registrados em papel. A seguir, apresentamos as funcionalidades sugeridas:

- **Aumentar/diminuir quantidade de quentinhas rapidamente:** o gerenciamento da quantidade disponível de quentinhas é automático para reservas, porém, caso o vendedor faça uma venda sem reserva ou receba mais quentinhas, deve editar o número manualmente. Para facilitar esse processo, botões para aumentar e diminuir as quantidades disponíveis rapidamente serão adicionados na tela de gerenciamento de cardápio.
- ***Progressive web app*:** como mencionado na seção 2.2, a criação de *web apps* seria útil para os vendedores e principalmente para os clientes. Assim, os usuários conseguiriam utilizar os aplicativos sem se preocupar com espaço de armazenamento do dispositivo. Além disso, possibilitaria o envio de um *link* direto para o cardápio de um vendedor, permitindo divulgá-lo por outros meios. Visto que o Flutter também pode ser usado para gerar aplicações *web*, a implementação de *web apps* será simples. Depois, serão hospedados no Firebase, que oferece essa solução.
- **Sistema de fidelização:** para melhorar os relacionamentos entre clientes e vendedores, será implementado um sistema de fidelização, levando em consideração quantias gastas ou número de quentinhas compradas. Por exemplo: a cada R$ 100,00 gastos ou dez quentinhas que um cliente comprar do vendedor, receberá uma refeição grátis.
- **Opções de bebidas no momento da reserva:** para oferecer aos clientes uma maneira fácil de escolher bebidas, será exibida uma lista simples com as opções após escolher a quentinha para reserva.
- **Opção de adicionar uma observação no pedido:** em algumas situações, os clientes podem desejar adicionar uma observação no pedido de uma quentinha, solicitando a retirada de algum ingrediente, por exemplo. Para isso, um campo de texto será adicionado na tela de reserva de quentinha, onde o usuário poderá digitar a observação.
- **Promoções condicionais:** para incentivar a compra de quentinhas pelos clientes, será implementada para os vendedores a opção de configuração de promoções gerais ou em

quentinhas específicas. Por exemplo, poderá ser definido um desconto por pagamento em dinheiro, ou uma redução de preço em certos horários.

- **Separação entre horários de reservas e horários de funcionamento:** na versão final do aplicativo do vendedor desenvolvido neste projeto, um usuário é capaz de configurar os horários de funcionamento, somente. Considerando a ocasião em que, por exemplo, desejar receber pedidos de reserva até as 10 horas, mas continuar aberto além desse horário para vendas no local, sem reserva, a separação entre horários de reservas e horários de funcionamento será implementada. Para isso, será adicionada ao aplicativo uma tela semelhante à já existente, permitindo essa separação, caso o vendedor deseje.
- **Entregas:** para implementar a funcionalidade de entregas, que podem ser requisitadas pelos clientes, diversas modificações deverão ser feitas no aplicativo do vendedor e do cliente. O cliente deverá poder definir o endereço onde deseja que a entrega da quentinha seja feita, e o vendedor precisará distinguir se e aonde este pedido deve ser entregue. Os detalhes desta implementação ainda não foram estudados.
- **Integração com impressora não fiscal:** para ter maior controle dos pedidos, a conexão entre o aplicativo do vendedor e uma impressora não fiscal poderá ser implementada, através de *bluetooth*. Assim, no momento em que uma reserva for realizada, um papel será impresso com: nome do vendedor, nome do cliente, número e hora do pedido, nome e quantidade da quentinha, bebida e preço total. Isto será especialmente útil para entregas, quando esse papel impresso pode ser fixado à embalagem, para facilitar a entrega e cobrança do pedido.

REFERÊNCIAS

APPLE. **iOS 14**. Disponível em https://www.apple.com/br/ios/ios-14. Acesso em: 25 ago. 2021a.

APPLE. **NSUserDefaults**. Disponível em: https://developer.apple.com/documentation/foundation/nsuserdefaults. Acesso em: 27 ago. 2021b.

ATLASSIAN. **Trello**. Disponível em: https://trello.com. Acesso em: 27 ago. 2021.

AXOSOFT. **GitKraken**. Disponível em: https://www.gitkraken.com. Acesso em: 27 ago. 2021.

Como apps de entrega estão levando pequenos restaurantes à falência. **G1**, 08 fev. 2020. Disponível em: https://g1.globo.com/economia/tecnologia/noticia/2020/02/08/como-apps-de-entrega-estao-levando-pequenos-restaurantes-a-falencia.ghtml. Acesso em: 5 ago. 2020.

GITHUB. **GitHub**. Disponível em: https://github.com. Acesso em: 27 ago. 2021.

GOOGLE. **Android**. Disponível em: https://www.android.com/intl/pt-BR_br. Acesso em: 25 ago. 2021a.

GOOGLE. **Dart**. Disponível em: https://dart.dev. Acesso em: 25 ago. 2021b.

GOOGLE. **Documentação Cloud Firestore**. Disponível em: https://firebase.google.com/docs/firestore. Acesso em: 2 ago. 2021c.

GOOGLE. **Documentação Firebase Cloud Messaging**. Disponível em: https://firebase.google.com/docs/cloud-messaging. Acesso em: 27 ago. 2021d.

GOOGLE. **Documentação Firebase Test Lab**. Disponível em: https://firebase.google.com/docs/test-lab. Acesso em: 27 set. 2021e.

GOOGLE. **Firebase**. Disponível em: https://firebase.google.com/?hl=pt. Acesso em: 25 ago. 2021f.

GOOGLE. **Flutter**. Disponível em: https://flutter.dev. Acesso em: 25 ago. 2021g.

GOOGLE. **Formulários Google**. Disponível em: https://www.google.com/intl/pt-BR/forms/about. Acesso em: 25 ago. 2021h.

GOOGLE. **Google Drive**. Disponível em: https://www.google.com/intl/pt-BR/drive. Acesso em: 27 ago. 2021i.

GOOGLE. **Google Meet**. Disponível em: https://meet.google.com. Acesso em: 27 ago. 2021j.

GOOGLE. **Pub.dev**. Disponível em: https://pub.dev. Acesso em: 27 ago. 2021k.

GOOGLE. **Regras de segurança do Firebase**. Disponível em: https://firebase.google.com/docs/rules?hl=pt-br. Acesso em: 26 ago. 2021m.

GOOGLE. **Shared Preferences**. Disponível em: https://developer.android.com/training/data-storage/shared-preferences?hl=pt-br. Acesso em: 27 ago. 2021n.

GOOGLE. **Transações e gravações em lote**. Disponível em: https://firebase.google.com/docs/firestore/manage-data/transactions?hl=pt-br. Acesso em: 26 ago. 2021o.

INTERNET ENGINEERING TASK FORCE. **RFC 6749 - The OAuth 2.0 Authorization Framework**. 2012. Disponível em: https://tools.ietf.org/html/rfc6749. Acesso em: 26 ago. 2021.

MEIRELLES, Fernando S. **31ª Pesquisa Anual do FGVcia: Uso da TI nas Empresas**, 31. ed. São Paulo, 2020. Disponível em: https://eaesp.fgv.br/sites/eaesp.fgv.br/files/u68/fgvcia2020pesti-resultados.pdf. Acesso em: 5 ago. 2020.

GLOSSÁRIO

Android – Sistema operacional de celulares e tablets mantido pela Google.

API – Em inglês, *Application Programming Interface*, é um conjunto de padrões e rotinas prontos que permitem um desenvolvimento mais prático de aplicações.

iOS – Sistema operacional de celulares e tablets mantido pela Apple.

Framework – Conjunto de pacotes de códigos prontos que auxiliam no desenvolvimento de aplicações.

NoSQL – Termo que se refere aos bancos de dados não relacionais.

Requisição POST – Uma requisição é um pedido que uma aplicação faz a um servidor, sendo a do tipo POST, aquela que indica que a aplicação deseja enviar dados para o servidor.

APÊNDICES

APÊNDICE A – FORMULÁRIO DE PESQUISA COM MANÁ CASEIRO QUENTINHAS

1. **Você usa algum desses aplicativos para anunciar ou vender suas quentinhas?** (múltipla escolha)
   ( ) iFood
   ( ) Rappi
   ( ) Uber Eats
   ( ) Facebook Marketplace
   ( ) WhatsApp/Telegram
   ( ) Instagram
   ( ) Goomer
   ( ) Delivery App
2. **Por quê usa esses ou nenhum?**
3. **Quais são os pontos negativos de usar esses apps, caso use?**
4. **Você entra em contato de alguma forma diretamente com seus clientes?**
   ( ) Sim
   ( ) Não
5. **Se sim, como?**
6. **Você realiza a reserva de quentinhas?**
   ( ) Sim
   ( ) Não
7. **Como é feita a reserva?**
8. **Você gostaria que as reservas fossem feitas de forma automática?**
   ( ) Sim
   ( ) Não
9. **Como você lida nos casos onde o cliente reserva a quentinha mas acaba não comprando?**
10. **Você usaria um aplicativo grátis com as seguintes funcionalidades?**
    Criar cardápio; Ligar/desligar reservas; Definir quantidade de quentinha para reservas; Definir e mostrar sua localização; Definir horários de funcionamento;

Definir formas de pagamento; Ver pedidos do dia, confirmando ou cancelando eles; Chat dentro do aplicativo, ou abrir no WhatsApp; Ver um rápido gráfico de vendas por dia; Sem pagamento ou delivery integrado.

( ) Sim

( ) Não

11. **Você realiza o serviço de delivery das quentinhas?**

( ) Sim

( ) Não

12. Se sim, como é feito? (se entrega na hora, se a entrega é feita a cada quantidade de pedidos, se cobra taxa extra, etc)

13. Se não, porque não faz?

APÊNDICE B – FORMULÁRIO DE PESQUISA DE DEMANDA DO APLICATIVO PARA CLIENTES

14. **Qual é o seu vínculo com a UFRJ?**
    ( ) Estudante
    ( ) Professor
    ( ) Servidor
    ( ) Outro
15. **Onde trabalha/estuda na UFRJ?**
    ( ) CCMN
    ( ) CT
    ( ) Letras
    ( ) Reitoria
    ( ) CCS
    ( ) EEFD
    ( ) Parque Tecnológico
    ( ) Hospital Universitário
    ( ) Outro
16. **Costuma comprar quentinhas?**
    ( ) Sim
    ( ) Não

Seção "Não costuma comprar quentinhas", caso quem respondeu tenha marcado "Não" na questão 3 da primeira seção:

1. Por qual(s) motivo(s) você não compra quentinhas?
2. Se um dos motivos é por restrição alimentar, qual você possui?
    ( ) Vegetariano
    ( ) Vegano
    ( ) Intolerante a lactose
    ( ) Intolerante a glúten
    ( ) Outro

3. Você usaria um aplicativo que facilitaria a procura de quentinhas?
   ( ) Sim
   ( ) Não

Seção “Costuma comprar quentinhas”, caso quem respondeu tenha marcado “Sim” na questão 3 da primeira seção:

1. Com que frequência você compra quentinhas?
   ( ) 1 vez por semana
   ( ) 2-4 vezes por semana
   ( ) Todos os dias da semana
   ( ) Ocasionalmente
2. Onde costuma comprar as quentinhas? (múltipla escolha)
   ( ) CCMN
   ( ) CT
   ( ) Letras
   ( ) Reitoria
   ( ) CCS
   ( ) EEFD
   ( ) Parque Tecnológico
   ( ) Outros
3. Você costuma comprar sempre no mesmo local/vendedor?
   ( ) Sim
   ( ) Não
4. Gostaria de conhecer mais opções?
   ( ) Sim
   ( ) Não
5. Quais motivos te levam a comprar determinada quentinha? (múltipla escolha)
   ( ) Preço
   ( ) Fila
   ( ) Localização
   ( ) Gosto
   ( ) Tamanho
   ( ) Restrição alimentar

( ) Variedade

( ) Outro

6. Você reserva suas quentinhas?

   ( ) Sim

   ( ) Não

   ( ) Ocasionalmente

7. Como você realiza as reservas das quentinhas?
8. Você se considera satisfeito em como são feitas as reservas de quentinhas?

   ( ) Sim

   ( ) Não

9. Por qual(s) motivo(s) você reserva ou deixa de reservar?
10. Você se interessaria por um aplicativo que visa facilitar o uso dos serviços de quentinhas?

    ( ) Sim

    ( ) Não

Seção “Sobre o aplicativo”:

1. Você conhece algum aplicativo para quentinhas?

   ( ) Sim

   ( ) Não

2. Se sim, qual?
3. E por que usa ou não usa?
4. Quais serviços você espera que um aplicativo desse tipo deve fornecer? (múltipla escolha)

   ( ) Cardápios das quentinhas

   ( ) Possibilidade de reservas

   ( ) Mapa mostrando localização das quentinhas

   ( ) Busca de quentinhas por filtros (nome, preço máximo-mínimo, quentinhas especiais, etc)

   ( ) Outro

5. Você gostaria de avaliar os vendedores onde já comprou quentinha?

   ( ) Sim

   ( ) Não

APÊNDICE C – RESULTADOS DA PESQUISA DE DEMANDA DO APLICATIVO PARA CLIENTES

Figura C-1 - Respostas da Pergunta 1 da primeira seção

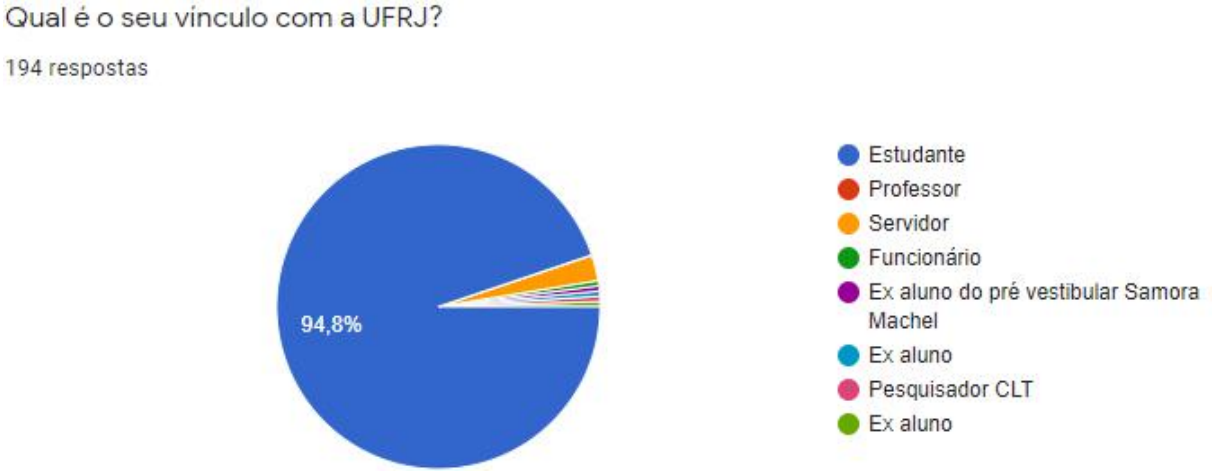


Figura C-2 - Respostas da Pergunta 2 da primeira seção

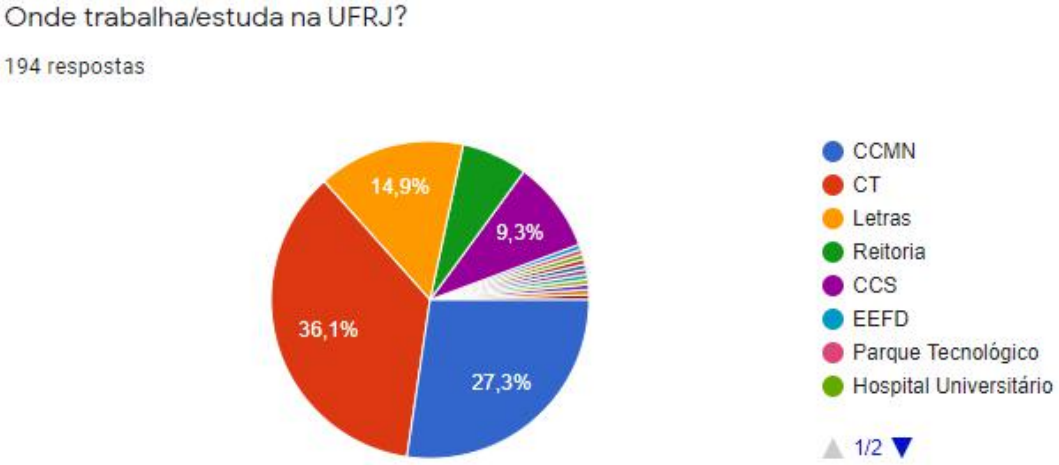


Figura C-3 - Respostas da Pergunta 3 da primeira seção

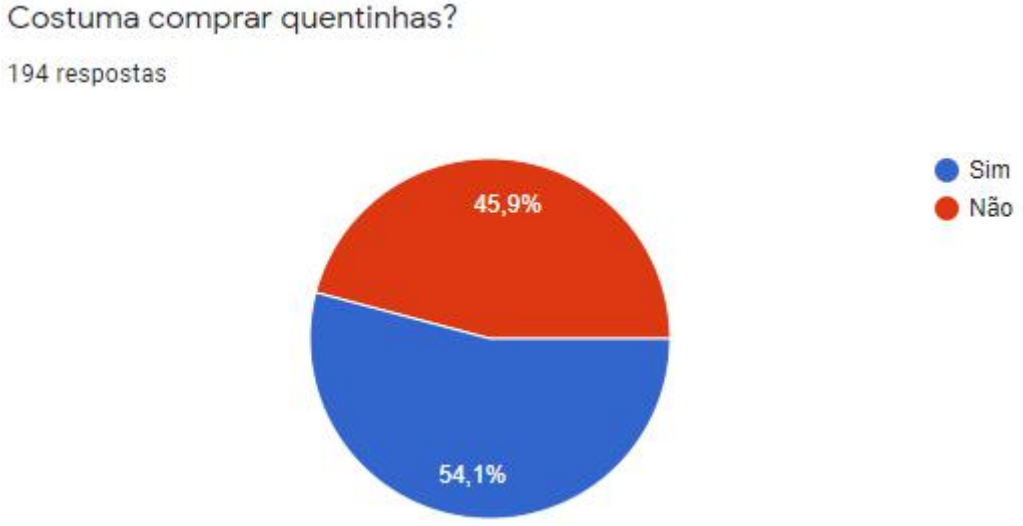


Figura C-4 - Respostas da Pergunta 1 da seção “Não costumo comprar quentinhas”

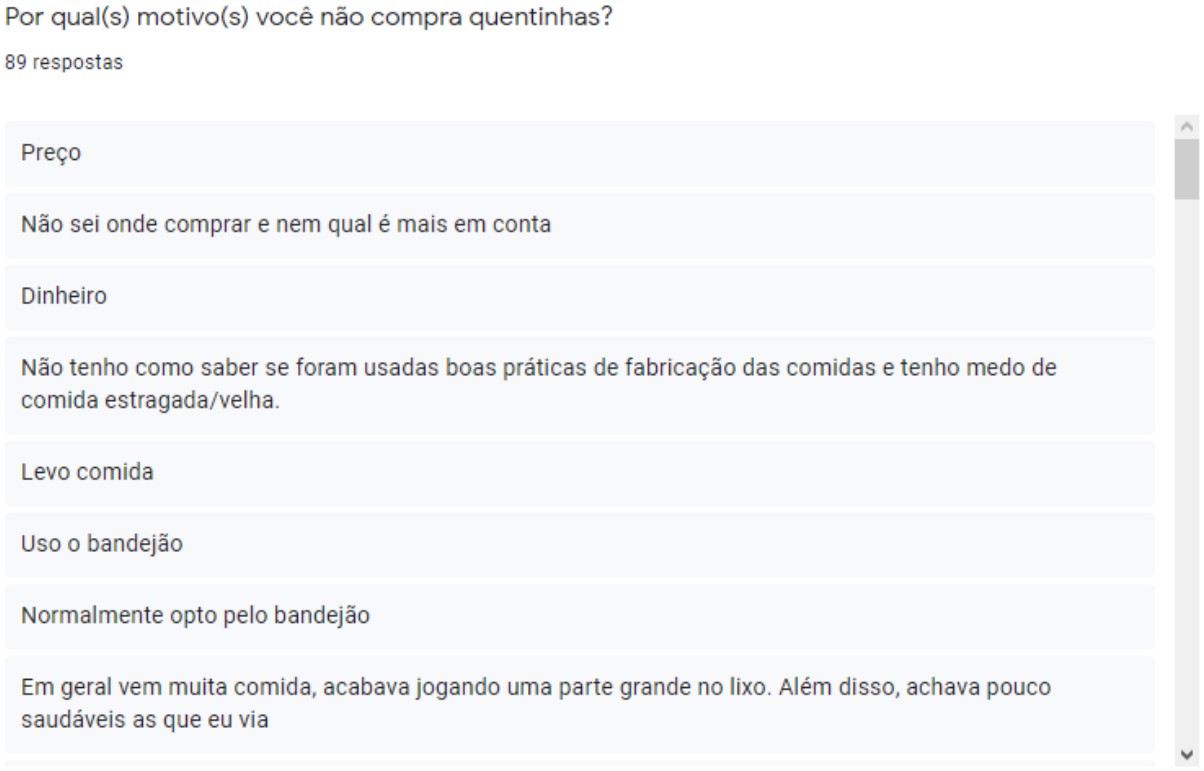


Figura C-5 - Respostas da Pergunta 2 da seção “Não costumo comprar quentinhas”

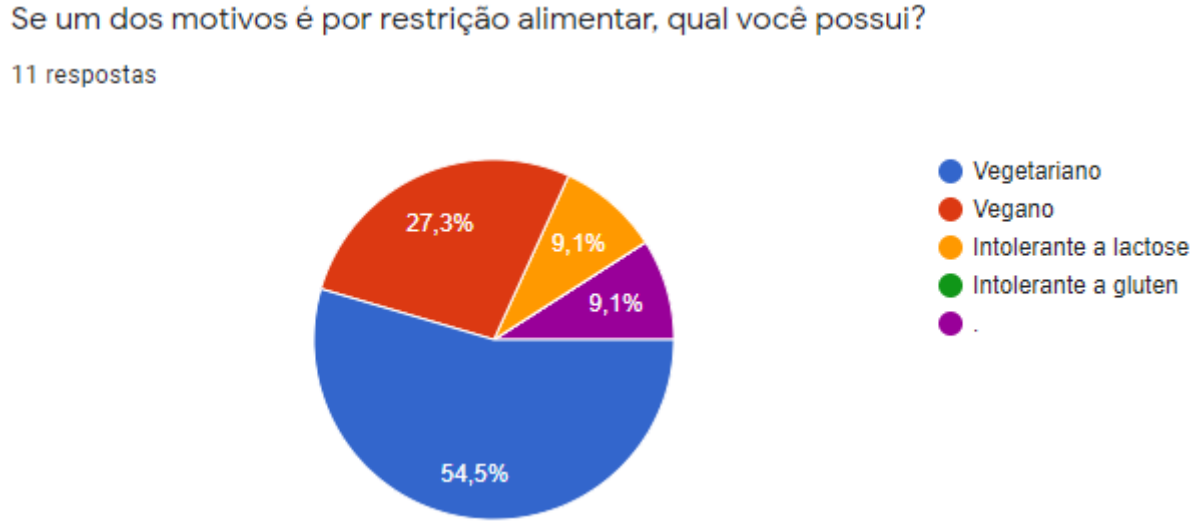


Figura C-6 - Respostas da Pergunta 3 da seção “Não costumo comprar quentinhas”

Figura C-7 - Respostas da Pergunta 1 da seção "Costumo comprar quentinhas"

Figura C-8 - Respostas da Pergunta 2 da seção "Costumo comprar quentinhas"

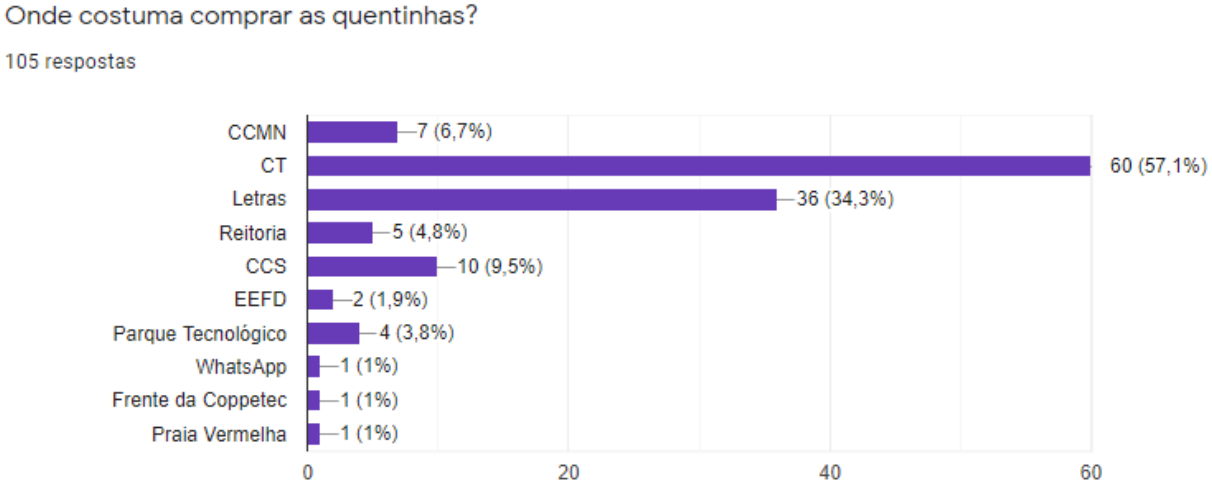


Figura C-9 - Respostas da Pergunta 3 da seção "Costumo comprar quentinhas"

Figura C-10 - Respostas da Pergunta 4 da seção "Costumo comprar quentinhas"

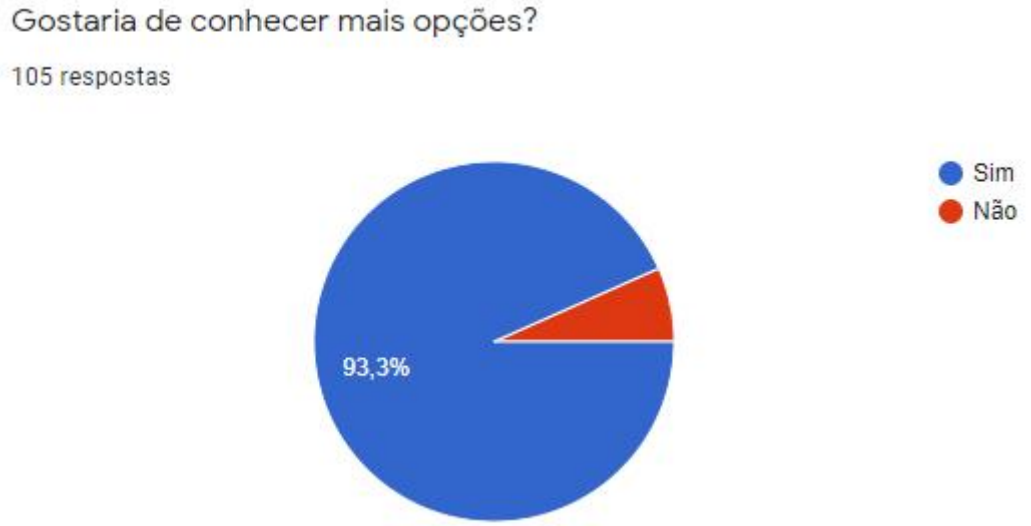


Figura C-11 - Respostas da Pergunta 5 da seção "Costumo comprar quentinhas"

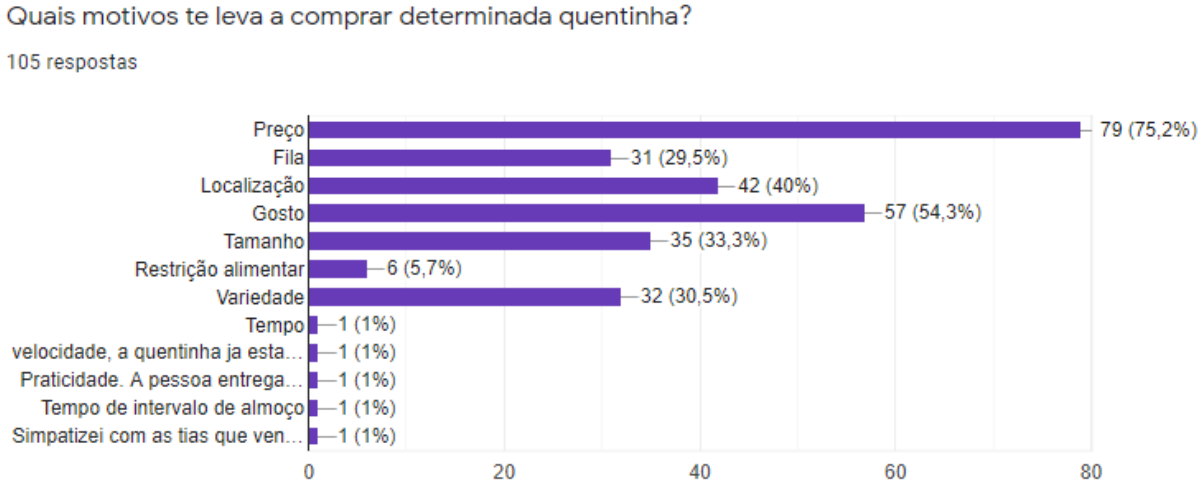


Figura C-12 - Respostas da pergunta 6 da seção "Costumo comprar quentinhas"

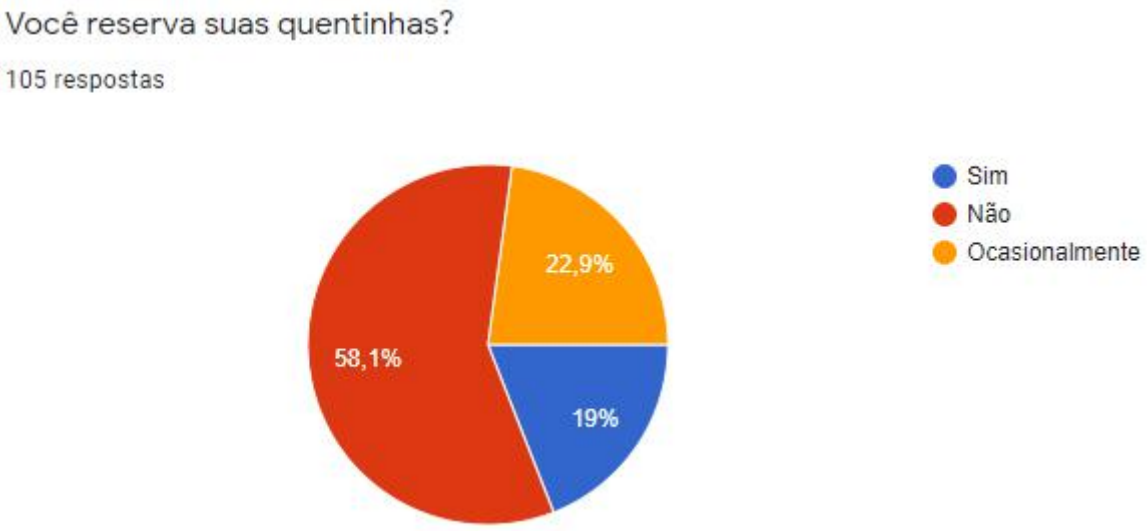


Figura C-13 - Respostas da pergunta 7 da seção "Costumo comprar quentinhas"

Figura C-14 - Respostas da Pergunta 8 da seção "Costumo comprar quentinhas"

Figura C-15 - Respostas da Pergunta 9 da seção "Costumo comprar quentinhas"

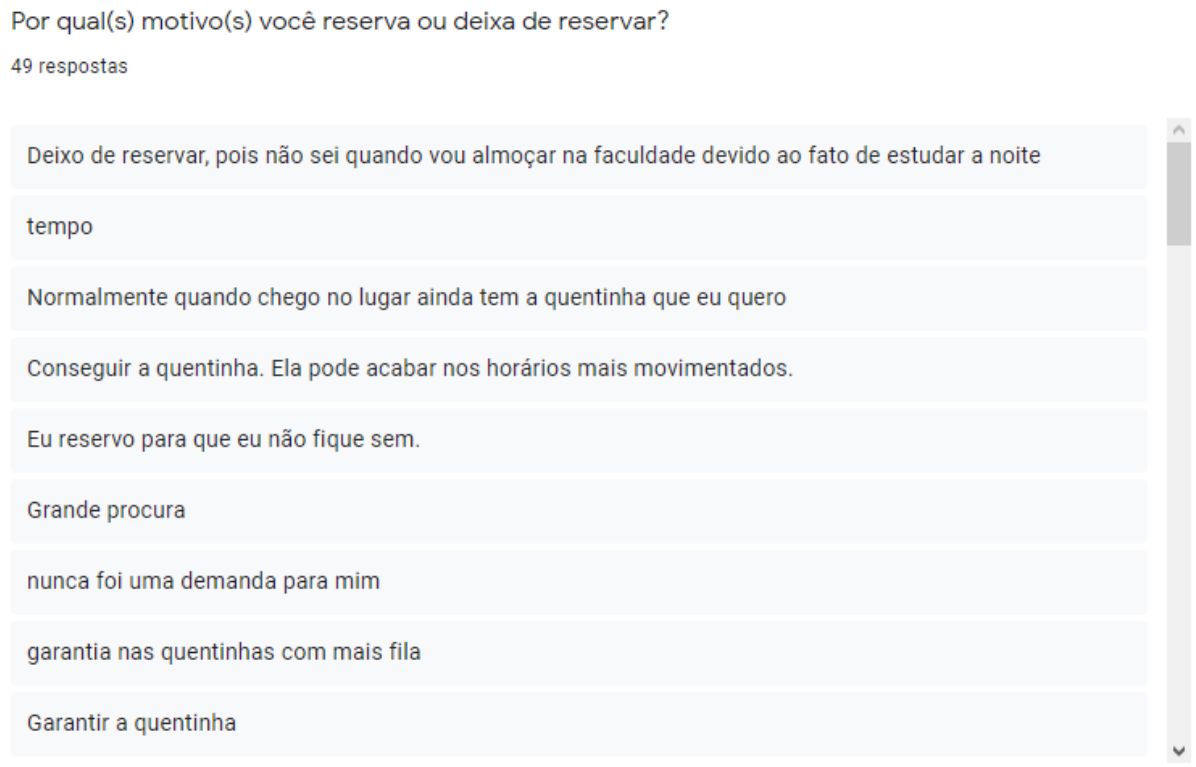


Figura C-16 - Respostas da Pergunta 10 da seção "Costumo comprar quentinhas"

Figura C-17 - Respostas da Pergunta 1 da seção "Sobre o aplicativo"

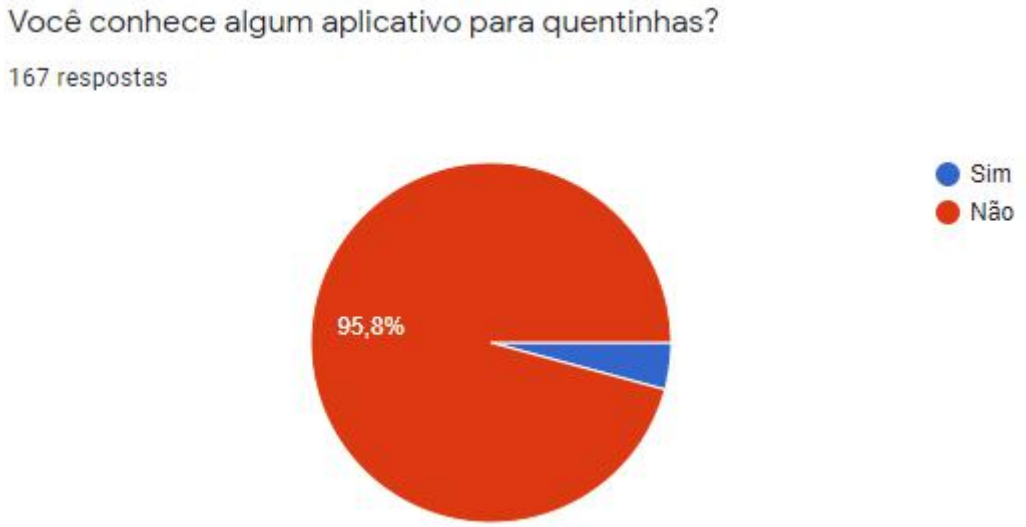


Figura C-18 - Respostas da Pergunta 2 da seção "Sobre o aplicativo"

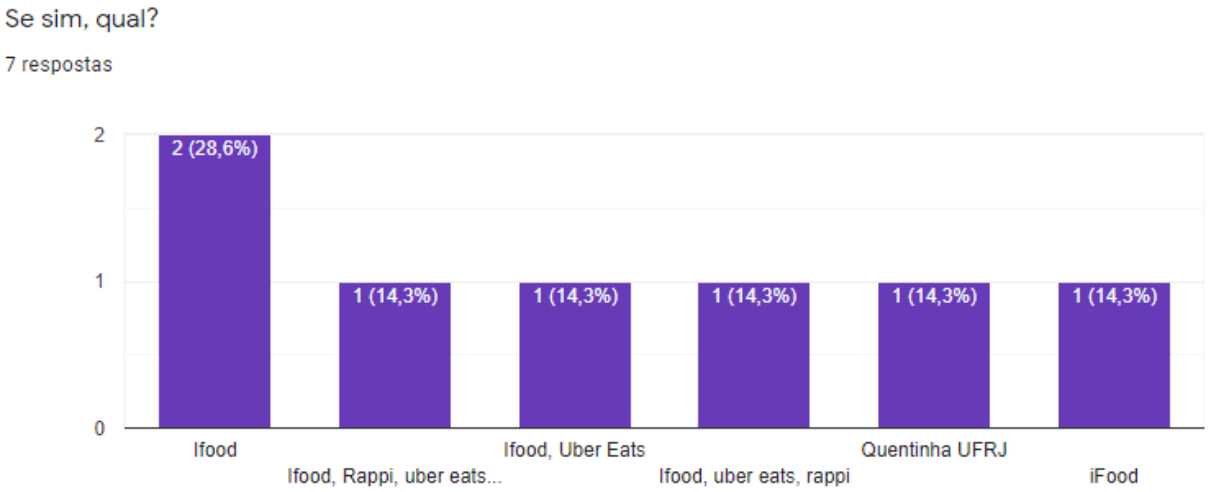


Figura C-19 - Respostas da Pergunta 3 da seção “Sobre o aplicativo”

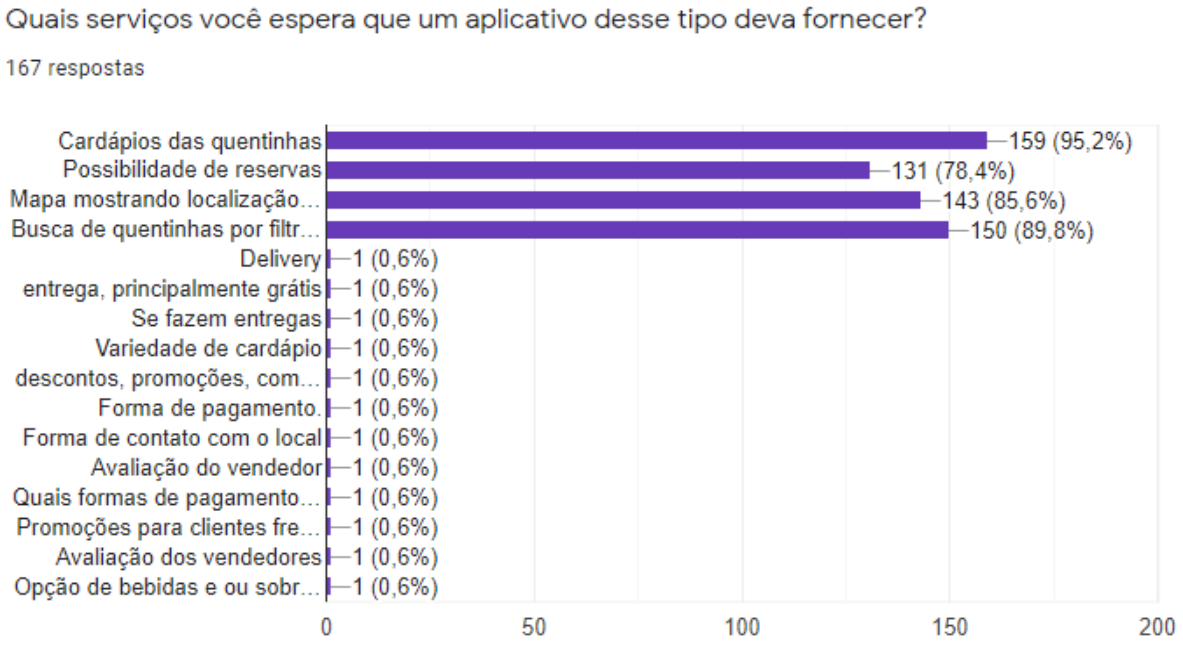
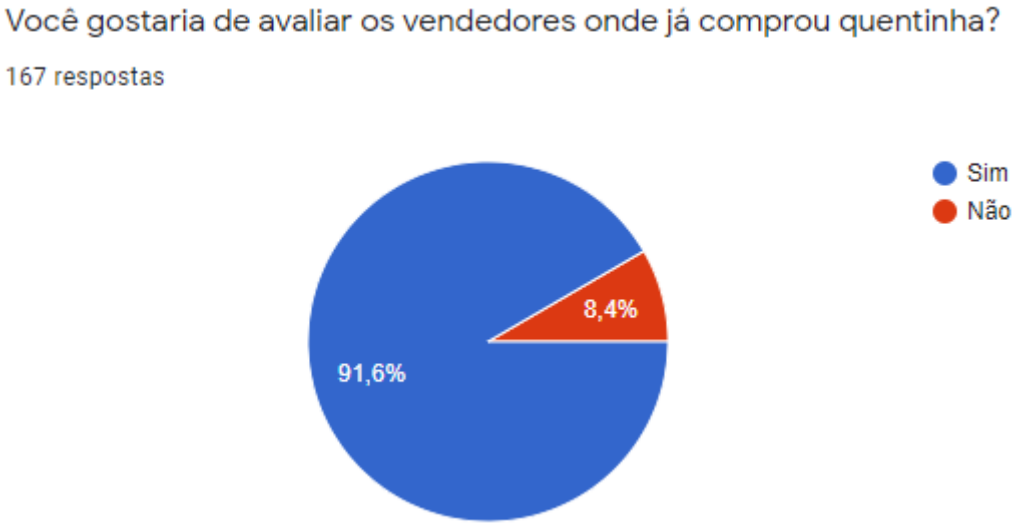


Figura C-20 - Respostas da Pergunta 4 da seção “Sobre o aplicativo”



APÊNDICE D – FORMULÁRIO DE PESQUISA DE DEMANDA DO APLICATIVO PARA VENDEDORES

1. Onde você vende suas quentinhas?
   ( ) CT
   ( ) CCMN
   ( ) Letras
   ( ) Reitoria
   ( ) CCS
   ( ) EEFD
   ( ) Parque Tecnológico
   ( ) Outro:
2. Você entra em contato de alguma forma com seus clientes?
   ( ) Sim
   ( ) Não
3. Qual a forma de contato com seus clientes?
   ( ) WhatsApp
   ( ) Facebook
   ( ) Instagram
   ( ) Celular/telefone
   ( ) Outro
4. Você realiza a reserva de quentinhas?
   ( ) Sim
   ( ) Não
5. Como é feita a reserva?
   ( ) WhatsApp
   ( ) Facebook
   ( ) Instagram
   ( ) Celular/telefone
   ( ) Outro
6. Você gostaria que as reservas fossem feitas de forma automática?
   ( ) Sim
   ( ) Não

7. Como você lida nos casos onde o cliente reserva a quentinha mas acaba não comprando?
8. Você preferiria usar algum outro meio para se comunicar e/ou realizar a reserva de quentinhas com seus clientes?
   ( ) Sim
   ( ) Não
9. Você gostaria de poder informar no aplicativo a quantidade de quentinhas disponíveis?
   ( ) Sim, mostrando a quantidade total de cada prato no dia (cliente não saberia se acabou sem entrar em contato diretamente)
   ( ) Sim, mostrando quantas quentinhas ainda tem disponível de um certo prato, para que o cliente saiba quando acabou
   ( ) Não
10. Você realiza o serviço de delivery das quentinhas?
    ( ) Sim
    ( ) Não
11. Se sim, como é feito? (se entrega na hora, se a entrega é feita a cada quantidade de pedidos, se cobra taxa extra, etc)
12. Se não, porque não faz?
13. Você usaria um aplicativo que permitisse anunciar suas quentinhas, mostrando a localização, opções de pratos, preços, etc?
    ( ) Sim
    ( ) Não
14. Você conhece algum aplicativo para quentinhas?
    ( ) Sim
    ( ) Não
15. Se sim, qual?
16. Podemos voltar a contactá-lo(a) nas próximas etapas do projeto? Se sim, por favor, nos informe seu email ou telefone de contato.

APÊNDICE E – RESULTADOS DA PESQUISA DE DEMANDA DO APLICATIVO PARA VENDEDORES

Figura E-1 - Respostas da Pergunta 1

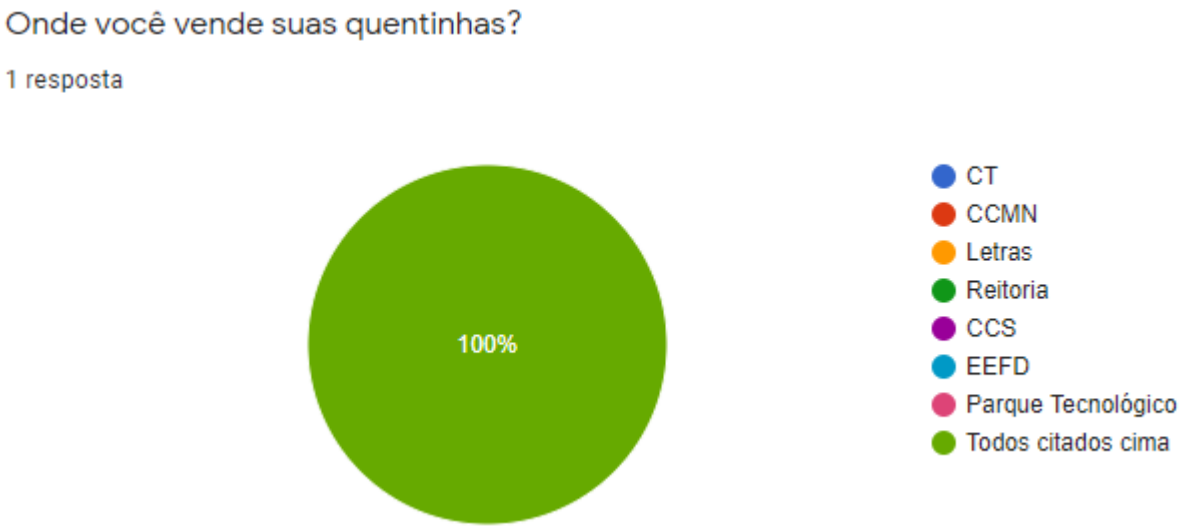


Figura E-2 - Respostas da Pergunta 2

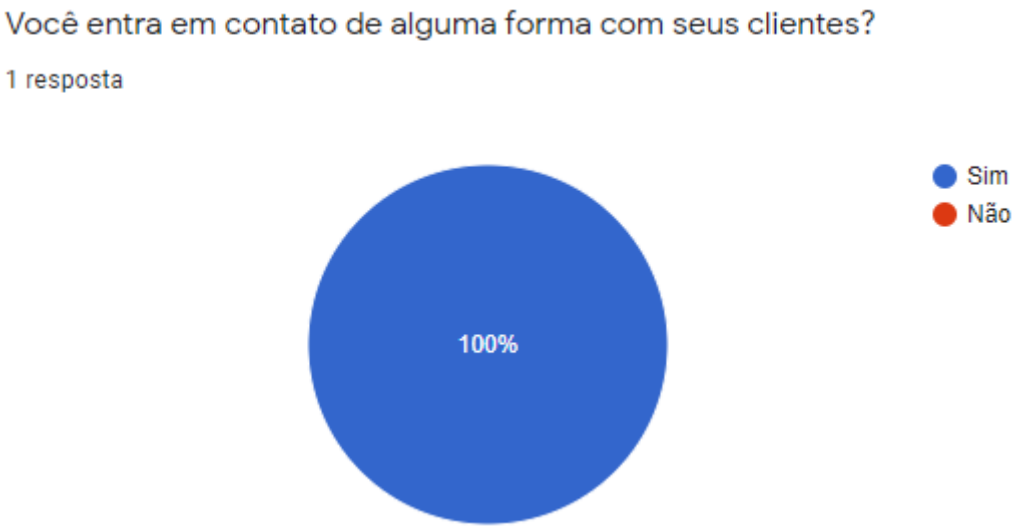


Figura E-3 - Respostas da Pergunta 3

Figura E-4 - Respostas da Pergunta 4

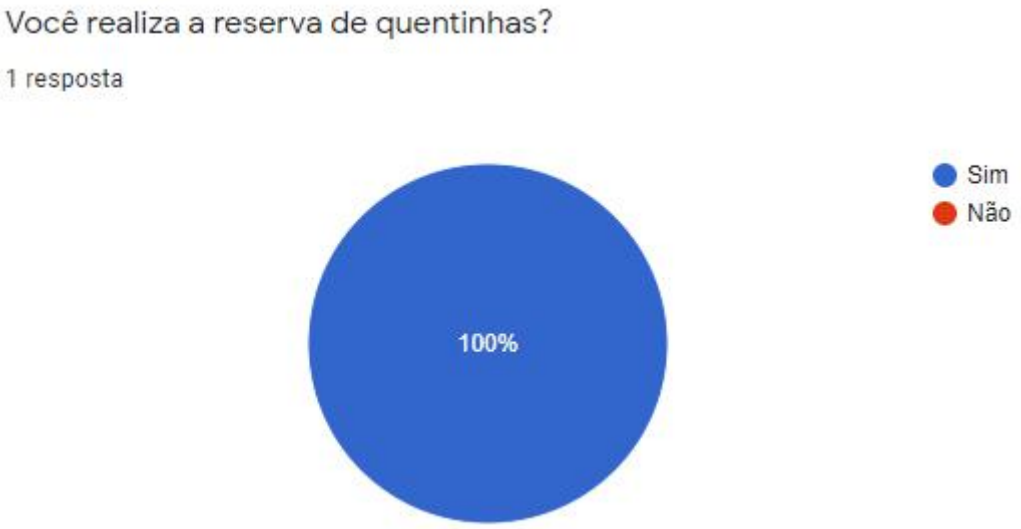


Figura E-5 - Respostas da Pergunta 5

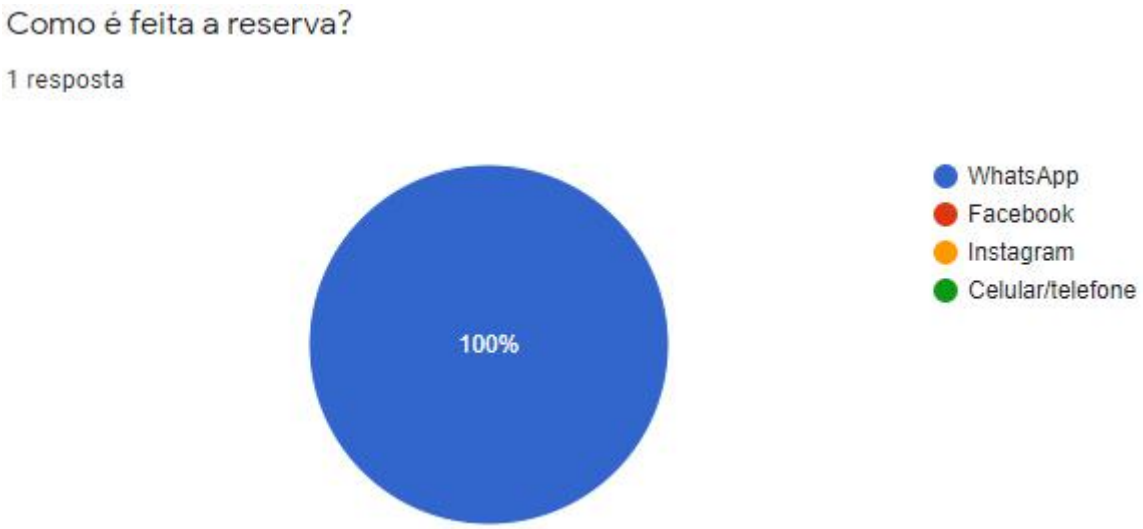


Figura E-6 - Respostas da Pergunta 6

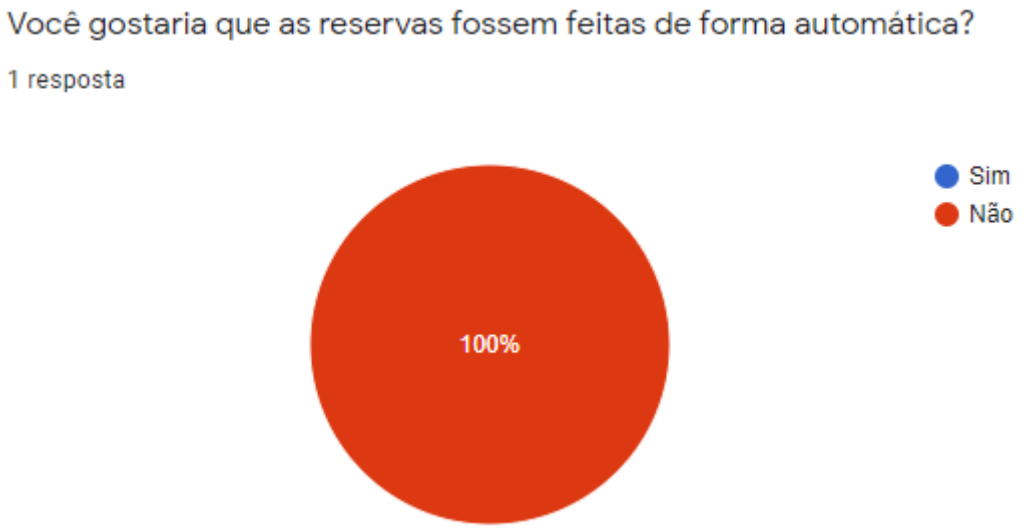


Figura E-7 - Respostas da Pergunta 7

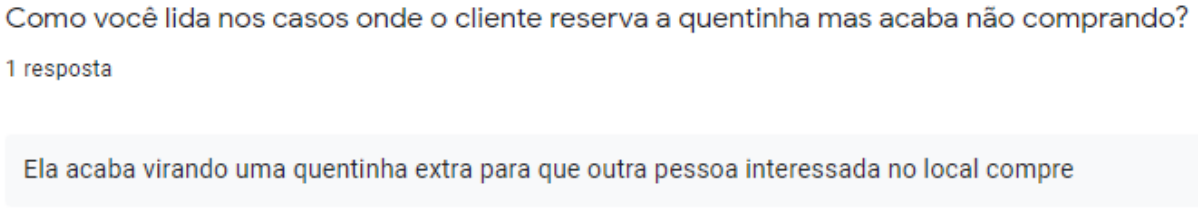


Figura E-8 - Respostas da Pergunta 8

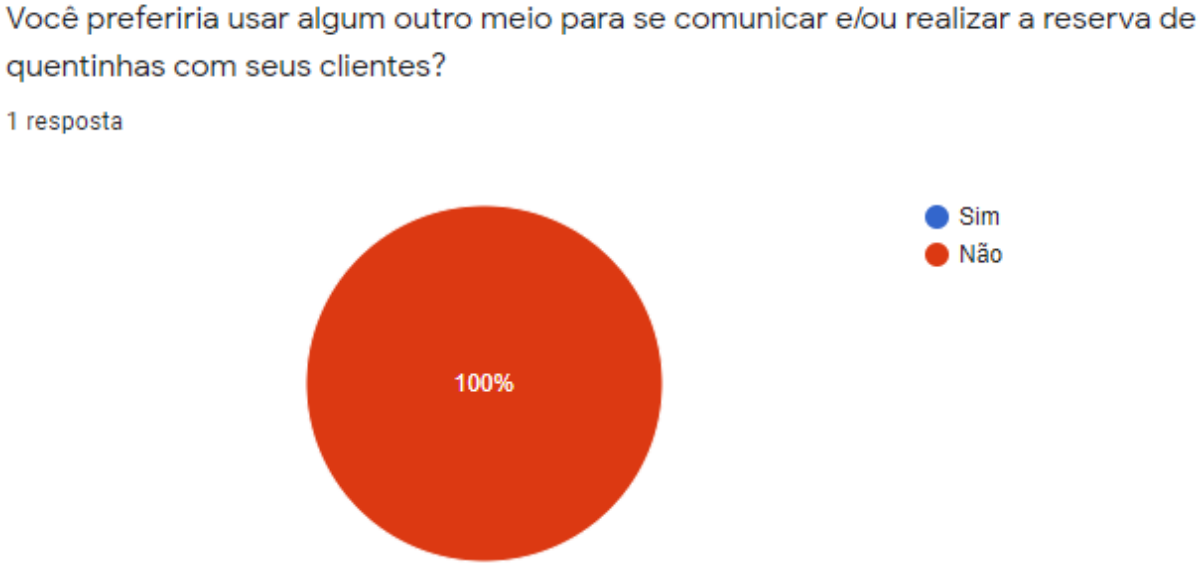


Figura E-9 - Respostas da Pergunta 9

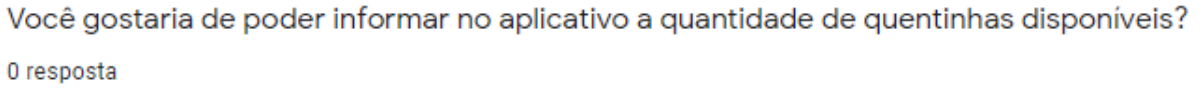


Figura E-10 - Respostas da Pergunta 10

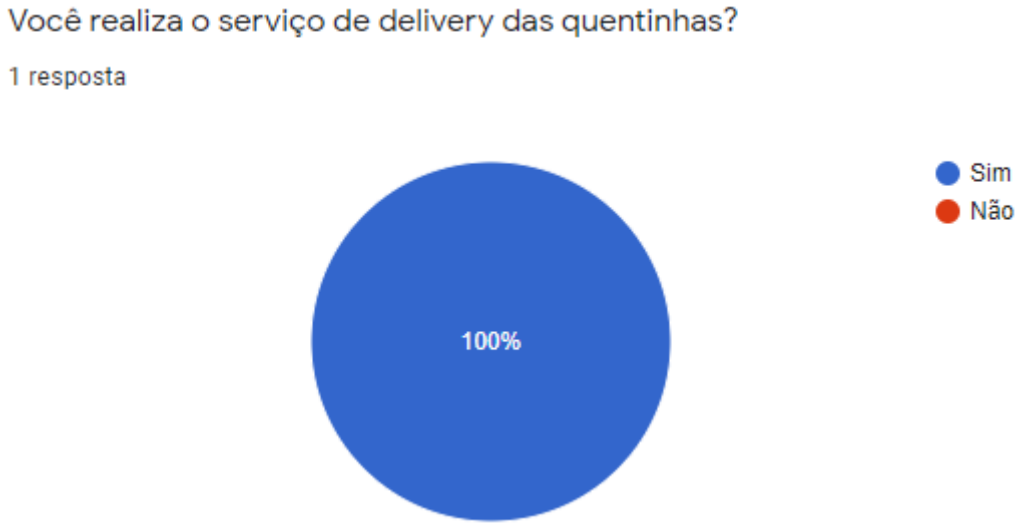


Figura E-11 - Respostas da Pergunta 11

Se sim, como é feito? (se entrega na hora, se a entrega é feita a cada quantidade de pedidos, se cobra taxa extra, etc)

1 resposta

A partir das 8:30 nós enviamos um cardapio que pode ser pedido até as 10:40 e cada local tem seu horario para chegar, seguimos uma rota e não cobramos taxa

Figura E-12 - Respostas da Pergunta 12

Se não, porque não faz?

0 resposta

Ainda não há respostas para esta pergunta.

Figura E-13 - Respostas da Pergunta 13

Você usaria um aplicativo que permitisse anunciar suas quentinhas, mostrando a localização, opções de pratos, preços, etc?

1 resposta

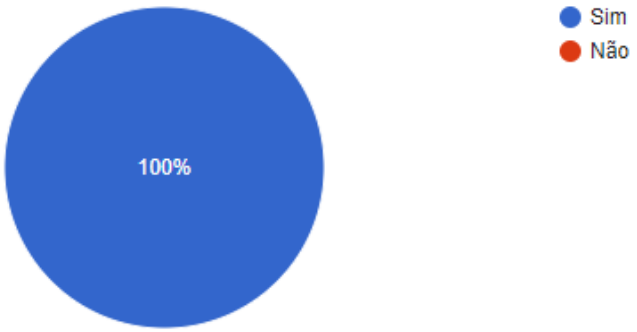


Figura E-14 - Respostas da Pergunta 14

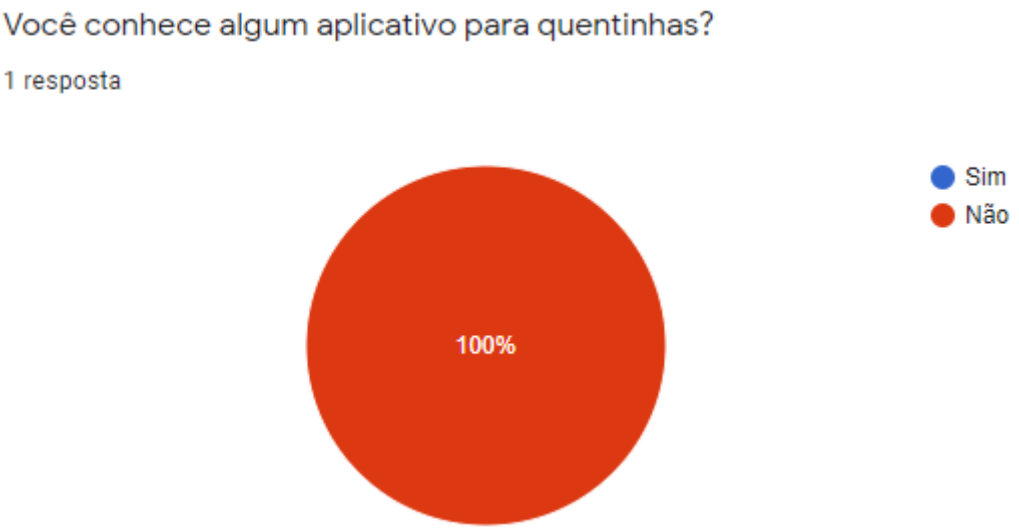
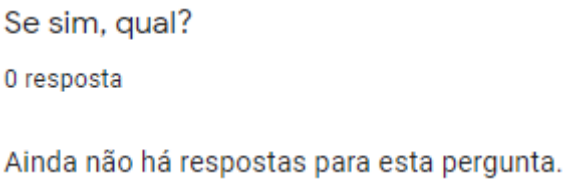


Figura E-15 - Respostas da Pergunta 15



APÊNDICE F – FORMULÁRIO DE AVALIAÇÃO DO APLICATIVO DO CLIENTE

1. Você conseguiu completar todos os testes?
   ( ) Sim
   ( ) Não
2. Se respondeu “Não” na pergunta acima, quais testes não conseguiu completar e por quê?
3. Qual nota você daria para a usabilidade do app?
   ( ) 1 (Muito ruim)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Muito boa)
4. Na sua opinião, o aplicativo Rango possui as funcionalidades necessárias para atender as demandas de um(a) comprador(a) de quentinhas?
   ( ) Sim
   ( ) Não
5. Se respondeu “Não” na pergunta anterior, qual funcionalidade você acha que falta?
6. Em uma escala de 1 a 5, você acredita que o Rango ajudaria um cliente a conhecer novos vendedores e pratos?
   ( ) 1 (Não ajudaria)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Ajudaria muito)
7. No eventual lançamento do app para o público geral, qual a probabilidade de você usá-lo?
   ( ) 1 (Pouco provável)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Muito provável)

8. Você tem algum bug para reportar? Se sim, tente descrever o que aconteceu e como podemos reproduzi-lo
9. Você tem algo mais a comentar?

APÊNDICE G – RESPOSTAS DA AVALIAÇÃO DO APLICATIVO DO CLIENTE

Figura G-1 - Respostas da Pergunta 1

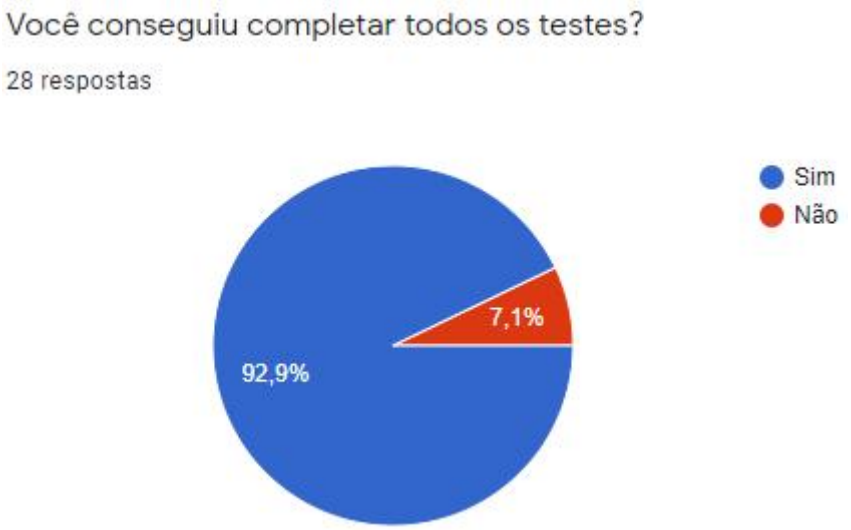


Figura G-2 - Respostas da Pergunta 2

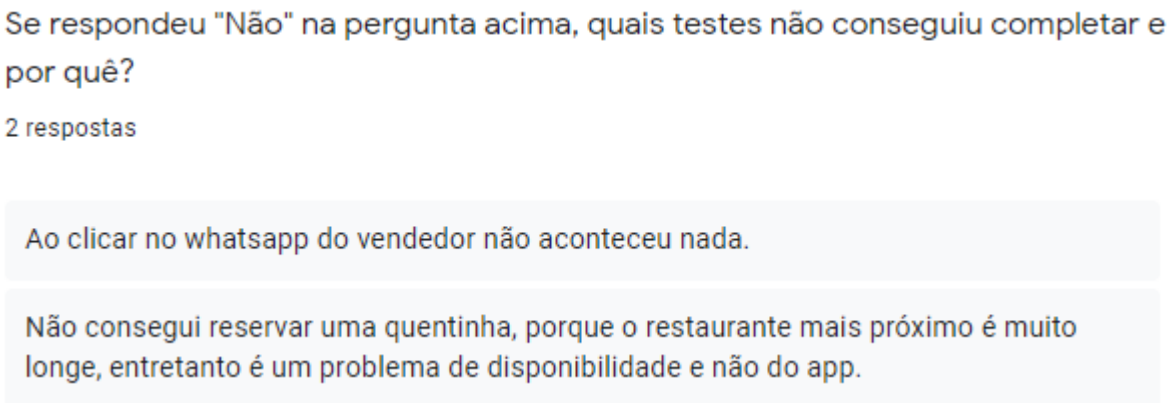


Figura G-3 - Respostas da Pergunta 3

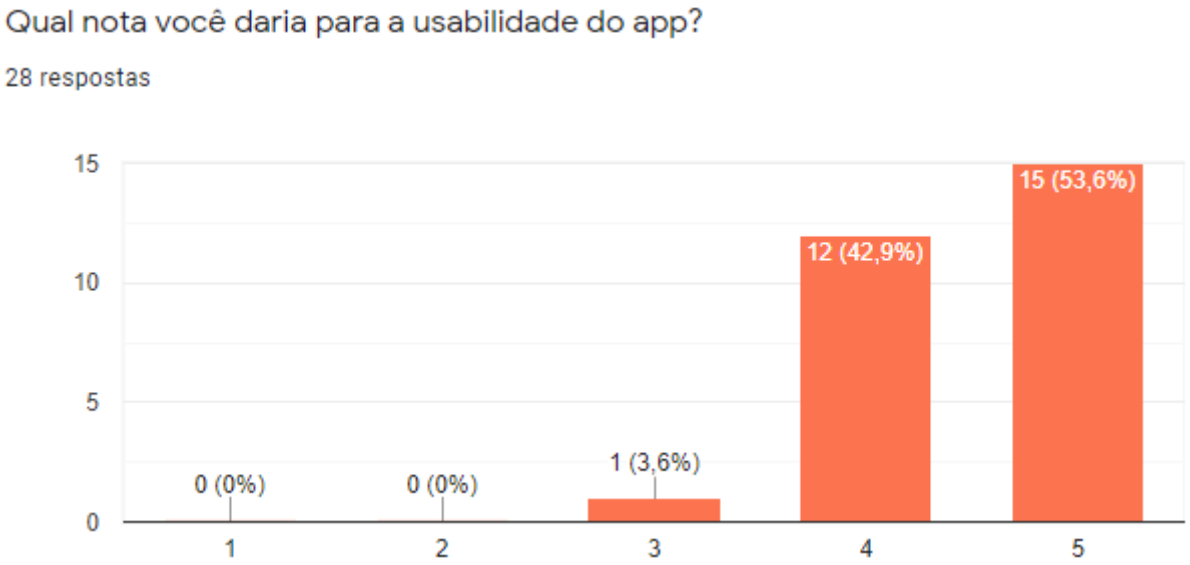


Figura G-4 - Respostas da Pergunta 4

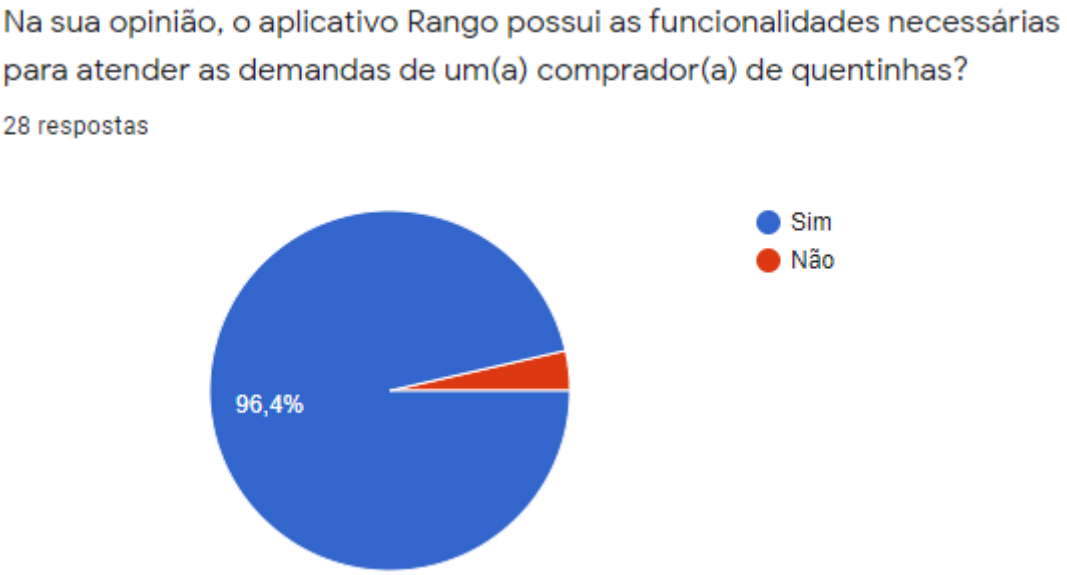


Figura G-5 - Respostas da Pergunta 5

Se respondeu "Não" na pergunta anterior, qual funcionalidade você acha que falta?

3 respostas

Falta opção.de escolher a data, e poder fazer reserva mesmo com a loja fechada.

Eu nao consigo colocar minha forma de pagamento, mesmo que seja dinheiro. Sei la vou pagar com 50 reais e quero troco.

Falta notificar que o pedido ta sendo preparado.

Faltou a busca dos restaurantes na home.

Poderia ter uma aba apenas para conhecer os restaurantes e assim dividir os pratos dos vendedores.

- Senti falta de um campo de busca para vendedor e de pratos. Talvez com filtros de distância, preços e tipos de pratos

- senti falta de um sistema de nota, para saber quais pratos são mais compras e tals.

Figura G-6 - Respostas da Pergunta 6

Em uma escala de 1 a 5, você acredita que o Rango ajudaria um cliente a conhecer novos vendedores e pratos?

28 respostas

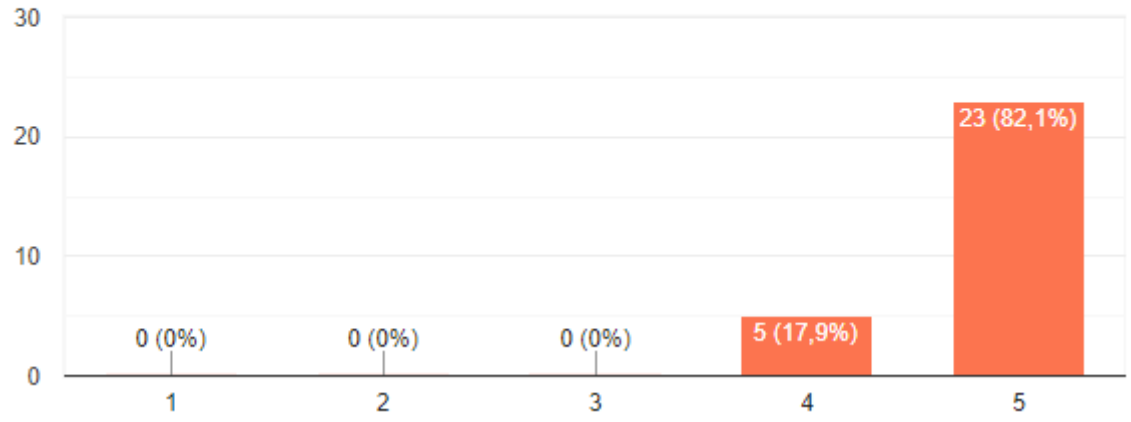


Figura G-7 - Respostas da Pergunta 7

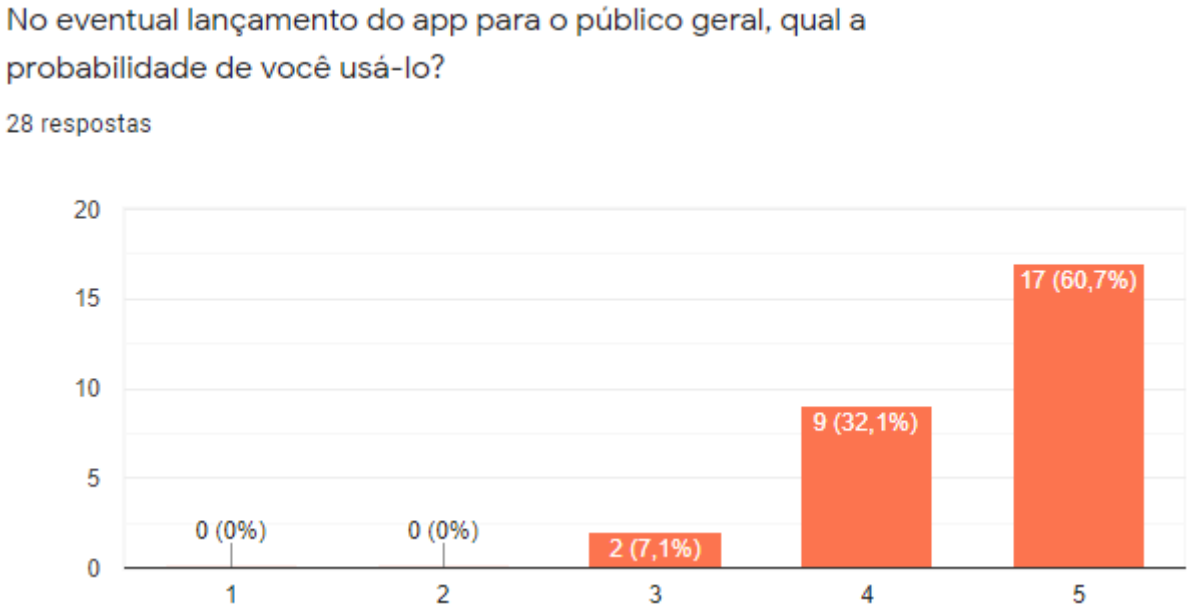


Figura G-8 - Respostas da Pergunta 8

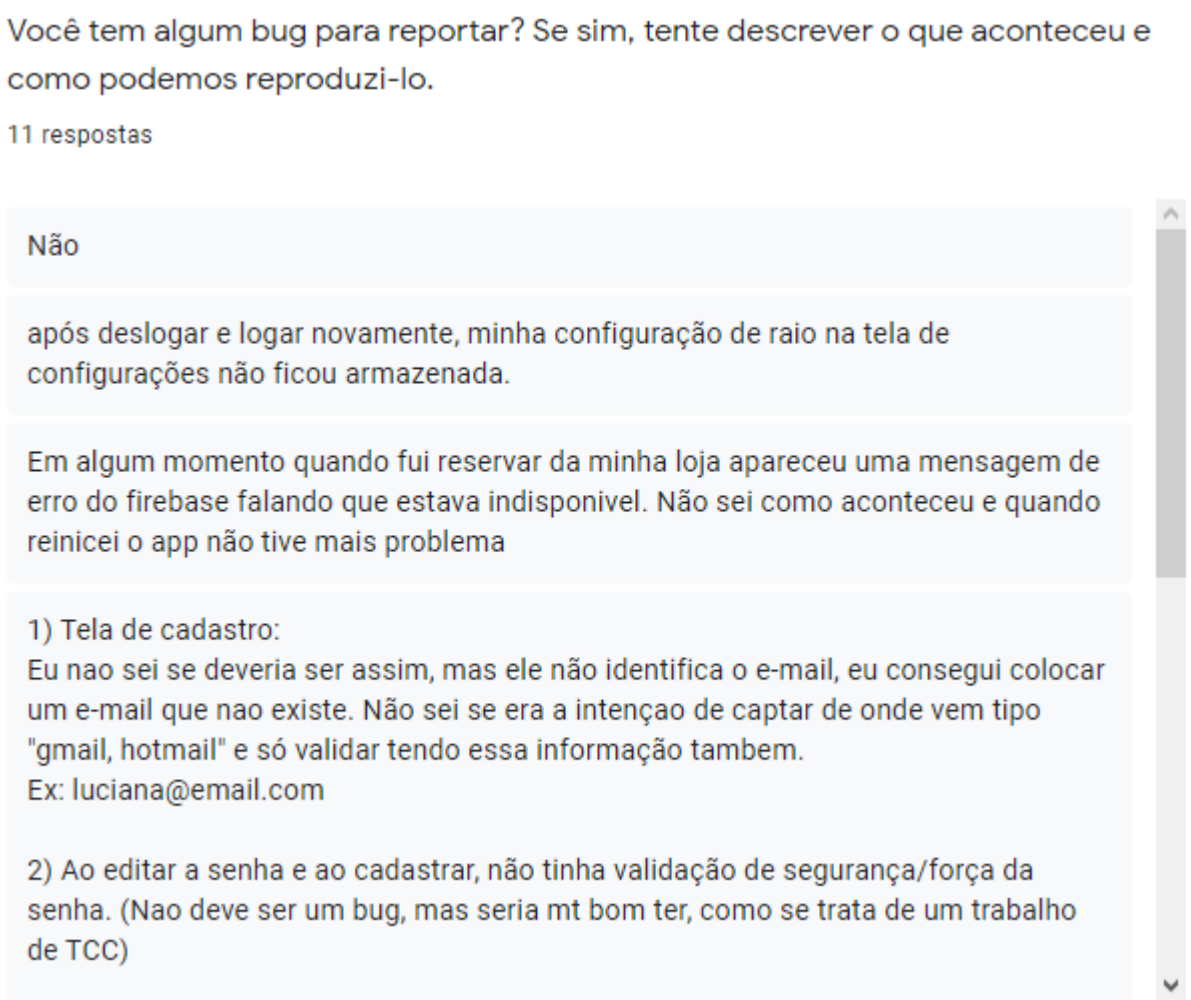
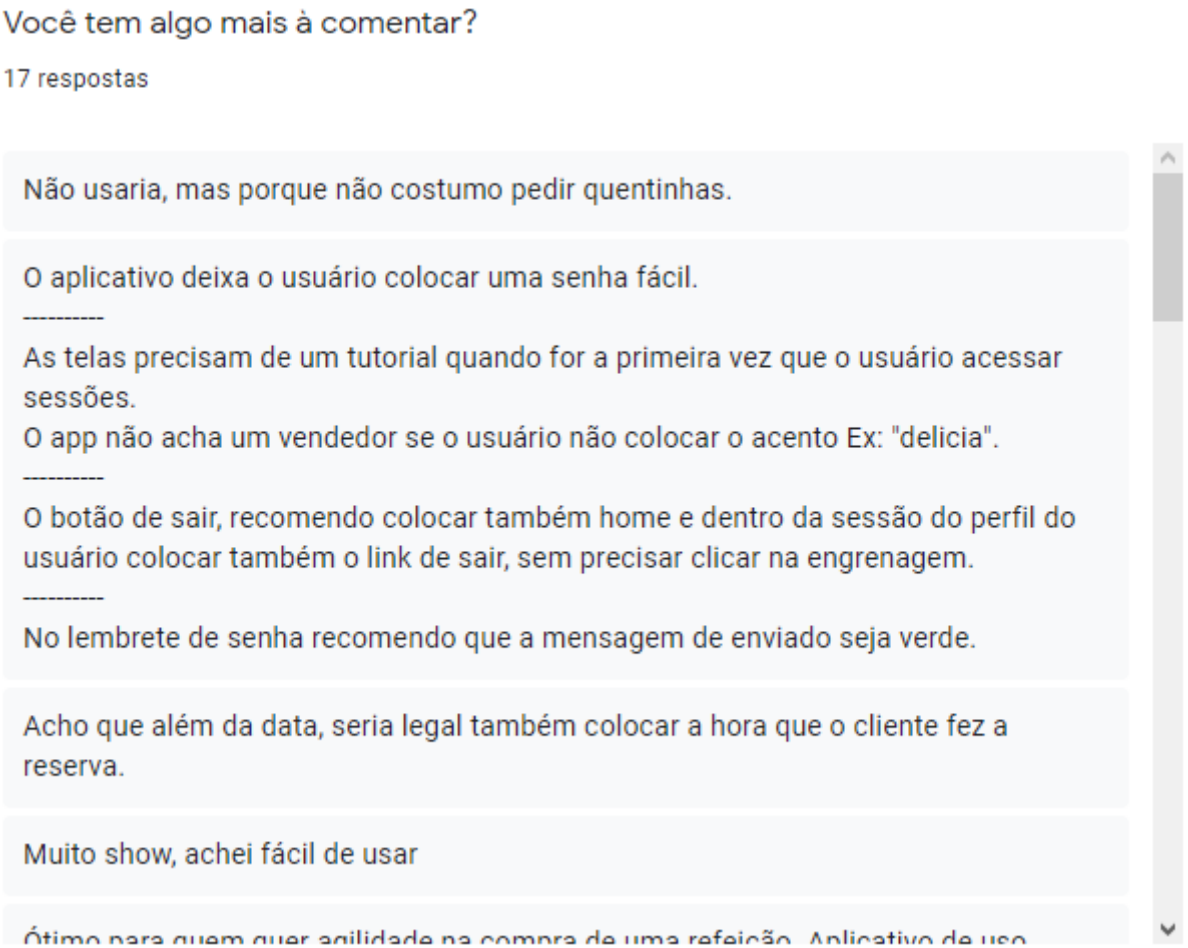


Figura G-9 - Respostas da Pergunta 9



APÊNDICE H – FORMULÁRIO DE AVALIAÇÃO DO APLICATIVO DO VENDEDOR

1. Você conseguiu completar todos os testes?
   ( ) Sim
   ( ) Não
2. Se respondeu “Não” na pergunta acima, quais testes não conseguiu completar e por quê?
3. Qual nota você daria para a usabilidade do app?
   ( ) 1 (Muito ruim)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Muito boa)
4. Na sua opinião, o aplicativo Rango possui as funcionalidades necessárias para atender as demandas de um(a) vendedor(a) de quentinhas?
   ( ) Sim
   ( ) Não
5. Se respondeu “Não” na pergunta anterior, qual funcionalidade você acha que falta?
6. Em uma escala de 1 a 5, você acredita que o Rango ajudaria um vendedor a conectar-se melhor com clientes, antigos e novos?
   ( ) 1 (Não ajudaria)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Ajudaria muito)
7. No eventual lançamento do app para o público geral, qual a probabilidade de você usá-lo?
   ( ) 1 (Pouco provável)
   ( ) 2
   ( ) 3
   ( ) 4
   ( ) 5 (Muito provável)

8. Você tem algum bug para reportar? Se sim, tente descrever o que aconteceu e como podemos reproduzi-lo
9. Você tem algo mais a comentar?

APÊNDICE I – RESPOSTAS DA AVALIAÇÃO DO APLICATIVO DO VENDEDOR

Figura I-1 - Respostas da Pergunta 1

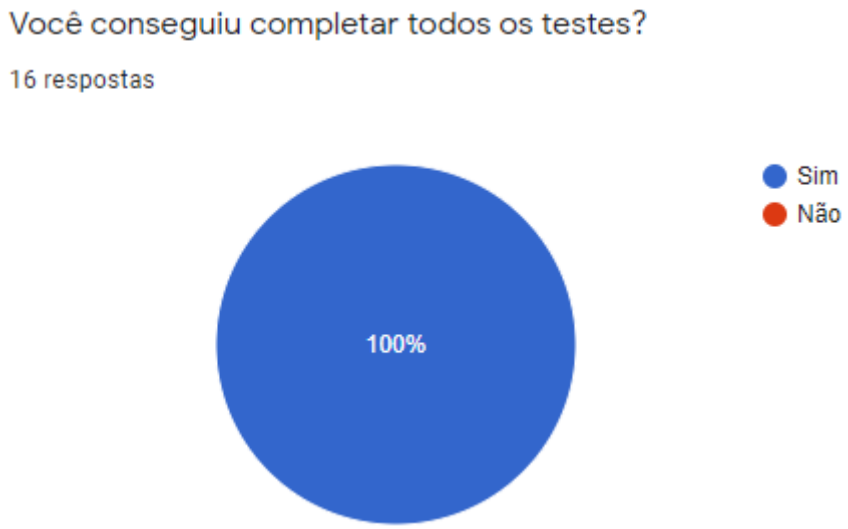


Figura I-2 - Respostas para a Pergunta 2

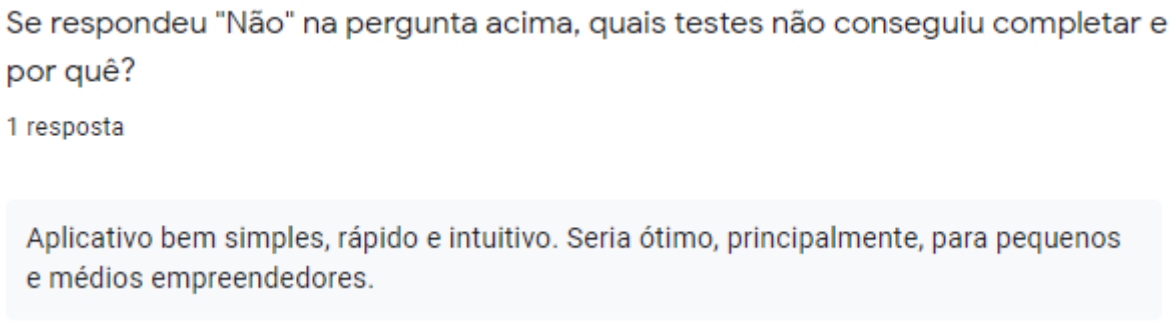


Figura I-3 - Respostas para a Pergunta 3

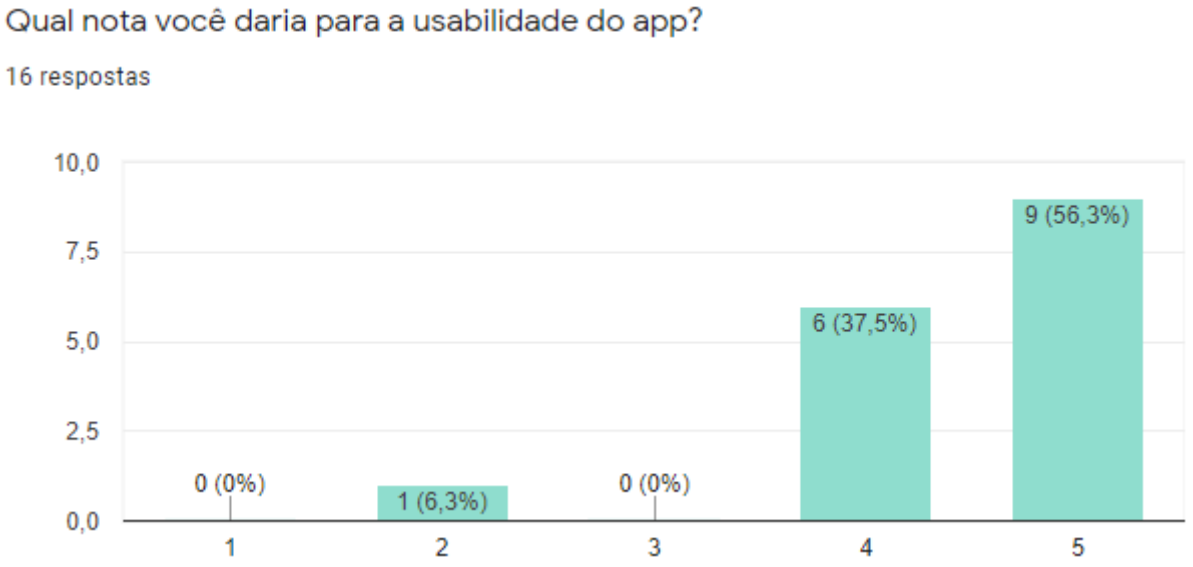


Figura I-4 - Respostas para a Pergunta 4

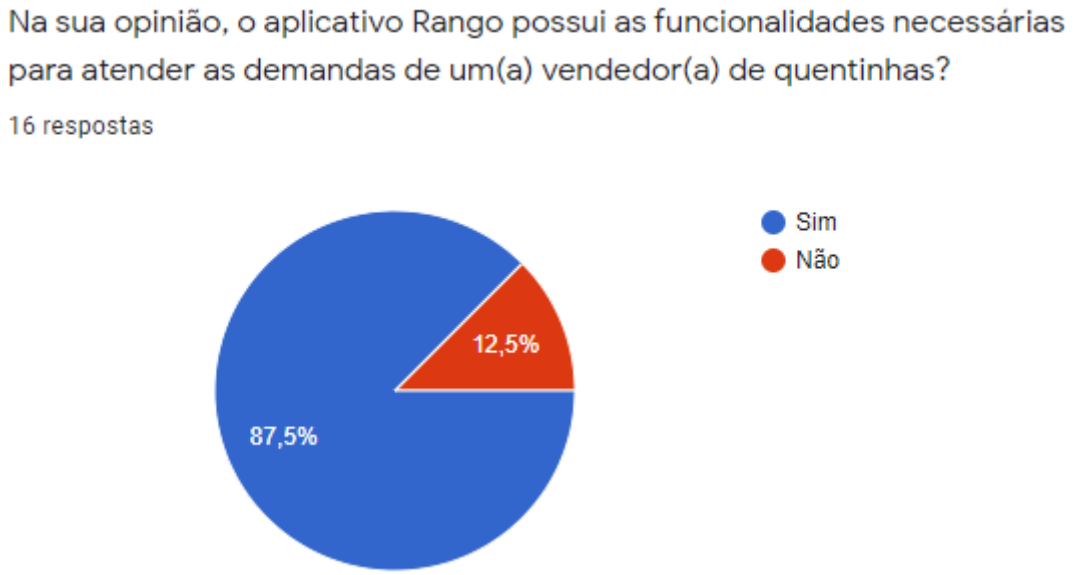


Figura I-5 - Respostas para a Pergunta 5

Se respondeu "Não" na pergunta anterior, qual funcionalidade você acha que falta?

3 respostas

Na verdade não sei dizer, pois não trabalho na área então não sei o que é necessário.

- Acho que seria legal se ao definir o horário de funcionamento fosse possível preencher todos os campos de uma só vez. E não todos os dias um por um.

Respondi "sim", mas acho que uma feature extra seria uma opção de adicionar comentários no momento que o cliente faz o pedido. Sei que dá pra fazer isso pelo chat, mas seria interessante que qualquer observação já "subisse" com o pedido.

Figura I-6 - Respostas para a Pergunta 6

Em uma escala de 1 a 5, você acredita que o Rango ajudaria um vendedor a conectar-se melhor com clientes, antigos e novos?

16 respostas

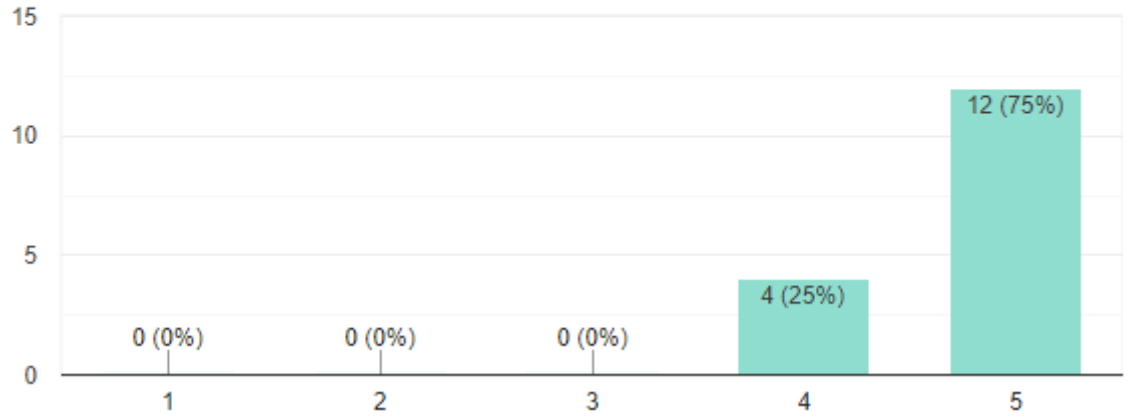


Figura I-7 - Respostas para a Pergunta 7

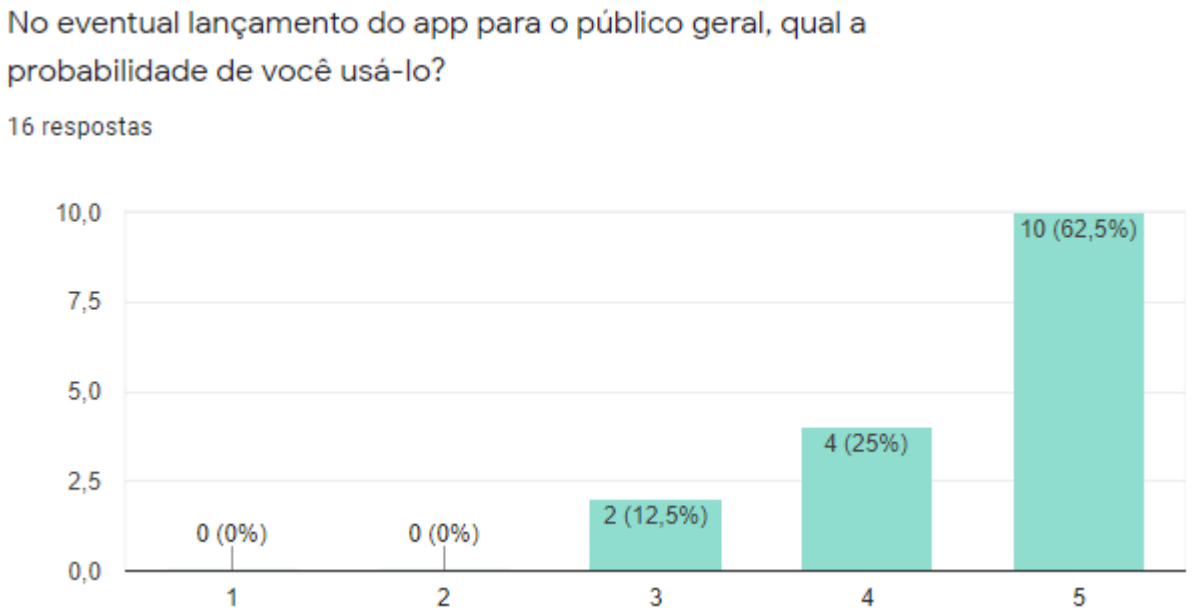


Figura I-8 - Respostas para a Pergunta 8

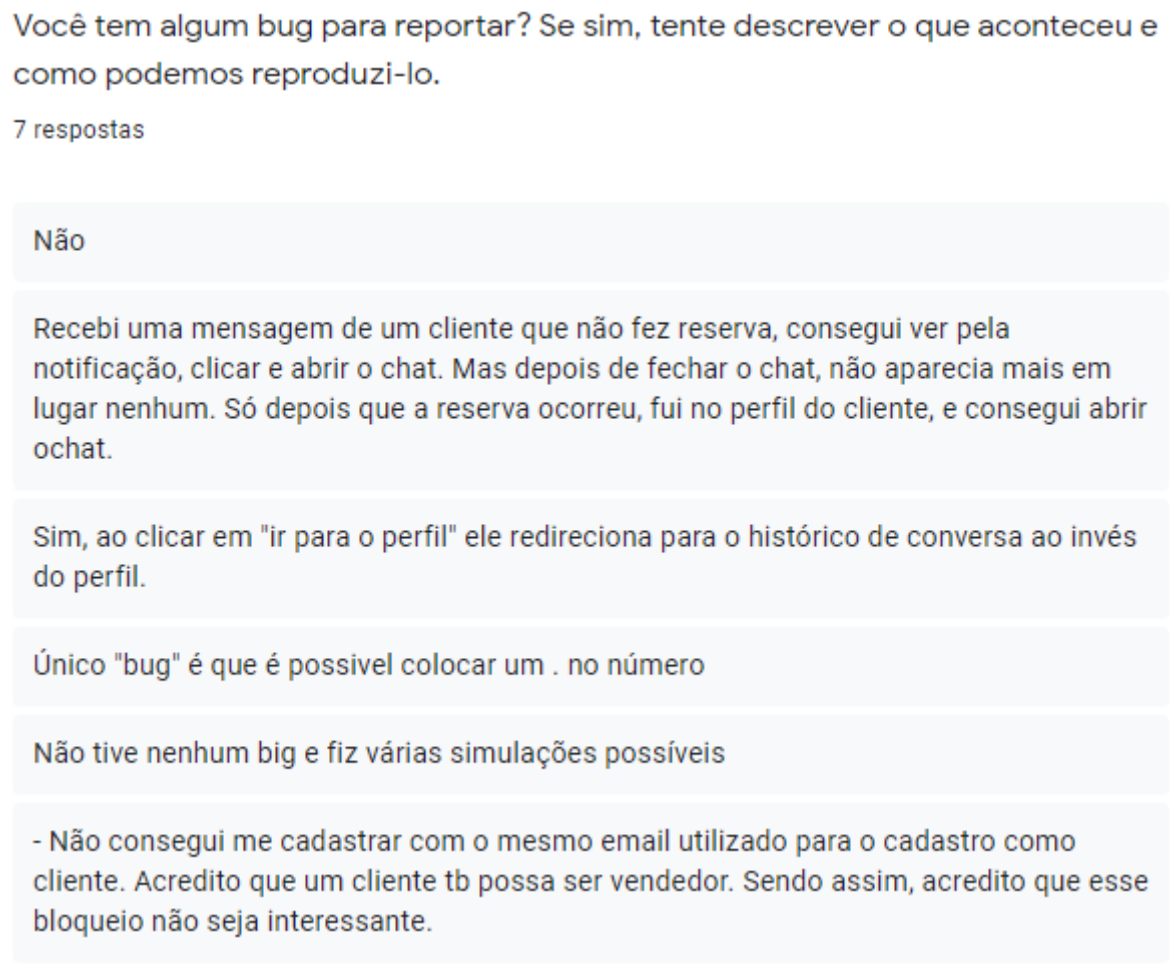
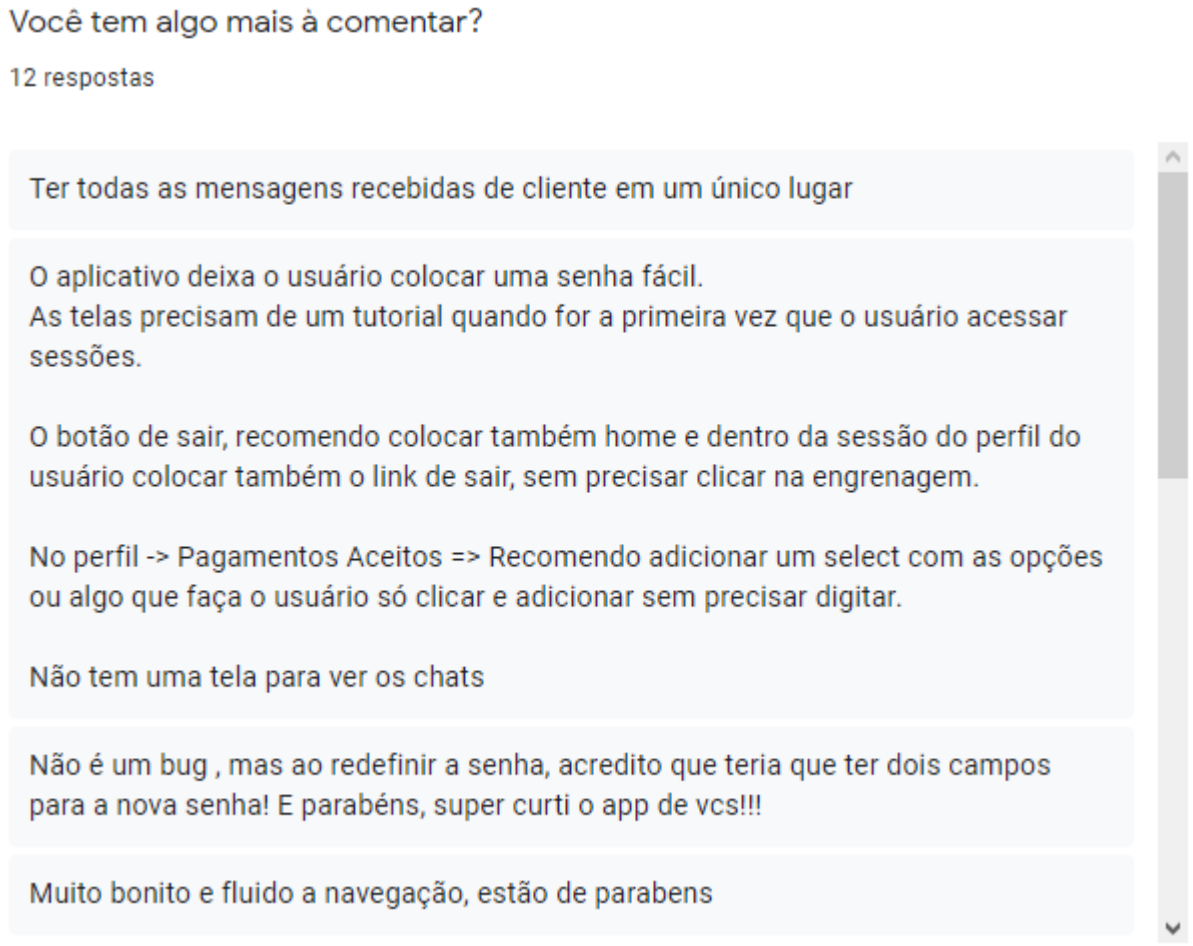


Figura I-9 - Respostas para a Pergunta 9



APÊNDICE J – GERENCIAMENTO DO PROJETO

De forma geral, a organização e comunicação foi feita principalmente através de troca de mensagens no aplicativo Telegram, onde tivemos discussões sobre a implementação da aplicação, problemas que tivemos e etc. Além disso, realizamos reuniões online a cada duas semanas via Google Meet (GOOGLE, 2021j) nas quais eram feitas discussões mais aprofundadas e também mostrada a evolução do projeto.

Abaixo, listamos algumas ferramentas que foram utilizadas para a organização e desenvolvimento desse projeto.

Controle de Versão

Utilizamos o GitHub (GITHUB, 2021) que é uma plataforma para hospedar códigos-fonte e arquivos usando o controle de versão Git. Para facilitar a publicação de alterações, foi utilizado o Git Kraken (AXOSOFT, 2021) que é uma interface gráfica que permite operar o Git de forma mais simples e intuitiva.

Quando se tem mais de uma pessoa na equipe, um controlador de versão se torna fundamental para que as alterações feitas, sejam visualizadas para ambos os membros da equipe.

Como foram desenvolvidos dois aplicativos, uma forma de organizar foi a de criarmos duas *branches* separadas, no mesmo repositório. *Branches* são ramificações no código fonte e podem ser modificadas de forma independente. Nomeamos a branch "*master*" como a ramificação onde o código fonte do app do cliente é armazenado, e a *branch* "*master-vendedor*" é a ramificação que armazenamos o código do aplicativo do vendedor. Como tivemos três pessoas distintas trabalhando ao mesmo tempo no projeto, cada uma criou a sua própria *branch* para evitar conflitos de código.

Trello

O Trello (ATLASSIAN, 2021) é uma ferramenta de gerenciamento de projetos, organizada por listas que em cada uma delas possuem cartões que representam as tarefas. Nele criamos a lista "*To Do*" que continha as tarefas gerais a serem feitas, fora do âmbito dos aplicativos, como listar tecnologias usadas, listar requisitos, entre outros. Criamos também

duas listas de tarefas para cada um dos aplicativos: Uma de tarefas a serem feitas e uma de tarefas sendo feitas. Por fim, também tínhamos uma lista "*Done*" com todas as tarefas finalizadas.

Google Drive

O Google Drive (GOOGLE, 2021i) foi o serviço escolhido para armazenamento de arquivos em nuvem, de forma que foi criada uma pasta compartilhada, onde todos tinham fácil acesso. Nela foram armazenadas as pesquisas, seus resultados, diagramas, a própria monografia e outros documentos.

APÊNDICE K – TELAS DO APLICATIVO DO CLIENTE

Figura K-1 - Tela inicial (esquerda) e de login (direita)

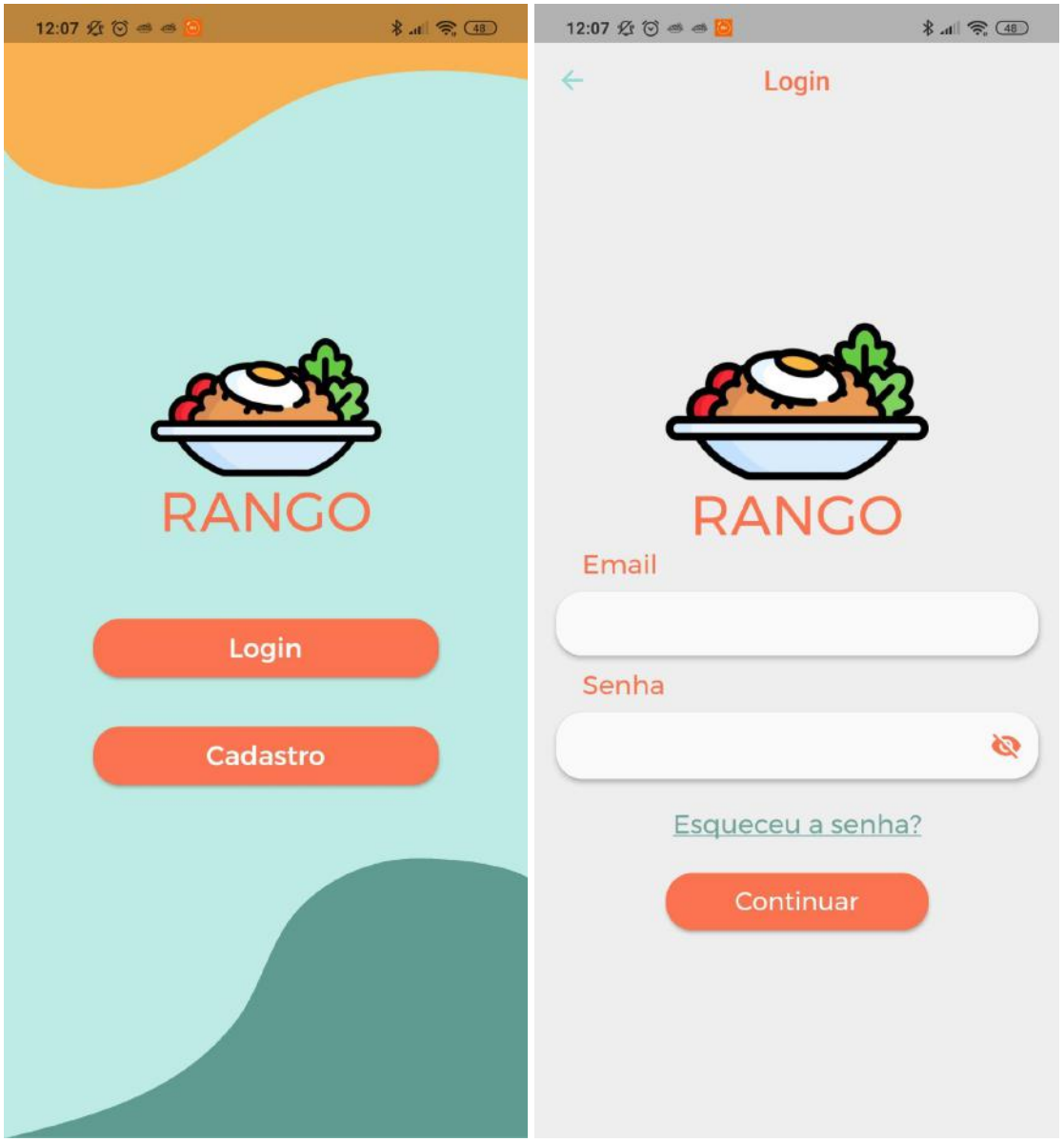


Figura K-2 - Tela de cadastro (esquerda) e de recuperação de senha (direita)

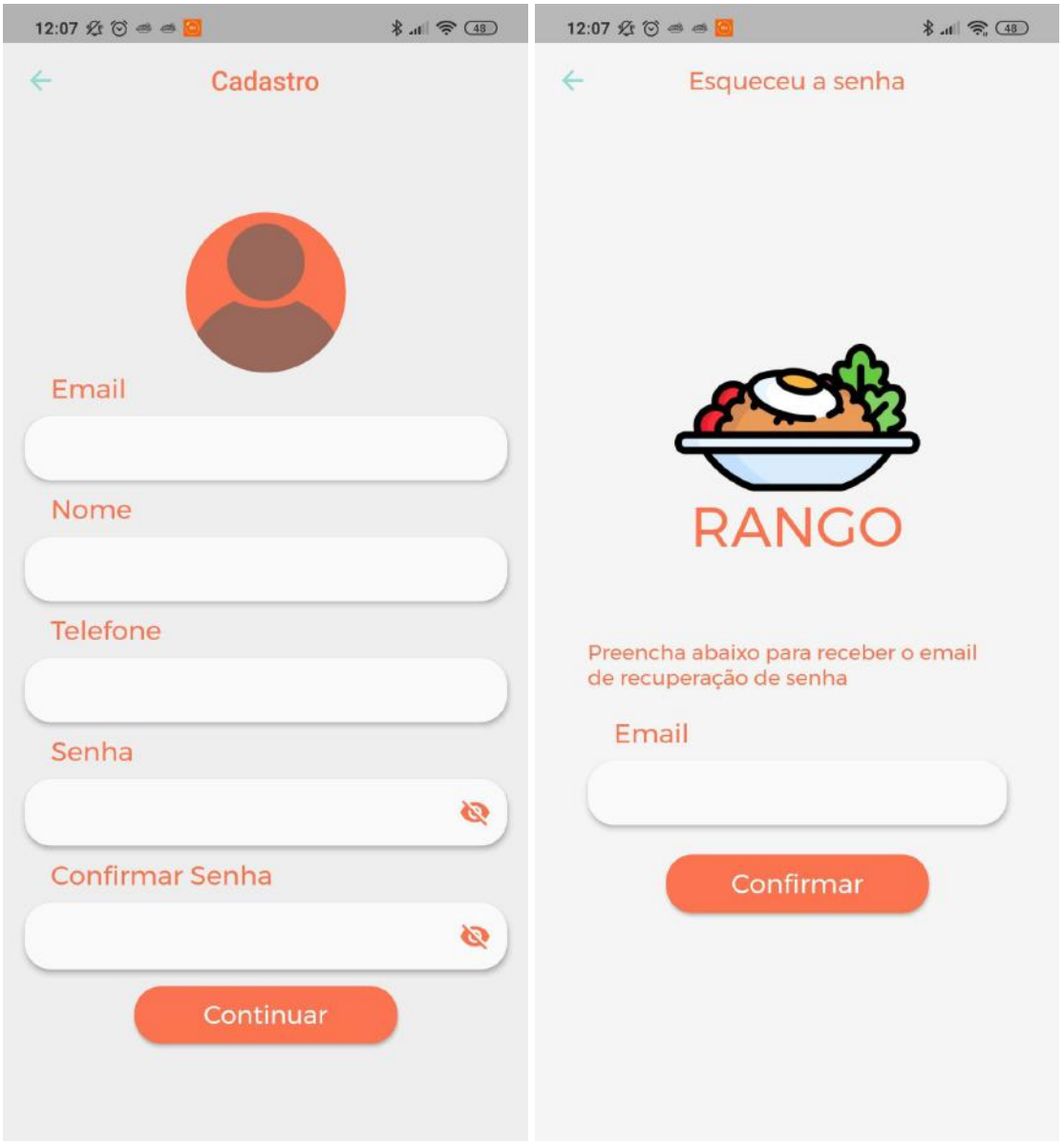


Figura K-3 - Tela de quentinhas e vendedores (esquerda) e de detalhes de quentinha (direita)

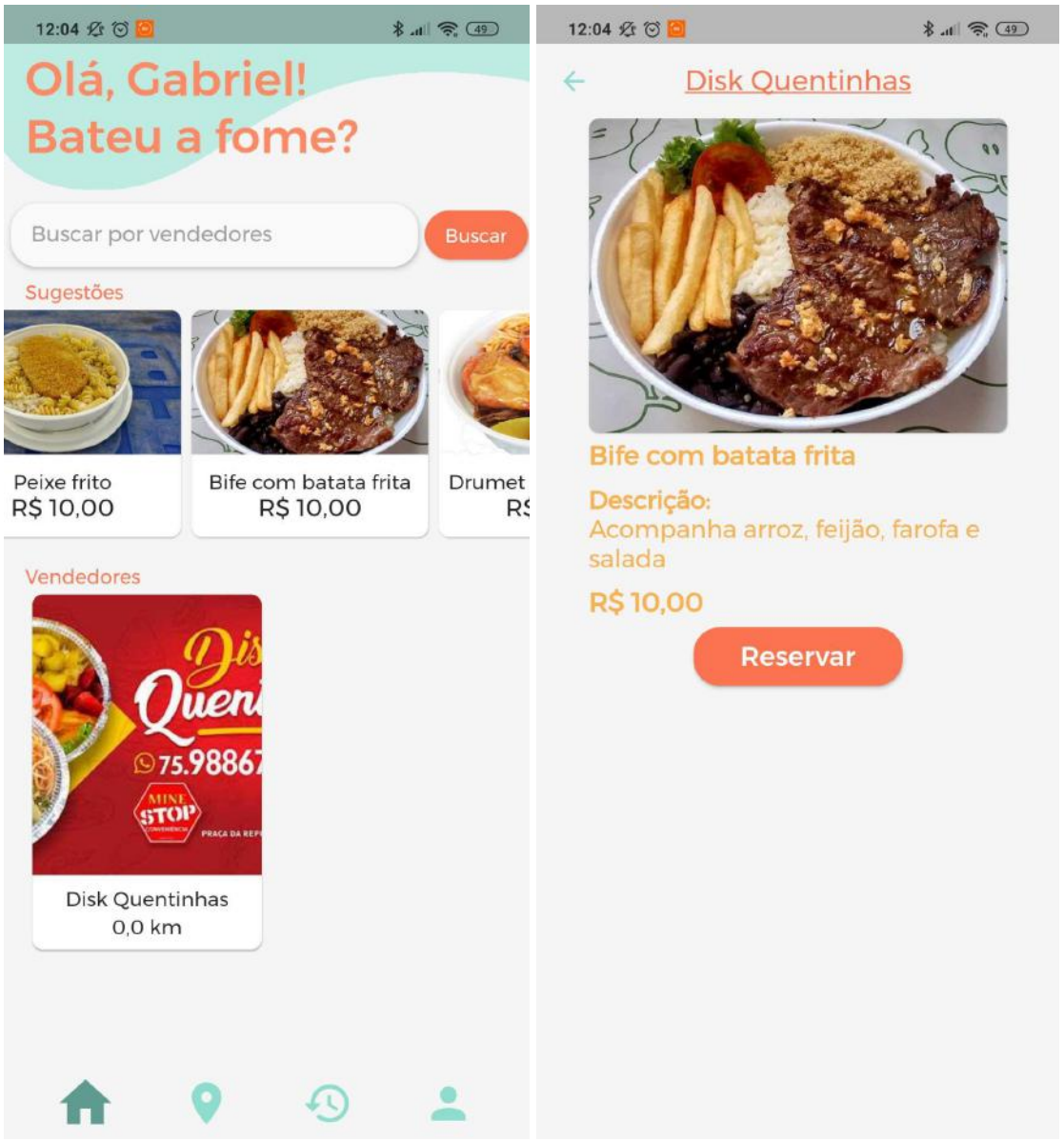


Figura K-4 - Tela de confirmação de reserva (esquerda) e de detalhes do vendedor (direita)

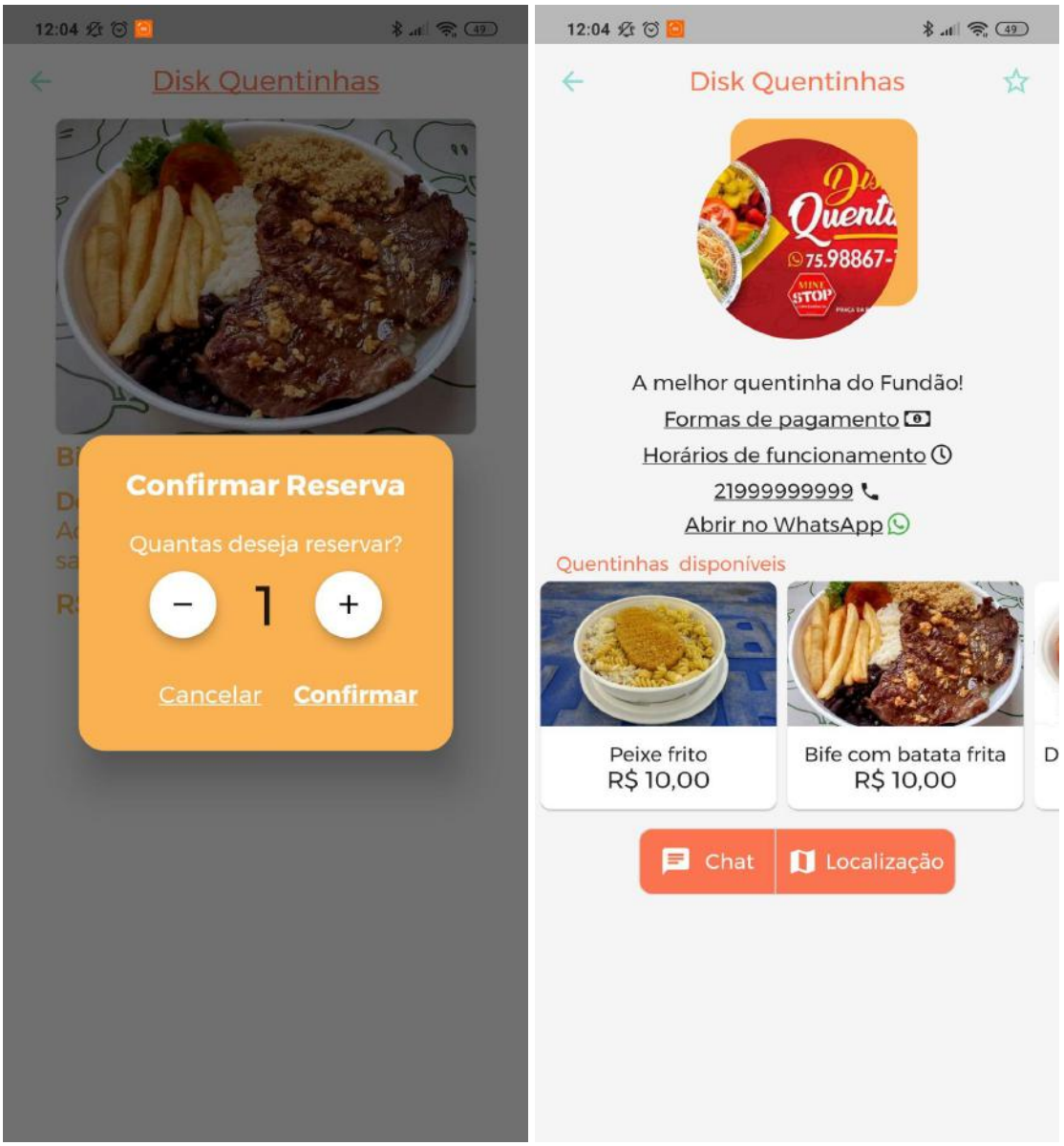


Figura K-5 - Formas de pagamento do vendedor (esquerda) e horários de funcionamento (direita)

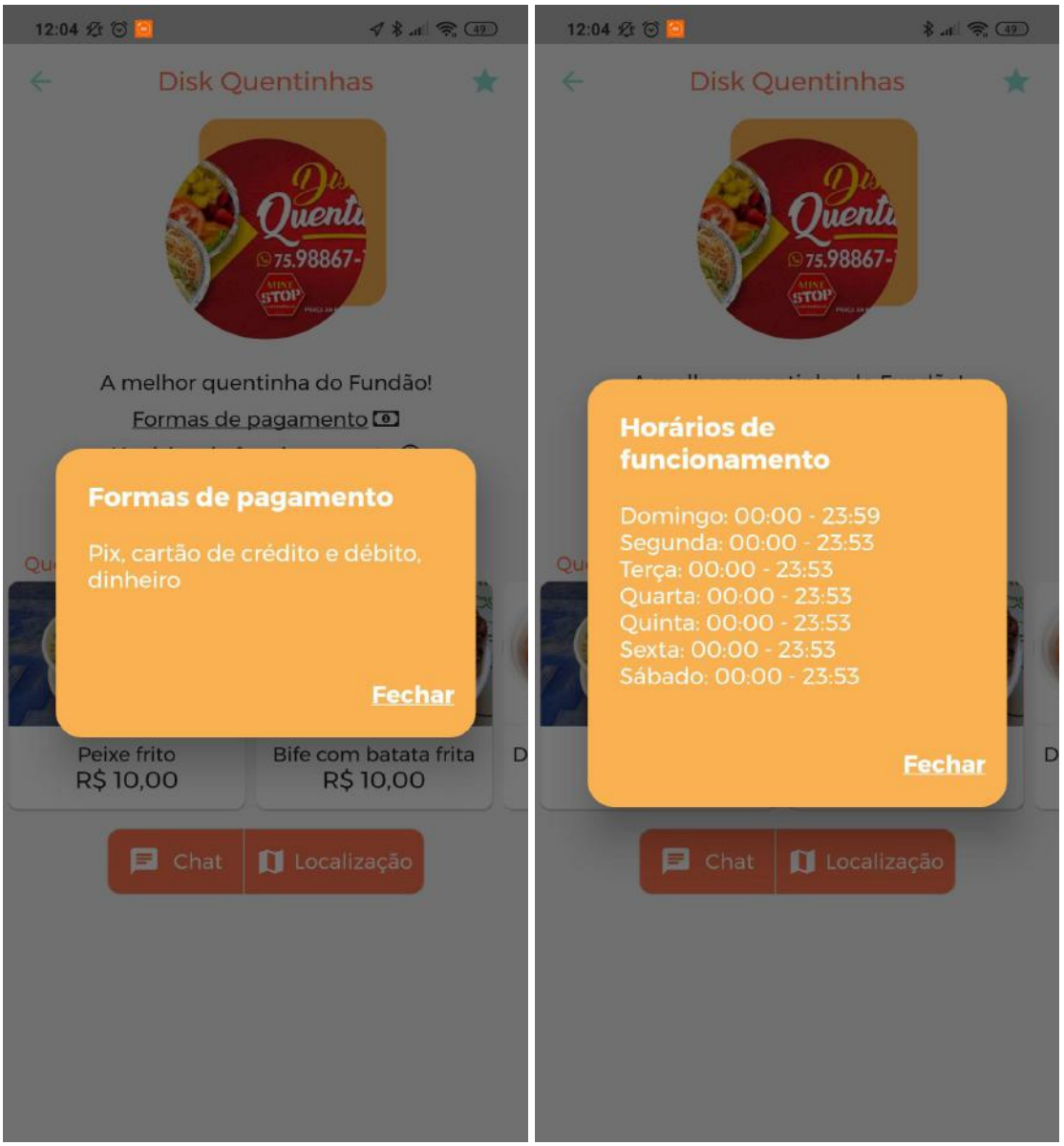


Figura K-6 - Tela de conversa com vendedor (esquerda) e tela do mapa (direita)

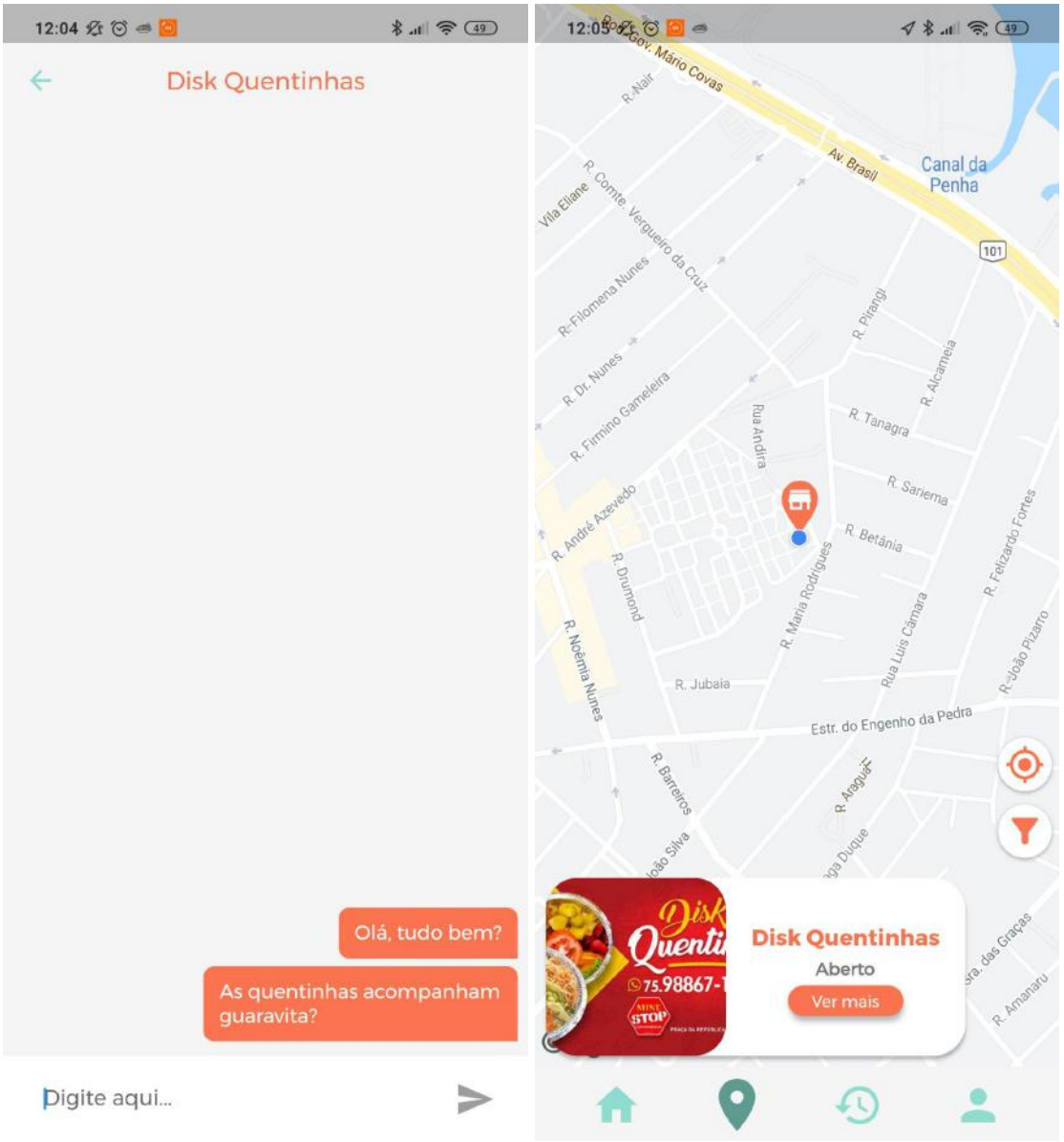


Figura K-7 - Tela de filtro do mapa (esquerda) e de histórico de pedidos (direita)

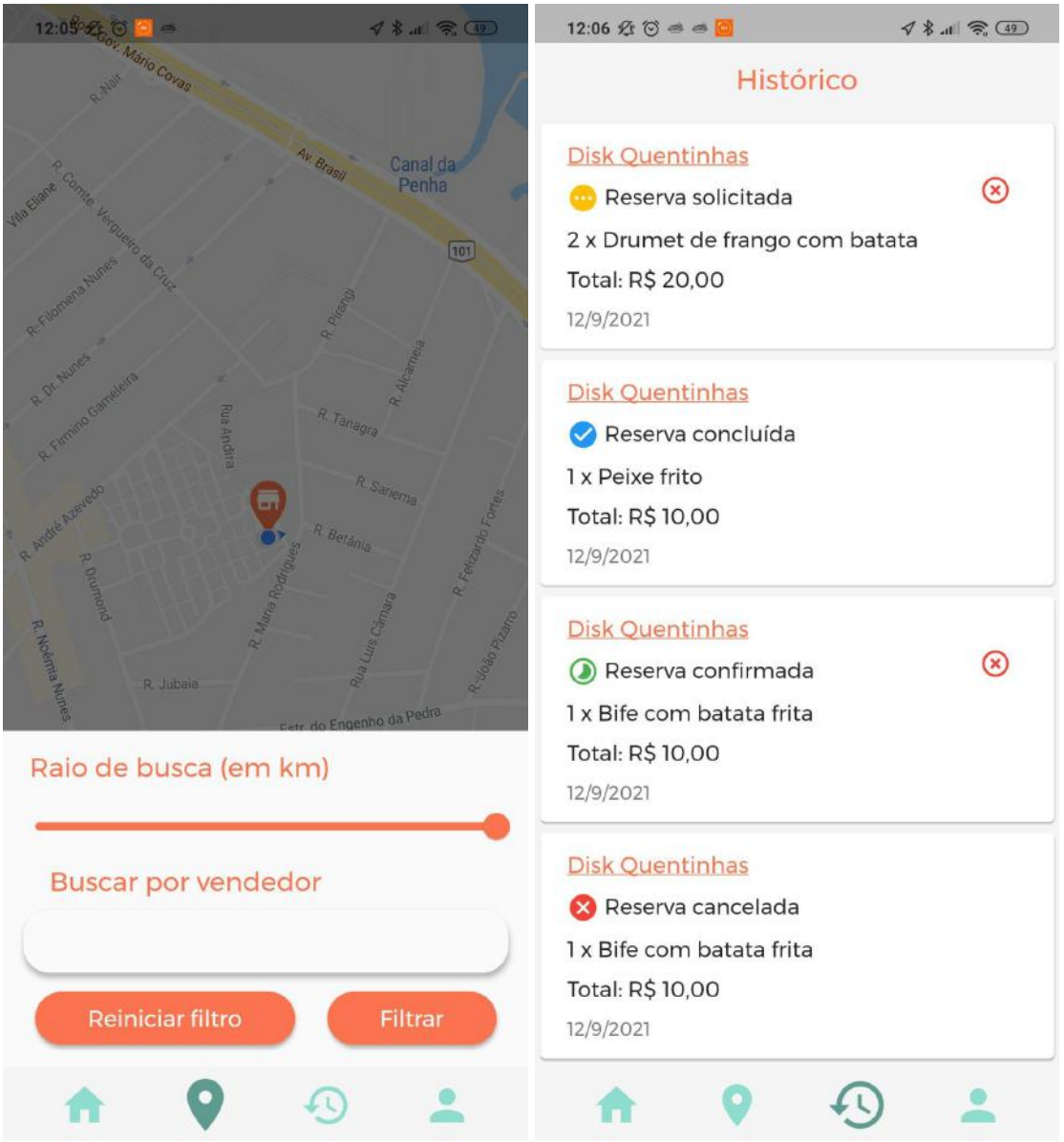


Figura K-8 - Tela de perfil do cliente (esquerda) e de edição de perfil (direita)

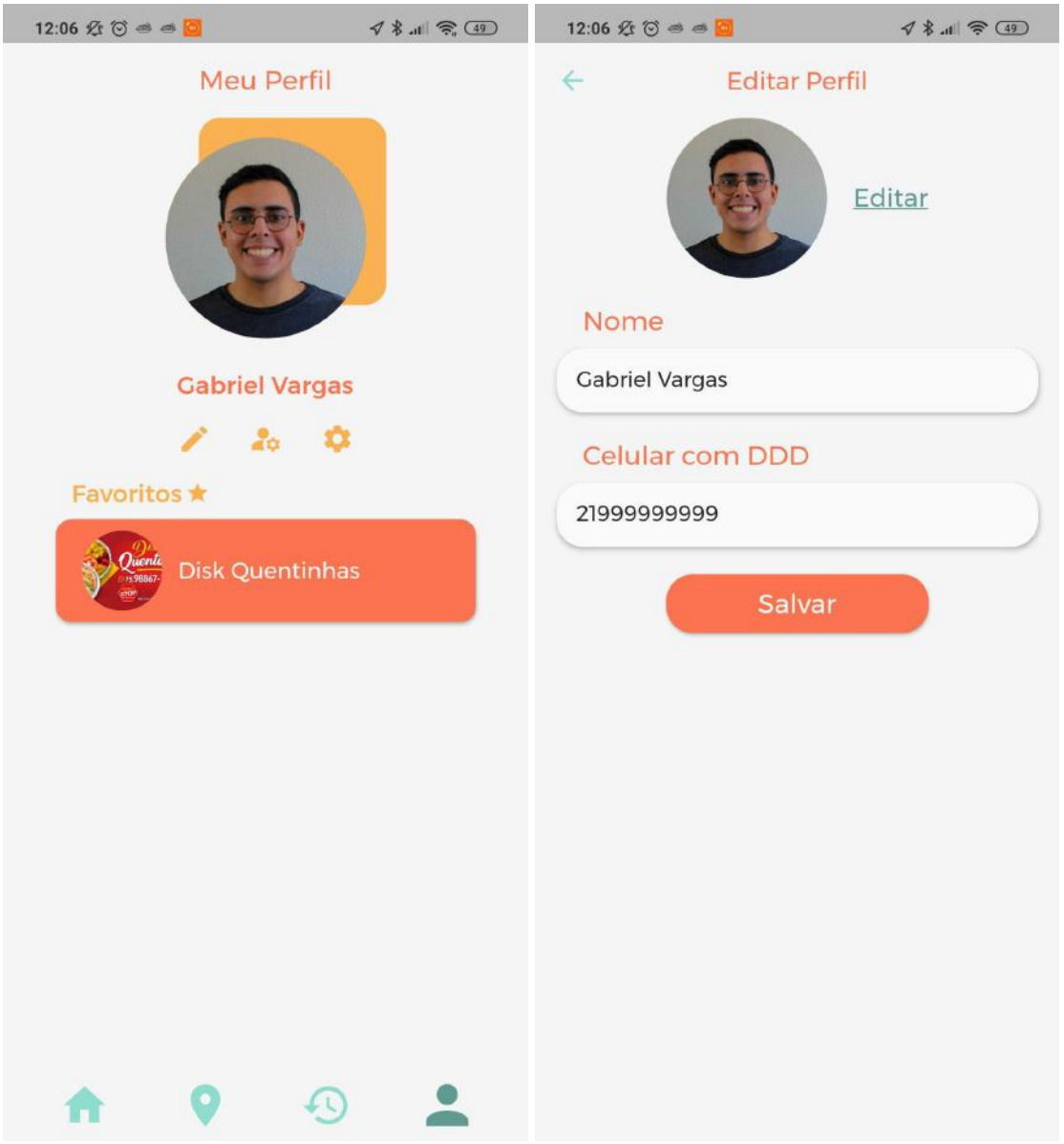


Figura K-9 - Tela de edição de conta (esquerda) e de configurações (direita)

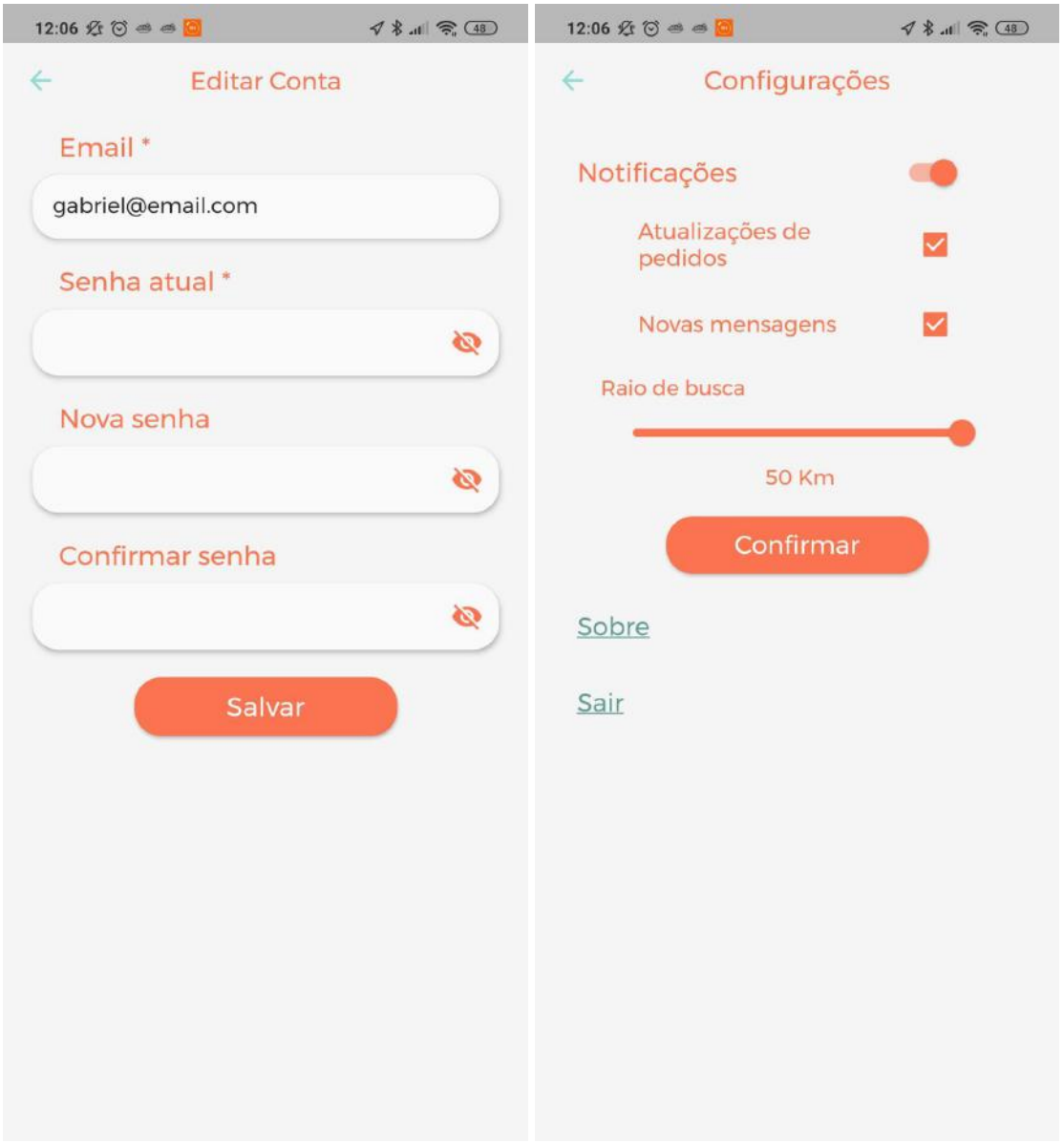
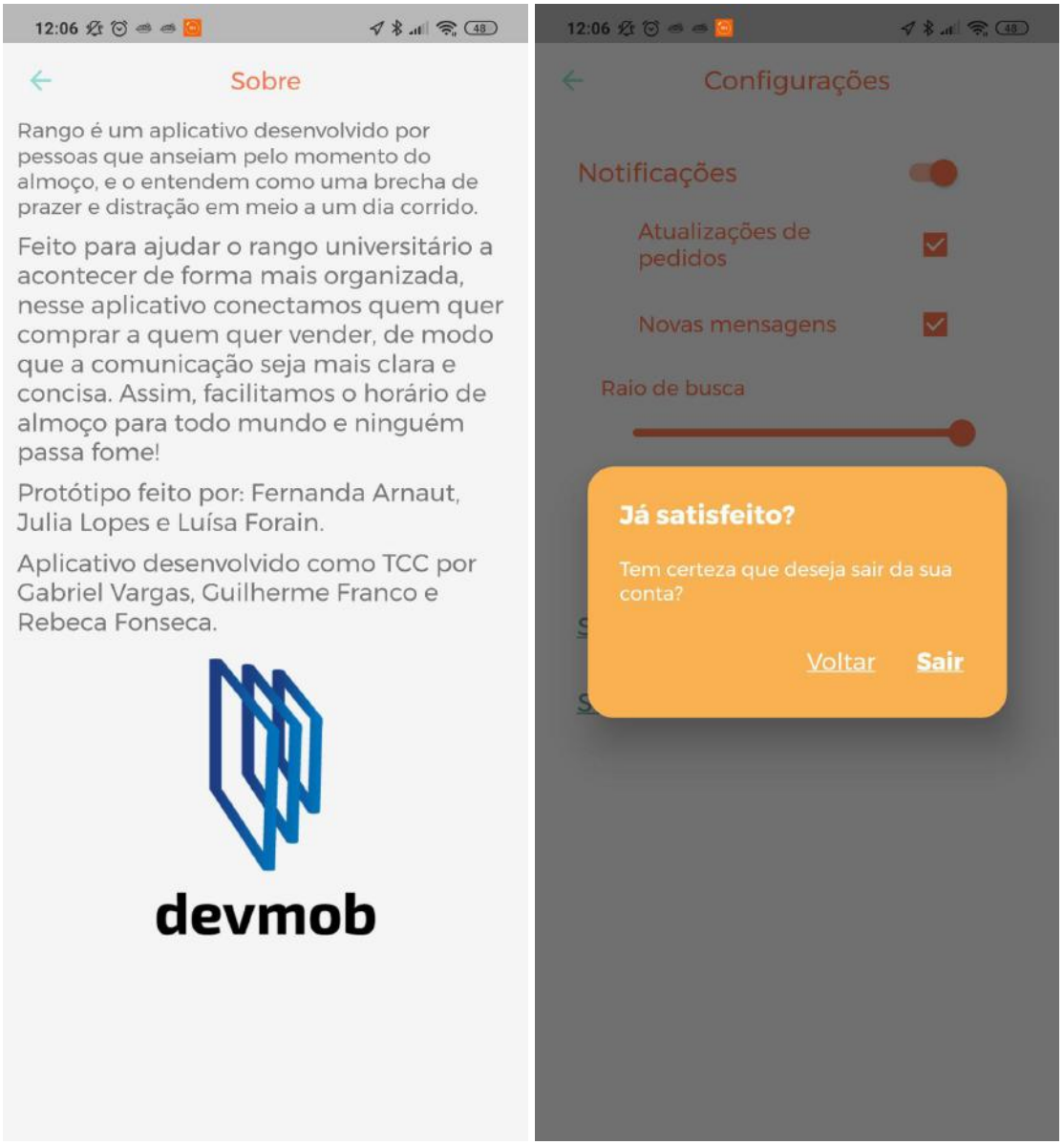


Figura K-10 - Tela de sobre (esquerda) e de sair da conta(direita)



APÊNDICE L – TELAS DO APLICATIVO DO VENDEDOR

Figura L-1 - Tela inicial (esquerda) e de login (direita)

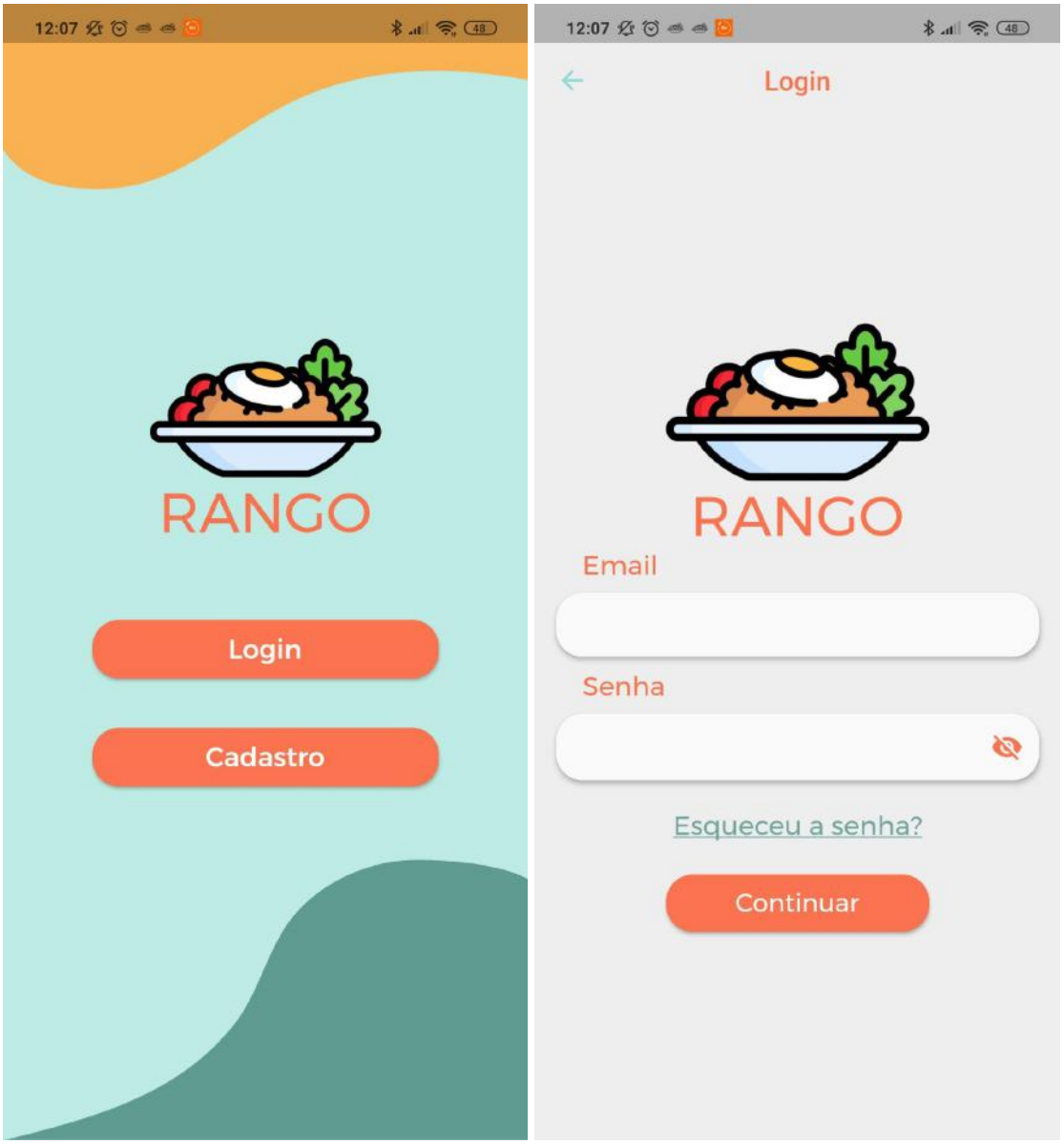


Figura L-2 - Tela de cadastro (esquerda) e de recuperação de senha (direita)

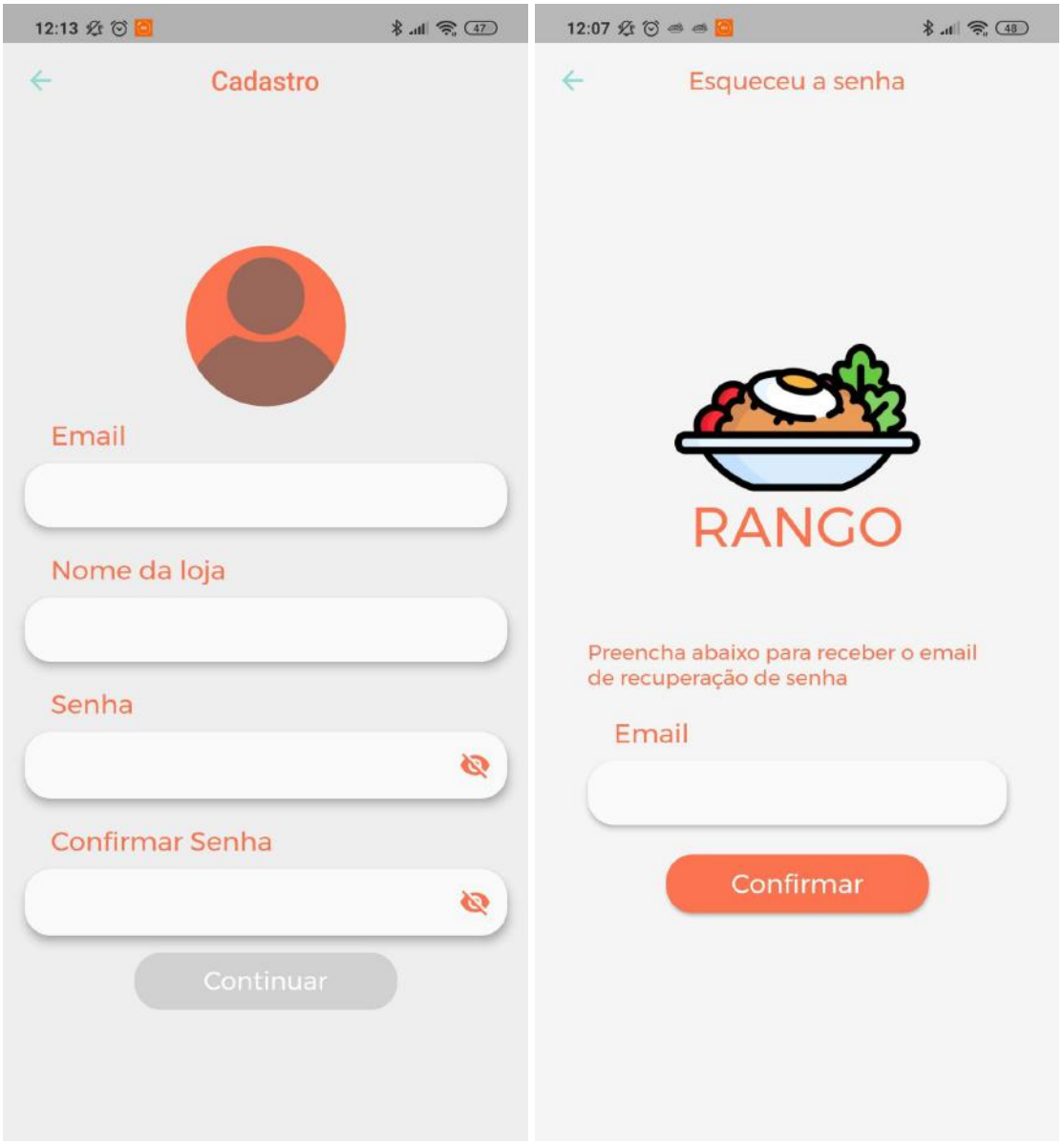


Figura L-3 - Tela de pedidos em aberto do dia (esquerda) e de pedidos concluídos do dia (direita)

Figura L-4 - Tela de perfil de cliente (esquerda) e de conversa com cliente (direita)

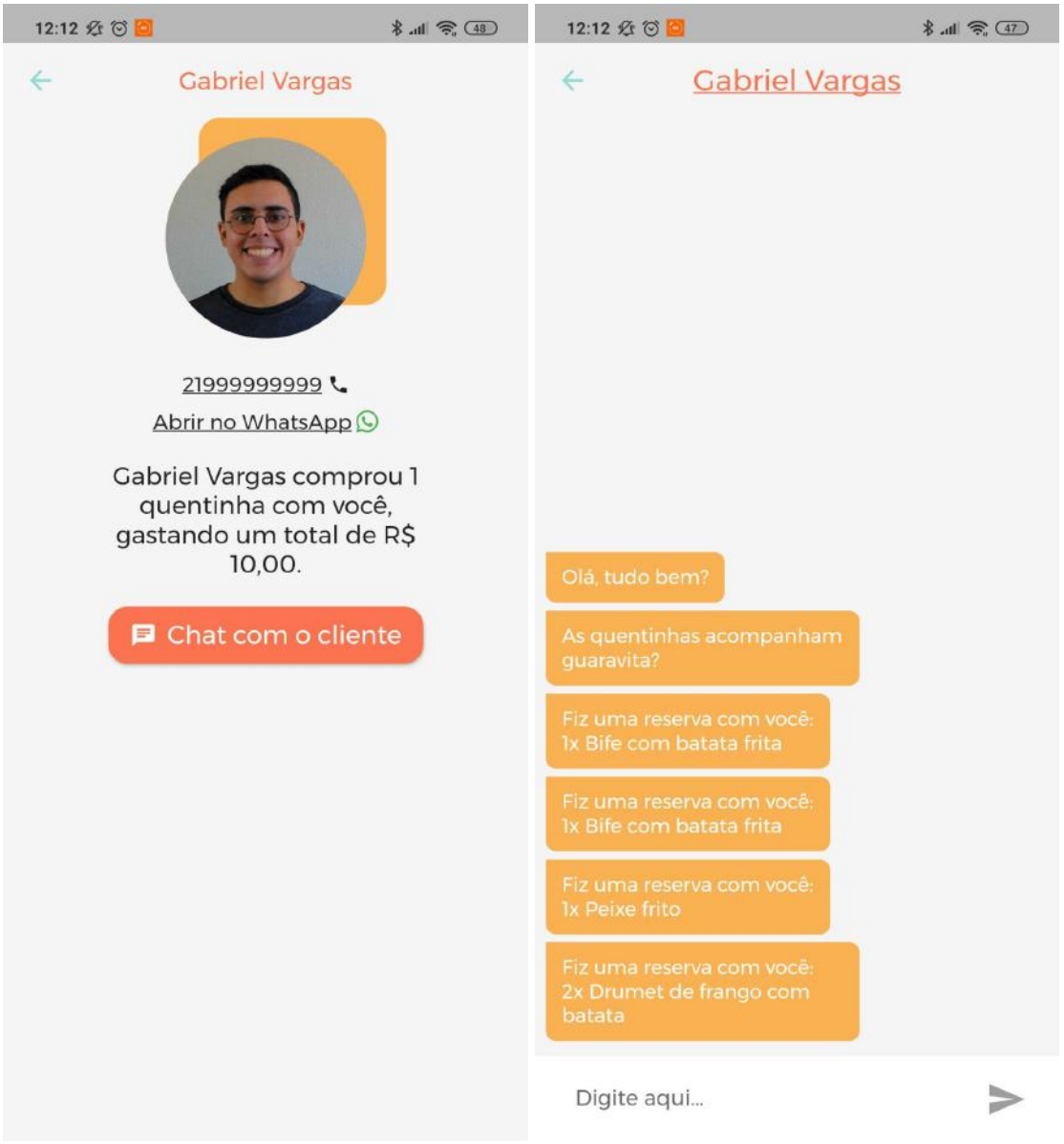


Figura L-5 - Tela de gerenciamento de quentinhas (esquerda) e de conversas recentes (direita)

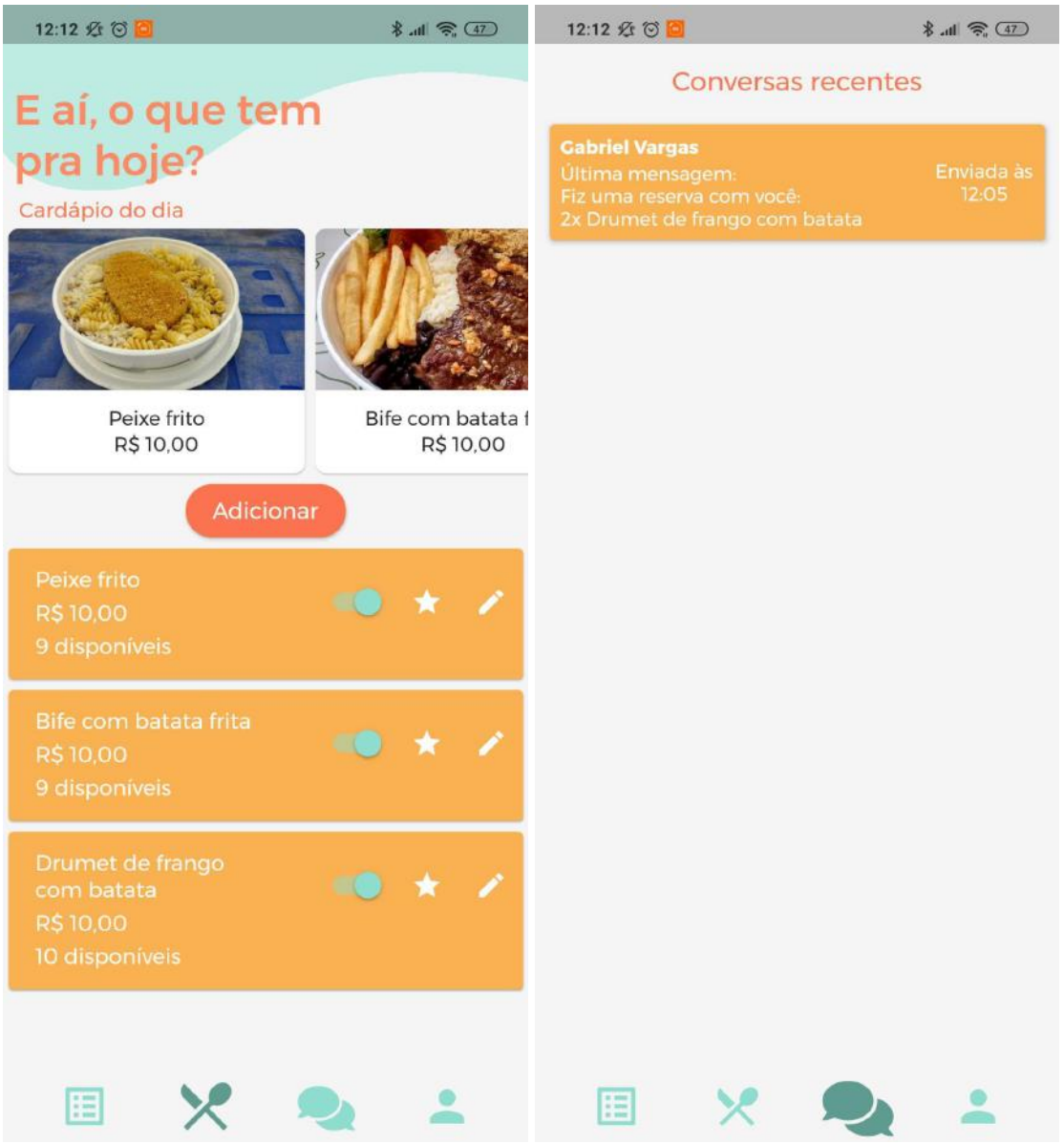


Figura L-6 - Tela de nova quentinha (esquerda) e de edição de quentinha (direita)

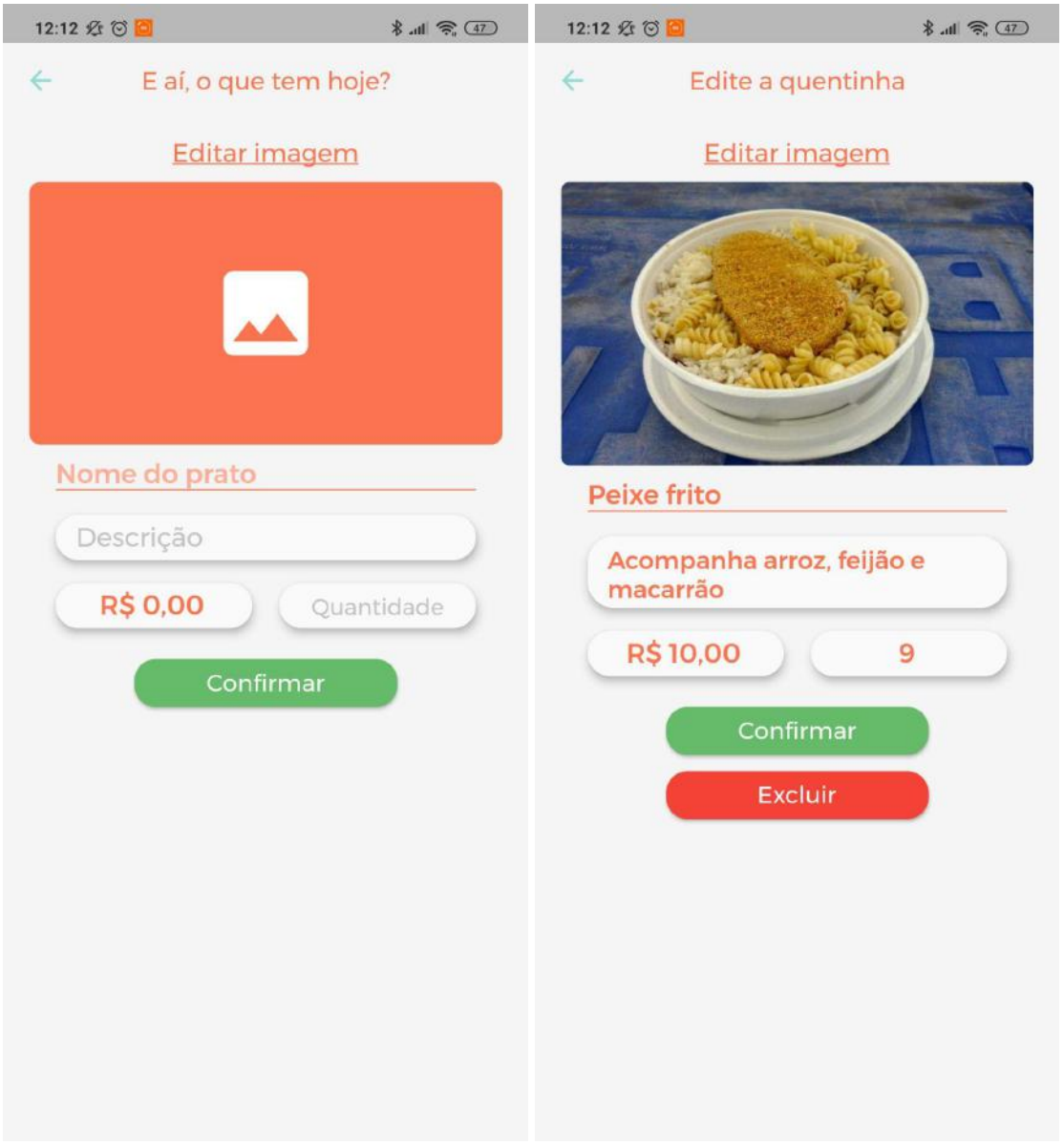


Figura L-7 - Tela de perfil (esquerda) e de edição do perfil (direita)

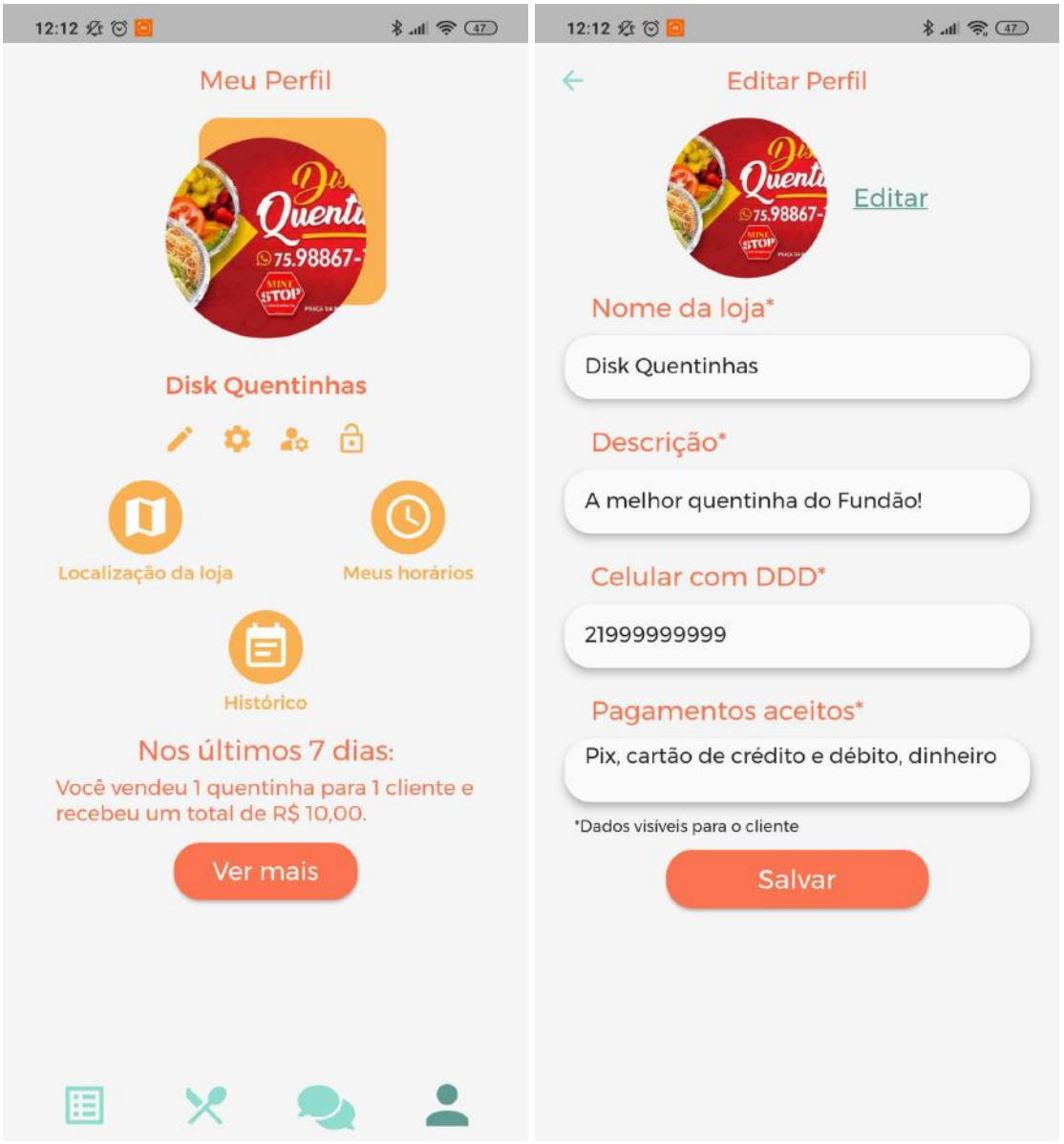


Figura L-8 - Tela de configurações (esquerda) e de edição de conta (direita)

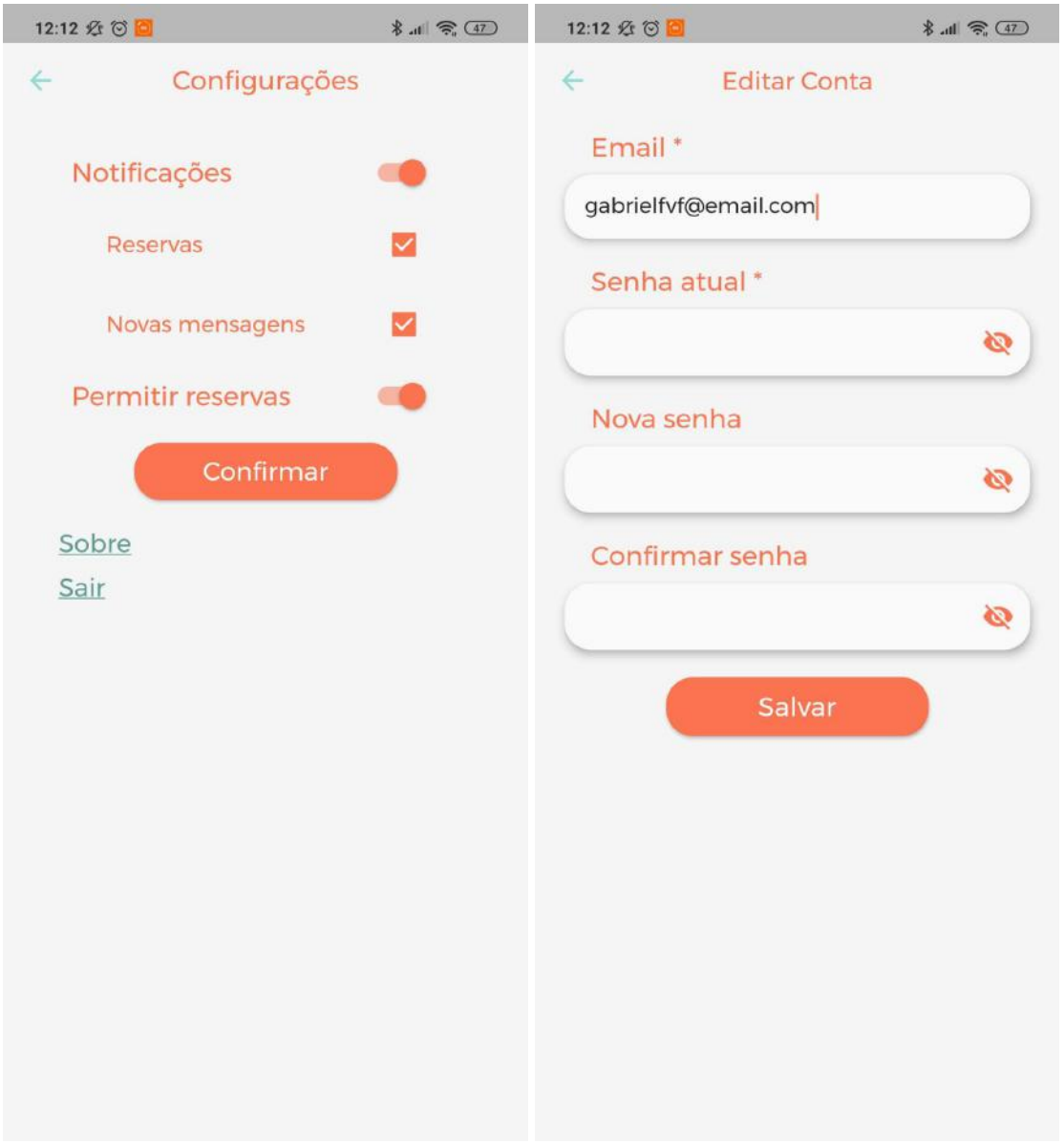


Figura L-9 - Tela de fechar a loja (esquerda) e de escolha de local da loja (direita)

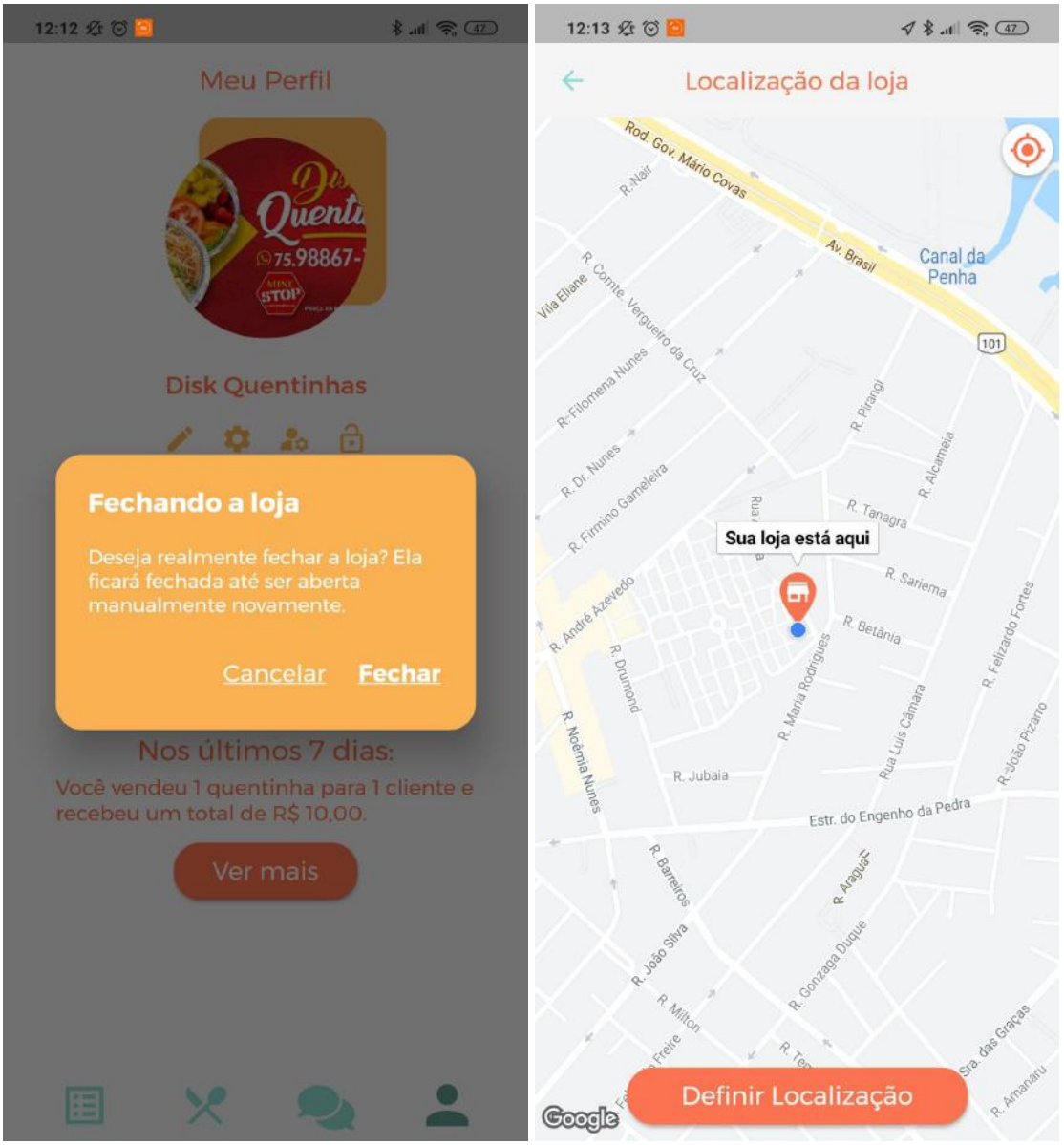


Figura L-10 - Tela de escolha do horário de funcionamento (esquerda) e de histórico de pedidos (direita)

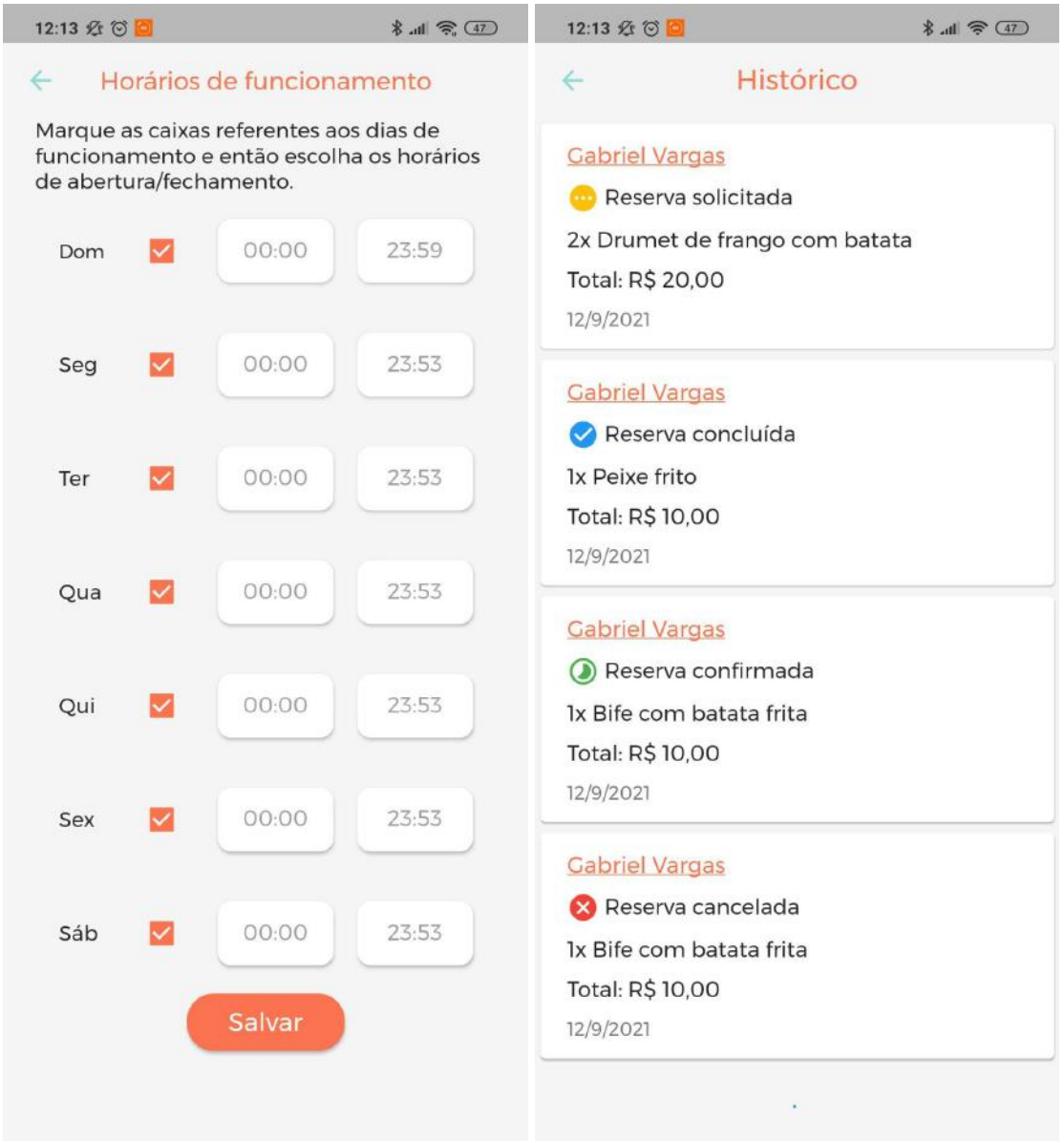


Figura L-11 - Tela de relatório

APÊNDICE M – REGRAS DE SEGURANÇA

Figura M-1 - Configuração de regras de segurança do banco de dados

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /clients/{userId} {
      allow read: if request.auth != null || request.query.limit == 1;
      allow write: if request.auth.uid == userId;
    }

    match /sellers/{userId} {
      allow read: if request.auth != null || request.query.limit == 1;
      allow write: if request.auth.uid == userId;

      match /meals/{mealId} {
        allow read: if request.auth != null;
        allow create: if request.auth.uid == userId;
        allow delete: if request.auth.uid == userId;
        allow update: if request.auth != null;
      }
    }

    match /orders/{orderId} {
      allow read: if request.auth != null &&
                  (request.auth.uid == resource.data.sellerId ||
                   request.auth.uid == resource.data.clientId);
      allow write: if request.auth != null &&
                   (request.auth.uid == request.resource.data.sellerId ||
                    request.auth.uid == request.resource.data.clientId);
    }

    match /chat/{chatId}/{all=**} {
      allow read: if request.auth != null &&
                     (request.auth.uid == chatId.split('_')[0] ||
                      request.auth.uid == chatId.split('_')[1] ||
                      request.auth.uid == resource.data.sellerId ||
                      request.auth.uid == resource.data.clientId);
      allow write: if request.auth != null &&
                      (request.auth.uid == chatId.split('_')[0] ||
                       request.auth.uid == chatId.split('_')[1] ||
                       request.auth.uid == request.resource.data.sellerId ||
                       request.auth.uid == request.resource.data.clientId);
    }
  }
}
```

Figura M-2 - Configuração de regras de segurança do armazenamento de arquivos

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /users/{userId}/{all=**} {
      allow read: if request.auth != null;
      allow write: if request.auth.uid == userId;
    }
  }
}
```