INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS BAMBUÍ
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Luiz Augusto Silva Veloso

DESENVOLVIMENTO DE UM APLICATIVO PARA *DELIVERY* DE BEBIDAS

Bambuí - MG
2023

LUIZ AUGUSTO SILVA VELOSO

DESENVOLVIMENTO DE UM APLICATIVO PARA *DELIVERY* DE BEBIDAS

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Marcos Roberto Ribeiro

Bambuí - MG
2023

Catalogação na Fonte Biblioteca IFMG - Campus Bambuí

V443d Veloso, Luiz Augusto Silva.
Desenvolvimento de um aplicativo para delivery de bebidas. / Luiz Augusto Silva Veloso. – 2023.
58 f. : il. ; color.

Orientador: Doutor Marcos Roberto Ribeiro.
Trabalho de Conclusão de Curso (graduação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG, Curso Bacharelado em Engenharia de Computação, 2023.

1. Aplicativo. 2. Delivery. 3. Bebidas. I. Ribeiro, Marcos Roberto. II. Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG. III. Título.

CDD 005.636

Elaborada por Douglas Bernardes de Castro- CRB-6/2802

Luiz Augusto Silva Veloso

DESENVOLVIMENTO DE UM APLICATIVO PARA *DELIVERY* DE BEBIDAS

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí para obtenção do grau de Bacharel em Engenharia de Computação.

Aprovado em 01 de Dezembro de 2023 pela banca examinadora:

Prof. Dr. Marcos Roberto Ribeiro – IFMG – Campus Bambuí – (Orientador)
Prof. Dr. Ciniro Aparecido Leite Nametala – IFMG - Campus Bambuí
Mestre Felipe Lopes de Melo Faria – IFMG - Campus Bambuí

Documento assinado eletronicamente por Marcos Roberto Ribeiro, Professor, em 01/12/2023, às 08:57, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por Ciniro Aparecido Leite Nametala, Professor, em 01/12/2023, às 09:18, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por Felipe Lopes de Melo Faria, Professor, em 01/12/2023, às 09:19, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador 1750128 e o código CRC D35FA433.

Bambuí - MG
2023

RESUMO

Diante de uma análise no mercado de *delivery*, foi observado que os aplicativos de sucesso não atendem às pequenas cidades do Brasil. Entretanto, em lojas de aplicativos, ou mesmo em trabalhos acadêmicos, é possível encontrar aplicativos de *delivery* que atendem aos estabelecimentos com serviços de refeições. Assim, o objetivo deste trabalho é informatizar a entrega de bebidas por meio do desenvolvimento de um aplicativo para dispositivo móveis, atendendo, também, as pequenas cidades do Brasil. No decorrer do trabalho, foram realizadas análises com os aplicativos existentes no mercado e com os aplicativos desenvolvidos em trabalhos acadêmicos, e, a partir disso, foram definidos os requisitos. Posteriormente, desenvolveram-se os protótipos das interfaces do usuário, a modelagem do banco de dados NOSQL, a implementação do aplicativo, utilizando-se o *framework Flutter* e a linguagem de programação *Dart*, sendo que, ao final, executaram-se os testes com as funcionalidades desenvolvidas. No trabalho, foi realizada a implementação do aplicativo para os clientes, e os estabelecimentos e os produtos foram adicionados diretamente no banco de dados.

Palavras-chave: Aplicativo. *Delivery*. Bebidas.

ABSTRACT

In an analysis of the delivery market, was observed that a gap exists where successful delivery apps in the industry do not cater to small towns in Brazil. However, in app stores or even in academic works, it is possible to find delivery apps that serve meal establishments. Thus, the objective of this work is to computerize the delivery of beverages by developing a mobile application that also caters to small towns in Brazil. During the development of the project, an analysis was conducted with existing market apps and those developed in academic works, allowing the definition of requirements. Subsequently, user interface prototypes were created, a NoSQL database was modeled, and, in the end, implementation was done using the Flutter framework and Dart programming language. In this project, the implementation of the app for customers was carried out, with establishments and products being added to the database. At the end of the implementation, a desk test was conducted to verify the functionality of the app.

Keywords: App. Delivery. Beverages.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

HTML - *HyperText Markup Language*

CSS - *Cascading Style Sheets*

PWA - *Progressive Web Applications*

NDK - *Native Development Kit*

LLVM - *Low Level Virtual Machine*

SGBD - Sistemas de gerenciamento de banco de dados

NoSQL - *Not only structured Query Language*

JSON - *JavaScript Object Notation*

AWS - *Amazon Web Services*

IoT - internet das Coisas

HAL - *Hardware Abstraction Layer*

LGPD - Lei Geral de Proteção de Dados

GPS - *Global Positioning System*

SUMÁRIO

1 INTRODUÇÃO

Em um mundo cada vez mais digital, determinadas tecnologias buscam ajudar e melhorar a qualidade de vida das pessoas, tornando suas tarefas diárias menos cansativas. No Brasil, em 2021, a Pesquisa Nacional por Amostra de Domicílios, realizada pelo IBGE, mostrou que 90,0% dos domicílios nacionais possuem acesso à *internet*. Essa pesquisa apontou que há 65,6 milhões de domicílios conectados e 155,7 milhões de usuários na *internet*. Na mesma pesquisa, foi mostrado que houve um aumento no uso da *internet* tanto no ambiente urbano quanto no rural, sendo que, em 2019, no ambiente urbano, havia 88,1% de domicílios conectados à *internet*, enquanto, em 2021, 92%. Já nos ambientes rurais, em 2019, havia 57,7%, e, em 2021, 92,3% (IBGE, 2022).

Segundo o trabalho de (MELLO, 2020), que teve por objetivo identificar quais as cidades do Brasil atendidas pelos aplicativos de *delivery* de sucesso, os aplicativos analisados pelo autor do trabalho foram *IFood*, *UberEats* e *Rappi*. Foi possível observar que eles estão concentrados em atender ao litoral brasileiro e às capitais dos Estados. Os motivos apontados foram a densidade demográfica e a qualidade das redes informacionais.

Em paralelo com o aumento no uso da *internet*, os pedidos via *delivery* aumentaram consideravelmente com o passar dos anos. Durante a pandemia, de 2020 a 2022, os aplicativos de *delivery* tiveram números recordes em vendas. Uma pesquisa realizada pelo Instituto *Foodservice* Brasil, em conjunto com o GS&NPD, mostrou que o mercado de *delivery*, em 2021, movimentou 40,5 bilhões de reais, o que representa 24% a mais que no ano anterior (IFB, 2023).

Com base nessas informações, foi observado um espaço no mercado para atender a muitas cidades. Também, em função do crescimento tecnológico, os aplicativos de *delivery* ficaram mais presentes na rotina das pessoas. A palavra *delivery* é uma tradução direta para entregar ou distribuir. Neste trabalho, quando houver a ocorrência da palavra *delivery*, a tradução será entrega.

Segundo Botelho, Cardoso e Canella (2020), há dois aspectos que motivaram o aumento no uso de aplicativos de *delivery* no Brasil. O primeiro é a expansão do acesso à *internet*, que possibilitou mais aparelhos conectados, e o segundo, a disseminação da cultura digital. Entretanto, muitos comércios não estão digitalizados, tendo os seus registros anotados em papéis, o que pode ser um causador de perda de dados de clientes e de vendas (ADAM, 2019).

Os dados de vendas e de clientes são de grande relevância para a alavancagem dos negócios, sendo possível estruturar o que oferecer para uma base de clientes, quando realizar promoções de determinado produto, entre outras possíveis ações. Para Adam (2019), os aplicativos são ferramentas que podem auxiliar o comércio local com o intuito de fazer *marketing* digital, uma vez que eles podem extrapolar limites geográficos, mostrando os produtos a todos os clientes e atraindo outros em diversas regiões. Além de beneficiar os estabelecimentos, os aplicativos de *delivery* trazem grandes vantagens para a comunidade,

devido à facilidade de deslocamento do produto até o consumidor e agilidade no processo de entrega.

Dessa forma, o presente trabalho propôs a informatização do *delivery* de bebidas por meio do desenvolvimento de um aplicativo para dispositivos móveis. O aplicativo foi desenvolvido para o cliente final, possibilitando visualizar os estabelecimentos cadastrados, comprar os produtos do estabelecimento selecionado e realizar outras ações. Os estabelecimentos e produtos serão cadastrados diretamente no banco de dados, a fim de realizar os testes das funcionalidades implementadas, e o aplicativo para os estabelecimentos será desenvolvido em outro projeto, fora do escopo deste trabalho.

1.1 Objetivo geral

O objetivo geral deste trabalho é informatizar as entregas de bebidas por meio do desenvolvimento de um aplicativo de *delivery* para dispositivos móveis.

1.2 Objetivos específicos

Os objetivos específicos do presente trabalho foram os seguintes:

- Analisar as principais funcionalidades dos aplicativos de sucesso no mercado e, assim, definir os requisitos do aplicativo;
- Desenvolver os protótipos das telas do usuário;
- Analisar as arquiteturas de banco de dados e realizar posteriormente a modelagem;
- Implementar o código do aplicativo e validar os requisitos com teste de mesa.

1.3 Justificativa

Um estudo realizado por Mobile Time (2019) mostrou a porcentagem de pessoas que pediram refeições por meio de um aplicativo no *smartphone*. Em 2015, foram registrados 25%; em 2016, 38%; em 2017, 47%; em 2018, 54%; e, em 2019, 62%. É possível perceber que, com o passar dos anos, houve um aumento considerável de pedidos por aplicativos. Com a pandemia, esse crescimento tornou-se ainda mais relevante. Em 2022, a consultoria *Kantar*, uma empresa especializada em informações, com sede no Reino Unido, realizou uma pesquisa na qual foi mostrado que, entre 2020 e 2022, o *delivery* no Brasil aumentou de 80% para 89%.

Uma pesquisa efetuada pela Consumo Online no Brasil, em parceria com a agência *Edelman*, entrevistou mil pessoas. Dos entrevistados, 93,5% confirmaram ter gostado da experiência de utilizar aplicativos de *delivery* durante o período pandêmico e pretendem continuar fazendo seus pedidos pelo *smartphone*. Além disso, 84% justificaram

o uso para economizar tempo; 63,9%, para evitar o contágio pela COVID-19; e 68,6% escolheram pela segurança dos pagamentos *online* (VALENTE, 2021).

O trabalho de Adam (2019) comparou as vendas de dois comércios com foco na entrega de refeições durante um período de 16 dias na cidade de São Leopoldo-RS. Durante esse período, foram anotados os pedidos diários realizados de forma tradicional, por ligações telefônicas, pedidos efetuados no balcão e também aqueles solicitados pelo aplicativo de *delivery*. Na primeira empresa, a soma de pedidos realizados tradicionalmente foi de 219, enquanto os de *delivery* totalizaram 550. Assim, os pedidos via *delivery* tiveram, aproximadamente, um aumento de 2,51 vezes sobre aqueles efetuados pelo meio tradicional. Na segunda empresa, os pedidos na forma tradicional chegaram a um total de 313, e os realizados pelo aplicativo, 537, um aumento médio de 1,71 vezes sobre os pedidos efetuados tradicionalmente.

Os aplicativos são úteis aos usuários que possuem uma rotina apressada, pelo fato de não haver necessidade de deslocamento, esperas em filas de caixa e/ou por um atendimento. Os aplicativos de *delivery* são um intermediário que fornece esses privilégios. Além disso, há outras vantagens, como a possibilidade de escolher em qual local comprar, baseado em diversos parâmetros, como preço, localidade do estabelecimento, avaliações dos usuários, tempo de entrega, se o comércio realiza entrega grátis ou não e variedades nas opções de pagamentos.

Os comerciantes também possuem benefícios, como alocar sua marca em tais aplicativos, uma vez que poderá ter maior segurança sobre as transações. O aplicativo também diminui consideravelmente questões relacionadas a roubo e assaltos, sendo possível gerenciar todas as movimentações de forma simples, à base de relatórios sobre pedidos diários, quais os tipos de pagamentos mais utilizados, entre outros (TDP, 2019).

Em Mello (2020), é possível observar que há oportunidade de mercado para um aplicativo que atende às pequenas cidades, que possuem muitos de seus comércios não informatizados.

1.4 Resultados esperados

Espera-se que, com o aplicativo finalizado, seja possível preencher a lacuna que há no mercado de *delivery* de bebidas. Inicialmente, o aplicativo será disponibilizado para as regiões do centro-oeste de Minas Gerais. Após a validação nessa região, esperar-se que ele alcance mais cidades, de outras regiões. Cumprindo o objetivo geral e os objetivos específicos, almeja-se elevar o número de vendas para o comércio de pequenas cidades que possuem o serviço de entregas de bebidas.

Para o cliente final, espera-se melhorar a comunicação com estes estabelecimentos, obtendo todas as informações necessárias para a tomada de decisão.

1.5 Estrutura do trabalho

Este trabalho adotou a seguinte estrutura: Introdução, Referencial Teórico, Metodologia, Desenvolvimento e, por fim, as Considerações Finais.

No capítulo do Referencial Teórico, o foco foi o estudo do desenvolvimento para dispositivos móveis, quais as maneiras de se desenvolver um aplicativo e quais ferramentas e linguagens são utilizadas para o desenvolvimento de aplicativos para dispositivos móveis.

No capítulo de metodologia, foi realizada a classificação do trabalho quanto à abordagem, à natureza, aos objetivos e procedimentos, e, para a metodologia computacional, foi classificado de acordo com Wazlawick (2009). Nas demais seções, aborda-se como os métodos e ferramentas foram utilizados no presente estudo.

No capítulo de desenvolvimento, é mostrada a aplicação dos métodos e ferramentas tanto no desenvolvimento do trabalho quanto no desenvolvimento do aplicativo.

No capítulo de considerações finais, são expostas as conclusões do trabalho e quais os trabalhos futuros a serem desenvolvidos.

2 REFERENCIAL TEÓRICO

Neste capítulo, são apresentados os fundamentos teóricos que serão abordados durante a execução do trabalho e os principais trabalhos acadêmicos correlatos. A Seção 2.1 aborda o desenvolvimento de aplicativos para dispositivos móveis. Na Seção 2.2, é discutido como funcionam os bancos de dados relacionais e não relacionais. Em seguida, a Seção 2.5 compara as ferramentas dos principais aplicativos de *delivery*. Na Seção 2.6, são mostrados trabalhos acadêmicos voltados para o desenvolvimento de aplicativos de *delivery*.

2.1 Desenvolvimento para aplicativos móveis

Segundo El-Kassas *et al.* (2017), a programação para dispositivos móveis se difere de outros tipos de *software* nos seus requisitos e restrições. Muitas dessas restrições se dão por meio de *hardware*, pois, em comparação com os *desktops*, os *smartphones* possuem menor poder de processamento e menor capacidade de memória RAM. Assim, os desenvolvedores de aplicativos móveis devem buscar maneiras para otimizar o código-fonte. Para os *smartphones*, existem dois principais sistemas operacionais, Android e *iOS*.

O sistema operacional Android foi desenvolvido com base no código aberto do Linux. Atualmente, o Android é o sistema operacional mais utilizado em todo o mundo, com 34,78% do mercado. Os aplicativos desenvolvidos para esse sistema operacional são distribuídos em lojas virtuais como *Play Store* ou *F-Droid*. A *Play Store*, loja oficial do Android, conta com 3,5 milhões de aplicativos disponíveis.

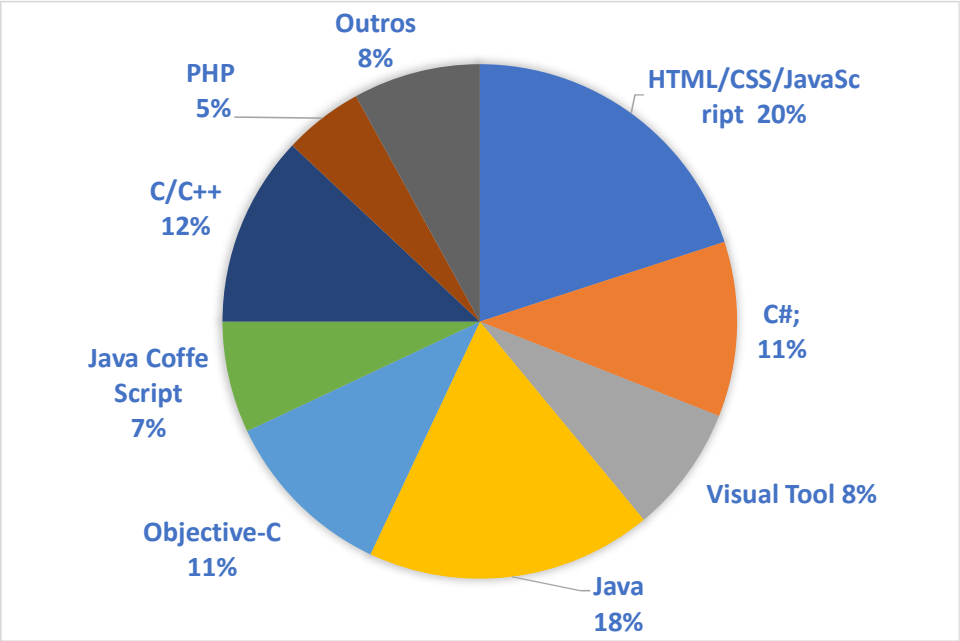
O sistema operacional *iOS* foi desenvolvido com base no sistema operacional *MAC OS X* para atender aos *smartphones* da empresa *Apple*. Atualmente, o *iOS* é o terceiro sistema operacional mais utilizado do mundo. Sua loja oficial é a *Apple Store* e possui 2,2 milhões de aplicativos (STATCOUNTER, 2023).

O desenvolvimento para aplicativos *mobile* pode ser classificado como desenvolvimento nativo ou multiplataforma. Segundo Smutný (2012), a escolha de como será realizado o desenvolvimento de um aplicativo pode depender de seu contexto. Se o aplicativo irá funcionar de forma *offline*, a melhor escolha é o desenvolvimento nativo. Caso o aplicativo tenha interações *online*, a melhor escolha é o desenvolvimento multiplataforma. Como ferramentas para desenvolvimento nativo, podem ser usadas as linguagens *Kotlin* ou *Java*, para o sistema operacional Android, e *Objective-C* ou *Swift* para o sistema operacional *iOS*. No caso do desenvolvimento multiplataforma, podem ser utilizados os *frameworks Flutter* com a linguagem de programação *Dart*; *React Native* com linguagem de programação *JavaScript* ou *TypeScript*, *Kotlin* multiplataforma; *Xamarin* com a linguagem de programação .NET, entre outros.

Litayem, Dhupia e Rubab (2015) realizaram um estudo em que foram levantadas as linguagens de programação mais utilizadas para os dispositivos móveis, conforme

mostrado na Figura 1. Esse mesmo estudo mostrou que, em 2021, houve um aumento de 3% dos desenvolvedores que utilizam o *framework flutter*, com o total de 42%, ultrapassando o *framework React Native*.

Figura 1 – Linguagens para desenvolvimento Nativo



Fonte: Elaborado pelo Autor, 2023.

2.1.1 Desenvolvimento Nativo

Para Prezotto e Batista Boniati (2014), aplicações nativas são desenvolvidas para um sistema operacional utilizando-se ferramentas e linguagem de programação específicas para uma plataforma. Caso a aplicação seja utilizada para diferentes plataformas, ela então deverá ser implementada em diferentes linguagens de programação e em diferentes ambientes. Aplicativos desenvolvidos nativamente possuem melhor desempenho ao serem programados para utilizar eficientemente os recursos de *hardware* e o sistema operacional.

Em relação ao sistema operacional Android, é mais fácil de se desenvolver aplicativos para sua plataforma, uma vez que todos os sistemas operacionais de computadores suportam e permitem o desenvolvimento e compilação para a plataforma Android. O *iOS* possui seu código-fonte fechado; logo, para o desenvolvimento de aplicativos, é necessário possuir um ambiente de produção com sistema operacional da *Apple* para ser possível a compilação do aplicativo. O desenvolvimento de um mesmo aplicativo em ambas as plataformas pode ser uma desvantagem, visto que é necessário codificar em diferentes linguagens de programação, utilizar diferentes ferramentas e, no pior caso, utilizar diferentes sistemas computacionais.

2.1.2 Desenvolvimento multiplataforma

De acordo com Shah, Sinha e Paypal (2019), as ferramentas multiplataforma devem permitir a compilação de um código-fonte escrito em uma linguagem de programação e traduzi-lo para a linguagem de referência do sistema operacional. Por exemplo, ao se desenvolver aplicativos com a ferramenta *Flutter* e linguagem de programação *Dart*, é solicitado, no início do projeto, quais linguagens de programação para Android e *iOS* serão utilizadas de referência para a tradução: *Kotlin* ou *Java*, para o Android, e *Objective-C* ou *Swift*, para *iOS*. Assim, o uso do *Flutter* irá permitir a compilação para os sistemas operacionais Android e *iOS*. Segundo Shah, Sinha e Paypal (2019), no desenvolvimento multiplataforma, existem categorias de aplicativos *Web apps*, *Progressive web apps* (PWA), aplicativos híbridos, aplicativos interpretados e aplicativos de compilação cruzada.

Os *Web apps* são desenvolvidos utilizando *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript*, com o auxílio de ferramentas como *Angular*, *Vue*, *React*, *jQuery*, *Django*, *Ruby*, entre outras. Os *Web apps* não precisam ser instalados e podem ser acessados por navegadores *Web*.

Aplicativos PWA são uma evolução dos *Web apps*, podendo ser instalados e também acessados no modo *offline*.

Aplicativos híbridos são desenvolvidos com HTML, CSS e *JavaScript*, sendo embutidos em *containers* que possuem o código nativo para determinado sistema operacional e bibliotecas nativas do sistema.

Os aplicativos são interpretados em tempo de execução para instruções nativas de cada sistema operacional; dessa forma, a interface possui certa semelhança com aplicações nativas.

Os aplicativos de compilação cruzada são desenvolvidos para o sistema operacional-alvo, e sua eficiência é semelhante à de aplicações nativas.

2.1.3 O framework Flutter

O *Flutter* é um *framework* desenvolvido pela Google que permite o desenvolvimento multiplataforma e também é uma estrutura de interface de usuário.

Segundo Shah, Sinha e Paypal (2019), os aplicativos desenvolvidos com *Flutter* são classificados como aplicativos baseados em *widgets*. Nesse cenário, um *widget* é tudo o que pode ser tratado como uma estrutura de interface do usuário, seja um botão, cores de texto, margens, mapas, caixas de texto, etc. Os *widgets* podem ser classificados de duas formas: com estado ou sem estado.

Os *widgets* com estados são os *StatefulWidget*, e os *widgets* sem estado são os *StatelessWidget*. Assim, um componente que deve apenas aparecer para o usuário e não realizar nenhuma mudança é tratado como *StatelessWidget*. Quando se deseja realizar mudanças nos aplicativos, atualizar informações e mostrar ao usuário, utilizam-se

os *StatefulWidget*. Um exemplo com esse *widget* é o aplicativo gerado no início de um projeto *Flutter*. Este aplicativo possui um botão e um texto, sendo que, quando o usuário aperta o botão, o texto irá mostrar quantas vezes o botão foi pressionado. Se o projeto fosse desenvolvido com *StatelessWidget*, o texto não seria alterado. Caso fosse desenvolvido com *StatefulWidgets*, o texto mostraria a quantidade de vezes que o botão foi pressionado.

A arquitetura do *framework Flutter* é dividida em duas partes. A primeira é totalmente desenvolvida em *Dart*, no qual ficam todos os *widgets* e as bibliotecas. A segunda é a *engine* gráfica implementada nas linguagens de programação C/C++. O código C/C++ é compilado para Android utilizando o kit de desenvolvimento nativo deste, ou *Native Development Kit* (NDK). Para o *iOS*, é utilizada uma máquina virtual de baixo nível, ou *Low Level Virtual Machine* (LLVM).

No desenvolvimento com *Flutter*, é utilizada a linguagem de programação *Dart*, a qual é multiparadigma, sendo orientada a objetos e possuindo algumas características de paradigma funcional. O *Dart* foi desenvolvido pela Google e possui uma baixa curva de aprendizagem. A linguagem é otimizada para todas as plataformas, sendo possível compilá-la para dispositivos móveis, *desktops*, *back-end* e para *JavaScript web* (DART, 2023).

2.2 Bancos de dados

Ramakrishnan e Gehrke (2008) define banco de dados como "... uma coleção de dados que, tipicamente, descreve as atividades de uma ou mais organizações relacionadas". Com o auxílio dos Sistemas de Gerenciamento de Banco de Dados (SGBDs), é possível gerenciar as estruturas do banco de dados e controlar os acessos aos dados que estão armazenados, sendo possível adicionar, alterar, excluir e selecionar. Existem diversos tipos de banco de dados: relacionais, hierárquicos, de rede, orientados a objetos, por objetos relacionais e não relacionais.

Uma das formas mais simples de armazenar os dados é a arquitetura pessoal, ou *Stand Alone*. Esse tipo de armazenamento é útil para aplicativos *offline*, quando é possível armazenar uma pequena quantidade de dados localmente no próprio dispositivo do usuário. Também existe a arquitetura cliente-servidor, uma das mais populares, onde todo o sistema de *back-end* fica armazenado em um servidor e, via rede, uma aplicação comunica com o servidor cliente. Essa arquitetura traz vantagens para bancos de dados que necessitam realizar transações de informações do servidor para um cliente de forma *online* (TAKAI; ITALIANO; FERREIRA, 2005).

Os banco de dados com armazenamento em nuvem utilizam diversas arquiteturas, como a arquitetura centralizada, que pode ser usual para pequenas demandas. Porém, com projeção de escalabilidade, nas arquiteturas distribuídas, os dados são salvos em vários servidores, com diferentes localizações geográficas. Essa arquitetura distribuída possibilita maior escalabilidade do sistema, disponibilidade e desempenho.

Os bancos de dados não relacionais surgiram com o propósito de resolver problemas onde os bancos relacionais não são adequados. Os principais requisitos para sua utilização são a facilidade para lidar com grandes volumes de dados estruturados, semiestruturados ou não estruturados e a necessidade de alta disponibilidade e escalabilidade no armazenamento. Com base nesses requisitos, existem características específicas que diferenciam o modelo de bancos de dados tradicional do de bancos de dados não relacional. Algumas dessas características são: altas disponibilidade, escalabilidade horizontal e execução em *hardware* comum.

Os SGBDs para tecnologias *Not only structured Query Language* (NoSQL) podem se beneficiar da arquitetura em nuvem. Essas tecnologias buscam armazenar um dado em mais de um computador, possibilitando, assim, que o *software* não fique indisponível caso aconteça algum incidente em um dos computadores. Também é comum armazenar o mesmo dado em mais de uma tabela, ou documentos, o que não é desejável nos bancos de dados relacionais (RIBEIRO, 2022).

De acordo com Lóscio, Oliveira e Pontes (2011), as características para bancos de dados *NoSQL* são definidas como escalabilidade horizontal e *hardware* comum, mapeamento objeto-relacional, alto rendimento e *APIs* para acesso aos dados.

Com abundância de dados, é necessário aumentar a escalabilidade e melhoria no desempenho, sendo que a solução de escalabilidade horizontal é o aumento no número de máquinas disponíveis para o armazenamento e processamento de dados.

O mapeamento objeto-relacional elimina a necessidade de um esquema estrito para definir a estrutura dos dados modelados, o que, por sua vez, simplifica a escalabilidade e aumenta a disponibilidade.

A característica de alto rendimento permite a replicação nativamente e diminui o tempo gasto para a recuperação da informação. Existem dois métodos principais, *Controller-Worker* e *Multi-Master*. É possível desenvolver APIs para facilitar o acesso aos dados armazenados.

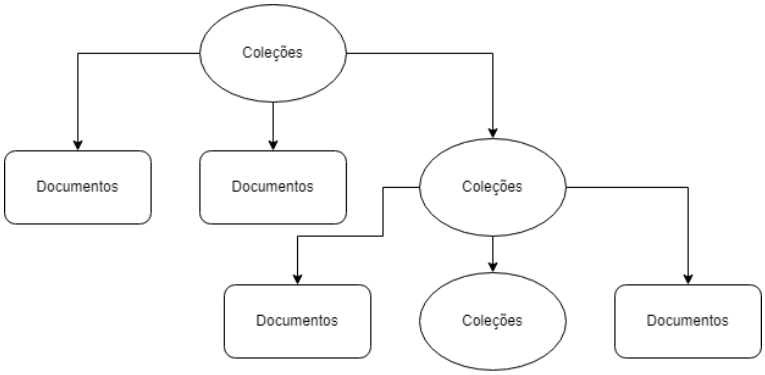
Um dos bancos de dados que utiliza a arquitetura *NoSQL* é o *Cloud Firestore*, que é um banco de dados *NoSQL* hospedado em nuvem e oferecido como um dos serviços do *Firebase*. Por ser um banco de dados não relacional, o *Firestore* se torna flexível e escalável para o desenvolvimento de aplicativos móveis e *web*. Os aplicativos que utilizam o *Cloud Firestore* não se comunicam com servidores, e sim diretamente com o banco de dados. O aplicativo recebe dados via *WebSocket*, que é uma comunicação mais rápida do que o protocolo HTTP. Os bancos de dados *Firestore* possuem vários recursos, como suporte *offline*. Tal suporte permite que os dados mais acessados sejam armazenados em cache, sendo possível utilizá-los no modo *offline*. Quando há alterações no banco de dados, os aplicativos conectados são atualizados em tempo real (MALLIK *et al.* 2020).

A arquitetura de armazenamento do *Cloud Firestore* é orientada a coleção e documentos, onde cada coleção possui um ou mais documentos, e cada documento pode

possuir uma coleção, ou, como é comumente chamado, subcoleções. Os documentos são armazenados no formato *JavaScript Object Notation* (JSON). A Figura 2 mostra como é a arquitetura de armazenamento do *Cloud Firestore*.

Os bancos de dados *NoSQL* permitem a duplicação destes, ou seja, armazenando-os em diferentes locais, o que simplifica as pesquisas que poderiam ser complexas. No entanto, a desvantagem da duplicação de dados está na manutenção, já que qualquer alteração precisa ser realizada em todos os locais nos quais os dados estão presentes. O *Cloud Firestore* possui um limite gratuito para operações de manipulação de dados no banco de dados. Entretanto, quando esse limite é ultrapassado, é cobrado um valor para cada cem mil documentos por operação. No momento do desenvolvimento deste trabalho, a operação de leitura possuía a cota de 50.000 leituras gratuitas de documentos por dia; ultrapassando-se este limite, é cobrado US$0,045 a cada cem mil documentos. A operação de gravação possui uma cota gratuita de 20.000 documentos por dia, e, extrapolando-se este limite, é cobrado US$0,135 a cada cem mil documentos. A cota para a exclusão de documento é de 20.000 por dia; ultrapassando-se este limite, é cobrado US$0,015 a cada cem mil documentos.

Figura 2 – Estrutura de armazenamento do Banco de dados *Firestore*



Fonte: Elaborado pelo Autor, 2023.

2.3 Metodologia ágil

A metodologia ágil ganhou reconhecimento em 2001 quando especialistas em Engenharia de *Software* se reuniram e estabeleceram os princípios para um desenvolvimento ágil. Essa metodologia veio como resposta à metodologia anterior, conhecida como metodologia tradicional.

De acordo com Soares (2004), a metodologia tradicional foi composta por métodos derivados atuantes nas áreas de Engenharia Civil, Engenharia Elétrica, Engenharia Naval, entre outras, sendo conhecida por suas atividades serem feitas sequencialmente. Assim, uma nova fase começa apenas quando a fase anterior estiver terminada. A Figura 6 mostra as fases da metodologia tradicional, ligadas por setas, indicando que uma nova

fase deve ser iniciada apenas quando a anterior estiver totalmente finalizada. Com isso, não é possível voltar em nenhuma fase encerrada.

A metodologia ágil, em resposta à metodologia tradicional, propõe que, ao se desenvolver um *software*, deve-se preocupar primeiramente com os usuários, com a interação entre o usuário e o *software*, e, em segundo plano, com as ferramentas que serão utilizadas, documentação do *software* e a implementação.

O *Scrum* é um dos métodos ágeis que se destaca pela maneira de desenvolver *software* incrementalmente, sendo utilizado em ambientes flexíveis, complexos, onde os requisitos do sistema estão em constante mudança e pequenas equipes. O *Scrum* é baseado em três pilares: transparência, inspeção e adaptação (SILVA; SOUZA; CAMARGO, 2013).

A transparência é uma garantia de que todo o processo fique claro para todas as partes envolvidas no projeto. A inspeção deve ocorrer por todo o período do projeto, verificando possíveis variações, para que possam ser evitados futuros problemas. A adaptação é uma consequência da inspeção, em que se deve adequar o procedimento às variações detectadas.

O *Scrum* possui, basicamente, três diferentes equipes, o *product owner*, *scrum master* e a equipe de desenvolvimento. O coração do *Scrum* é o *product backlog*. O *product backlog* são listas de requisitos e itens que o cliente deseja, utilizando a terminologia do próprio cliente. Por ser um método de desenvolvimento incremental, no *Scrum*, há interações diárias entre os membros das equipes; esses encontros são chamados *sprints* (KNIBERG, 2007).

Cada *sprint* é a fase em que as informações são passadas para a equipe, definindo o escopo, a estimativa e a importância das informações. Cada *sprint* tem, em média, 30 dias, e, durante esse período, são realizadas reuniões semanais para atualização da situação do projeto (KNIBERG, 2007). A Figura 7 mostra os processos que o método *Scrum* emprega no desenvolvimento do *software*.

Para auxiliar e mostrar as atividades de desenvolvimento do sistema, foi utilizado o Kanban, que é um método ágil e popular no qual é possível verificar as tarefas que devem ser realizadas, as que estão em andamento, as urgentes e as concluídas.

2.4 Requisitos funcionais e não funcionais

Os requisitos são o conjunto de funcionalidades ou componentes que constituem um *software*. Cada requisito visa alcançar um objetivo específico, sendo que o conjunto de requisitos objetiva alcançar o objetivo geral do software (ENGHOLM JUNIOR, 2010). Os requisitos podem ser divididos em dois grupos: funcionais e não funcionais. Os requisitos funcionais descrevem o comportamento de funcionalidades do *software* e como elas devem reagir à entrada de dados do usuário, e os requisitos não funcionais determinam as restrições do *software*, por exemplo, restrições de *hardware* e regras de negócios (TAVEIRO, 2016).

2.5 Comparativo de aplicativos existentes

Esta seção apresenta uma comparação de alguns aplicativos de *delivery* de sucesso no mercado, os serviços utilizados para a implementação da infraestrutura e principal linguagem de programação. Os aplicativos comparados foram *IFood*, Zé *Delivery* e *Rappi*. Ao final, foi realizada uma comparação dos diferentes requisitos entre os aplicativos de *delivery* com aqueles desenvolvidos em trabalhos acadêmicos.

O Zé *Delivery* é uma subsidiária da empresa Anheuser-Busch InBev, uma das maiores produtoras internacionais de bebidas, que possui uma extensa rede de distribuição em todo o Brasil. O Zé *delivery* optou por utilizar os serviços da *Amazon Web Services* (AWS) (AWS, 2021).

Assim como o Zé *Delivery*, o *IFood* também utiliza os serviços da AWS para o desenvolvimento de sua infraestrutura. A linguagem de programação utilizada no *back-end* pelo *IFood* é a *Rust*. O *IFood* conta com planos variados para estabelecimentos que desejam se cadastrar no aplicativo. O plano básico possui uma mensalidade de R$100,00 e uma comissão sobre o pedido de 12%. Caso o usuário pague pela plataforma, são cobrados 3,2% de taxa sobre o pedido. O segundo plano possui uma mensalidade de R$130,00 e uma comissão de 23%, e, caso o usuário pague pela plataforma, são cobrados 3,2% de taxa sobre o pedido. Por ser uma mensalidade mais cara, há benefícios que a mensalidade básica não possui (IFOOD, 2021).

O aplicativo de *delivery Rappi* tem como proposta a entrega de produtos de restaurantes e supermercados. A infraestrutura do aplicativo é baseada nos serviços da AWS (AWS, 2022). Para os estabelecimentos que desejam se cadastrar no aplicativo *Rappi*, é gratuito; entretanto, são cobrados uma comissão e um imposto sobre o valor agregado a cada venda (RAPPI, 2022).

O trabalho de Mello (2020) demonstra a distribuição espacial nas cidades brasileiras onde os aplicativos de *delivery* possuem atuação. Os aplicativos analisados foram *Uber Eats*, *Rappi* e *IFood*. A metodologia usada no trabalho foi exploratória, de caráter quantitativo, buscando hipóteses para justificar a desigualdade da oferta dos serviços no Brasil e a estratégia dessas empresas para expansão territorial. A Figura 3 mostra as cidades de atuação de cada empresa citada.

O trabalho concluiu que as grandes empresas de *delivery* alimentício possuem maior concentração do serviço no litoral brasileiro, e a distribuição desigual no território do país está ligada com a densidade demográfica, com a densidade técnica das redes informacionais e com as desigualdades socioespaciais da renda e da produção.

2.6 Estado da arte

Nesta seção, são apresentados trabalhos acadêmicos sobre aplicativos de *delivery* e quais ferramentas e linguagens de programação foram utilizadas no desenvolvimento.

Figura 3 – Cidades atendidas por *IFood*, *Rappi*, *UberEats*

(a) – IFood (b) – Rappi (c) – UberEats

Fonte: MELLO, 2020.

O trabalho de Melo e Ferreira (2021) propõe a criação de aplicativo para *delivery* com foco em entregas de refeições para a cidade de Córrego Fundo - MG, que possui aproximadamente 6,25 mil habitantes. Para auxiliar no desenvolvimento do aplicativo, foram utilizados o *Android Studio* como IDE; *Java* como linguagem de programação para desenvolvimento em Android, e o banco de dados fornecido pelo *Firebase*.

No trabalho de Melo (2019), foi desenvolvido um aplicativo *delivery* para atender a todos os tipos de produtos e estabelecimentos na cidade de Alagoa Nova - PB, que possui aproximadamente 21 mil habitantes. Para o *back-end*, foram utilizados os serviços do *Firebase*, como *Cloud Storage*, *Authentication* e *Realtime Database*. No desenvolvimento para o *front-end*, empregou-se o *framework React Native* com a linguagem de programação *JavaScript*.

O trabalho de Fontana Junior (2013) propôs o desenvolvimento de um protótipo de aplicativo de *delivery* para o sistema operacional Android. Foram utilizados a linguagem de programação Java e o banco de dados SQL *Server*. Para tomadas de decisões empresariais, optou-se por desenvolver uma aplicação *Web* utilizando tecnologias ASP.NET MCV3 e a linguagem de programação .NET. Para auxiliar no desenvolvimento do aplicativo móvel, utilizou-se o IDE Eclipse, e, para o desenvolvimento da aplicação Web, o editor de texto *Visual Studio Code*.

O trabalho de Kohlbeck *et al.* (2021) propôs o desenvolvimento de um aplicativo *delivery* para refeições, com foco nas opções que os usuários podem escolher em relação aos alimentos, seja sem restrições, para pessoas com intolerâncias alimentares, veganas ou orgânicas. O desenvolvimento foi realizado no *Android Studio*, utilizando-se os serviços do *Firebase* para armazenamento dos dados e autenticação do usuário.

O trabalho Almeida *et al.* (2023) envolveu o desenvolvimento de um aplicativo de *delivery* para pizzarias para a cidade de Princesa Isabel e regiões. A linguagem de programação *Java* foi empregada para o desenvolvimento no ambiente Android.

Diante dos trabalhos acadêmicos voltados para o desenvolvimento de aplicativos

de *delivery*, o presente trabalho destaca-se pelo desenvolvimento de um aplicativo de *delivery* para os estabelecimentos que realizam entregas de bebidas. Diferenciando-se das demais pesquisas, a abordagem abrange uma análise voltada para a escalabilidade do produto.

3 METODOLOGIA

Este capítulo descreve como foram realizados os processos de desenvolvimento do presente trabalho. A Seção 3.1 descreve a classificação da pesquisa. Na Seção 3.2, são mostradas as funcionalidades comparadas nos aplicativos existentes. A Seção 3.3 apresenta o caso de uso do aplicativo e quais são os principais aspectos a serem tratados. Na Seção 3.4, são mostradas as configurações do computador utilizado no trabalho e as ferramentas. A Seção 3.5 apresenta como foi realizado o desenvolvimento das interfaces do usuário e a modelagem do banco de dados. Na Seção 3.6 é detalhado as etapas realizadas no desenvolvimento.

3.1 Classificação da Pesquisa

Este trabalho, pode ser classificado quanto a sua abordagem em pesquisa qualitativa. Isso se deve a análise dos funcionamentos de todos os requisitos definidos, ao estudo de caso para o uso de aplicativos de *delivery* e uma análise do mercado desses aplicativos. Quanto à natureza, pode ser classificado como uma pesquisa aplicada. Por ser desenvolvido um aplicativo utilizando técnicas e conhecimentos obtidos durante o curso de Engenharia da Computação. Quanto aos objetivos, pode ser definida como pesquisa exploratória. Pois aos objetivos, foi realizado um estudo no desenvolvimento de aplicativos de *delivery* e análises das ferramentas utilizadas no desenvolvimento. Quanto aos procedimentos, a pesquisa pode ser classificada como pesquisa-ação, sendo que há a participação do pesquisador no problema e ser investigado (GERHARDT; SILVEIRA, 2009).

Para Wazlawick (2009) o trabalho pode ser classificado como *apresentação de algo diferente*, uma vez que foram analisadas as funcionalidades dos aplicativos de *delivery* existentes para a criação de um único aplicativo. Diante do fato que o aplicativo fornece um serviço intermediário entre o comércio e o cliente final, este trabalho pode ser classificado como um serviço inovador que, segundo Carvalho, Reis e Cavalcante (2011), um serviço inovador, "... é aquele que proporciona ao cliente uma experiência única que o satisfaz e o faz sentir vontade de buscar a empresa novamente".

3.2 Funcionalidades comparadas

Analisando os aplicativos Zé *Delivery*, *IFood* e *Rappi* no lado do cliente, foi realizada uma comparação de algumas suas funcionalidades com a solução proposta. As funcionalidades para aplicativos de *delivery* como adicionar produto em carrinhos, confirmar endereço, buscar por um produto ou estabelecimentos, não foram comparadas, pois estas são essenciais a estes aplicativos. As funcionalidades comparadas foram:

- Escolher localidade do usuário por GPS ou CEP;

- Mostrar estabelecimentos a partir da localidade escolhida;
- Colocar observações nas compras;
- Pagamentos;
- Histórico de Pedidos;
- Múltiplos Endereços;
- Navegar pelo aplicativo antes de se cadastrar;

No Quadro 1 é mostrado uma tabela comparativa com as funcionalidades analisadas dos aplicativos da Seção 2.5.

Quadro 1 – Funcionalidades comparadas

Aplicativos	Escolher localidade	Mostrar estabelecimentos próximos	Colocar observações	Formas de pagamentos	Histórico de pedidos	Navegar sem cadastro	Atender pequenas cidades
Zé *delivery*	X	X	X	X	X	X	
IFood	X	X	X	X	X	X	
UberEats	X	X		X	X		
BOM *DELIVERY*	X			X	X		X
Protótipo Aplicativo Lanches	X	X		X			
CCFOOD				X	X		X
Chico's *Pizza*				X			X

Fonte: Elaborado pelo Autor, 2023.

3.3 A solução

O aplicativo desenvolvido nesse trabalho busca resolver possíveis problemas com as entregas de bebidas e oferecer uma maior comidade para pedir produtos nos locais desejado e se beneficiar de todas as vantagens que um aplicativo possa trazer. O aplicativo foi desenvolvido com uma linguagem multiplataforma e será disponibilizado nas principais plataformas de aplicativos para *smartphone*.

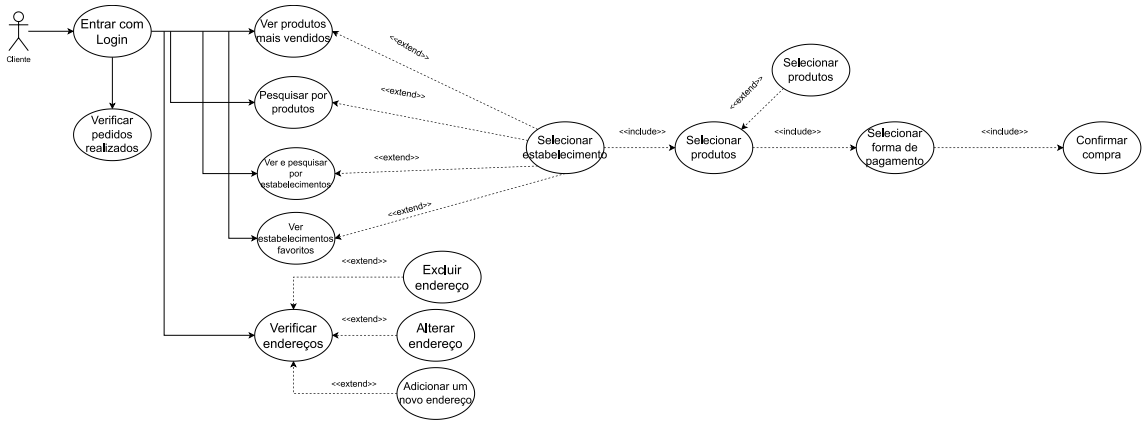
Para que todos os requisitos da solução sejam atendidos, o usuário deve possuir uma conexão a internet, uma vez que os produtos e estabelecimentos serão armazenados

em um banco de dados na nuvem. Dessa forma, alguns cuidados foram tomados em relação aos dados inseridos pelo usuário, como CPF, dados de endereço, formas de pagamento e entre outros. Os dados foram tratados para ficar conforme a Lei Geral de Proteção de Dados (LGPD) e ao entrar no aplicativo pela primeira vez, o usuário deve aceitar um termo e condições, no qual é informado como e quando os dados são utilizados.

A Figura 4 mostra o caso de uso para usuário no aplicativo. Para o usuário acessar o aplicativo deve realizar o cadastro via e-mail e senha pessoal. O uso do aplicativo deve ser de forma natural e intuitiva. Inicialmente, ao realizar o cadastro via e-mail e senha pessoal, é pedido também um endereço. A tela inicial permite ao usuário ver estabelecimentos ou produtos, ao escolher uma das opções, ele deve selecionar o produto ou estabelecimento, a quantidade do produto que deseja comprar e posteriormente na tela de confirmação de comprar, é selecionado a forma de pagamento e opcionalmente colocar alguma observação. Após todos os campos obrigatários serem preenchidos, deve-se confirmado a compra. Na tela inicial, é possível selecionar a opção para ver os estabelecimentos marcados como favorito.

No menu, existem as opções de verificar os endereços cadastrados e cadastrar um novo se assim desejar. Também há a opção de alterar a senha e verificar os pedidos realizados em um período selecionado.

Figura 4 – Caso de uso para o usuário.



Fonte: Elaborado pelo Autor, 2023.

3.4 Materiais e tecnologias

Nesta seção, são mostradas as tecnologias usadas, tanto *software* quanto *hardware*, tais tecnologias são possíveis verificar no Quadro 2.

3.5 Métodos e procedimentos

Na Seção 3.5.1, são vistos alguns conceitos importantes para modelar o sistema adequadamente. Diante do objetivo do trabalho de desenvolver um *software*, foi utilizada

Quadro 2 – Materiais e ferramentas

Processador	*Ryzen* 5 3400G
Memoria RAM	16 GB
Placa de vídeo	*NVIDIA GeForce GTX 1660 SUPER*
Memoria secundaria	1 TB HD 256SSD
Sistema operacional	*Windows 10 Pro — 21H2*
IDE	*Android Studio 2022.3.1 Patch 2¨*
Editor de Texto	*VSCode 1.83.1*
Framework	*Flutter 75.1.2*
Linguagem de programação	*Dart 223.8950*
Software Design Telas	Pencil 3.1.1
Banco de Dados	*Cloud Firestore*

Fonte: Elaborado pelo Autor, 2023.

a metodologia ágil para desenvolvimento de *software*, explicada em maior detalhe na Seção 3.5.2.

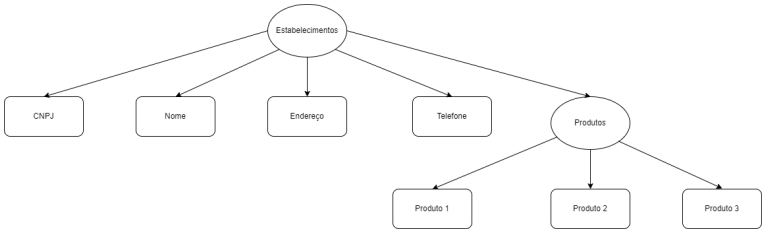
3.5.1 Modelagem do sistema

A modelagem do aplicativo foi dividida na criação de interface do usuário, onde foi utilizado a ferramenta Pencil. Na modelagem do banco de dados utilizando o padrão JSON.

O processo de *Design* para interfaces possui um papel crucial para a experiência dos usuários, por ser dessa forma que o usuário interage com o aplicativo, possibilitando o controle e acesso aos dados. As interfaces foram modeladas e desenvolvidas por meio de análises dos componentes e funcionalidades dos aplicativos de *delivery* existentes.

Para a modelagem do banco de dados foi utilizado o padrão JSON. Por se tratar de um banco de dados não relacional, não foi encontrada nenhuma ferramenta para realizar a modelagem graficamente. Desta forma, a modelagem com o padrão JSON pode ser realizada em um editor de texto, seguindo o padrão estabelecido para criação de coleções e documentos do *Firestore*. Na Figura 5, é mostrado um exemplo genérico de uma das possíveis forma de modelar os estabelecimentos para o banco de dados *Firestore*.

Figura 5 – Estrutura genérica para armazenamento de estabelecimentos no Banco de dados *Firestore*



Fonte: Elaborado pelo Autor, 2023.

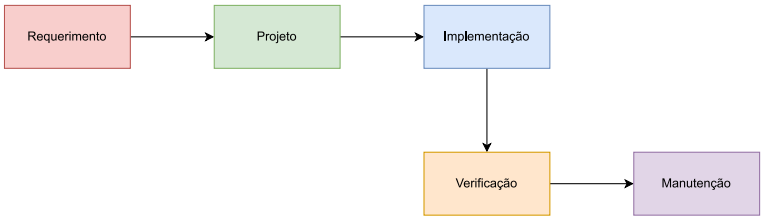
3.5.2 Método Scrum e Kanban

O quadro *Kanban* foi realizado na plataforma do *GitHub*, no qual também foi utilizado para fazer o versionamento do *software*. As tarefas no quadro *Kanban* foram organizadas da seguinte forma:

- **a fazer:** tarefas que devem ser inicializadas;
- **fazendo:** tarefas que já estão em processo de desenvolvimento;
- **revisão do código:** revisar como o sistema ficou após o desenvolvimento da tarefa;
- **concluídas:** tarefas que já foram analisadas e incrementadas no código-fonte

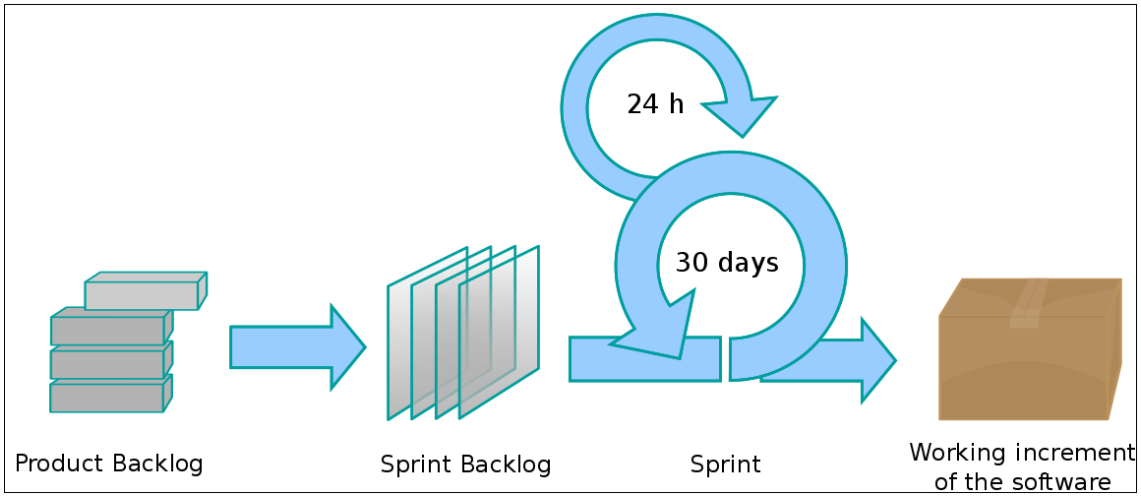
As atividades foram elaboradas pelo aluno e orientador, onde foi realizado o acompanhamento de todo o processo até a conclusão de cada tarefa. O acompanhamento foi realizado utilizando o método *Scrum* por meio de reuniões semanais com duração de 30 minutos, nas reuniões foram discutidos sobre o andamento das tarefas e quais os próximos passos a serem tomados tanto no desenvolvimento do trabalho, quanto no desenvolvimento do aplicativo.

Figura 6 – Processos do Modelo tradicional para Desenvolvimento de *Software*



Fonte: Elaborado pelo Autor, 2023.

Figura 7 – Processos do Método *Scrum* para Desenvolvimento de *Software*



Fonte: DEVMEDIA, 2023.

3.6 Etapas do trabalho

No desenvolvimento do código-fonte foi utilizado o *Framework Flutter* e a linguagem de programação *Dart*. Ambos foram desenvolvidos pela *Google LLC* e pelos seus serviços oferecidos, pela facilidade de integração foi utilizado alguns dos destes serviços do *Firebase* desenvolvido pela mesma empresa.

Inicialmente, o projeto deveria possuir alguns requisitos analisados nos aplicativos de sucesso no mercado com o mesmo contexto, com foco em requisitos considerados diferentes.

Foi feito o projeto de interfaces utilizando a ferramenta *Pencil*. A modelagem do banco de dados ocorreu de forma que fosse possível o aplicativo escalar, sem a necessidade de uma futura manutenção na arquitetura. Ela respeitou as restrições de armazenamento do *Firestore*, em que cada coleção pode possuir um ou mais documentos e um documento pode possuir apenas uma coleção, ou também denominado subcoleção.

Após a conclusão das interfaces do usuário e modelagem do banco de dados, foi iniciado o desenvolvimento do aplicativo utilizando as ferramentas citadas. Por ser um desenvolvimento de aplicativo para o usuário, os produtos e estabelecimentos para fins de testes de requisitos foram adicionados diretamente no banco de dados. Ao final foi testada a eficácia dos requisitos.

4 DESENVOLVIMENTO

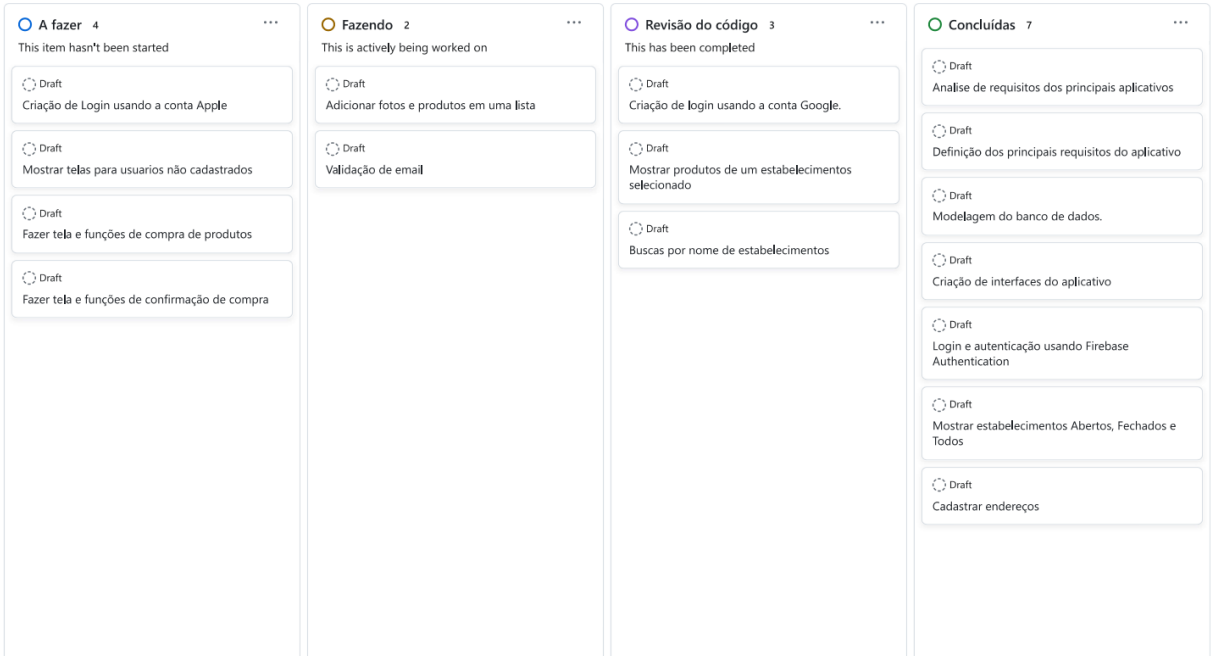
Este capítulo aborda o processo de desenvolvimento do trabalho, desde a análise de requisitos até o desenvolvimento do aplicativo. A Seção 4.1 apresenta como foram realizados o gerenciamento do projeto e a ferramenta utilizada. Na Seção 4.2, apresentam-se os detalhes da análise de requisitos, e a Seção 4.3 demonstra como foi realizada a modelagem do banco de dados.

4.1 Gestão do projeto

A gestão das atividades do projeto foi monitorada pelo *Kanban* do GitHub. Inicialmente, foi criado um repositório no *GitHub* para armazenar todo o código desenvolvido. Com essa mesma ferramenta, foi possível criar um quadro *Kanban* para monitorar o estágio de cada atividade. A Figura 8 mostra um estado do quadro *kanban* do projeto com as colunas *A fazer*, *Fazendo*, *Revisão do código* e *Concluídas*.

A utilização do *Kanban* foi importante para dar prioridade às atividades a serem realizadas e acompanhar aquelas em execução. Dessa forma, foi possível analisar eventuais problemas e fazer correções para que o fluxo das atividades não fosse prejudicado. A visualização clara do progresso das tarefas permite planejamento e acompanhamento mais eficazes do trabalho.

Figura 8 – Tarefas *Kanban*



Fonte: Elaborado pelo Autor, 2023.

4.2 Desenvolvimento dos requisitos

Nesta seção, são mostrados os protótipos das interfaces desenvolvidas, sendo detalhadas todas as ações possíveis em cada uma das telas.

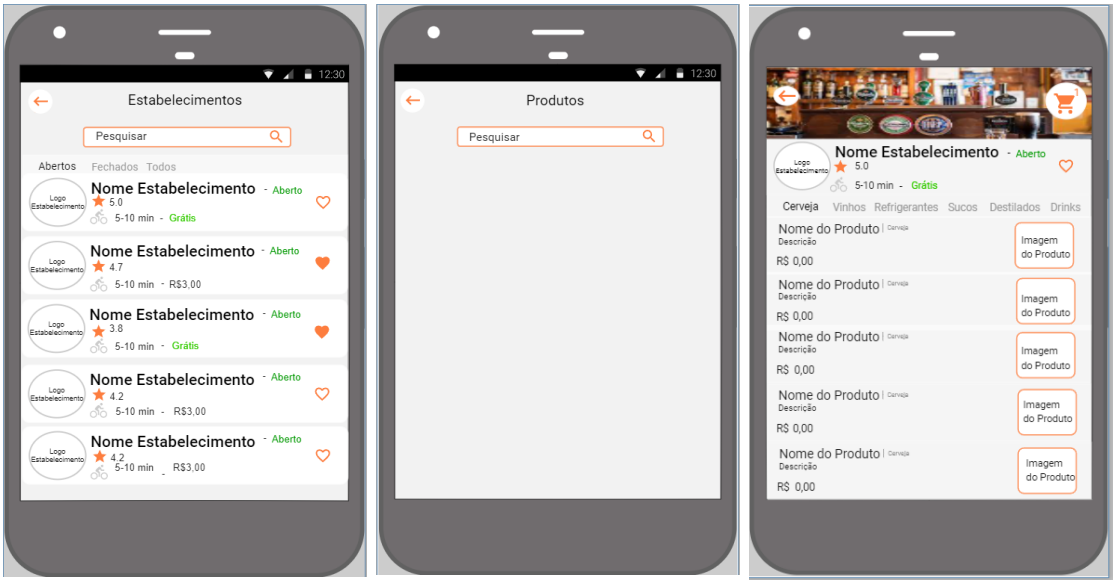
4.2.1 Mostrar estabelecimentos por localidade

Ao se cadastrar no aplicativo, com suas credenciais, é pedido que o usuário escolha uma cidade para cadastrar seu endereço. Este primeiro endereço é considerado o principal, sinalizado para o usuário na tela de "meus endereços". Com as cidades salvas nos endereços, é possível realizar um filtro na coleção de estabelecimento, e, como resultado, serão exibidos todos os estabelecimentos que possuírem cadastro com aquela cidade. Caso o usuário possua outros endereços cadastrados, com cidades diferentes, ao selecionar um endereço, serão mostrados os estabelecimentos da cidade correspondente ao endereço selecionado. Essa funcionalidade está presente nas telas de estabelecimentos, produtos, detalhes do estabelecimento. As Figuras 9(a),9(b) e 9(c) mostram as respectivas telas.

Na tela de estabelecimentos, é possível verificar todos os estabelecimentos cadastrados na cidade. A primeira lista mostra todos os abertos, e, ao lado, é possível ver todos os fechados. Por último, há uma opção para mostrar ambos. Caso deseje, o usuário pode procurar por um estabelecimento específico pela barra de pesquisa, como representando na Figura 9(a).

Na tela de pesquisa de produtos, o usuário pode buscar um produto por meio de uma barra de pesquisar. O resultado da pesquisa retorna os estabelecimentos que vendem o produto, ordenados pelo preço, como representando na Figura 9(b).

Figura 9 – Telas de estabelecimentos, produtos e detalhes do estabelecimento.



(a) – Estabelecimentos (b) – Produtos (c) – Detalhes

Fonte: Elaborado pelo Autor, 2023.

Ao selecionar um estabelecimento, é possível verificar todos os produtos separados por um filtro de tipos. Ao selecionar um tipo, são mostrados produtos referente a ele. A tela é composta por cartões de produtos, sendo que, em cada cartão, são informados o nome do produto, o tipo, uma breve descrição, o valor unitário e uma imagem referente ao produto, como representando na Figura 9(c).

4.2.2 Pesquisa por estabelecimentos ou produtos

É possível pesquisar por estabelecimentos ao selecionar o botão para mostrá-los na tela *home*. A nova tela de estabelecimentos, além de mostrar todos, possui um campo de pesquisa onde é possível pesquisar o estabelecimento pelo seu nome. Para cada resultado da pesquisa, são mostrados os nomes dos estabelecimentos, se está aberto ou fechado, tempo de entrega e avaliações dos usuários.

Caso o usuário escolha ver os produtos na tela *home*, é mostrada uma barra de pesquisa onde é possível pesquisar pelo nome de um produto. O resultado da pesquisa irá retornar os estabelecimentos que possuem o produto ordenados pelo menor valor. Dessa forma, é necessário realizar uma pesquisa composta pelos campos de nome e valor de cada estabelecimento. Essas funcionalidades estão presentes nas telas de estabelecimentos e produtos. As Figuras 9(a) e 9(b) mostram as respectivas telas.

4.2.3 Colocar observações nas compras

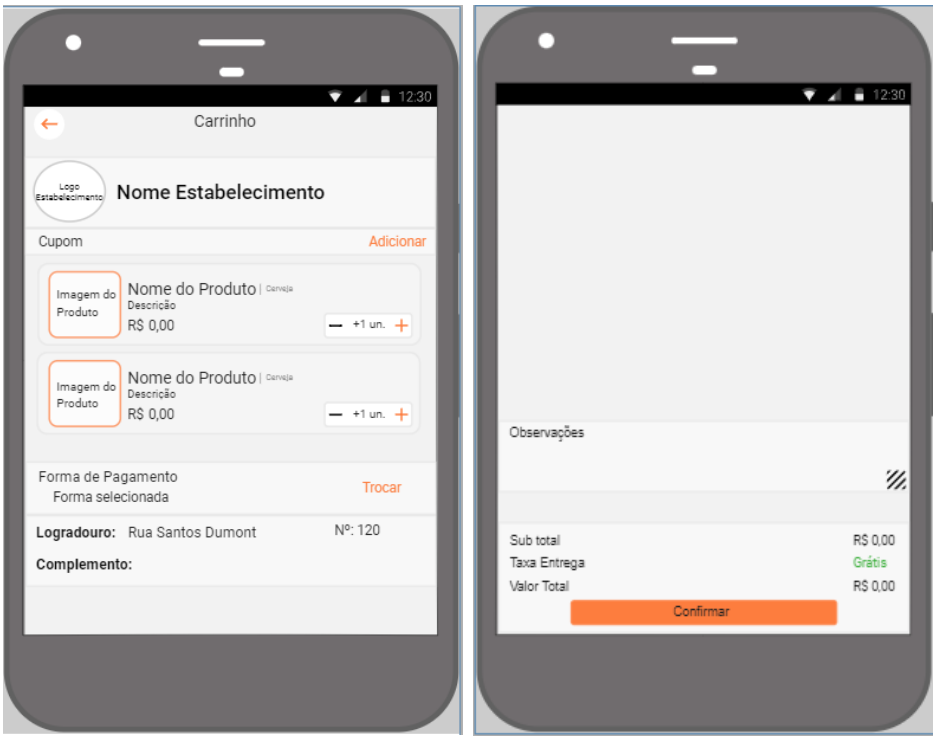
Antes de finalizar a compra, o usuário terá a opção de adicionar observações que ele achar necessário. Por exemplo, se quiser a bebida na temperatura ambiente, ou observações em relação à entrega, ou valores do pagamento. O preenchimento do campo não é obrigatório, e, nesse caso, ficará em branco. Essa funcionalidade, presente nas telas de confirmação da compra, é mostrada nas Figuras 10(a) e 10(b), as quais exibem as respectivas telas.

As telas mostram todas as informações da compra, incluindo os produtos selecionados e o endereço de entrega. Nessa tela, o usuário deve selecionar a forma de pagamento desejada, sendo que, no final, há um campo de observações disponível, caso o usuário queira adicionar informações complementares.

4.2.4 Histórico de pedido

Toda ordem de pedido contém as informações sobre os nomes dos produtos comprados, como valor unitário, quantidade, valor da taxa de entrega, dia e horário da finalização da compra. Se assim desejar, o usuário poderá verificar os detalhes das compras, sendo exibidos o nome do estabelecimento, a logomarca, o endereço de entrega e a forma de pagamento utilizada.

Figura 10 – Tela de confirmação de compra.



(a) – Tela do carrinho (b) – Tela do carrinho 2

Fonte: Elaborado pelo Autor, 2023.

O histórico de pedidos é realizado por períodos; logo, o usuário deverá escolher uma data inicial e uma final, e o resultado serão todos os pedidos efetuados naquele intervalo de tempo. Essa funcionalidade está presente na tela de "meus pedidos". A Figura 11(a) mostra o projeto da tela. Na tela, é possível ver o histórico de pedidos do usuário. Inicialmente, é uma tela vazia; após a escolha de uma data final e inicial, são mostrados todos os pedidos daquele período.

4.2.5 Múltiplos endereços

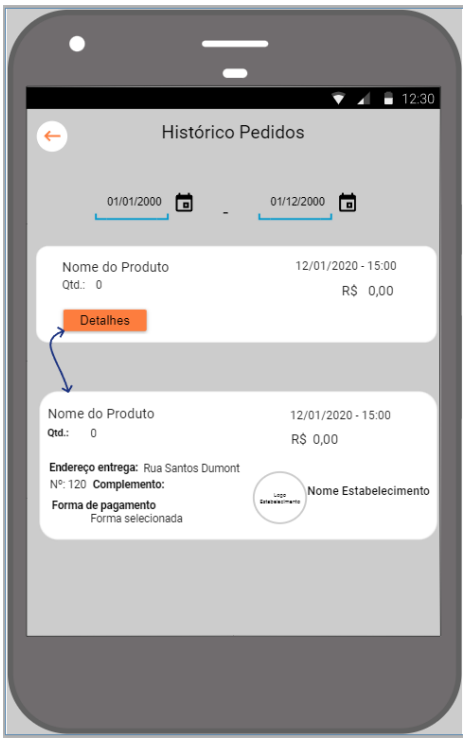
A funcionalidade de múltiplos endereços permite que o usuário cadastre mais de um endereço sem restrições de cidades. Isso significa que, mesmo se houver mais de um endereço cadastrado com a mesma cidade, não afetará a funcionalidade. Caso o limite de endereços seja alcançado e seja necessário cadastrar outro endereço, deverá ser realizada a exclusão de um registro.

É obrigatório que o usuário tenha um endereço selecionado, o qual é visível por meio de um ícone verde. Os demais endereços serão sinalizados com um ícone vermelho. Essa funcionalidade está presente na tela de "meus endereços". A Figura 11(b) mostra o projeto da tela.

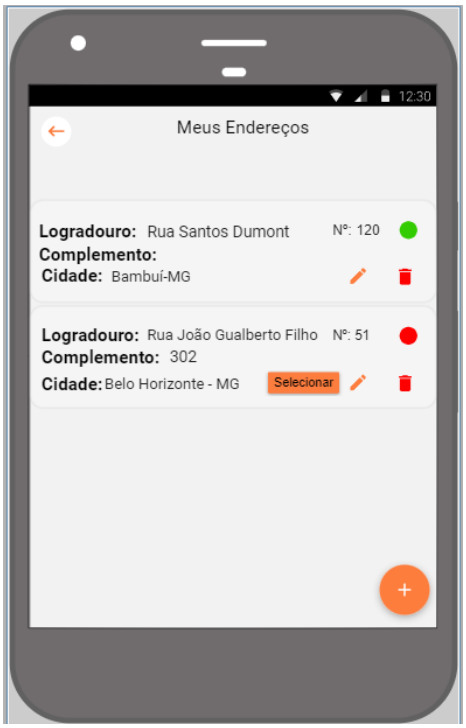
Cada endereço é composto pelo logradouro, número, cidade e complemento, se houver, e cada um possui suas ações de deletar e editar. Caso um endereço seja selecionado, é mostrado um círculo verde, e, se não for, são exibidos um círculo vermelho e um botão

para selecioná-lo. No canto inferior direito, há um botão para adicionar um novo endereço.

Figura 11 – Tela de pedidos e endereços cadastrados



(a) – Tela de pedidos



(b) – Tela de endereços

Fonte: Elaborado pelo Autor, 2023.

4.2.6 Login e cadastro

A Figura 12(a), inicialmente, mostra componentes nos quais é possível o usuário cadastrar uma nova conta por um *e-mail* e senha ao clicar no texto "criar uma nova conta".

A Figura 12(b) possui os campos necessários para o usuário preencher e cadastrar uma nova conta. Nessa tela, a única opção que não é obrigatória é o campo de complemento, no qual há a possibilidade de essa informação não ser relevante para o usuário.

4.2.7 Home e menu

A Figura 13(a) mostra dois botões iniciais, nos quais é possível o usuário realizar as pesquisas por produtos, verificar os estabelecimentos cadastrados na cidade e também pesquisar por estabelecimento. Na barra de navegação, estão as opções para ir à tela *Home* ou buscar os estabelecimentos marcados como favoritos.

Na Figura 13(b), é mostrado o *Menu* para o usuário. Quando o usuário estiver logado, são exibidas as informações cadastradas, como Nome, Endereço e Telefone. Nas demais opções, o usuário é redirecionado para suas respectivas telas.

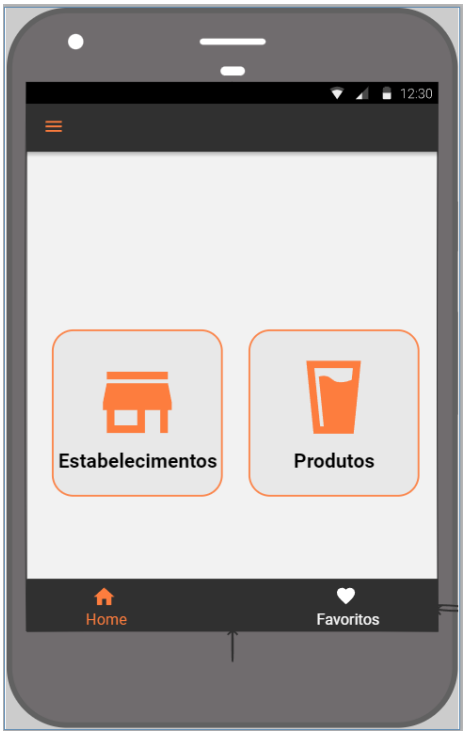
Figura 12 – Tela de *login* e registro

(a) – Tela de *Login*

(b) – Tela de Registro

Fonte: Elaborado pelo Autor, 2023.

Figura 13 – Tela de *home* e *menu*



(a) – Tela *Home*

(b) – Tela *Drawer Login*

Fonte: Elaborado pelo Autor, 2023.

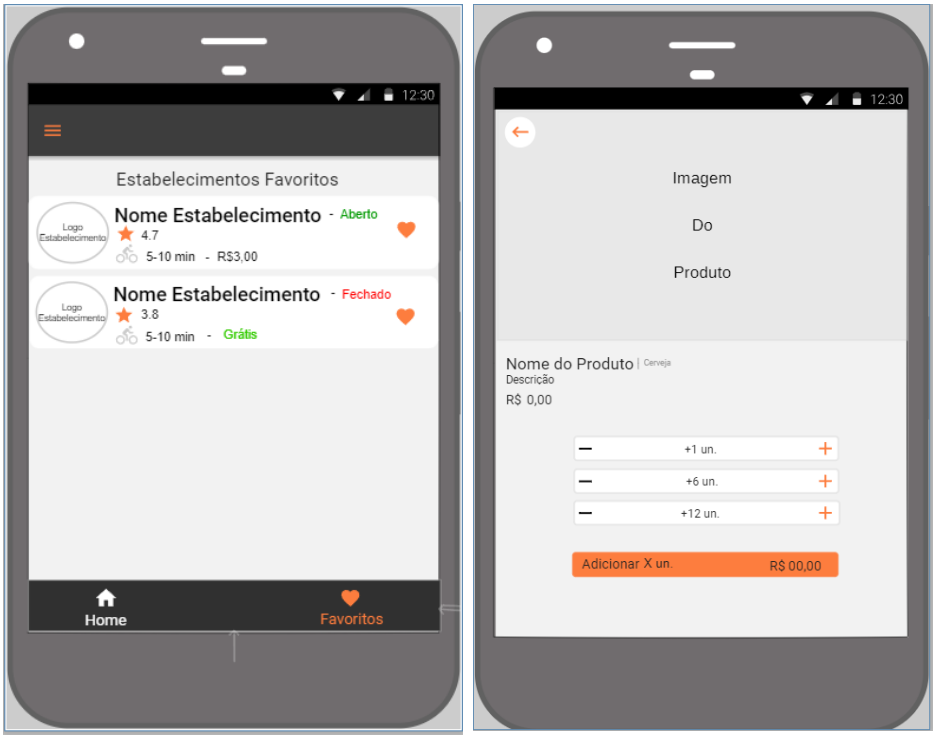
4.2.8 Telas de favoritos

A Figura 14(a) mostra os cartões dos estabelecimentos marcados como favoritos pelo usuário. Se o coração estiver com o fundo branco, significa que o estabelecimento não está como favorito; se estiver com o fundo laranja, que está como favorito. Os cartões de estabelecimento mostram informações do nome do estabelecimento, logomarca, se está aberto ou fechado, avaliação, tempo de entrega e valor da entrega.

4.2.9 Detalhes dos produtos

A Figura 14(b) representa os detalhes do produto quando selecionado. Nessa tela, são exibidos a imagem do produto, o nome, o tipo, uma breve descrição e o valor unitário. Por padrão, é possível acrescentar unidades adicionais do produto, com opções de mais 1, mais 6 e mais 12. O botão de adicionar mostrará o valor total em unidades.

Figura 14 – Tela de estabelecimentos favoritos e detalhes do produto



(a) – Estabelecimentos favoritos (b) – Detalhes produtos

Fonte: Elaborado pelo Autor, 2023.

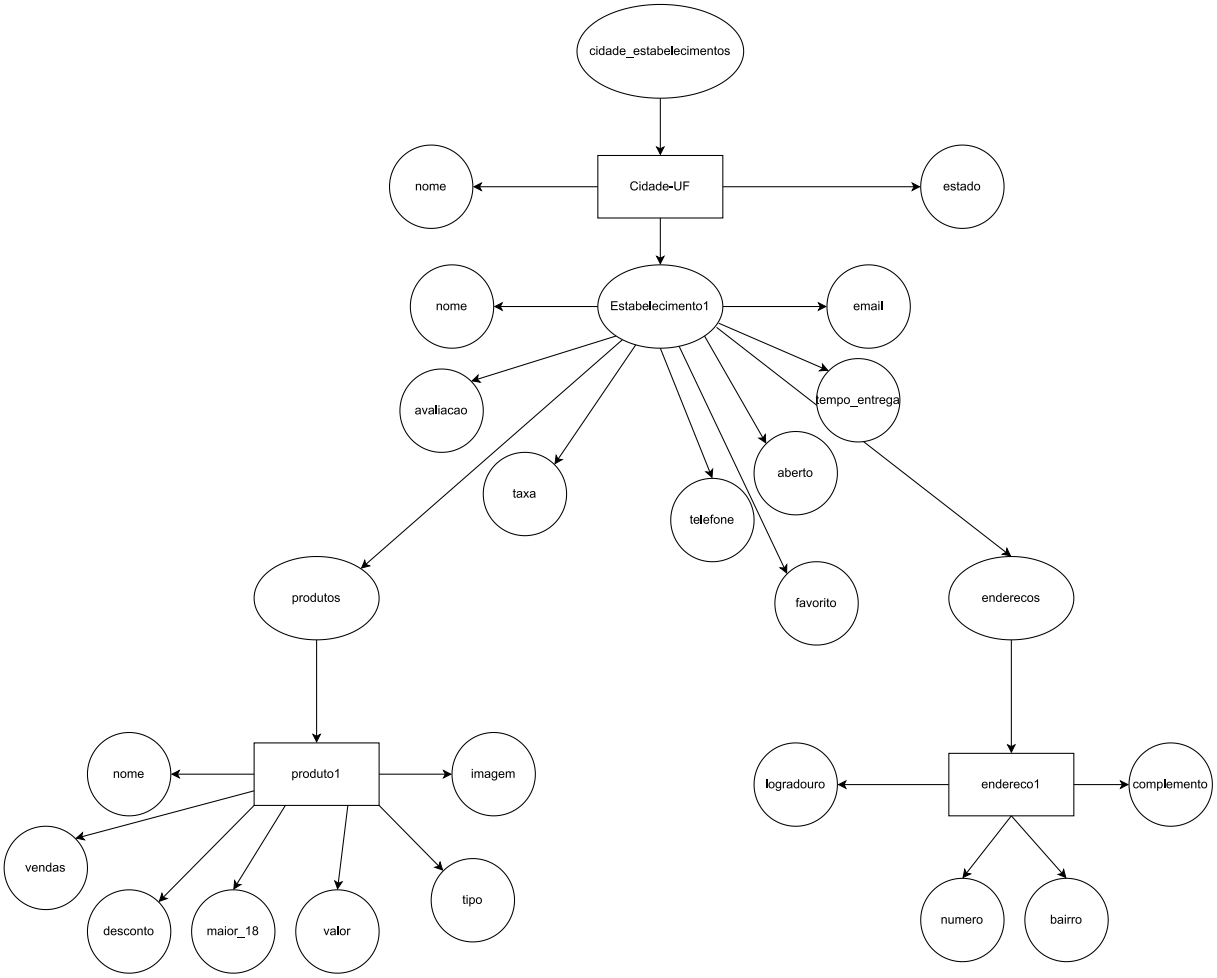
4.3 Modelagem do banco de dados

Para a modelagem do banco de dados, optou-se pelo uso do padrão JSON, por meio do editor de texto *VSCode*. Os documentos são armazenados no *Firestore* seguindo o mesmo padrão JSON. Devido à falta de uma ferramenta gráfica de modelagem específica para bancos de dados *NoSQL*, adotou-se esse padrão para a modelagem.

Durante a modelagem, foi identificado um cenário em que ocorrem duplicações de dados, especificamente com relação aos endereços selecionados pelos usuários. Os endereços são cadastrados em uma subcoleção específica para armazená-los, mas também são incluídos diretamente no documento principal do usuário. Essa abordagem foi adotada visando facilitar a manipulação desses dados em diferentes contextos. Essa duplicação de dados é benéfica no momento de utilizar as informações relacionadas aos endereços, como na tela de confirmação de pagamentos ou ao filtrar os estabelecimentos conforme a cidade cadastrada no endereço.

Essa estratégia permite uma melhor eficiência na recuperação e utilização desses dados, uma vez que não seria necessário fazer consultas adicionais em outras coleções para se obter as informações essenciais para esses casos específicos. No Apêndice A, é mostrado como foi realizada a modelagem do banco de dados, e as Figuras 15, 16 e 17 exibem os diagramas das respectivas modelagens.

Figura 15 – Diagrama da modelagem de cidades por estabelecimento



Fonte: Elaborado pelo Autor, 2023.

Para armazenar os estabelecimentos, foi criada uma coleção que pode ser denominada “coleção raiz”, com nome `cidades_estabelecimentos`. Tal coleção indica

cada cidade e seus estabelecimentos correlacionados. Entretanto, cada cidade é um documento que possui uma subcoleção de estabelecimentos, facilitando, assim, o filtro de pesquisa dos estabelecimentos.

Todo estabelecimento possui informações básicas, como CNPJ, nome, telefone, entre outras, as quais são consideradas atributos de chave e valor do documento. Já para armazenar os produtos de um estabelecimento, foi desenvolvida uma subcoleção com o nome "produtos", na qual é possível armazenar cada produto; assim, um produto é um documento da subcoleção de produtos.

Para atender ao requisito de pesquisar um produto e correlacioná-lo com o estabelecimento que o possui, foi necessária a criação de uma nova coleção raiz `produtos_estabelecimentos`. Todos os seus documentos são baseados nos nomes das cidades, e, posteriormente, todos os documentos de uma cidade possuirão uma subcoleção de nomes de produtos que armazenam todos os produtos daquela cidade.

Para armazenar os dados do cliente, foi necessária a criação de uma terceira coleção raiz com o nome "clientes". Cada cliente é armazenado por um documento que contém suas informações inscritas na tela de registro.

Para atender ao requisito de o usuário ter múltiplos endereços, foi criada uma subcoleção para registrá-los. Cada endereço possui um atributo lógico que determina se ele está como selecionado ou não. Dessa forma, é possível controlar em qual endereço o usuário deseja que a entrega seja realizada. A ordem de pedidos é uma subcoleção do cliente, onde são armazenados os pedidos finalizados.

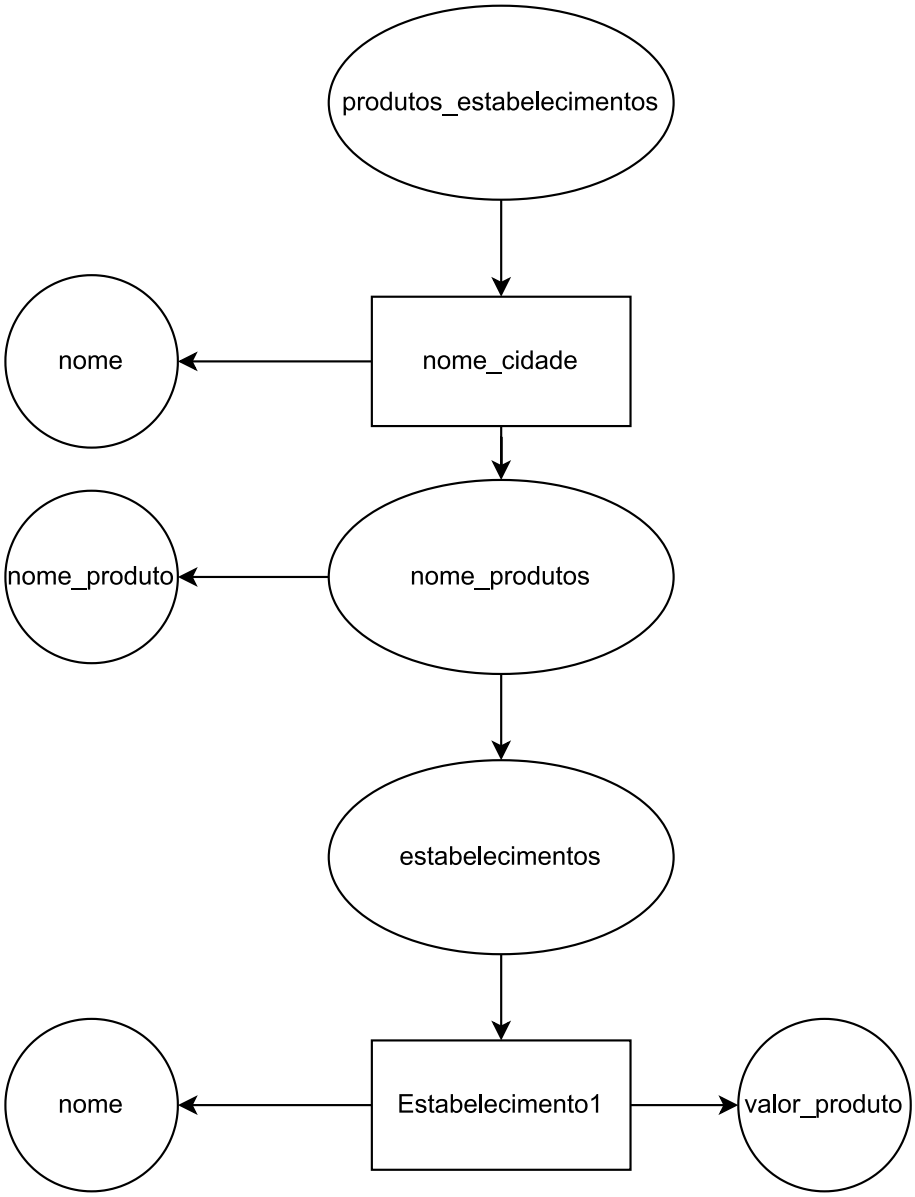
4.4 Detalhes da implementação

Esta seção descreve alguns detalhes da implementação do aplicativo, utilizando a IDE *Android Studio*, linguagem de programação *Dart*, *framework Flutter* e os serviços *Cloud Firestore* e *Firebase Authentication*.

4.4.1 Criação do projeto

O *Android Studio* permite o desenvolvimento com *Flutter* por meio da instalação de um *plugin*. Quando o projeto é criado, por padrão, são geradas as pastas `lib`, *android*, *ios*, *linux*, *macos*, *web* e *windows*. A pasta `lib` é o local em que os códigos *Dart* são armazenados, e, nas demais pastas, armazenam-se os arquivos de configuração para cada plataforma. Um exemplo do uso desses arquivos de configuração é quando o aplicativo necessita utilizar a câmera do celular, ou o GPS, sendo necessário incluir as permissões nesses arquivos. Também é criado o `pubspec.yaml`, arquivo no qual os desenvolvedores devem colocar as bibliotecas externas utilizadas no projeto. Na Figura 19(a), exibem-se essas pastas e arquivos.

Figura 16 – Diagrama da modelagem de produtos por estabelecimento

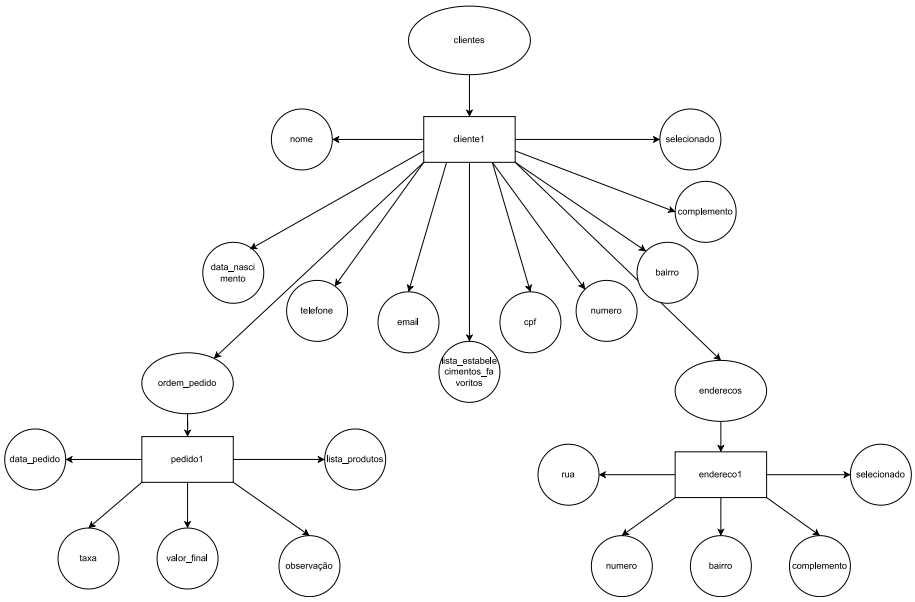


Fonte: Elaborado pelo Autor, 2023.

4.4.2 Bibliotecas externas

As bibliotecas externas utilizadas no aplicativo são mostradas na Figura 19(b). Elas são incluídas no arquivo `pubspec.yaml`, sendo que, posteriormente, é utilizado o comando `pub get`. `Pub` é o gerenciador de pacotes dos projetos *Flutter*; assim, quando é adicionada uma nova dependência ao projeto, é necessário usar o comando `pub get` para a biblioteca ser adicionada ao projeto. O *Flutter* armazena as versões de cada dependência no arquivo `pubspec.lock`. Desse modo, quando se utiliza o `pub get`, é garantido que as mesmas versões são adicionados ao projeto em diferentes ambientes.

Figura 17 – Diagrama da modelagem de dados do cliente



Fonte: Elaborado pelo Autor, 2023.

Figura 18 – Pastas iniciais e bibliotecas utilizadas

```
.dart_tool
.idea
android
ios
lib
    main.dart
linux
macos
test
web
windows
analysis_options.yaml
pubspec.lock
pubspec.yaml .1 README.md
```

(a) – Pastas iniciais

```
dependencies:
  flutter:
    sdk: flutter
  firebase_auth: ^4.6.2
  google_sign_in: ^6.1.4
  provider: ^6.0.5
  cloud_firestore: ^4.8.0
  get: ^4.6.5
  material_floating_search_bar_2:
  ↪  ^0.5.0
  shared_preferences: ^2.2.0
  badges: ^3.1.1
  intl: ^0.17.0
  fluttertoast: ^8.2.2
  auto_size_text: ^3.0.0
  mask_text_input_formatter: ^2.5.0


  cupertino_icons: ^1.0.2
```

(b) – Bibliotecas utilizadas

Fonte: Elaborado pelo Autor, 2023.

4.4.3 Construção de telas com widgets

No *Flutter*, as telas são desenvolvidas com *widgets*. Um exemplo de construção da tela com *widgets* é visto no código do Apêndice B.

No código, é mostrado como são criadas as telas utilizando-se *widgets*. Os *widgets* utilizados nesse código foram `Scaffold`, `SingleChildScrollView`, `Padding`, `EdgeInsets`, `Column`, `Row`, `SizedBox`, `TextField`, `InputDecoration`,

`UnderlineInputBorder, BorderSide, Color, TextStyle e Align`. Cada *widget* possui atributos que esperaram receber um valor ou outro *widget*. Por exemplo, o *widget* `Row` contém o atributo `mainAxisAligmnet`, que recebe um valor, `MainAxisAlignment.spaceBetween`, e também outro atributo, `children`, que recebe uma lista de *widgets*.

Essa tela é composta por campos que o usuário deseja atualizar em um endereço previamente selecionado. Cada campo possui um *widget* `TextField`, no qual é possível escrever um valor. Cada `TextField` é conectado a uma variável *controller*, que fica escutando alterações no campo.

O último *widget* da lista de *widgets* de `Column` é um `Container` com um filho `GestureDetector`. O `GestureDetector` possui vários atributos que capturam movimentos na tela. O atributo utilizado é o `onTap`, que espera receber uma função, sendo acionado quando seu filho é clicado. No código, quando o textttContainer é clicado, todos os valores dos campos são atualizados no *Cloud Firestore*, e o usuário é redirecionado para a tela de endereços cadastrados.

4.4.4 Navegação entre telas

O *Flutter* oferece a flexibilidade de implementar várias formas de navegação entre as telas. No Apêndice C, é apresentado o código `main.dart`, onde estão definidos nomes para os caminhos usados na navegação nomeada, juntamente com os respectivos contextos de cada página. Vale ressaltar que, nesse cenário, a rota inicial está configurada para direcionar o aplicativo para a tela de *Login*.

Também no código do Apêndice B, é possível observar uma das técnicas de navegação entre telas, na linha 19.

Nesse ponto específico, quando o usuário atualiza o endereço, o evento `onTap()` é acionado e executa a sua instrução. Essa abordagem é utilizada por se tratar de uma página nomeada.

Já no Apêndice D, é mostrado um código com exemplo da navegação entre telas, ao lidar com o evento `onPressed`, que espera receber uma função quando acionado, executando a linha 6. Essa chamada de função tem o efeito de retornar para a tela anterior na pilha de navegação, e, como resultado, passa a quantidade de itens na lista de compras.

4.4.5 Conexão com banco de dados e login

Para integrar o *Cloud Firestore* ao projeto *Flutter*, é necessário gerar um arquivo de configuração no console do *Firebase*, quando é criado um novo projeto. Ao gerar o arquivo, ele deve ser colocado na pasta mostrada na Figura 20. O arquivo `google-services.json` contém todas as informações para a configuração do *Cloud Firestore*. Após ter sido realizada a parte da configuração, no arquivo `main.dart`, é necessário

inicializar o *Cloud Firestore*, seguindo-se as linhas 2 e 3 do código mostrado no Apêndice E.

Após a inicialização do *Cloud Firestore*, é possível instanciar o contexto do banco de dados e utilizar os métodos disponibilizados pelo *Firestore*.

No cadastro do usuário e na autenticação de *login*, foi utilizado o *Firebase Authentication*, um serviço disponibilizado pelo *Firebase*. Com o uso deste serviço, é possível a autenticação utilizando senhas, números de telefones, contas do *Google*, do *Facebook*, do *Twitter*, do *GitHub*, *e-mails* e senhas.

Neste trabalho, a autenticação foi implementada por *e-mail* e senha. Ao realizar o *login* pela primeira vez, o *Firebase Authentication* permite que os *logins* subsequentes sejam realizados de forma automática. Para efetuar o cadastro usando *e-mail* e senha, basta apenas usar o método `createUserWithEmailAndPassword()`, do *Firebase Authentication*, e passar os valores do `TextField` de *e-mail* e senha.

4.5 Alcance dos objetivos específicos e objetivo geral

Os requisitos da solução foram definidos mediante a análise de alguns aplicativos consolidados no mercado de aplicativos de *delivery*.

O segundo objetivo específico foi alcançado após a definição dos requisitos, utilizando-se a ferramenta *Pencil* para o desenvolvimento dos protótipos das telas.

Posteriormente, analisou-se melhor a arquitetura do banco de dados, e, como o armazenamento ocorre nele, foi possível realizar a modelagem no banco de dados utilizando o padrão *JSON*, alcançando o terceiro objetivo.

Ao final, foi implementado o aplicativo e gerado um *APK*, no qual foi possível testar todas as funcionalidades. Como mencionado na Seção 1, os estabelecimentos e produtos foram inseridos diretamente no banco de dados por meio do *console* do *Firebase*.

Figura 20 – Pasta de configuração do *Cloud Firestore*

```
.dart_tool
.idea
android
    .gradle
    app
        src
            build.gradle
            google-services.json
        gradle
```

Fonte: Elaborado pelo Autor, 2023.

4.6 Desafios encontrados no desenvolvimento

Durante o desenvolvimento deste trabalho, foram encontradas algumas dificuldades, especialmente na modelagem do banco de dados e em determinadas etapas do desenvolvimento do sistema.

A modelagem do banco de dados precisava ocorrer de forma que a escalabilidade não se tornasse um problema futuro para o cadastro de estabelecimentos em diversas cidades. Diante disso, a solução encontrada foi a criação de uma coleção específica para atender às cidades, em que cada cidade é representada por um documento. Outro desafio encontrado na modelagem foi a forma de armazenar os endereços cadastrados e selecionados pelo usuário, cuja solução foi a criação de uma subcoleção para os endereços, e o endereço selecionado teria seus dados armazenados no documento do usuário, sendo essa uma abordagem que visa evitar acessos desnecessários sempre que o usuário acessa o aplicativo.

Na implementação do código-fonte, houve alguns desafios a serem solucionados. O primeiro foi a implementação da barra de pesquisa por produtos filtrada por estabelecimentos e com o menor valor. Esse problema ocorre, uma vez que a pesquisa deve conter um filtro com coleções distintas. Outro desafio foi a navegação entre as telas, no qual, dependendo da ação do usuário, o carrinho deveria ter os itens removidos ou mostrar a quantidade de produtos nele. Um exemplo é quando o usuário seleciona e confirma a quantidade de um produto - ele deverá ser direcionado para a tela de detalhes do estabelecimento, mostrando a quantidade de produtos adicionados no carrinho. Para resolver este problema, a quantidade de produtos foi passada por parâmetro no método de navegação da pilha, sendo atualizado o estado atual da tela de detalhes do estabelecimento.

5 CONSIDERAÇÕES FINAIS

Ao final do prazo estipulado para a entrega do trabalho, todas as funcionalidades do aplicativo do cliente foram cumpridas e, após o teste das funcionalidades, foi possível verificar que todas cumprem suas ações corretamente. A funcionalidade de cadastrar um novo endereço e selecionar qual endereço deve ser entregue à bebida reflete corretamente na tela de confirmação de compra. O requisito de mostrar os estabelecimentos a partir da localidade escolhida acontece no momento em que o usuário cadastra o endereço na tela de cadastro de um novo usuário ou quando ele seleciona um endereço diferente na tela de endereços cadastrados.

O histórico de pedidos é mostrado ao usuário após a seleção de um período com uma data inicial e uma data final. Assim, é realizada uma busca que contém todos os pedidos no período. O requisito de pesquisar por estabelecimento ou produto é atendido em telas separadas. Quando o usuário busca por um produto, são listados todos os estabelecimentos que contêm esse produto, do menor valor ao maior valor. O requisito de adicionar observações na compra é atendido na tela de confirmação de compra, sendo possível ou não adicionar alguma observação.

O trabalho se estende além do desenvolvimento do aplicativo, e busca preencher uma lacuna existente nas cidades na qual não há plataformas de *delivery* de bebidas, uma realidade comum em muitas cidades do Brasil. O aplicativo visa trazer comodidade para o usuário e contribuir para o crescimento econômico comércio local.

5.1 Trabalhos futuros

Como trabalho futuro, será realizado o desenvolvimento do aplicativo para o lado do estabelecimento, no qual todas as configurações do estabelecimento que aparecem no aplicativo do cliente serão manipuladas pelo próprio estabelecimento. O desenvolvimento de novas funcionalidades no aplicativo do cliente, adicionando possibilidades de realizar o cadastro com as contas do *Google* e *Apple*, a criação de uma nova tela para usuários que esqueceram a senha.

Também será necessário realizar a integração dos aplicativos, uma vez que o pedido feito pelo cliente deverá ser aceito no estabelecimento, mostrando ao cliente todo o processo de entrega, por exemplo, se o pedido foi aceito, se o pedido está a caminho e se o pedido foi entregue.

REFERÊNCIAS

ADAM, J. d. S. Estratégia de marketing com utilização de aplicativos de terceiros nas empresas. **Repositório Universitário da Ânima (RUNA)**, 2019. Disponível em: https://repositorio.animaeducacao.com.br/handle/ANIMA/3633. Acesso em: 23 fev. 2023.

ALMEIDA, W. P. de *et al.* Desenvolvimento de um aplicativo de food service para pizzarias do município de Princesa Isabel-PB. **Research, Society and Development**, Itajubá-MG, v. 12, n. 3, 2023. DOI: https://doi.org/10.33448/rsd-v12i3.40697.

AMAZON WEB SERVICES (AWS). **Caso prático do Rappi**. 2022. Disponível em: https://aws.amazon.com/es/solutions/case-studies/Rappi/. Acesso em: 28 fev. 2023.

_____. **Zé Delivery cria serviço de entrega de bebidas na AWS e sustenta seu crescimento exponencial**. 2021. Disponível em: https://aws.amazon.com/pt/solutions/case-studies/ze-delivery/. Acesso em: 24 fev. 2023.

BOTELHO, L. V.; CARDOSO, L. d. O.; CANELLA, D. S. COVID-19 and the digital food environment in Brazil: Reflections on the pandemic's influence on the use of food delivery apps. **Cadernos de Saúde Pública**, SciELO Brasil, São Paulo, v. 36, 2020. DOI: https://doi.org/10.1590/0102-311X00148020. Acesso em: 23 fev. 2023.

CARVALHO, H. G. d.; REIS, D. R. d.; CAVALCANTE, M. B. **Gestão da Inovação**. Curitiba-PR: Aymará Educação, 2011.

DART. **Paint your UI to life**. 2023. Disponível em: https://dart.dev/. Acesso em: 24 jun. 2023.

DEVMEDIA. **Desenvolvimento ágil com Scrum:** uma visão geral. Rio de Janeiro-RJ. Disponível em: https://www.devmedia.com.br/desenvolvimento-agil-com-scrum-uma-visao-geral/26343#modulo-mvp. Acesso em: 22 ago. 2023.

ENGHOLM JUNIOR, H. **Engenharia de software na prática**. São Paulo-SP: Novatec Editora, 2010.

FONTANA JUNIOR, S. A. **Protótipo de um aplicativo android para pedidos de lanches e um portal Web para gestão e monitoramento**. 2013. Trabalho de Conclusão de Curso (Graduação em Sistema de Informação) – Universidade do Planalto Catarinense, Lages-SC, 2013.

GERHARDT, T. E.; SILVEIRA, D. T. **Métodos de pesquisa**. Porto Alegre-RS: Editora da UFRGS, 2009.

IBGE. **Internet chegou a 90% dos domicílios brasileiros no ano passado**. 2022. Disponível em: https://www.gov.br/pt-br/noticias/educacao-e-pesquisa/2022/09/internet-chegou-a-90-dos-domicilios-brasileiros-no-ano-passado. Acesso em: 28 fev. 2023.

IFOOD. **Um plano e várias vantagens para você**. 2021. Disponível em: https://parceiros.ifood.com.br/restaurante/planos. Acesso em: 19 jun. 2023.

INSTITUTO FOODSERVICE BRASIL (IFB). **Delivery IFB**. Disponível em: https://foodbizbrasil.com/wp-content/uploads/2021/04/IFB-Delivery-2020.pdf. Acesso em: 24 fev. 2023.

EL-KASSAS, W. S. *et al.* Taxonomy of Cross-Platform Mobile Applications Development Approaches. **Ain Shams Engineering Journal**, ScienceDirect, Cairo-Egito, v. 4, p. 869–877, 2017. DOI: https://doi.org/10.1016/j.asej.2013.03.005.

KNIBERG, H. **Scrum y XP desde las trincheras**. C4Media, editora da InfoQ, 2007. p. 122.

KOHLBECK, E. *et al.* Elaboração de um aplicativo de delivery usando como base o ciclo de vida de sistemas produto-serviço. **Produto & Produção**, Rio Grande do Sul, v. 22, n. 1, p. 98–116, 2021.

LITAYEM, N.; DHUPIA, B.; RUBAB, S. Review of cross-platforms for mobile learning application development. **International Journal of Advanced Computer Science and Applications**, Reino Unido, v. 6, n. 1, 2015. DOI: doi.org/10.14569/IJACSA.2015.060105.

LÓSCIO, B. F.; OLIVEIRA, H. R. D.; PONTES, J. C. d. S. NoSQL no desenvolvimento de aplicacões Web colaborativas. In: SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS (SBSC), VIII., 2011, Rio de Janeiro. **Anais do VIII Simpósio Brasileiro de Sistemas Colaborativos**. Paraty-RJ, 2011. p. 11.

MALLIK, R. *et al.* Development of an android application for viewing covid-19 containment zones and monitoring violators who are trespassing into it using firebase and geofencing. **Transactions of the Indian National Academy of Engineering**, v. 5, p. 163–179, 2020. Acesso em: 12 abr. 2023.

MELLO, M. B. S. **Você tem forme de quê?:** Análise da distribuição espacial dos principais aplicativos de delivery no Brasil. 2020. Trabalho de Conclusão de Curso (Graduação em Geografia) – Universidade Federal Fluminense Instituto de Geociências, Niterói-RJ, 2020.

MELO, C. F. d. A. **Bom Delivery:** Um aplicativo de delivery genérico para cidades do interior. 2019. Trabalho de Conclusão de Curso (Graduação em Ciência de Computação) – Universidade Federal de Campina Grande, Campina Grande-PB, 2019.

MELO, M. J.; FERREIRA, W. M. **CFFOOD:** Desenvolvimento de Aplicativo de Delivery para a Cidade de Córrego Fundo. 2021. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro universitário de Formiga, Formiga - MG, 2020.

MOBILE TIME. **Comércio Móvel no Brasil**: Setembro de 2019. 2019. Disponível em: https://www.mobiletime.com.br/. Acesso em: 24 jun. 2023.

PREZOTTO, E. D.; BATISTA BONIATI, B. Estudo de Frameworks Multiplataforma Para Desenvolvimento de Aplicações Mobile Híbridas. In: ENCONTRO ANUAL DE TECNOLOGIA DA INFORMAÇÃO, V., 2014, Frederico Westphalen-RS. **Anais do EATI - Encontro Anual de Tecnologia da Inf e Semana Acadêmica de Tecnologia da Informação**. Santa Maria -RS: Universidade Federal de Santa Maria, 2014.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de gerenciamento de banco de dados**. 3. ed. São Paulo-SP: McGraw-Hill Interamericana do Brasil LTDA, 2008.

RAPPI. **Cadastre seu restaurante no Rappi**. 2022. Disponível em: https://merchants.rappi.com/pt-br/junte-se-a-nos. Acesso em: 28 out. 2023.

RIBEIRO, M. R. **Big Data Básico**. Belo Horizonte – MG: Instituto Federal de Minas Gerais, 2022. *E-book*.

SHAH, K.; SINHA, H.; PAYPAL, M. Analysis of Cross-Platform Mobile App Development Tools. In: 2019 IEEE INTERNATIONAL CONFERENCE FOR CONVERGENCE IN TECHNOLOGY (I2CT), V., 2019, Bombay, India. **Proceedings** [**...**] Piscataway, Nova Jersey, EUA: IEEE, 2019. p. 1–7. DOI: 10.1109/I2CT45611.2019.9033872. Acesso em: 24 fev. 2023.

SILVA, D. E. d. S.; SOUZA, I. T. de; CAMARGO, T. Metodologias Ágeis para o desenvolvimento de software: Aplicação e o uso da metodologia SCRUM em contraste ao modelo tradicional de Gerenciamento de Projetos. **Revista Computação Aplicada-UNG-SER**, Guarulhos-SP, v. 2, n. 1, p. 39–46, 2013.

SMUTNÝ, P. Mobile development tools and cross-platform solutions. In: PROCEEDINGS OF THE 13TH INTERNATIONAL CARPATHIAN CONTROL CONFERENCE (ICCC), XIII., 2012, High Tatras, Slovakia. **Proceedings** [**...**] Piscataway, Nova Jersey, EUA: IEEE, 2012. p. 653–656. DOI: 10.1109/CarpathianCC.2012.6228727. Acesso em: 24 fev. 2023.

SOARES, M. d. S. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, Campo Largo-PR, v. 3, n. 1, 2004. DOI: https://doi.org/10.21529/RESI.2004.0301006.

STATCOUNTER. **Operating System Market Share Worldwide**. Disponível em: https://gs.statcounter.com/os-market-share#monthly-200901-202303. Acesso em: 24 fev. 2023.

TAKAI, O. K.; ITALIANO, I. C.; FERREIRA, J. E. **Introdução a Banco de Dados**. São Paulo, SP, 2005.

TAVEIRO, F. T. **Análise do impacto de um requisito não funcional relacionado a Usabilidade**. 2016. Trabalho de Pós-graduação (Especialização em Gestão da Tecnologia) – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, São Paulo-SP, 2016.

TDP SISTEMAS (TDP). **Conheça as 5 principais vantagens de um aplicativo para Delivery**. Dois Córregos-SP, 2019. Disponível em: https://www.tdp.com.br/conheca-as-5-principais-vantagens-de-um-aplicativo-para-delivery/. Acesso em: 24 fev. 2023.

VALENTE, J. **Pesquisa revela aumento de pedidos de comida por app durante pandemia**: Alternativa permite economizar tempo, afirmam entrevistados. Brasília-DF, 2021. Disponível em: https://agenciabrasil.ebc.com.br/geral/noticia/2021-12/pesquisa-revela-aumento-de-pedidos-de-comida-por-app-durante-pandemia. Acesso em: 24 fev. 2023.

WAZLAWICK, R. S. **Metodologia de pesquisa para ciência da computação**. Rio de Janeiro-RJ: Elsevier, 2009.

APÊNDICES

APÊNDICE A – MODELAGEM DO BANCO DE DADOS NO FORMATO JSON

```
"cidades_estabelecimentos":{
   "Cidade-UF":{
       "nome": "Cidade",
       "estado": "UF",
       "estabelecimentos":{
           "Estabelecimento1" :{
               "nome": "Estabelecimento1",
               "avaliacao": 4.5,
               "telefone": "00-00000-0000",
               "email": "estabelecimento1@email.com",
               "cnpj": "00.000.000/0000-00",
               "taxa": 0.01,
               "aberto":true,
               "tempo_entrega": "5",
               "enderecos": {
                   "logradouro": "Rua Tal",
                   "numeroEndereco": "000",
                   "bairro": "Bairro",
                   "complemento": "Complemento"
               },
               "favorito": 0,
               "produtos": {
                   "Produto1" : {
                       "nome": "",
                       "valor": 1.00,
                       "tipo": "Tipo do produto",
                       "imagem": "imagem_do_produto",
                       "maior_18": true,
                       "desconto": 0.00,
                       "vendas": 0
                   },
                   "..." : "..."
               }
           },
           "..." : "..."
       }
   },
   "..." : "..."
},
```

```
"produtos_estabelecimentos":{
   "Nome_cidade":{
       "nome": "nome_cidade",
       "nome_produtos":{
           "nome_produto": "nome",
           "estabelecimentos":{
               "Estabelecimento1" :{
                   "nome": "Estabelecimento1",
                   "valor_produto": 4.5
               },
               "..." : "..."
       },
       "..." : "..."
       }
   },
   "..." : "..."
},
```

```
"clientes":{
    "cliente1":{
        "nome": "Nome Completo",
        "data_nascimento": "01/01/2000",
        "telefone": "DDD-9 9999-9999",
        "e-mai": "example@email.com",
        "cpf": "100.000.000-00",
        "rua": "Nome da rua",
        "numero": "numero casa",
        "bairro": "Bairro",
        "complemento": "Complemento e observações",
        "selecionado": true,
        "enderecos": {
            "endereco1": {
                "rua": "Nome da rua",
                "numero": "numero casa",
                "bairro": "Bairro",
                "complemento": "Complemento e observações",
                "selecionado": true
            },
            "..." : "..."
        },
       "ordem_pedido":{
           "pedido1":{
                "data_pedido":"dd/mm/yyyy hh:mm:ss",
                "taxa": 0,
                "valor_final": 100,
                "observacao": "obervações",
                "lista_produtos":[
                    "valor_final":100,
                    "nome":"Nome_produto",
                    "quantidade":2,
                    "valor":50
                ],
            },
            "...":"..."
        },
        "lista_estabelecimentos_favoritos": [
            "estabelecimento1",
            "..."
        ],
    },
    "...":"...",
}
```

APÊNDICE B – CÓDIGO DA TELA DE ATUALIZAR ENDEREÇOS

```
@override
      Widget build(BuildContext context) {
        var datas = widget.datas;
        var documentReference = widget.documentReference;
        return Scaffold(
          backgroundColor: const Color(0xFFDCDCDC),
          body: SingleChildScrollView(
            child: Padding(
              padding: const EdgeInsets.only(left: 25, right: 25, top: 15),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      SizedBox(
                        width: 200,
                        child: TextField(
                          decoration: const InputDecoration(
                            border: InputBorder.none,
                            enabledBorder: UnderlineInputBorder(
                                borderSide: BorderSide(
                                  color: Color(0xFFFD7D3E),
                                )
                            ),
                            hintText: 'Ex: Rua Tal',
                            labelText: 'Rua',
                            labelStyle: TextStyle(
                              color: Color(0xFFFD7D3E),
                            ),
                          ),
                          controller: addressControllerText,
                        ),
                      ),
                      SizedBox(
                        width: 100,
                        child: TextField(
                          decoration: const InputDecoration(
                            border: InputBorder.none,
                            enabledBorder: UnderlineInputBorder(
                                borderSide: BorderSide(
                                  color: Color(0xFFFD7D3E),
                                )
                            ),
                            hintText: 'Ex: 999',
                            labelText: 'Número',
                            labelStyle: TextStyle(
                              color: Color(0xFFFD7D3E),
                            ),
                          ),
                          controller: numberAddressController,
                        ),
                      ),
                    ],
                  ),
```

```
TextField(
    decoration: const InputDecoration(
        border: InputBorder.none,
        enabledBorder: UnderlineInputBorder(
```

```
        borderSide: BorderSide(
            color: Color(0xFFFD7D3E),
            )
        ),
        hintText: 'Ex: Bairro',
        labelText: 'Bairro',
        labelStyle: TextStyle(
        color: Color(0xFFFD7D3E),
        ),
    ),
    controller: neighborhoodController,
),
Align(
  alignment: Alignment.centerLeft,
  child: SizedBox(
    width: 200,
    child: TextField(
      decoration: const InputDecoration(
        border: InputBorder.none,
        enabledBorder: UnderlineInputBorder(
            borderSide: BorderSide(
              color: Color(0xFFFD7D3E),
            )
        ),
        hintText: 'Ex: Complemento',
        labelText: 'Complemento',
        labelStyle: TextStyle(
          color: Color(0xFFFD7D3E),
        ),
      ),
      controller: complementController,
    ),
  ),
),
```

```
Padding(
    padding: const EdgeInsets.only(top: 30, bottom: 30),
    child: Container(
        decoration: BoxDecoration(
            color: const Color(0xFFFD7D3E),
            borderRadius: BorderRadius.circular(8)),
        height: 50,
        child: GestureDetector(
          onTap: () {
            addressController.updateAddress(
                datas,
                documentReference,
                addressControllerText.text,
                numberAddressController.text,
                neighborhoodController.text,
                complementController.text
            );
            Navigator.of(context).pushReplacementNamed('/my_address_page');
          },
          child: const Center(
            child: Text(
              'Atualizar',
              style: TextStyle(
                color: Colors.black,
                fontSize: 22,
              ),
            ),
          ),
```

```
29          ),
30      ),
31  )
32  ],
33  ),
34  ),
35  ),
36  appBar: AppBar(
37    leading: IconButton(
38        onPressed: () {
39          Navigator.of(context).pushReplacementNamed('/my_address_page');
40        },
41        icon: const Icon(Icons.arrow_back_sharp)),
42    centerTitle: true,
43    backgroundColor: const Color(0xFF303030),
44    title: const Text(
45      "Atualizar Endereço",
46      style: TextStyle(
47        fontSize: 28,
48      ),
49    ),
50  ),
51  );
52  }
```

APÊNDICE C – NOMEAÇÃO DE ROTAS

```
@override
Widget build(BuildContext context) {
    return ChangeNotifierProvider(
        create: (context) => GoogleSingInProvider(),
        child:
        MaterialApp(
          debugShowCheckedModeBanner: false,
          initialRoute: '/',
          routes: {
            '/': (context) => const LoginPage(),
            '/requestPage': (context) => const MyRequestsPage(),
            '/home': (context) => const HomePage(),
            '/register': (context) => const RegisterPage(),
            '/my_address_page': (context) => const MyAddress(),
            '/change_password_page': (context) => const ChangePassword(),
            '/forgout_password_page': (context) => const ForgoutPassword(),
          },
        ));
}
```

APÊNDICE D – EXEMPLO DO USO DE NAVEGAÇÃO ENTRE TELAS

```
1  return Scaffold(
2      backgroundColor: const Color(0xFFFFFFFF),
3      appBar: AppBar(
4          leading:  IconButton(
5              icon: const Icon(Icons.arrow_back, color: Colors.white),
6          onPressed: () => Navigator.of(context).pop(itensList.length),
7          ),
```

APÊNDICE E – INICIALIZAÇÃO *CLOUD FIRESTORE* E *FIREBASE AUTHENTICATION*

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}
```

```
Padding(
    padding: const EdgeInsets.only(top: 30, bottom: 30),
    child: Container(
      decoration: BoxDecoration(
          color: const Color(0xFFFD7D3E),
          borderRadius: BorderRadius.circular(8)),
      height: 40,
      child: GestureDetector(
        onTap: () {
            FirebaseAuth.instance
              .createUserWithEmailAndPassword(
                  email: emailController.text,
                  password: passwordController.text);
            Navigator.of(context).pushReplacementNamed('/home');
        },
        child: const Center(
          child: Text(
            'Registrar-se',
            style: TextStyle(
              color: Colors.black,
              fontSize: 22,
            ),
          ),
        ),
      ),
    ),
)
```