UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ALEXANDRE RODRIGUES COELHO

SISTEMA DE DELIVERY PARA SUPERMERCADOS

PATO BRANCO

2022

ALEXANDRE RODRIGUES COELHO

SISTEMA DE DELIVERY PARA SUPERMERCADOS

Delivery System for Supermarkets

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Profª. Andréia Scariot Beulke.

Pato Branco

2022

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

ALEXANDRE RODRIGUES COELHO

SISTEMA DE DELIVERY PARA SUPERMERCADOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnológo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 29/novembro/2022

Andréia Scariot Beulke
Mestra
Universidade Tecnológica Federal do Paraná

Vinicius Pegorini
Mestre
Universidade Tecnológica Federal do Paraná

Robison Cris Brito
Doutor
Universidade Tecnológica Federal do Paraná

Pato Branco

2022

RESUMO

A alta taxa de adoção do uso de *smartphones*, atrelado ao fácil acesso à Internet permitiu a expansão das aplicações móveis. Devido à pandemia mundial ocorrida no início de março de 2020 causada pelo vírus SARS-CoV-2 ou Novo Coronavírus, foi necessário adotar medidas de isolamento social e, dessa maneira, o uso de aplicativos que possibilitam a aquisição de produtos de forma *on-line* se tornaram mais utilizados, visando reduzir os riscos de contaminação para o usuário final e para as pessoas que trabalham nas empresas. Diante disso, um dos setores atingidos foi o de supermercados, visto que é um local com um fluxo constante de pessoas. Uma das possíveis soluções para amenizar esse problema seria a introdução de uma aplicação móvel para a realização das compras pelos clientes, buscando evitar a aproximação entre eles e, ainda, proporcionado mais comodidade e conforto, por descartar a necessidade de locomoção até o local. Antes mesmo da pandemia, o uso de aplicativos de *delivery*, já era adotado pela população e por empresas, no entanto, isso se intensificou durante a pandemia. As cidades pequenas com menor fluxo de pessoas e estrutura urbana, muitas vezes, não possuem um sistema de *delivery* para atender os habitantes. No entanto, a população dessas cidades e, também, as empresas, podem ter interesse nesse tipo de sistema. Nesse sentido, o desenvolvimento desse trabalho pode também essa população a fim de impulsionar o movimento de *delivery* no interior. Para a implementação do sistema, as principais tecnologias utilizadas foram a linguagem Java, o ecossistema Spring (Boot, Data, Security, Web), o *framework* Flutter e o banco de dados PostgreSQL. O resultado foi uma aplicação móvel para os clientes realizarem suas compras, e uma interface *desktop* ou *web*, na mesma aplicação, para que os funcionários possam gerenciar suas mercadorias e pedidos.

Palavras-chave: comércio eletrônico; Spring; *delivery* para supermercados; Flutter.

ABSTRACT

The high rate of adoption of smartphones, coupled with easy access to the Internet, allowed the expansion of mobile applications. Due to the global pandemic that occurred in early March 2020 caused by the SARS-CoV-2 virus or Novel Coronavirus, it was necessary to adopt measures of social isolation and, thus, the use of applications that make it possible to purchase products online. have become more used, aiming to reduce the risks of contamination for the end user and for the people who work in the companies. In view of this, one of the sectors affected was supermarkets, since it is a place with a constant flow of people. One of the possible solutions to alleviate this problem would be the introduction of a mobile application for making purchases by customers, seeking to avoid approaching and, still, providing more convenience and comfort, by discarding the need to travel to the place. Even before the pandemic, the use of delivery apps was already adopted by the population and companies, however, this intensified during the pandemic. Small cities with less flow of people and urban structure often do not have a delivery system to serve the inhabitants. However, the population of these cities and also companies may be interested in this type of system. In this sense, the development of this work can also this population in order to boost the delivery movement in the interior. For the implementation of the system, the main technologies used were the Java language, the Spring ecosystem (Boot, Data, Security, Web), the Flutter framework and the PostgreSQL database. The result was a mobile application for customers to make their purchases, and a desktop or web interface, in the same application, for employees to manage their merchandise and orders.

Keywords: e-commerce; Spring; delivery to supermarkets; Flutter.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

ABRASEL	Associação Brasileira de Bares e Restaurantes
API	*Application Programming Interface*
B2B	*Business to Business* B2B
B2C	*Business to Consumer*
BloC	*Business Logic Component*
C2B	*Consumer to Business*
C2C	*Consumer to Consumer*
CNDL	Confederação Nacional de Dirigentes Lojistas
CRUD	*Create, Read, Update, Delete*
DER	Diagrama entidade-relacionamento
DTO	*Data Transfer Object*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
ORM	*Object-Relational Mapping*
REST	*Representational State Transfer*
RF	Requisitos Funcionais
RNF	requisitos não funcionais
SGBD	Sistema Gerenciador de Banco de Dados
SPC	Serviço de Proteção ao Crédito
SQL	*Structured Query Language*
UI	*User Interface*
URL	*Uniform Resource Locator*
USP	Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. No final do capítulo está a organização do texto por meio de uma breve apresentação dos seus capítulos.

1.1 Considerações Iniciais

Com a difusão da Internet e dos *smartphones*, os aplicativos de *delivery* tornaram-se protagonistas na revolução tecnológica e nos hábitos de consumo, crescendo exponencialmente nos últimos anos.

No primeiro semestre de 2020, o mundo vivenciou um cenário caótico no início da pandemia, pois o risco de contágio da Covid-19 inviabilizava a locomoção aos estabelecimentos para a compra de produtos essenciais. Além disso, em função das normas e medidas de isolamento social, grande parcela do ramo alimentício foi obrigada a interromper temporariamente o atendimento presencial ao público. Assim, o mercado de *delivery* se tornou fundamental para que as pessoas pudessem adquirir suas compras sem sair de casa e de forma mais segura. Além disso, estimulou o consumo e influenciou hábitos da população, pois aumentou o consumo de uma mesma pessoa e novos usuários adquiriram esse serviço. De acordo com o Jornal da Universidade de São Paulo (USP), de março a abril de 2020, início da pandemia causada pelo Covid-19, o número de usuários que passaram a utilizar os serviços de delivery aumentou em 155% e os pedidos aumentaram em 975% no ano de 2020, transformando, assim uma tendência em necessidade (JORNAL..., 2021).

Dentre os meios de venda de produtos, o *e-marketplace* se destaca por dispor de diversas marcas e modelos de mercadorias oferecidas por múltiplas lojas em um único ambiente e de forma *on-line*. O gerente comercial da Ebit/Nielsen, durante o *Marketplace Conference* 2021 ressaltou sobre as tendências e oportunidades dos *marketplaces* no Brasil. A afirmação é que o *marketplace* cresceu 52% em 2020, acima do total do mercado, o que resultou em R$ 73, 2 bilhões para a categoria. Para comparação, o mercado de *e-commerce* como um todo cresceu 41% em 2020.

> "É um crescimento super agressivo em número de pedidos e com *ticket* médio acima da média total do *e-commerce*, ou seja, são resultados super positivos para o *marketplace*, que se consolida cada vez mais no cenário de pandemia como uma alternativa para o varejo tradicional que está impossibilitado de vender *off-line*" (RONDINELLI, 2021).

O *smartphone* é o dispositivo mais utilizado nas compras pela internet (87%) em 2021, um avanço de 20 pontos percentuais em relação ao estudo realizado em 2019. O dado é da pesquisa “Consumo online no Brasil”, realizada pela Confederação Nacional de Dirigentes Lojistas (CNDL) e pelo Serviço de Proteção ao Crédito (SPC Brasil), em parceria com a Offer Wise Pesquisas (RONDINELLI, 2021).

Os principais *marketplaces* do Brasil oferecem aplicativo próprio para *smartphones*. Por meio de um aplicativo próprio o varejista é capaz de oferecer recursos exclusivos não disponíveis em uma página web e até mesmo garantir a segurança dos usuários. Os aplicativos também são ferramentas de vendas que prestigia os clientes fiéis, uma excelente forma de facilitar as compras e meio eficiente para divulgar ofertas relevantes.

Os habitantes de cidades menores carecem de um sistema de *delivery* e, por vezes, são submetidos ao uso de aplicativos de empresas pequenas que, quando existentes, são ainda limitados, não oferecendo a mesma diversidade de produtos e a facilidade de uso de empresas maiores.

Assim, este trabalho propõe o desenvolvimento de um sistema de *delivery* para supermercados. O cadastro inicial dos estabelecimentos será de responsabilidade do administrador, posteriormente o sistema permitirá que funcionários do supermercado possam cadastrar os produtos por meio de uma interface *desktop* e *web*. Os clientes dos estabelecimentos poderão utilizar um aplicativo móvel para adquirir seus produtos, e solicitar a retirada de produtos no local ou por meio de entrega em domicílio.

1.2 Objetivos

A seguir são apresentados os objetivos do sistema proposto neste trabalho, sendo o objetivo geral o resultado principal esperado e, os objetivos específicos, as principais funcionalidades principais do sistema.

1.2.1 Objetivo Geral

Desenvolver um sistema web e móvel de *delivery* para comercialização de produtos de supermercados.

1.2.2 Objetivos Específicos

- Proporcionar aos varejistas de regiões menores, a utilização de um sistema virtual da sua empresa para a venda de seus produtos via *delivery*;
- Possibilitar que o cliente realize o cadastro de dados pessoais e a escolha entre entrega em domicílio ou retirada no local;
- Permitir que os clientes registrem produtos como favoritos;
- Fornecer gráficos com métricas para controle dos varejistas.

1.3 Justificativa

Os serviços de entrega a domicílio (*delivery*) vem se desenvolvendo nos últimos anos e apresentaram um avanço ainda maior no início da pandemia ocasionada pela Covid-19. Muitas organizações precisaram criar novas estratégias para se manterem no mercado. E uma das estratégias que mais se destacou foi o serviço de entrega a domicílio. Com isso, os sistemas de *delivery* ganharam novos usuários tornando-os um negócio de sucesso e, ainda, ajudou a minimizar a crise provocada pela pandemia (OLIVEIRA; ABRANCHES; LANA, 2020).

O aumento do número de usuários nesse setor já era observado mesmo antes da pandemia. No entanto, isso se intensificou durante a pandemia, pois as pessoas preferiram optar pelo uso de aplicativos de *delivery* para adquirir suas compras, especialmente, os produtos de supermercado, tanto para atender as medidas de contenção, como para garantir uma segurança maior e evitar sair de casa e escapar do contágio causado pelo Covid-19.

Os aplicativos de *delivery* ainda não estão disponíveis para toda a população brasileira devido às diferenças socioeconômicas de cada região. Esse tipo de negócio deve ser adaptado para a realidade local, visando explorar as vantagens de uma comunidade menor.

Nesse contexto, o desenvolvimento do sistema proposto nesse trabalho, se justifica pela necessidade que a população dessas cidades também possui em realizar suas compras em supermercado por meio do uso de aplicativos.

O sistema proposto visa atender tanto aos supermercados quanto aos clientes desses estabelecimentos. O cliente poderá usufruir da praticidade e segurança que o aplicativo propõe, que é realizar suas compras e recebê-las sem sair de casa. Para o supermercado o uso do sistema será relevante para alavancar seus negócios e competir no mercado, visto que os serviços de entrega estão se tornando uma tendência e, assim poderá aumentar suas vendas e gerar mais lucros.

1.4 Estrutura do Trabalho

Este trabalho está organizado em capítulos. Este é o primeiro capítulo e apresenta as considerações iniciais com o contexto do sistema a ser desenvolvido, os seus objetivos e a justificativa. O Capítulo 2 apresenta a fundamentação teórica que compõe os conceitos necessários para o desenvolvimento deste trabalho. No Capítulo 3 estão as ferramentas e as tecnologias utilizadas na modelagem do sistema e que serão utilizadas na implementação subsequente do sistema. No Capítulo 4 está o resultado da realização do trabalho que é a modelagem, desenvolvimento das interfaces e da implementação do sistema. Por fim, está a conclusão e as referências utilizadas no trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta a fundamentação teórica deste trabalho cujo conteúdo explana sobre os tipos de negócios que podem ocorrer por meio da Internet e o uso de aplicativos de terceiros nas empresas.

2.1 Negócios na Internet

A Internet tem se mostrado uma opção interessante para se fazer negócios. Uma de suas principais vantagens é a quebra de barreira entre o cliente e a empresa proporcionada pelo fácil acesso aos produtos em qualquer tempo e lugar podendo efetuar uma compra sem sair casa. As compras podem ser realizadas em qualquer estabelecimento do mundo e em qualquer horário. As formas de pagamento são praticamente as mesmas dos estabelecimentos convencionais, podendo ser realizado por meio de transferência bancária, cartão de crédito, boleto ou *pix*. Ainda permite o parcelamento ou pagamento à vista. Os preços das lojas *on-line* são bastante competitivos e, geralmente menores que das lojas convencionais, pois os custos com estocagem e funcionários são menores. Além disso, o cliente pode optar por receber o seu produto em domicílio ou retirar na loja mais próxima de sua residência. Torres (2013) afirma que o comércio ocorre pela troca de produtos e serviços visando obter lucros de maneira sustentável e que que cada tipo de negócio na Internet possibilita que sejam implementados modelos de negócios diferentes.

Os tipos de negócios na Internet mais comuns baseados em comércio eletrônico variam de acordo com as características que definem as relações comerciais. Para Turban et. al (2004), os tipos mais conhecidos são:

a) *Business to Business* (B2B): relações estabelecidas entre a empresa e vendas corporativas.
b) *Business to Consumer* (B2C): transações que ocorrem entre uma pessoa jurídica e uma pessoa física.
c) *Consumer to Consumer* (C2C): transações que ocorrem entre pessoas físicas.
d) *Consumer to Business* (C2B): relações que ocorrem entre uma pessoa física e uma pessoa jurídica.

O B2C é um modelo de negócio no qual as empresas disponibilizam produtos ou serviços para o consumidor final que são pessoas físicas, sendo, portanto, o modelo de negócio adotado nesse trabalho. Geralmente o mercado B2C oferece maior volume de operações no qual o consumidor final adquire produtos variados e heterogêneos. O B2C permite que os

consumidores não estejam presentes fisicamente na loja e, consequentemente, não precisarem de locomoção para adquirirem os seus produtos.

No entanto, um comércio eletrônico não se limita apenas às vendas *on-line*, pois também envolve (POWARCZUK, 2012): a realização de orçamentos, disponibilização de catálogos eletrônicos, plano de acesso aos pontos de venda, gestão em tempo real do estoque dos produtos, a gestão do pagamento *on-line,* o acompanhamento da entrega e os serviço pós-venda. Essas características são essenciais para o negócio possam fluir de forma satisfatória para o cliente e para empresa.

Ao adquirir um produto ou serviço, o cliente busca novidades e facilidade no relacionamento com as empresas para que suas necessidades sejam atendidas. Para Silveira (2018) esse relacionamento pode se tornar mais propício com a utilização de aplicativos que possibilitem um contato direto com os envolvidos.

2.2 Aplicativos de terceiros no ambiente corporativo

A comunicação do cliente com o mercado tem se tornado cada vez eficiente por meio do uso de aplicativos. Geralmente, esses aplicativos de *delivery* são utilizados para alavancar as vendas, reduzir custos e aumentar os lucros. Esses aplicativos conectam consumidores e empresas para a entrega de refeições ou encomendas. Portanto, o relacionamento com o cliente pode ser expandido com o uso desses aplicativos, pois eles oferecem um meio de interação entre os clientes e a empresa sem a limitação geográfica e em grande velocidade.

Os aplicativos de *delivery* têm se tornado importantes no ambiente corporativo, principalmente os que trabalham com produtos alimentícios, pois oferece a entrega dos produtos em domicílio em curto prazo. Eles funcionam como intermediários entre os consumidores e os estabelecimentos. Nesse contexto, França et al. (2014, p. 1) destacam que "as pessoas estão com a vida cada vez mais corrida, entre trabalho e estudo, além das horas que perdem por dia no trânsito da cidade durante a semana, deixando as pessoas ainda mais cansadas e propícias a pedirem comida em casa ou no trabalho".

Nesse sentido, os aplicativos de *delivery* se expandem para outras áreas que envolvem a entrega de encomendas, como as de produtos farmacêuticos e de supermercados, por exemplo. Uma das vantagens desse serviço é que os consumidores conseguem acompanhar desde o processo da compra até a entrega do produto, possibilitando um maior empoderamento no relacionamento do cliente com a empresa.

Segundo a Associação Brasileira de Bares e Restaurantes (ABRASEL) (2018), o faturamento do mercado de *delivery* no Brasil cresce a cada ano numa escala anual de pelo menos 1 bilhão de reais. Esse mercado se intensificou ainda mais com a crise da pandemia ocasionada pela Covid-19 em 2020, onde diversas empresas fecharam suas portas devido às medidas de contingência impostas pelo governo. No entanto, o aumento do número de usuários desse tipo de serviço já era observado antes mesmo da pandemia.

Montenegro (2020) afirma que o as pessoas passaram a utilizar mais esses serviços a partir de abril de 2020, atingindo um total de 22 milhões de aparelhos celulares no país. Marino (2020) afirma que a plataforma iFood atingiu 39 milhões de pedidos no mês de março de 2020. O iFood é uma das maiores plataformas de *delivery* possuindo mais de 236 mil restaurantes parceiros, estando presente em mais de 1.000 cidades do país (MENIGHINI et al. 2021).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para o desenvolvimento deste trabalho. Os materiais elencam as ferramentas e as tecnologias utilizadas na modelagem, banco de dados, linguagem de programação e *frameworks* utilizados. O método está relacionado a um modelo de processo que visa ordenar e estruturar o desenvolvimento do aplicativo.

3.1 Materiais

No Quadro 1 estão listados os materiais que foram utilizados para o desenvolvimento do trabalho e implementação da aplicação.

Quadro 1 - Lista de ferramentas e tecnologias

Ferramenta / Tecnologia	Versão	Finalidade
Dart	2.18.4	Linguagem de programação.
Flutter	3.3.7	Toolkit para Desenvolvimento *web*, *desktop* e *mobile*.
flutter_bloc	8.1.1	Gerenciamento de estado.
flutter_modular	5.0.3	Injeção de dependência e gerenciamento de rotas.
Intellij Idea Ultimate	2022.2.3	*Integrated Development Environment* (IDE) de desenvolvimento.
Java	17	Linguagem de programação *back-end.*
pgAdmin 4	6.12	Ferramenta para administração do banco de dados.
PostgreSQL	14.6	Banco de dados.
Spring Boot	2.7.5	*Framework* para criação de aplicações.
Spring Data	2.7.5	*Framework* de implementação de repositórios e *Object-Relational Mapping* (ORM).
Spring Security	5.7.5	*Framework* de autenticação e segurança da aplicação.
Visual Paradigm	16.3	Modelagem do banco de dados e do sistema.

Fonte: Autoria própria (2022).

A aplicação foi desenvolvida utilizando Flutter, um kit de desenvolvimento de interface de usuário, baseado na linguagem de programação Dart. Com o Flutter foi possível desenvolver tanto a aplicação *desktop*/*web* quanto a aplicação *mobile*. A seguir é apresentada

uma descrição dessas tecnologias por serem as mais importantes utilizadas no projeto e por não terem sidos estudadas durante o período de integralização do curso.

3.1.1 Flutter

Flutter é o kit de ferramentas de interface do usuário portátil do Google para criar aplicativos bonitos e compilados nativamente para dispositivos móveis, *web* e *desktop* a partir de uma única base de código. É um projeto de código aberto, com contribuições da Google e de outras empresas e indivíduos. Ele é diferente da maioria das outras opções para criar aplicativos móveis porque não depende da tecnologia do navegador da Web nem do conjunto de widgets que acompanham cada dispositivo. Em vez disso, o Flutter usa seu próprio mecanismo de renderização de alto desempenho para desenhar *widgets*.

O Modular é uma solução para modularizar as rotas e o sistema de injeção de dependência, fazendo com que cada escopo tenha suas próprias rotas e injeções independentes de qualquer outro fator na estrutura. São criados objetos para agrupar as rotas e as injeções, chamados de Módulos. O *flutter_modular* é o pacote que contém essas implementações. Ele foi usado para melhorar o gerenciamento de recursos e separação das funcionalidades existentes no sistema.

O padrão Business Logic Component (BLoC) foi desenhado por Paolo Soares e Cong Hui, do Google e apresentado pela primeira vez durante a DartConf 2018. Ele vem com uma ideia base de desacoplar as funções lógicas da camada de apresentação (*Presentation Layer*), de modo que os componentes de *User Interface* (UI) deverão apenas se preocupar com a UI e não com regras de negócio. Eles são construídos basicamente em cima da utilização de *Streams*, que são fluxos de dados, que recebem informações, trafegam por um “canal” e depois emitem essa informação para quem estiver “ouvindo” esse fluxo. O *flutter_bloc* é um pacote que facilita a implementação do padrão BLoC a partir de *widgets* já prontos.

3.1.2 Dart

Dart é uma linguagem otimizada para o cliente para desenvolver aplicativos rápidos em qualquer plataforma. Seu objetivo é oferecer a linguagem de programação mais produtiva para desenvolvimento multiplataforma, combinada com uma plataforma de execução flexível para *frameworks* de aplicativos. Dart também forma a base do Flutter.

O Dart fornece a linguagem e os tempos de execução que impulsionam os aplicativos Flutter e, também, oferece suporte às tarefas principais do desenvolvedor, como formatação, análise e teste de código.

3.2 Método

Para a implementação do método foi utilizado o modelo cascata, no qual o processo é como uma série de passos previsíveis, ou um roteiro, que ajuda na criação de um sistema de alta qualidade (HIGOR, 2013). Apesar deste modelo ser recomendado para adaptações em sistemas já existentes, a ideia do desenvolvimento sequencial facilitou na resolução deste projeto. Outras metodologias, como o Scrum, por exemplo, poderiam apresentar melhor gerenciamento do projeto, no entanto poderia exigir maior esforço e tempo para utilização. O desenvolvimento foi desenvolvido conforme as seguintes etapas:

a. **Requisitos:** nessa fase, começou-se a idealizar como seria o sistema a partir da análise de requisitos, tais como suas funcionalidades e requisitos funcionais e não funcionais. Parte da coleta de requisitos foi realizada observando as funcionalidades visíveis no *FOGRO*, que é um conjunto de interfaces de usuário (UI *Kit*) para auxiliar no processo de desenvolvimento de aplicativos de comida e mercearia, com foco em dispositivos móveis. Com os requisitos em mão, seguiu-se para a criação do diagrama de caso de uso, de modo a demonstrar as diferentes maneiras de como o usuário iria interagir diretamente com o sistema. Foi utilizada a ferramenta Visual Paradigm para a elaboração dos artefatos dessa fase.
b. **Projeto:** a partir dos requisitos definidos, foi criado um cronograma para realização das tarefas. Baseado nesse cronograma, foi estipulado uma estimativa para finalização de cada etapa. Nesta etapa também foi formulado o diagrama entidade-relacionamento (DER), representando a estrutura do banco de dados. Foi utilizado a ferramenta Visual Paradigm para modelagem. Para realização do projeto, foi usado como base o *FOGRO*.
c. **Implementação:** nesta etapa consiste na codificação do sistema. Ela foi realizada inteiramente em conjunto com a IDE IntelliJ Idea. A parte servidora da aplicação foi construída com a linguagem Java usando o *framework* Spring. O vasto ecossistema do Spring permite um desenvolvimento de forma mais ágil, por possuir diversos projetos integrados. Dentre alguns desses projetos, os principais utilizados foram o Spring Boot, que possibilita uma rápida criação de aplicações Spring e facilita a

integração com os outros projetos, o Spring Data, para criação do banco de dados a partir do mapeamento objeto-relacional, e o Spring Security, responsável pela parte de autenticação e autorização do sistema. Já a parte cliente, foi desenvolvida com a linguagem Dart, utilizando o Flutter para construção da interface de usuário. Ao usar o Flutter, possibilitou-se construir uma aplicação tanto *desktop* quanto *web* usando a mesma base de código, além da construção da aplicação *mobile*, usando a mesma linguagem e dependências.

d. **Verificação:** nessa etapa testadas as funcionalidades principais do sistema, averiguando se estavam funcionando corretamente, conforme especificado. O ponto fraco dessa etapa foi a falta de testes realizados diretamente em código, tais como testes unitários, testes de integração e até mesmo testes de UI (*Widgets Testing*), tanto no servidor quanto na aplicação cliente.
e. **Manutenção:** um dos pontos fracos da escolha deste modelo de desenvolvimento, foi que esta etapa se mostrou meio obsoleta, visto que por ainda não possuir usuários finais, só o próprio desenvolvedor é capaz de detectar mudanças necessárias e assim realizá-las.

4 RESULTADOS

Este capítulo apresenta o resultado da modelagem e o desenvolvimento do sistema apresentado nesse trabalho.

4.1 Escopo do Sistema

O sistema deve gerenciar todos os processos de um comércio virtual, desde o cadastro do supermercado e de seus produtos, até a compra e o meio de entrega, pelo cliente do estabelecimento.

Os atores previstos no sistema são: administrador, funcionário e cliente. O administrador será responsável por realizar o cadastro do supermercado e de seus funcionários. Inicialmente, esse cadastro será feito diretamente no banco de dados. O funcionário do supermercado é a pessoa responsável por cadastrar os produtos e ofertas. O administrador também poderá manipular esses produtos, caso seja necessário. O controle de preços e estoque deve ser realizado pelo funcionário do supermercado.

O sistema permitirá que os clientes realizem seu cadastro por meio do aplicativo móvel. O cadastro poderá ser realizado somente por e-mail. Esse cadastro será compartilhado entre todos os supermercados disponíveis, de modo que o cliente possa realizar compras em diferentes supermercados se cadastrando apenas uma vez.

Os clientes poderão adicionar os produtos no carrinho de compras e escolher o momento em que a compra será finalizada, pois quando a aplicação for fechada de forma abrupta, os produtos devem permanecer no carrinho. Além disso, o valor da compra será calculado automaticamente, somando o valor dos itens, o valor dos serviços e frete.

As compras poderão ser entregues em domicílio ou retiradas no próprio supermercado. Para cada compra, poderá ser cobrada uma taxa de serviços, que inclui o pedido *on-line* e a separação de produtos e, caso seja uma entrega, poderá ser cobrado um valor de frete que poderá variar de acordo com o endereço informado pelo cliente.

Os clientes poderão filtrar os produtos por categorias, preço e nome do produto. Também poderão ordenar os produtos por preço, mais vendidos e por percentual de desconto. As compras dos clientes devem ficar registradas por meio de um histórico de pedidos que pode ser visto acessando o perfil do usuário e clicando na opção “Meus Pedidos”.

Os clientes poderão marcar produtos como favoritos, armazenando-os em uma lista que pode ser vista acessando o perfil do usuário e clicando na opção “Meus Favoritos”. Nessa lista, deve ser possível adicionar os produtos diretamente no carrinho.

Os clientes poderão acompanhar o *status* do pedido que esteja em aberto. Os funcionários do supermercado poderão usar gráficos para acompanhar algumas métricas relacionadas às vendas, tais como visão geral dos *status* dos pedidos, produtos mais e menos comprados, horários com maior incidência de compras, *ticket* médio, ou outros personalizados de acordo com a necessidade.

4.2 Modelagem do Sistema

Esta seção apresenta os requisitos funcionais e não funcionais e os diagramas desenvolvido para representar o software a ser desenvolvido.

No Quadro 2 são apresentados os requisitos funcionais (RF) definidos para o sistema, caso o usuário com perfil de cliente.

Quadro 2 – Requisitos funcionais para clientes

Identificação	Nome	Descrição
RF01	Manter usuário	Refere-se ao cadastro dos usuários que irão utilizar o sistema. Esses usuários podem ser classificados em três categorias, sendo: cliente, administrador e funcionário do supermercado. Os usuários com perfil de funcionário são cadastrados pelo próprio administrador, de modo que somente os clientes poderão se cadastrar manualmente pelo aplicativo.
RF02	Manter informações pessoais	Os clientes poderão alterar suas informações pessoais, como nome, e-mail, telefone, senha e a foto de perfil.
RF03	Manter endereços	Os clientes poderão cadastrar, editar e excluir endereços que poderão ser utilizados como endereços de entrega.
RF04	Manter favoritos	Os clientes poderão adicionar ou remover produtos como favoritos. Essa interação poderá ser feita ao clicar no ícone representado por um coração, que será apresentado na tela no topo superior dos produtos.
RF05	Visualizar lista de favoritos	Os clientes poderão acessar uma lista mostrando todos os produtos adicionados como favoritos e, a partir dessa lista, poderão adicionar estes produtos diretamente no carrinho.
RF06	Visualizar produtos	O sistema possibilitará que os clientes possam visualizar os produtos. Na tela inicial serão apresentados os produtos mais vendidos. Esses produtos poderão ser adicionados no carrinho ao clicar no botão representado com ícone de carrinho.
RF07	Visualizar categorias	O sistema possibilitará que os clientes possam visualizar as categorias existentes.
RF08	Visualizar produtos por categoria	Ao selecionar uma categoria, deverá ser exibida uma lista de produtos pertencentes àquela categoria.
RF09	Filtrar produtos	O sistema disponibilizará uma tela para aplicação de filtros, na qual será possível alterar o intervalo de preços, a ordenação (mais vendidos, preço e percentual de desconto) e a categoria. Os produtos também poderão ser filtrados por nome por meio de uma barra de busca. Caso nenhum produto atenda aos filtros especificados, deverá ser apresentada uma mensagem ao usuário.
RF10	Gerenciar carrinho	O cliente poderá gerenciar seu carrinho de compras, podendo adicionar, remover e alterar a quantidade dos produtos. O botão para acesso ao carrinho será visível em várias telas do sistema. A quantidade não poderá exceder o estoque do produto. Caso não tenha estoque suficiente para o produto, deverá ser apresentada

		uma mensagem ao usuário. O estoque do pedido deverá ser atualizado automaticamente após finalizado o pedido.
RF11	Escolher tipo de entrega	O sistema deverá mostrar uma opção para selecionar o tipo de entrega que pode ser retirada no local ou entrega à domicílio.
RF12	Escolher informações da entrega	Caso o tipo de entrega seja por entrega no local, o cliente poderá informar para qual endereço será entregue e em qual horário deseja receber a entrega. O cliente deverá registrar um novo endereço, caso não tenha nenhum cadastrado. Por padrão o endereço padrão virá selecionado automaticamente. O horário de entrega deverá ser um horário válido para o dia atual, de modo que não será possível escolher um horário retrógado ao horário atual.
RF13	Realizar pedido	O sistema deverá validar as informações do pedido e somente permitir finalizar o pedido se todas informações obrigatórias estiverem informadas. O total da compra deverá ser calculado automaticamente, levando em consideração o valor dos itens, a taxa de entrega, os descontos (para produtos em oferta) e a taxa de serviço.
RF14	Acompanhar pedido	Após a finalização do pedido, o sistema deverá redirecionar o cliente para uma tela na qual ele poderá acompanhar o *status* do pedido.
RF15	Visualizar histórico de pedidos	O sistema deverá possuir uma tela na qual serão apresentados todos os pedidos já realizados pelo cliente. Caso haja algum pedido que o *status* seja diferente de entregue, deverá existir um botão que ao ser clicado, redirecionará o cliente para a tela de acompanhamento do pedido.

Fonte: Autoria própria (2022).

No Quadro 3 são apresentados os RF definidos para o sistema, especificamente para usuário com perfil de administrador e de funcionário.

Quadro 3 – Requisitos funcionais para administrador e funcionário do supermercado

Identificação	Nome	Descrição
RF16	Acessar módulo de administrador	Ao acessar o sistema, caso o cargo do usuário seja funcionário ou administrador e ele estiver acessando o sistema por meio de uma aplicação *web* ou *desktop*, ele deve ser redirecionado para o módulo de administrador.
RF17	Visualizar gráficos de métricas	O sistema deverá apresentar gráficos, apresentando métricas relacionadas às vendas dos produtos do supermercado. Esses gráficos deverão aparecer logo após o funcionário acessar o sistema.
RF18	Manter produtos	O sistema deverá permitir que o funcionário consiga adicionar novos produtos, além de editar e excluir produtos já existentes.
RF19	Manter taxas de entrega por bairro	O sistema deverá permitir que o funcionário consiga adicionar, editar ou excluir preços para taxa de entrega para os bairros disponíveis. Não deverá ser possível adicionar mais de uma taxa de entrega para o mesmo bairro. Essa taxa deverá ser aplicada no campo “taxa de entrega”, durante a realização do pedido.
RF20	Manter pedidos	O sistema deverá listar o histórico de pedidos. Por meio desse histórico, deverá ser possível alterar o *status* dos pedidos em aberto. Os *status* podem ser: pendente, realizado, confirmado, pronto para retirada, pronto para entrega, saiu para entrega ou entregue.
RF21	Editar configurações	Permitir aos funcionários alterar os dados da empresa, esses dados contêm: nome, telefone, celular, horários de abertura e fechamento, horários de abertura e fechamento do delivery, valor

		da taxa de serviço, endereço do local, logo da empresa e os *banners* que serão apresentados na tela inicial do sistema do cliente.

Fonte: Autoria própria (2022).

No Quadro 4 estão listados os requisitos não funcionais (RNF) do sistema que são aqueles relacionados ao uso da aplicação em termos de desempenho, usabilidade e segurança, como, por exemplo, validação e preenchimento de campos e integridade de dados.

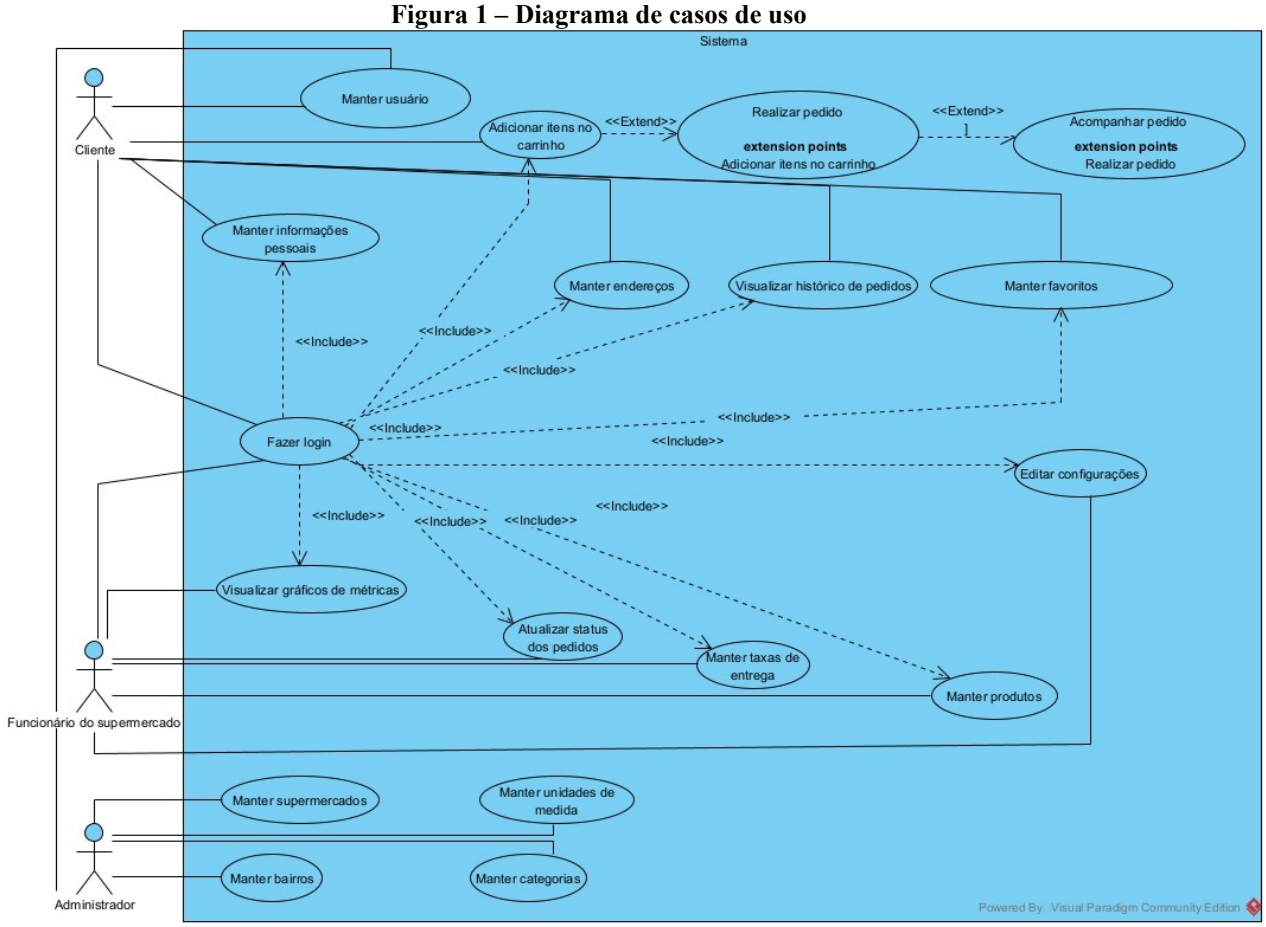
Quadro 4 – Requisitos não funcionais

Identificação	Nome	Descrição
RNF01	Acessar sistema	O acesso ao sistema é realizado através do e-mail e senha cadastrados. Ao acessar o sistema, deve-se escolher também qual mercado será acessado.
RNF02	Campos de preenchimentos obrigatórios	Os campos que são de preenchimento obrigatório serão validados por meio de uma função do sistema.
RNF03	Campos com máscaras de entrada	Os campos que possuem caracteres especiais, e que não serão armazenados no banco de dados, serão validados por meio de máscaras de entrada nos respectivos campos.
RNF04	Vínculo	O sistema não deve permitir a exclusão de registros que possuam dados vinculados. Por exemplo: excluir um endereço vinculado a um pedido.
RNF05	Atualização automática para acompanhamento do pedido	O sistema deve atualizar automaticamente a tela de acompanhamento do pedido, buscando trazer informações rápidas para o usuário em relação a seu pedido.
RNF06	Atualização automática para listagem de pedidos	O sistema deve atualizar automaticamente a tela de listagem de pedidos, buscando novos pedidos que possam ter sido realizados.
RNF07	Redirecionamento para tela de acompanhamento do pedido	Ao clicar no botão do carrinho e houver algum pedido em andamento, o sistema deve redirecionar o usuário diretamente para a tela de acompanhamento do pedido.
RNF08	Ícone de favorito para produtos	O ícone de coração localizado acima do produto, deve indicar se o produto está ou não na lista de favoritos. O coração preenchido deve representar que o produto está presente, e vice-versa.
RNF09	Alterar endereço para endereço padrão	O primeiro endereço cadastrado de cada usuário deve ser automaticamente configurado como endereço padrão. Ao adicionar outro endereço e marcar o novo endereço como padrão, todos os outros endereços devem ser marcados automaticamente como “não padrão”.
RNF10	Exclusão de endereços	Somente deve ser possível excluir o endereço, houver mais de um endereço cadastrado.

Fonte: Autoria própria (2022).

A Figura 1 apresenta o diagrama de caso de uso definidos para o sistema. Esse diagrama apresenta as funcionalidades essenciais do sistema, que são realizadas por seus atores, representados pelo cliente, administrador e funcionário do supermercado.

Figura 1 – Diagrama de casos de uso



Fonte: Autoria própria (2022).

O administrador é responsável por cadastrar as empresas e seus respectivos funcionários, além de manter os cadastros de categorias, bairros e unidades de medida, que serão comuns para todos os supermercados. Os usuários podem se cadastrar no sistema e realizar a autenticação, posteriormente poderão ter acesso às outras funcionalidades: gerenciar os itens do carrinho, realizar e acompanhar os pedidos, editar as informações pessoais, manter endereços, visualizar histórico de pedidos e manter os favoritos. O funcionário do supermercado poderá manter as configurações da empresa, visualizar gráficos de métricas, visualizar e atualizar os status dos pedidos existentes, alterar o preço da taxa de entrega para os bairros e manter os produtos.

Os quadros 5 a 8 apresentam a expansão dos casos de uso que constam na Figura 1. Os casos de uso de manutenção (manter) são representados pelas operações persistentes no banco de dados que são: incluir, alterar, excluir e pesquisar. Essas operações possuem comportamentos similares entre si para cada caso de uso que as compõe.

O Quadro 5 apresenta a expansão do caso de uso que representa a operação de inserção de um registro no banco de dados.

Quadro 5 – Operação para inserir um novo registro no banco de dados

Caso de uso: Inserir (representa a operação de inclusão nos casos de uso "manter"). **Descrição:** Incluir novos dados no sistema. **Atores:** Cliente, administrador ou funcionário do supermercado, dependendo das funções definidas no caso de uso. **Pré-condição:** Estar autenticado no sistema. **Sequência de Eventos:** 1. Ator acessa a tela respectiva do cadastro. 2. Ator preenche os campos em tela e pressiona o botão "Salvar" 3. Os registros são inseridos no banco de dados e é apresentado uma mensagem especificando o *status* da operação. **Pós-Condição:** O registro é inserido no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator clica no botão "Salvar" sem ter preenchido os campos obrigatórios 1.2. O sistema valida as informações e destaca os campos com informação faltante, apresentando uma mensagem embaixo de cada campo.

Fonte: Autoria própria (2022).

O Quadro 6 apresenta a expansão do caso de uso que representa a operação de alteração de um registro já existente no banco de dados.

Quadro 6 – Operação para alterar registro existente no banco de dados

Caso de uso: Alterar (representa a operação de alteração nos casos de uso "manter"). **Descrição:** Alterar dados existentes do sistema. **Atores:** Cliente, administrador ou funcionário do supermercado, dependendo das funções definidas no caso de uso. **Pré-condição:** Estar autenticado e possuir dados cadastrados no sistema. **Sequência de Eventos:** 1. O ator acessa a tela para visualização dos dados já cadastrados. 2. O ator clica no botão correspondente a edição. 3. Ator altera os dados e pressiona o botão "Editar" 4. O sistema valida as informações, subscreve no mesmo registro e é apresentado uma mensagem especificando o status da operação. **Pós-Condição:** O registro é alterado no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1 O ator clica no botão "Editar" sem ter preenchido os campos obrigatórios 1.2 O sistema valida as informações e destaca os campos com informação faltante, apresentando uma mensagem embaixo de cada campo.

Fonte: Autoria própria (2022).

O Quadro 7 apresenta a expansão do caso de uso que representa a operação para excluir um registro existente no banco de dados.

Quadro 7 – Operação para exclusão de registro do banco de dados

Caso de uso:
Excluir (representa a operação de exclusão nos casos de uso "manter").
Descrição:
Remover dados do sistema.
Atores:
Cliente, administrador ou funcionário do supermercado, dependendo das funções definidas no caso de uso.
Pré-condição:
Estar autenticado e possuir dados cadastrados no sistema.
Sequência de Eventos:
1. O ator acessa a tela para visualização dos dados já cadastrados.
2. O ator clica no botão correspondente a exclusão.
3. O ator clica na opção "Sim" na caixa de diálogo que é apresentada na tela.
4. Ator altera os dados e pressiona o botão "Editar"
5. O sistema exclui o registro do banco de dados e é apresentado uma mensagem especificando o status da operação.

Pós-Condição:
O registro é excluído do banco de dados.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Registro possui vínculos no sistema.	1.1 O ator clica no botão de confirmação para exclusão do registro. 1.2 O sistema verifica se o registro possui algum vínculo e apresenta uma mensagem de erro informando a impossibilidade da exclusão.

Fonte: Autoria própria (2022).

O Quadro 8 apresenta a expansão do caso de uso que representa a operação de consulta dos dados no banco.

Quadro 8 – Operação para consulta de registros do banco de dados

Caso de uso:
Consultar (representa a operação de consulta nos casos de uso "manter").
Descrição:
Consultar informações cadastradas no sistema.
Atores:
Cliente, administrador ou funcionário do supermercado, dependendo das funções definidas no caso de uso.
Pré-condição:
Estar autenticado e possuir dados cadastrados no sistema.
Sequência de Eventos:
1. O ator acessa a tela para visualização dos dados.
2. Os dados são exibidos para o usuário de acordo com a consulta.

Pós-Condição:
Os dados são exibidos para o ator.

Fonte: Autoria própria (2022).

O Quadro 9 apresenta o caso de uso para o cliente manter as suas informações pessoais no sistema.

Quadro 9 – Caso de uso manter informações pessoais

Caso de uso: Manter informações pessoais. **Descrição:** Permite alterar informações pessoais que não são apresentadas durante o cadastro. **Atores:** Cliente **Pré-condição:** Estar autenticado. **Sequência de Eventos:** 1. Ator acessa a tela "Sobre Mim" 2. Ator preenche os campos e clica no botão "Salvar" 3. Os dados são alterados e é apresentado uma mensagem especificando o status da operação. **Pós-Condição:** O usuário tem seus dados alterados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1 O ator clica no botão "Salvar" sem ter preenchido os campos obrigatórios 1.2 O sistema valida as informações e destaca os campos com informação faltante, apresentando uma mensagem embaixo de cada campo.
2. Usuário informa um e-mail que já está cadastrado	2.1 O ator altera o e-mail e clica no botão "Salvar" 2.2 O sistema valida as informações e verifica que o novo e-mail informado já pertence a outro usuário. 2.3 Uma mensagem de erro é apresentada e o usuário precisa informar um outro e-mail para prosseguir.

Fonte: Autoria própria (2022).

O Quadro 10 apresenta o caso de uso para o cliente adicionar produtos no carrinho.

Quadro 10 – Caso de uso adicionar itens no carrinho

Caso de uso: Adicionar itens no carrinho. **Descrição:** O ator adiciona os itens no carrinho. **Atores:** Cliente **Pré-condição:** Estar autenticado. **Sequência de Eventos:** 1. O sistema dispõe dos produtos disponíveis. 2. Ator seleciona qual produto quer adicionar no carrinho. 3. O sistema valida se há estoque para o produto selecionado. 4. O produto é adicionado ao carrinho. **Pós-Condição:** O carrinho com os itens adicionados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Estoque insuficiente.	1 O ator seleciona o produto para ser adicionado ao carrinho. 2 O sistema valida o estoque e informa que o produto selecionado não tem estoque suficiente.

Fonte: Autoria própria (2022).

O Quadro 11 apresenta o caso de uso para o cliente realizar pedido.

Quadro 11 – Caso de uso realizar pedido

Caso de uso:
Realizar pedido.
Descrição:
Após adicionados os itens no carrinho, o usuário segue em frente para realizar seu pedido.
Atores:
Cliente
Pré-condição:
Estar autenticado e ter produtos adicionados no carrinho.
Sequência de Eventos:

1. Sistema exibe os itens que estão no carrinho, suas quantidades e o subtotal calculado.
2. O cliente clica no botão de "Checkout".
3. Sistema exibe a tela com o resumo do pedido.
4. Sistema exibe opção do tipo de entrega.
   4.1 Cliente seleciona a opção "Retirada"
   Cliente informa o horário que irá retirar no campo "Observações".
   O sistema calcula o valor total do pedido, somando o valor total dos itens do carrinho e a taxa de serviço e subtraindo o valor do desconto de produtos que estão em oferta.

   4.2 Cliente seleciona a opção "Entrega"
   O sistema apresenta os campos relacionados a Entrega
   4.2.1 Cliente não possui endereço padrão cadastrado
   Cliente adiciona um novo endereço clicando em "Adicionar".
   O sistema configura esse endereço como endereço da entrega.

   4.2.2 Usuário já possui endereço padrão cadastrado
   O sistema já configura esse endereço como endereço da entrega.
   O Cliente pode alterar o endereço clicando na opção "Alterar" e selecionando outro endereço existente, ou adicionar um novo endereço e selecioná-lo.
   Cliente seleciona o endereço de entrega.
   O sistema verifica o valor da taxa de entrega do bairro do endereço selecionado.
   4.2.3 Bairro possui taxa de entrega
   O valor da taxa é adicionado ao total do pedido

   4.2.4 Bairro não possui taxa de entrega
   É apresentado o texto "Grátis" no resumo do pedido e não é adicionado valor ao total do pedido.
   Cliente informa o horário de Entrega.
   O sistema calcula o valor total do pedido, somando o valor total dos itens do carrinho, da taxa de entrega e da taxa de serviço e subtraindo o valor do desconto de produtos que estão em oferta.
5. O cliente clica no botão "Finalizar Pedido"
6. O sistema valida as informações selecionadas.

Pós-Condição:
O status do pedido é alterado para "Realizado" e já será visível para o supermercado.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1 O ator clica no botão "Realizar Pedido" sem ter preenchido os campos obrigatórios 1.2 O sistema exibe uma mensagem em tela, apresentando os campos que precisam ser preenchidos.

Fonte: Autoria própria (2022).

O Quadro 12 apresenta o caso de uso para que o cliente possa acompanhar o seu pedido.

Quadro 12 – Caso de uso acompanhar pedido

Caso de uso:
Acompanhar pedido.
Descrição:

O ator acompanha o status atual do pedido. **Atores:** Cliente **Pré-condição:** Estar autenticado e estar com algum pedido em aberto. **Sequência de Eventos:** 1. O cliente acessa a aplicação. 1.1 O usuário acessa a tela “Meus Pedidos”. 1.1.1 O usuário clica no botão de rastreio para acompanhar o pedido em aberto. 1.2 O usuário clica no botão do carrinho enquanto há algum pedido em aberto. **Pós-Condição:** A tela de acompanhamento do pedido é exibida.

Nome do fluxo alternativo (extensão)	**Descrição**
1. Estoque insuficiente.	1.1 O ator seleciona o produto para ser adicionado ao carrinho. 1.2 O sistema valida o estoque e informa que o produto selecionado não tem estoque suficiente.

Fonte: Autoria própria (2022).

O Quadro 13 apresenta o caso de uso para que o cliente possa visualizar o histórico dos pedidos.

Quadro 13 – Caso de uso visualizar histórico de pedidos

Caso de uso: Visualizar histórico de pedidos. **Descrição:** O ator vê um breve resumo de cada pedido que já realizou. **Atores:** Cliente **Pré-condição:** Estar autenticado e já ter feito algum pedido. **Sequência de Eventos:** 1. O cliente acessa a aplicação. 2. O cliente acessa a tela “Meus Pedidos”. **Pós-Condição:** Todos os pedidos já realizados pelo cliente são exibidos.

Fonte: Autoria própria (2022).

O Quadro 14 apresenta o caso de uso para que o cliente possa manter produtos favoritos.

Quadro 14 – Caso de uso manter favoritos

Caso de uso: Manter favoritos. **Descrição:** O ator adiciona ou remove produtos como favoritos. **Atores:** Cliente **Pré-condição:** Estar autenticado. **Sequência de Eventos:** 1. O cliente acessa a aplicação. 2. O cliente clica no ícone de coração no canto superior direito de algum produto em tela. 2.1 Produto não está na lista de favoritos Produto é adicionado na lista de favoritos. 2.2 Produto já está na lista de favoritos. Produto é removido da lista de favoritos.

Pós-Condição: Os produtos são adicionados ou removidos da lista de favoritos, de acordo com a condição inicial dos mesmos.

Fonte: Autoria própria (2022).

O Quadro 15 apresenta o caso de uso para que o funcionário possa visualizar gráficos de métricas.

Quadro 15 – Caso de uso visualizar gráficos de métricas

Caso de uso: Visualizar gráficos de métricas **Descrição:** O ator visualiza alguns gráficos que dão uma informação básica sobre as vendas. **Atores:** Funcionário do supermercado **Pré-condição:** Estar autenticado e possuir vendas para a montagem dos gráficos. **Sequência de Eventos:** 1. O funcionário acessa o *dashboard* da aplicação. **Pós-Condição:** São exibidos os gráficos.

Fonte: Autoria própria (2022).

O Quadro 16 apresenta o caso de uso para que o funcionário possa atualizar o *status* dos pedidos.

Quadro 16 – Caso de uso atualizar status dos pedidos

Caso de uso: Atualizar status dos pedidos **Descrição:** O ator pode alterar o status dos pedidos de acordo com a situação atual. **Atores:** Funcionário do supermercado **Pré-condição:** Estar autenticado e possuir pedidos com situação diferente de “Entregue”. **Sequência de Eventos:** 1. O funcionário acessa a aplicação. 2. O funcionário acessa a aba “Pedidos” 3. O sistema carrega os pedidos realizados. 4. O funcionário clica na ação correspondente de acordo com o pedido que quer alterar. **Pós-Condição:** O status do pedido selecionado é alterado.

Fonte: Autoria própria (2022).

O Quadro 17 apresenta o caso de uso para que o funcionário possa manter as taxas de entrega conforme o bairro do cliente.

Quadro 17 – Caso de uso manter taxas de entrega

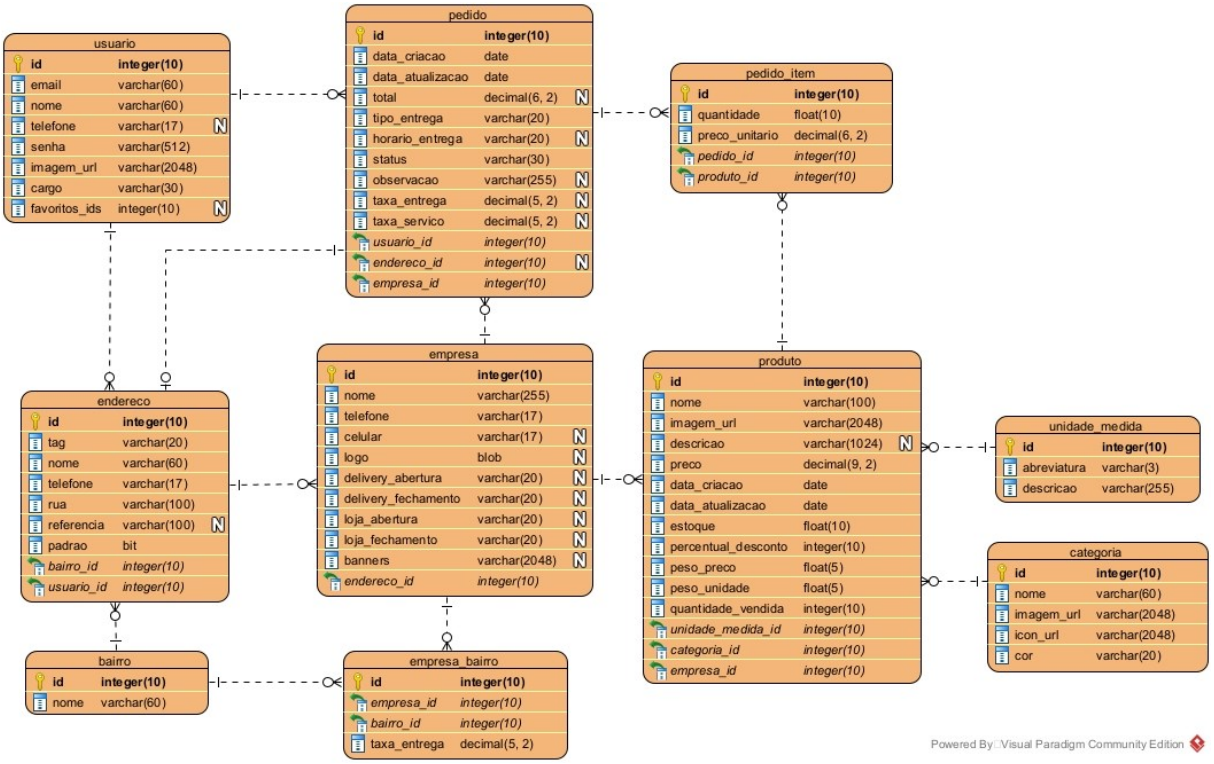
Caso de uso: Manter taxas de entrega **Descrição:** O ator consegue alterar o valor cobrado como taxa de entrega de cada bairro que esteja cadastrado no sistema. **Atores:** Funcionário do supermercado **Pré-condição:** Estar autenticado e possuir bairros cadastrados. **Sequência de Eventos:**

1. O funcionário acessa a aplicação. 2. O funcionário acessa a aba “Bairros” 1. O funcionário quer adicionar uma taxa para um bairro existente O funcionário acessa a aba “Adicionar Bairro” O sistema carrega todos os bairros que ainda não tem taxa para aquele supermercado. O ator preenche os campos e clica no botão “Cadastrar” 2. O funcionário que editar a taxa de um bairro já configurado. O funcionário acessa a aba “Listar Bairros” O ator seleciona a ação de edição. O ator informa a nova taxa de entrega e clica no botão “Editar” **Pós-Condição:** O valor da taxa de entrega dos bairros é alterado.

Fonte: Autoria própria (2022).

A Figura 2 apresenta o diagrama de entidades-relacionamentos (DER) que representa o banco de dados da aplicação.

Figura 2 – Diagrama de entidade relacionamento



Fonte: Autoria própria (2022).

O nível de acesso do usuário será definido pelo valor do “campo” cargo. A entidade “usuário” se relaciona com o pedido, sendo este o cliente do pedido. Os itens do pedido se relacionam com a entidade “pedido” e a entidade “produto”, e armazenam o preço unitário correspondente ao preço do produto na época em que ele foi vendido. A tabela “empresa” possui relacionamento com as tabelas “pedido”, “produto” e “bairros” por empresa, de modo que por

poder possuir várias empresas, esses registros deverão sempre serem filtrados pela empresa. A entidade “empresa_bairro” apresenta os valores das taxas de entrega por bairro para cada empresa. A tabela de endereço se relaciona com a de usuário, sendo esses os endereços cadastrados pelo usuário, com o pedido que será o endereço de entrega, quando o tipo de entrega do pedido não for por retirada, e com a empresa, denotando qual o endereço onde a empresa está localizada.

4.3 Apresentação do Sistema

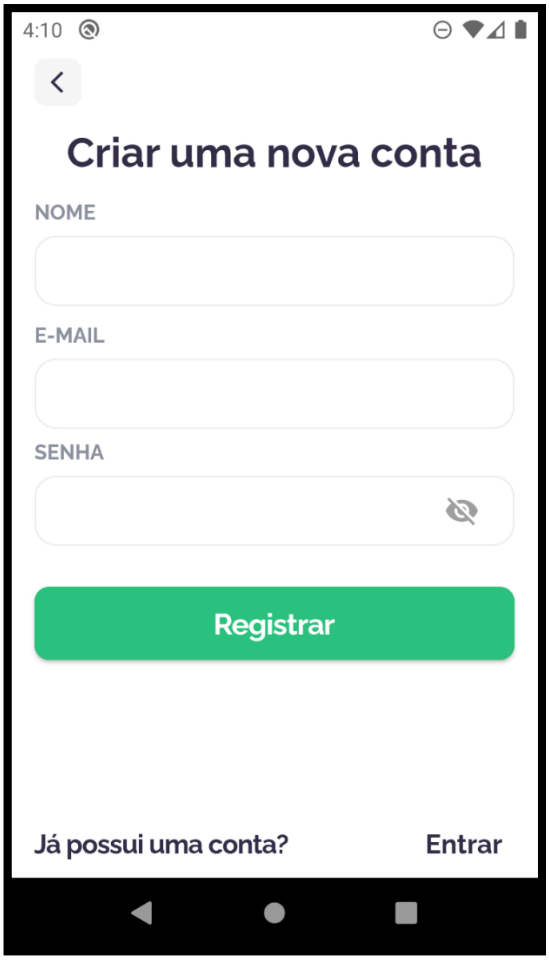
As telas a seguir apresentam as principais funcionalidades implementadas no sistema. As figuras numeradas sequencialmente de 3 a 21 apresentam as telas do aplicativo.

A Figura 3 apresenta a tela inicial do sistema, na qual o usuário pode optar por entrar ou se cadastrar e a Figura 4 apresenta a tela de cadastro do cliente.

Figura 3 - Tela de boas-vindas

Fonte: Autoria própria (2022).

Figura 4- Tela de cadastro de clientes

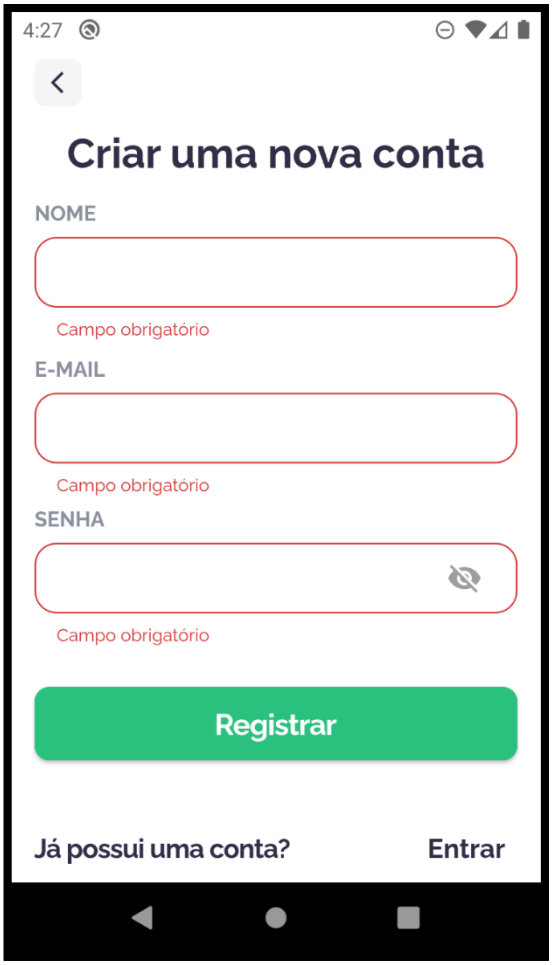


Fonte: Autoria própria (2022).

O acesso à tela de cadastro de clientes, apresentada na Figura 4, pode ser realizado por meio da opção “Criar uma conta” da tela de boas-vindas, apresentada na Figura 3. Para o registro inicial do cliente, são requisitadas as informações de nome, *e-mail* e senha.

Na Figura 5 é apresentado o padrão de preenchimento obrigatório utilizado no sistema, em que os campos recebem um contorno na cor vermelha para que o usuário possa identificar que ocorreram erros durante o preenchimento do formulário. Ainda, é apresentada uma mensagem informando qual foi o erro ocorrido. Ao corrigir o que foi requisitado na mensagem e tentar realizar a ação novamente, os campos devem voltar ao normal.

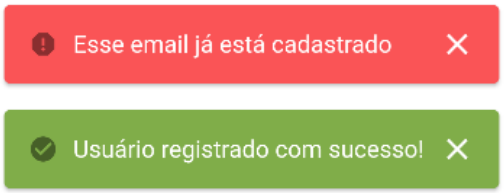
Figura 5 - Tela de cadastro de clientes apresentando erro



Fonte: Autoria própria (2022).

A Figura 6 apresenta os possíveis retornos de sucesso e falha para a tentativa de cadastro do cliente.

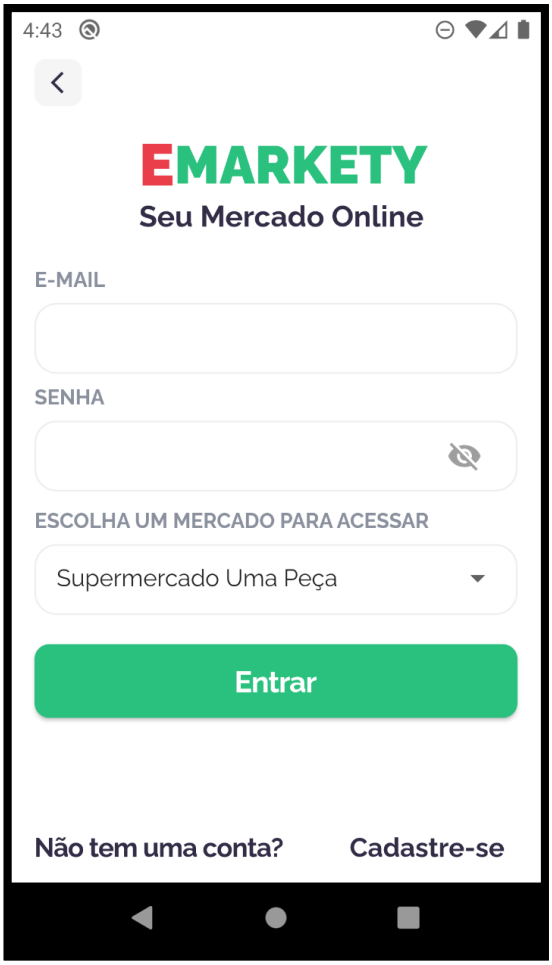
Figura 6 - Mensagens exibidas no cadastro do cliente



Fonte: Autoria própria (2022).

Na Figura 6, a primeira mensagem ocorre quando há tentativa de registrar um cliente com um *e-mail* que já está sendo utilizado. Já a segunda mensagem é apresentada ao realizar o cadastro com sucesso. A Figura 7 apresenta a tela para autenticação dos usuários no sistema.

Figura 7 - Tela de autenticação de usuários



Fonte: Autoria própria (2022).

Na Figura 7 são apresentados os campos necessários para realizar a autenticação no sistema. O campo *dropdown* "escolha um mercado para acessar" apresenta todas as empresas cadastradas no sistema, possibilitando que o usuário escolha qual delas deseja acessar. Algumas entidades do banco de dados possuem vínculo com a entidade que contém os dados da empresa,

portanto, se tornou necessária a informação de qual empresa está sendo acessada antes da autenticação, de modo a manter a integridade dos dados em relação às empresas.

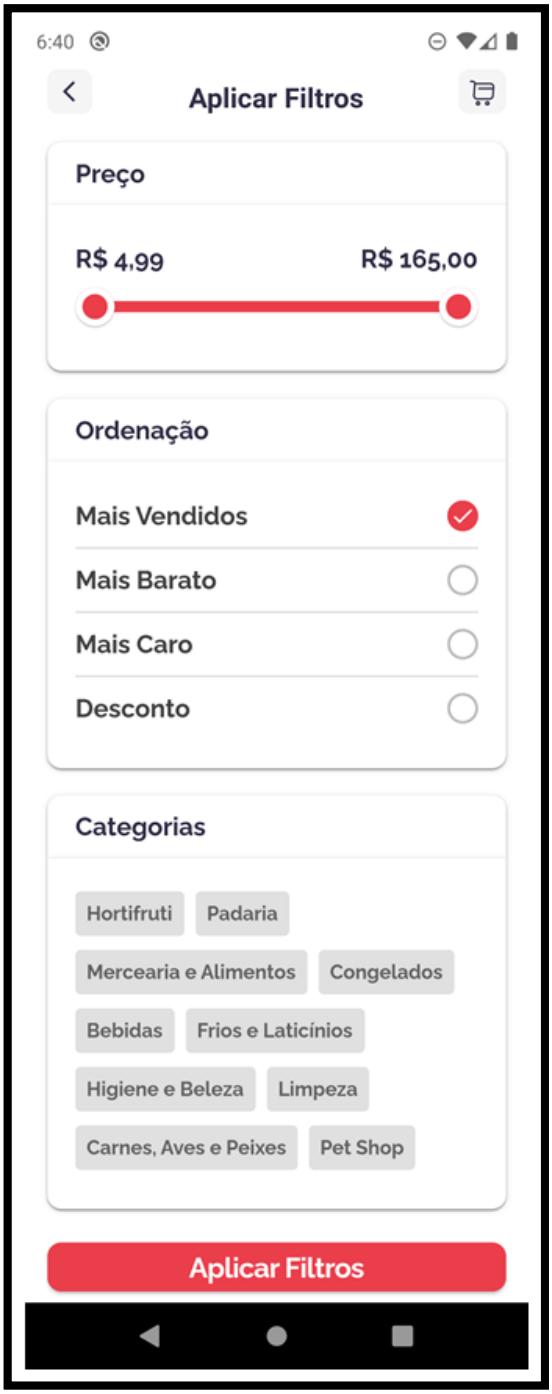
A Figura 8 apresenta a tela inicial do sistema quando os usuários estão autenticados correspondentes ao seu nome e o endereço que estiver cadastrado como padrão. Os *banners* cadastrados na empresa autenticada também são apresentados nesta tela, assim como algumas categorias e produtos mais vendidos. A barra de pesquisa e o botão de filtro, localizado na parte superior da tela, podem ser utilizados para filtrar os produtos de acordo com alguns critérios, como mostrado na Figura 9.

Na tela apresentada na Figura 9, é possível escolher o intervalo de preço dos produtos a serem exibidos, a ordem em que eles serão exibidos, sendo: mais vendidos, mais barato, mais caro ou desconto, e se pertencem a alguma categoria, como, hortifruti, padaria, mercearia e alimento, por exemplo, caso haja categoria selecionada. Ao pressionar o botão “Aplicar Filtros”, é realizada uma busca dos dados e o resultado é apresentado em uma nova tela, conforme ilustra a Figura 10. Se o usuário informar no campo de pesquisa uma palavra, um conjunto de palavras ou apenas metade de uma palavra, o sistema realiza a busca de acordo com a palavra-chave informada, trazendo todos os dados encontrados com o critério de busca informado.

Para a construção do componente de intervalo de preços, foi necessário realizar uma consulta buscando o produto com menor valor e o produto com maior valor, simbolizando os valores mínimo e máximo possíveis apresentados, respectivamente. O componente de ordenação, é um enumerador contendo as opções apresentadas, de modo que caso precise ser adicionado um novo tipo de ordenação, basta adicioná-lo ao enumerador e ele aparecerá nesta tela, visto que a ideia é percorrer todos os itens do enumerador e mostrá-los para seleção. O componente de categorias apresenta todas as categorias disponíveis que estão cadastradas, ele permite que o usuário possa escolher zero ou uma categoria, de modo que ao clicar numa nova categoria, a anterior é não será mais selecionada e a nova fica em destaque. Ao selecionar uma categoria que já está selecionada, a seleção é desfeita e é considerado que não há nenhuma categoria selecionada para a filtragem.

Figura 8 - Tela inicial para usuários autenticados

Figura 9 - Tela de filtros



Fonte: Autoria própria (2022).

Figura 10 - Tela de listagem dos produtos filtrados

Fonte: Autoria própria (2022).

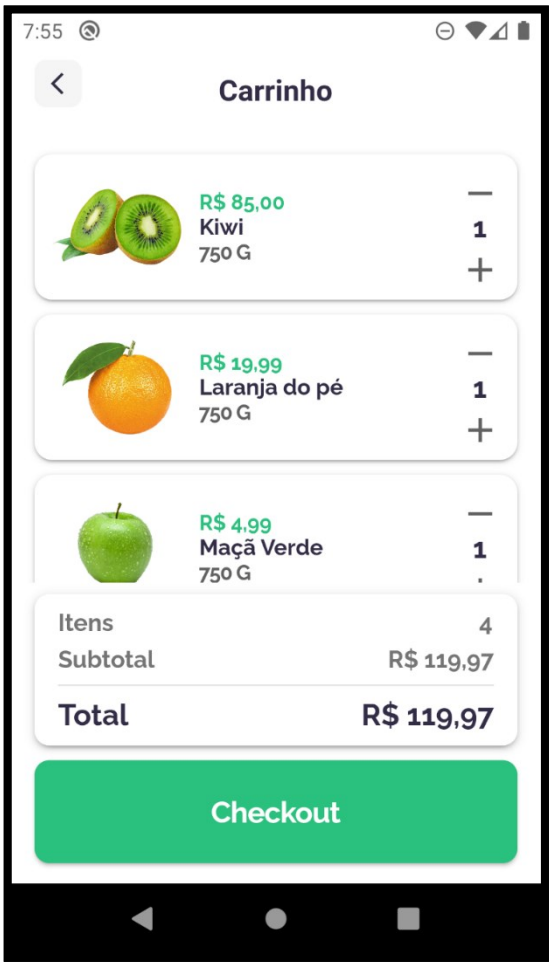
As figuras 11 e 12 apresentam as telas de carrinho de compras, demonstrando quando o carrinho está vazio e quando ele possui itens, respectivamente. Ao adicionar itens no carrinho, a tela de carrinho vazio é substituída pela tela com o produto adicionado. A Figura 12 apresenta o carrinho com alguns itens adicionados. Os valores de subtotal e total são calculados automaticamente. Caso haja desconto em algum produto, esse valor também é calculado automaticamente e apresentado no resumo dos valores.

Ao alterar a quantidade, é considerado o campo “Peso Unidade”, ou seja, para cada vez que é clicado no botão de acrescentar ou decrescer, a quantidade que será somada ou subtraída será de acordo com esse campo.

Figura 11 - Tela do carrinho vazio

Fonte: Autoria própria (2022).

Figura 12 - Tela do carrinho com produtos



Fonte: Autoria própria (2022).

É possível remover algum item do carrinho ao deslizar o *card* do item para a esquerda, pressionar o botão com ícone de lixeira e confirmação a exclusão por meio de uma caixa de diálogo, conforme pode ser observado na Figura 13.

Ao realizar a remoção de um produto do carrinho, é feita uma atualização e há a alteração automática nos valores do resumo. Caso todos os produtos do carrinho sejam removidos, a tela apresentada na Figura 11 substitui novamente a tela que apresenta a listagem dos itens do carrinho.

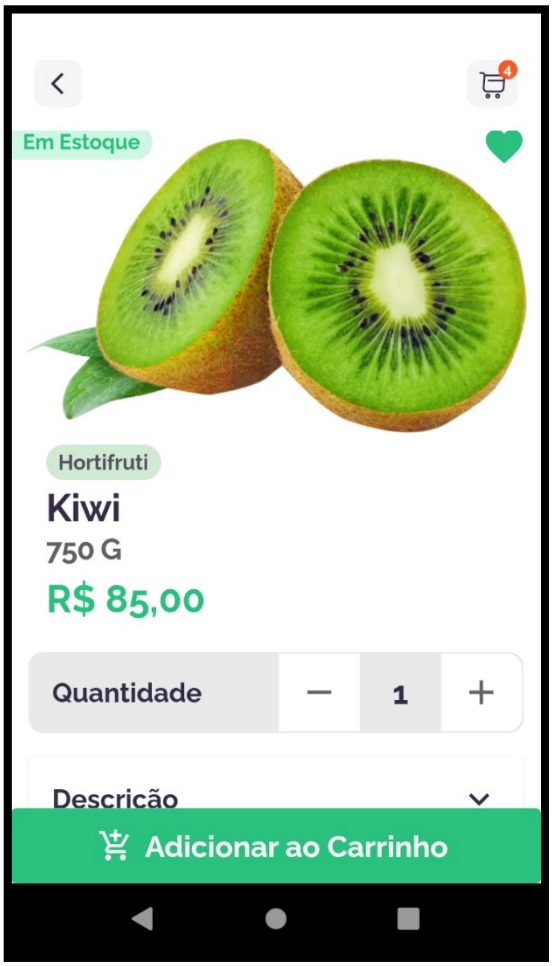
Figura 13 - Tela apresentando a exclusão de itens do carrinho

Fonte: Autoria própria (2022).

Na Figura 14 é apresentada a tela com os detalhes do produto, mostrando o valor promocional, caso haja, se existe estoque para o produto, descrição e possibilidade de alterar a quantidade antes de adicionar o produto no carrinho.

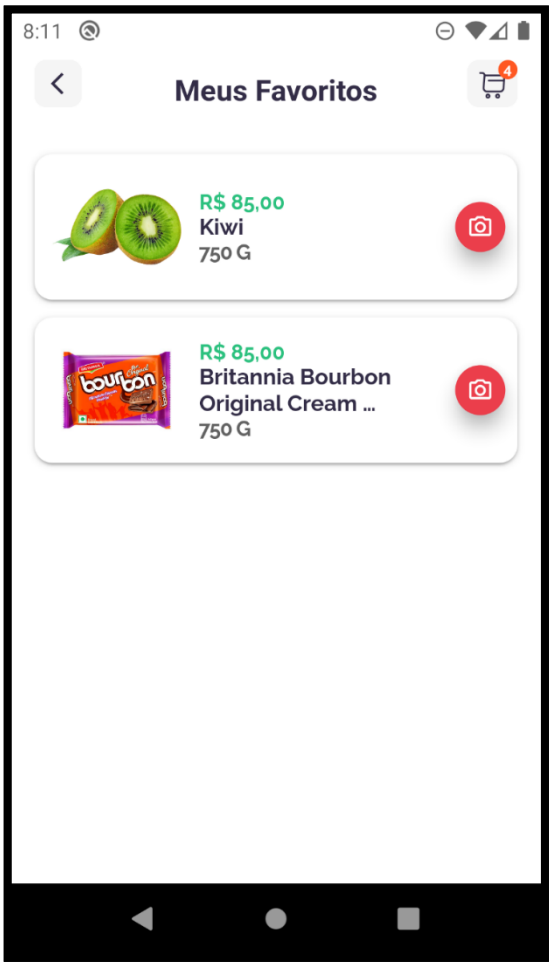
Na Figura 15 são apresentados os produtos que estão marcados como favoritos. Para adicionar algum produto nesta listagem, é necessário clicar no ícone representado por um coração apresentado no *card* do produto ou na tela de detalhe do produto. O ícone de coração preenchido representa que o produto está marcado como favorito, conforme pode ser visto na parte superior direita na Figura 14. Essa tela pode ser acessada por meio da tela de perfil do usuário.

Figura 14 - Tela de detalhes do produto



Fonte: Autoria própria (2022).

Figura 15 - Tela com listagem dos favoritos

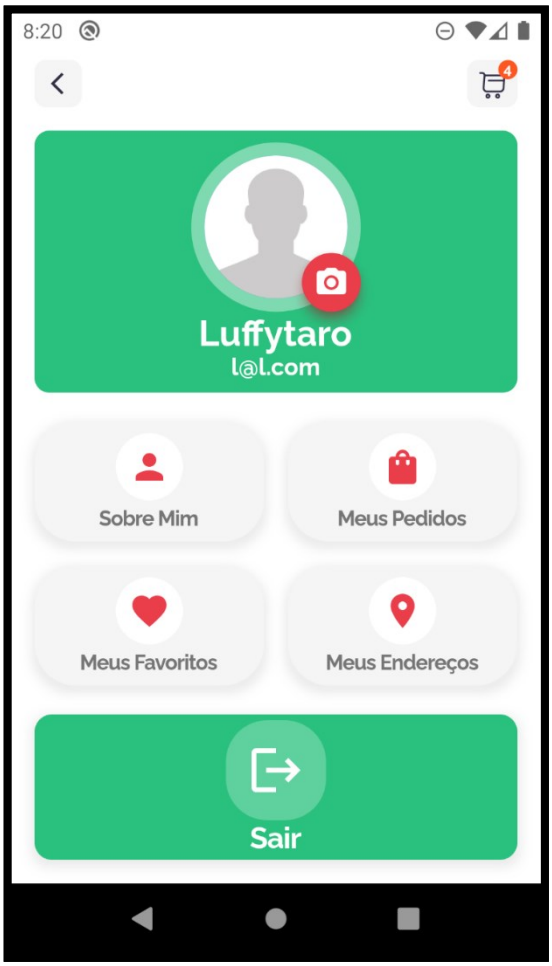


Fonte: Autoria própria (2022).

Do mesmo modo realizado na listagem de itens do carrinho, é possível remover itens da listagem de favoritos. Basta deslizar o *card* do produto para a esquerda e aparecerá o botão representado por um ícone de lixeira e, ao pressionar este botão e clicar na opção "sim" na caixa de diálogo que é apresentada, o produto será removido.

A Figura 16 apresenta a tela de perfil do usuário. A partir dessa tela, é possível acessar várias outras telas, como, por exemplo, a listagem de favoritos exibida na Figura 15. Também é possível alterar a imagem de perfil do usuário, ao pressionar o botão com ícone representado por uma câmera e informar uma URL de uma imagem válida.

Figura 16 - Tela de perfil do usuário



Fonte: Autoria própria (2022).

A Figura 17 apresenta a tela de *checkout* do pedido, com o tipo de entrega por “retirada”. No resumo da conta é realizado o cálculo dos valores e somada a taxa de serviço, caso houver. O cliente também pode adicionar observações, se desejar.

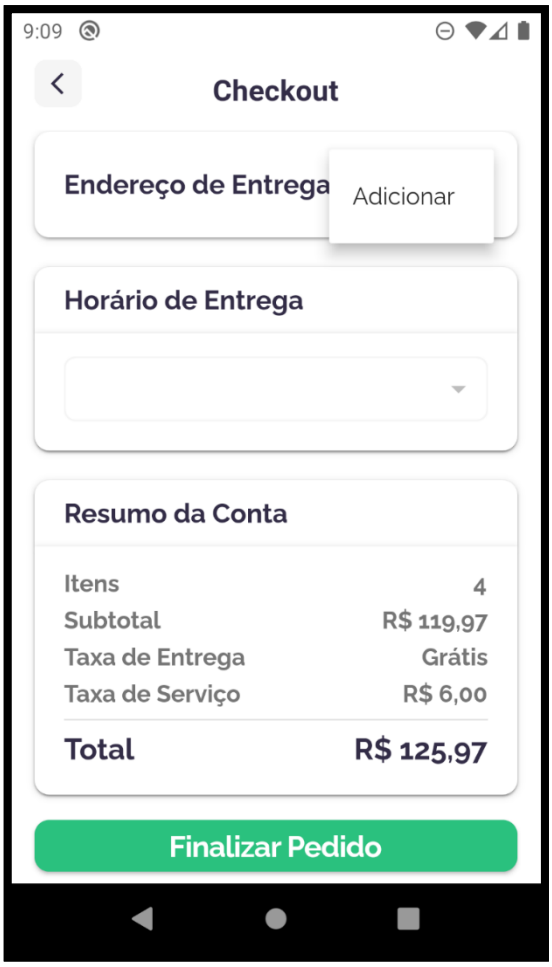
Na tela de *checkout* é possível reparar que o código do pedido é apresentado antes da realização do mesmo. Isso se deve porque o pedido já é criado momentos antes no banco de dados. Quando o usuário tenta adicionar produtos no carrinho pela primeira vez, é verificado se já existe algum pedido em aberto, caso não houver, ele é criado e mantido com o *status* de pendente. Esse *status* permanece até que o pedido seja finalizado na tela de *checkout*. O pedido com *status* de pendente não aparece na listagem de pedidos do usuário.

A Figura 18 apresenta a tela de *checkout* do pedido com tipo de entrega por “entrega”.

Figura 17 - Tela de checkout do pedido com tipo de entrega por retirada

Fonte: Autoria própria (2022).

Figura 18 - Tela de checkout do pedido com tipo de entrega por entrega

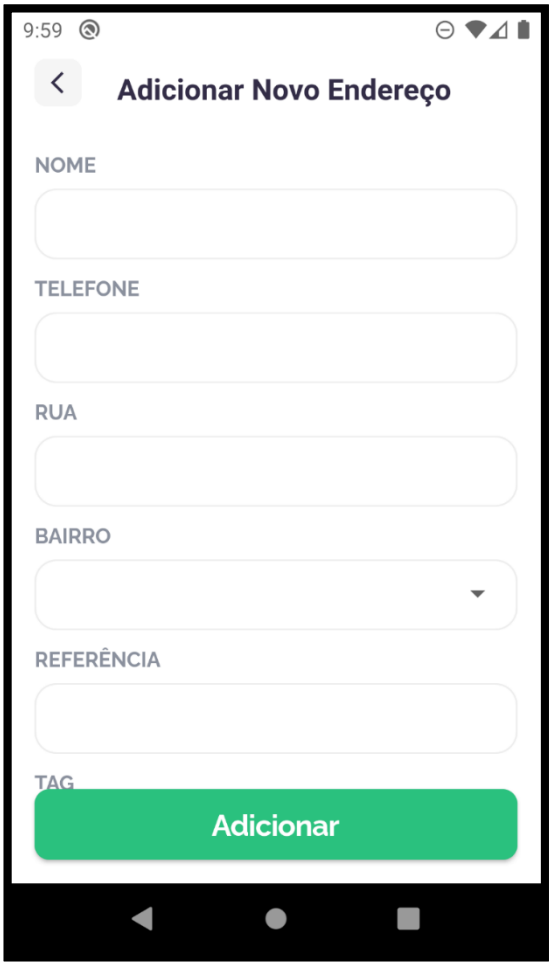


Fonte: Autoria própria (2022).

Na Figura 18, onde é apresentada a tela para finalização do pedido com tipo de entrega por entrega, são adicionados os campos “Endereço de Entrega” e “Horário de Entrega”. Caso o usuário já possui um endereço configurado como padrão, ele já virá carregado ao alterar o tipo de entrega para “entrega”. Não possuindo endereço cadastrado, a opção “Adicionar” é apresentada, conforme pode ser visto na Figura 18. O Horário de entrega é calculado baseado nos campos “Horário de abertura *delivery*” e “Horário de fechamento *delivery*”, pertecentes à empresa selecionada. Esse cálculo é feito de uma em uma hora, desde o ínicio até o fim do horário. Caso o horário atual for superior ao horário de fechamento do *delivery*, nenhum horário de entrega será carregado e portanto não será possível finalizar o pedido, retornando a mensagem de erro “Selecione um horário para entrega”.

A Figura 19 apresenta a tela para adição de um novo endereço. Ao salvar o primeiro endereço, ele automaticamente é salvo como o endereço padrão para o usuário.

Figura 19 - Tela para adição de um novo de endereço



Fonte: Autoria própria (2022).

A Figura 20 apresenta a tela para acompanhamento do pedido em aberto. Essa tela de acompanhamento é atualizada a cada 5 segundos, de forma a deixar o cliente ciente do estado atual de seu pedido. De acordo com as alterações realizadas pelos funcionários do supermercado, o pedido do usuário também é atualizado.

A Figura 21 corresponde ao histórico de pedidos, mostrando todos os pedidos que o cliente já realizou e, caso haja algum pedido em aberto, também é possível acompanhá-lo.

Também é possível clicar em cada pedido do histórico de pedidos, com essa ação será apresentada uma nova tela contendo detalhes do pedido selecionado. Nessa tela são apresentados todos os produtos que foram comprados para aquele e pedido e qual foi o preço pago para cada produto no momento da compra, visto que futuramente os preços dos produtos podem sofrer alteração.

Figura 20 - Tela para rastreio do pedido

Fonte: Autoria própria (2022).

Figura 21 - Tela de histórico dos pedidos

Fonte: Autoria própria (2022).

As figuras numeradas de 22 a 30 apresentam as telas que são exibidas nas aplicações *web* ou *desktop*, caso o usuário autenticado seja administrador ou funcionário. As figuras numeradas de 22 a 29 demonstram o uso do sistema a partir da aplicação *desktop*, enquanto a Figura 30 demonstra o acesso por meio do navegador Google Chrome. Essas aplicações estão presentes no mesmo projeto da aplicação *mobile*, de forma que, a partir de uma validação em código, as telas apresentadas para o usuário são diferentes dependendo de qual plataforma o sistema esteja sendo executado.

Existe uma mudança na tela de autenticação em relação à aplicação para dispositivos móveis. O campo para selecionar a empresa é oculto, portanto, os usuários com perfil de administrador e de funcionário não necessitam escolher qual empresa vão acessar, pois a ideia é que eles acessem a empresa em que estão vinculados. Esse vínculo é realizado por meio do endereço registrado para a empresa. Esse endereço está atrelado ao usuário, deste modo é possível saber de qual empresa esse usuário pertence.

A Figura 22 apresenta a tela de *dashboard* do módulo para administradores.

Figura 22 - Tela de *dashboard* com os gráficos

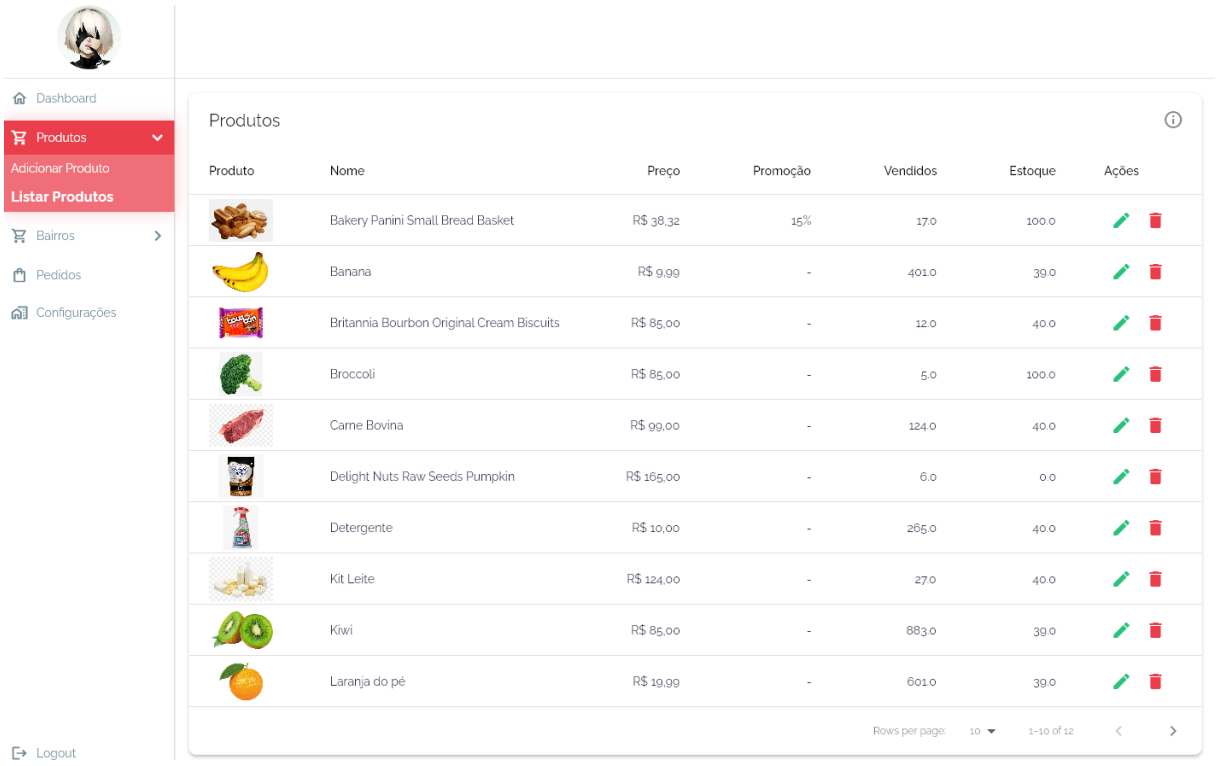
Fonte: Autoria própria (2022).

Na Figura 22 são apresentados alguns gráficos informativos, são eles: produtos mais vendidos, produtos menos vendidos, quantidade de pedidos por mês no último ano e o percentual geral dos status de pedido.

Esses gráficos só estão sendo carregados quando o funcionário se autentica na aplicação, ou seja, para que seus valores sejam atualizados, ele precisaria sair e realizar a autenticação na aplicação novamente. Isso é necessário para tentar reduzir o consumo de recursos.

A Figura 23 apresenta a listagem de todos os produtos cadastrados para a empresa autenticada. É possível realizar a ordenação dos campos clicando nos respectivos nomes das colunas. A coluna “Ações” apresenta dois ícones diferentes, permitindo a edição e a exclusão dos registros, respectivamente.

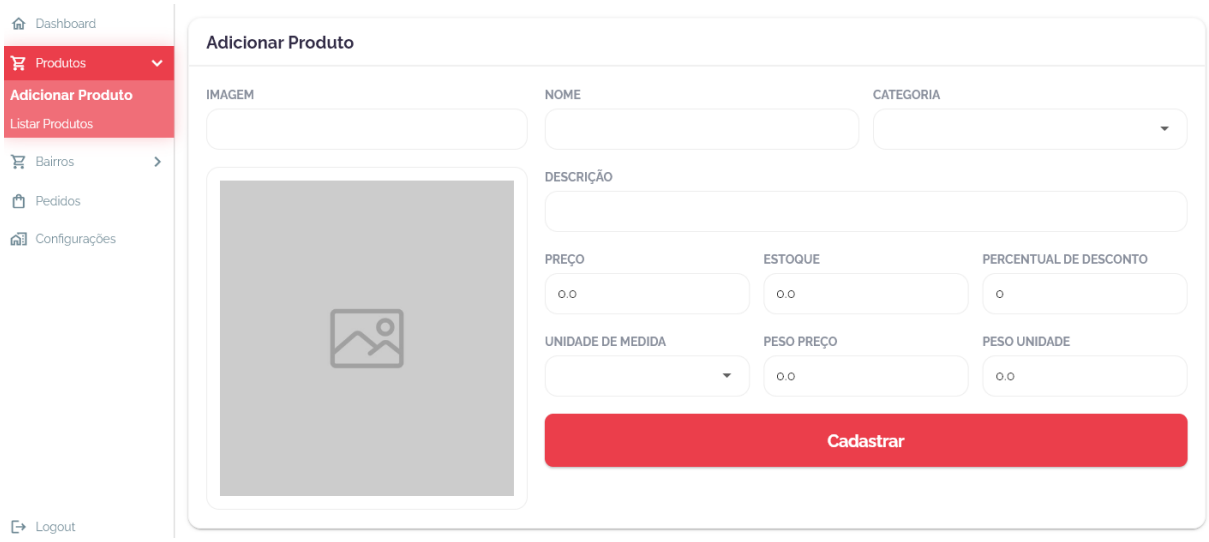
Figura 23 - Tela com listagem dos produtos



Fonte: Autoria própria (2022).

A Figura 24 exibe a tela para cadastro de um novo produto. Nessa tela é possível visualizar a imagem do produto quando informado a *Uniform Resource Locator* (URL) da imagem no campo “IMAGEM”. Caso o campo “PERCENTUAL DE DESCONTO” for salvo com o valor “0”, é considerado que o produto não terá redução no seu preço.

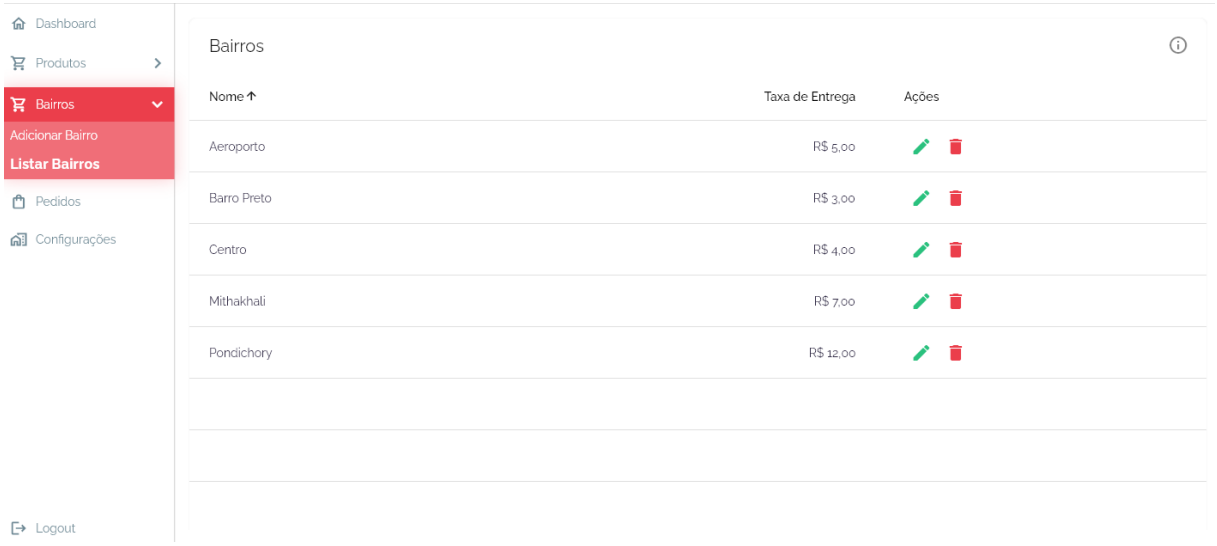
Figura 24 - Tela para cadastro de produtos



Fonte: Autoria própria (2022).

A Figura 25 apresenta a tela com a listagem dos bairros que possuem taxa de entrega cadastrados.

Figura 25 - Tela de listagem dos bairros com taxa de entrega configurados

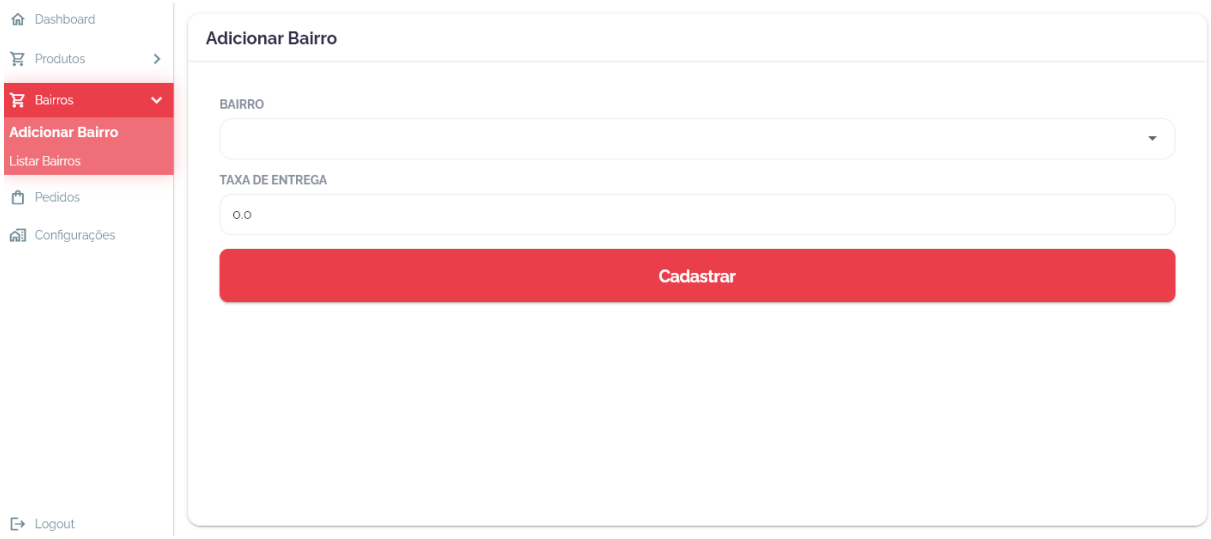


Fonte: Autoria própria (2022).

Do mesmo modo como a tabela de produtos, é possível fazer a edição e exclusão dos registros. A entrega de um pedido com tipo definido como “entrega”, só será possível se o bairro do endereço informado estiver cadastro nesta listagem. Caso contrário, ao tentar finalizar o pedido, será apresentada a seguinte mensagem “Este bairro não está cadastrado para esta empresa” e, consequentemente, não será possível finalizar o pedido.

A Figura 26 apresenta a tela para realização do cadastro da taxa de entrega para os bairros.

Figura 26 - Tela para adição de taxas de entrega para bairros

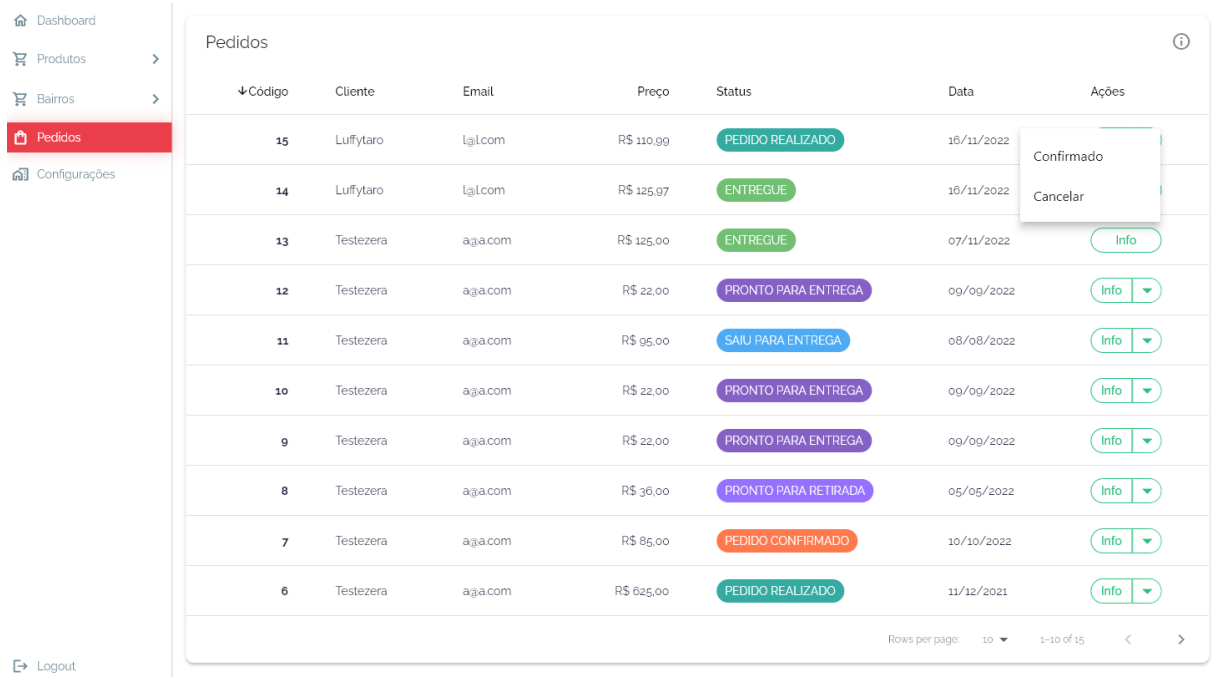


Fonte: Autoria própria (2022).

No *dropdown* dos bairros só são apresentados os bairros que ainda não possuem taxa de entrega configurada, de modo a não existir um mesmo bairro com várias taxas de entrega cadastradas.

A Figura 27 exibe a tela de listagem dos pedidos que já foram realizados para a empresa.

Figura 27 - Tela com a listagem de pedidos



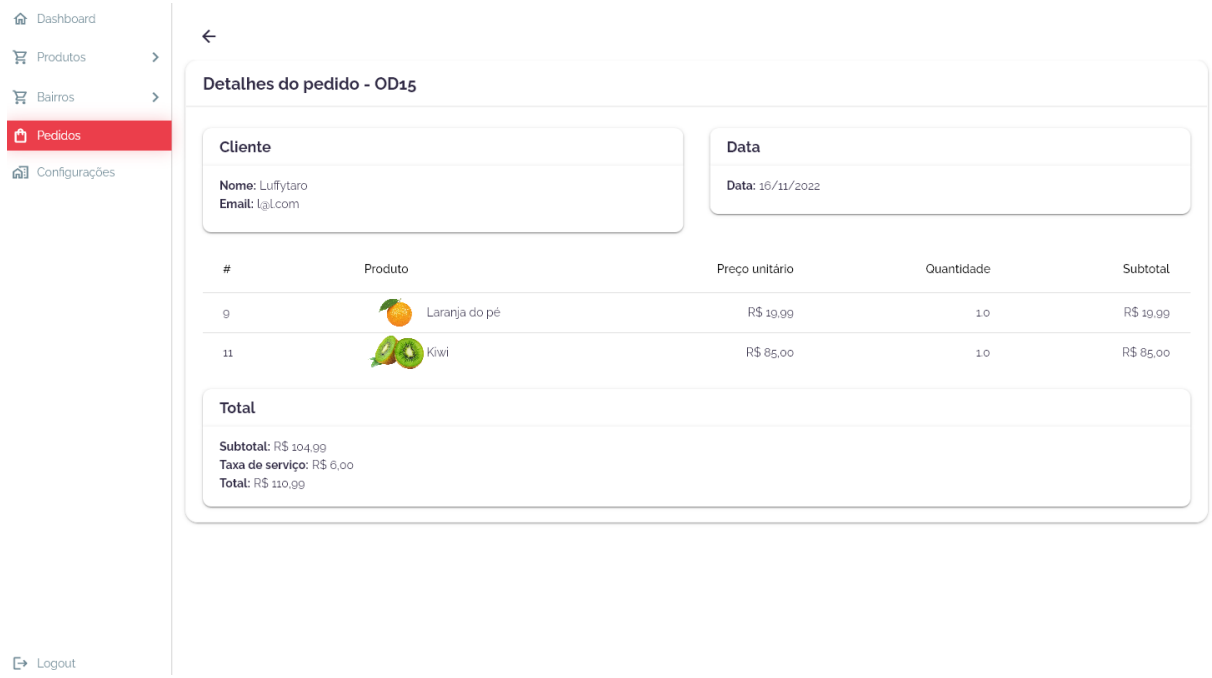
Fonte: Autoria própria (2022).

Para o botão de ações, ao clicar no botão com o ícone representado por uma seta apontada para baixo, são apresentadas as opções para alteração do *status* atual do pedido. É apresentada a opção de cancelar e o próximo *status*, de acordo com o *status* atual do pedido. O botão com a informação “Info”, é responsável por mostrar a tela que apresenta os detalhes do pedido em questão, conforme pode ser visto na Figura 28.

Pedidos pendentes não são apresentados nesta listagem e pedidos com *status* definido como “entregue” não podem mais ser alterados, somente visualizados.

A Figura 28 apresenta a tela com detalhes do pedido selecionado.

Figura 28 - Tela de detalhes do pedido

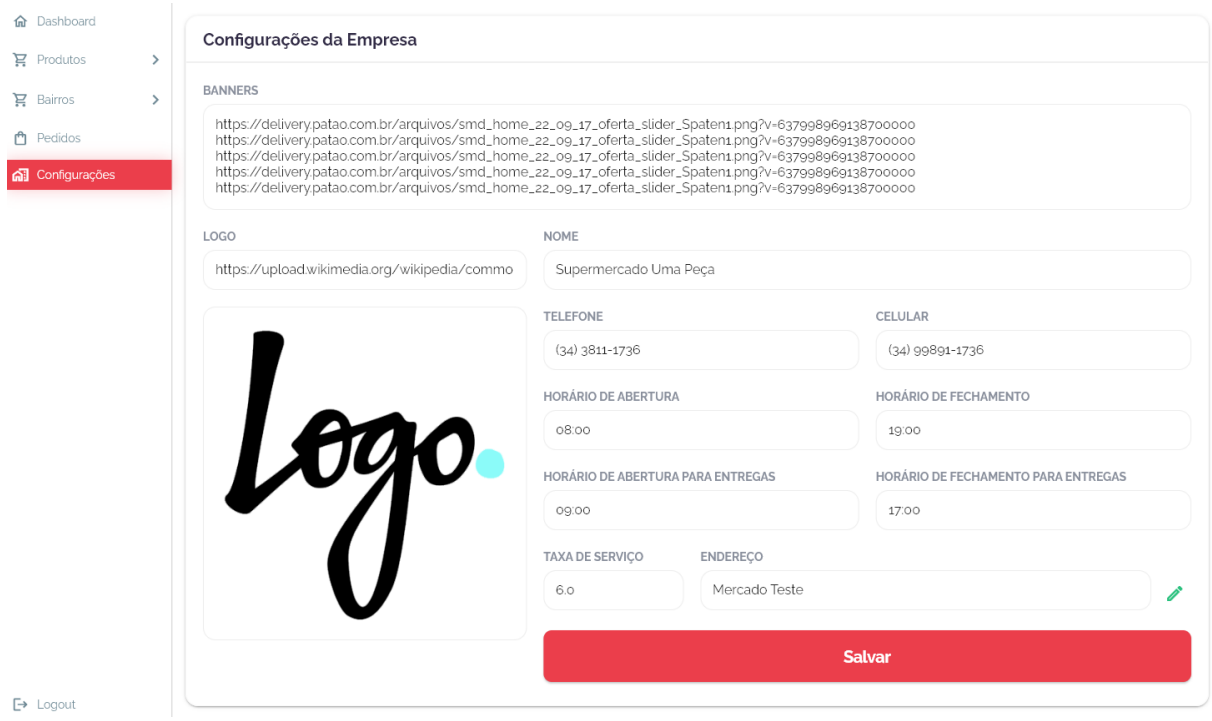


Fonte: Autoria própria (2022).

Na tela apresentada na Figura 28, são apresentadas algumas informações do pedido, tais como: qual cliente realizou o pedido, a data de realização, quais produtos foram comprados, o resumo do valor total e caso o tipo de entrega seja “entrega”, são apresentadas as informações de para qual endereço foi entregue o pedido.

A Figura 29 exibe a tela para alteração das informações da empresa. O campo “*BANNERS*” permite que sejam inseridas várias URLs de imagens que serão exibidas na tela inicial dos clientes. Várias podem ser cadastradas respeitando a regra de uma URL por linha, separando-as por meio da quebra de linha. A campo “NOME” refere-se o nome que será apresentado para se conectar nessa empresa, ao iniciar a aplicação por um dispositivo móvel. A taxa de serviço é um campo de preenchimento opcional, sendo o valor padrão definido como “0.00”, que será cobrada em qualquer pedido realizado.

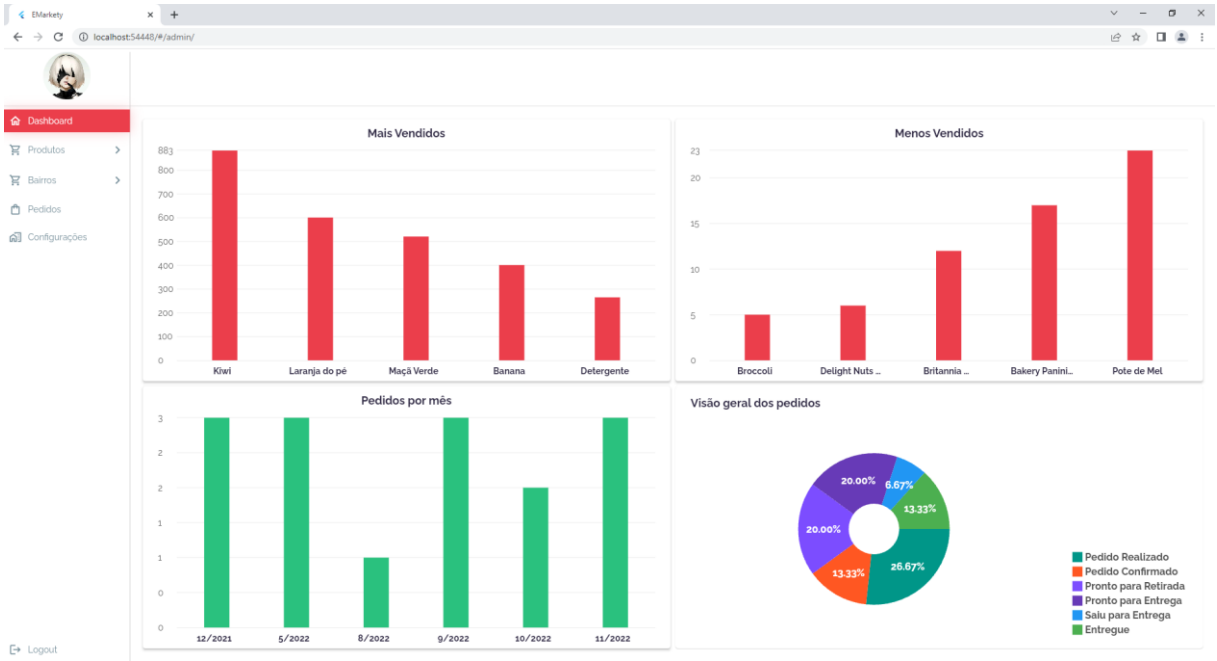
Figura 29 - Tela para edição das informações da empresa



Fonte: Autoria própria (2022).

A Figura 30 apresenta a tela de *dashboard* que será exibida ao acessar a aplicação por meio de um navegador.

Figura 30 - Tela de *dashboard* pela aplicação *web*



Fonte: Autoria própria (2022).

No quesito de visual, a aplicação é bem semelhante. Porém vale ressaltar que a maior diferença da aplicação web é a possibilidade de alterar a URL manualmente, visto que as outras plataformas não dão essa liberdade. Consequentemente, torna-se necessário um certo bloqueio de rotas, de modo a evitar acesso indevido de usuários em rotas não autorizadas.

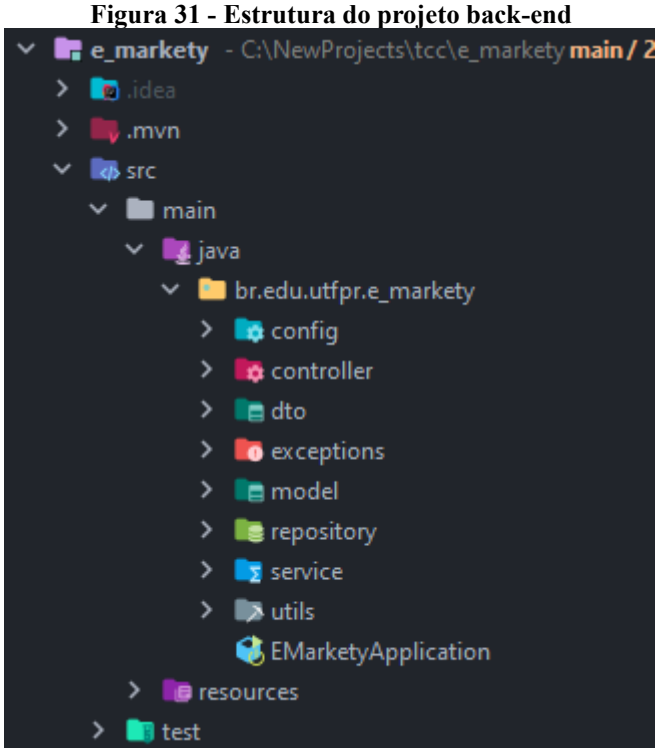
4.4 Implementação do Sistema

A apresentação da implementação do sistema é realizada por meio de partes relevantes do código desenvolvido, afim de exemplificar o uso das tecnologias presentes no Quadro 1 do Capítulo 3.

Primeiramente, foi desenvolvido o banco de dados, utilizando o sistema gerenciador de banco de dados (SGBD) PostgreSQL, juntamente com o pgAdmin como plataforma de administração e desenvolvimento do banco. Como o nicho do sistema em questão é um pouco específico, o administrador principal é capaz de adicionar certos registros que serão comuns para quaisquer empresas que possam vir a utilizar o sistema, e esses dados são adicionados por meio de um s*cript Structured Query Language* (SQL), que é executado posterior à criação do *back-end* do sistema.

Para o desenvolvimento do projeto *back-end*, foi utilizada a linguagem de programação Java, juntamente com o *framework* Spring. A Figura 31 apresenta como ficou definida a estrutura do projeto *back-end*.

Figura 31 - Estrutura do projeto back-end



Fonte: Autoria própria (2022).

Serão demonstrados pelo menos uma classe de cada pacote, afim de apresentar como foi composto este projeto. A Listagem 1 apresenta o código da classe SecurityConfiguration, localizada na pasta *security* do diretório *config*.

Listagem 1 - Configuração das permissões de acesso aos *endpoints* da aplicação

```
@EnableWebSecurity
@Configuration
@RequiredArgsConstructor
public class SecurityConfiguration {

    private final Filter authByTokenFilter;
    private final UsuarioService usuarioService;
    private final PasswordEncoder passwordEncoder;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http
            .authorizeHttpRequests(
                authorize -> {
                    try {
                        authorize
                            .antMatchers(HttpMethod.GET,
GET_URLS).hasAnyRole("ADMIN", "FUNCIONARIO")
                            .antMatchers(HttpMethod.POST,
POST_URLS).hasAnyRole("ADMIN", "FUNCIONARIO")
                            .antMatchers(HttpMethod.PUT,
PUT_URLS).hasAnyRole("ADMIN", "FUNCIONARIO")
                            .antMatchers(HttpMethod.DELETE,
DELETE_URLS).hasAnyRole("ADMIN", "FUNCIONARIO")
                            .antMatchers(HttpMethod.PATCH,
PATCH_URLS).hasAnyRole("ADMIN", "FUNCIONARIO")
                            .antMatchers(HttpMethod.GET, "/empresa",
"/usuario/current").permitAll()
                            .antMatchers(HttpMethod.POST, "/auth/**",
"/usuario").permitAll()
                            .anyRequest().authenticated()
                            .and().csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELES
S)
                            .and().addFilterBefore(authByTokenFilter,
UsernamePasswordAuthenticationFilter.class);
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }
            );
        return http.build();
    }

    // Demais métodos
}
```

Fonte: Autoria própria (2022).

Na Listagem 1, a anotação @EnableWebSecurity indica que a configuração de segurança está disponível e ao usar junto com a anotação @Configuration, que é responsável por indicar que essa classe possui métodos que definem @Beans, o Spring entende que essa classe será usada para configurar a parte de segurança da aplicação. A anotação @RequiredArgsConstructor é do Lombok e infere um construtor para a classe que tem como parâmetro todos os atributos que estiverem definidos com o modificador "final", pois devem ser inicializados no momento de construção dessa classe. Esses parâmetros são inicializados por injeção de dependência, quem faz essa injeção é o próprio Spring, mantendo o controle das instâncias durante o uso da aplicação. O método presente nesta listagem, atua como um *middleware* responsável por autorizar ou bloquear o acesso a rotas de acordo com os métodos de requisição *Hypertext Transfer Protocol* (HTTP) definidos para cada rota. Desse modo é possível permitir que o usuário precise estar autenticado para acessar certas rotas e, também, é possível que somente certos tipos de usuários possam acessar rotas específicas. As rotas bloqueadas para usuários específicos estão contidas na classe "ConfigUtils".

Na Listagem 2 é apresentada a classe GenericController, que é usada como base para os outros *controllers*.

Listagem 2 - Classe genérica para os *controllers*

```
public abstract class GenericController<ID, Y> {

    protected Sort sort = Sort.by("id");
    protected abstract GenericService<ID, Y> getService();

    @GetMapping("/page")
    public ResponseEntity<PageResponseDto<Y>>
getAllPaginated(@RequestParam int page,
                @RequestParam int size,
                @RequestParam(required = false) String order,
                @RequestParam(required = false) Boolean asc) {
        var paginatedData = getService().getAll(getPageable(page, size,
order, asc));
        return new ResponseEntity<>(PageResponseDto.of(paginatedData),
HttpStatus.OK);
    }

    protected Pageable getPageable(int page, int size, String order,
Boolean asc) {
        var pageRequest = PageRequest.of(page, size);

        if (order != null && asc != null) {
            pageRequest = PageRequest.of(page, size, asc ?
Sort.Direction.ASC : Sort.Direction.DESC, order);
        }

        return pageRequest;
    }
```

```
    @GetMapping("/{id}")
    public ResponseEntity<Y> getById(@PathVariable @NotNull ID id) {
        return new ResponseEntity<>(getService().getById(id),
HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<Y> save(@RequestBody @Valid Y dto) {
        return new ResponseEntity<>(getService().save(dto),
HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> update(@PathVariable ID id, @RequestBody
@Valid Y dto) {
        try {
            return new ResponseEntity<>(getService().update(id, dto),
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(e.getMessage(),
HttpStatus.BAD_REQUEST);
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable @NotNull ID id) {
        getService().delete(id);
        return ResponseEntity.noContent().build();
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Map<String, String>
handleValidationExceptions(MethodArgumentNotValidException ex) {
        return CustomValidator.handleMethodArgumentNotValidException(ex);
    }

    // Demais métodos
}
```

Fonte: Autoria própria (2022).

Essa é uma classe genérica, buscando reduzir a quantidade de código que comumente é igual para os outros *controllers* e dando a possibilidade de sobrescrita por meio da anotação @Override, quando necessário. Ao retornar a classe ResponseEntity nos métodos, é possível designar qual o código de *status* de resposta HTTP. As anotações @PostMapping, @PutMapping, @GetMapping e @DeleteMapping representam o verbo HTTP da requisição. A anotação @ExceptionHandler permite a sobrescrita quando a exceção especificada no parâmetro da anotação é lançada.

A Listagem 3 demonstra um *controller* herdando essa classe.

Listagem 3 - Classe responsável pelo *endpoint* de categorias

```
@RestController
```

```
@RequestMapping("categoria")
@RequiredArgsConstructor
public class CategoriaController extends GenericController<Long,
Categoria> {
    private final CategoriaService service;

    @Override
    protected GenericService<Long, Categoria> getService() {
        return service;
    }
}
```

Fonte: Autoria própria (2022).

Na Listagem 3 é apresentada a classe CategoriaController, responsável pelo *endpoint* de categorias. A anotação @RestController indica que é uma classe *Application Programming Interface* (API) *Representational State Transfer* (REST), já a anotação @RequestMapping("categoria") define o *endpoint* da requisição.

A Listagem 4 apresenta a interface GenericService.

Listagem 4 - Interface que possui a declaração dos métodos padrões para todos os *services*

```
public interface GenericService <ID, Y> {

    List<Y> getAll();

    Page<Y> getAll(Pageable pageable);

    Y save(Y dto);

    Y update(ID id, Y dto);

    Y getById(ID id);

    void delete(ID id);
}
```

Fonte: Autoria própria (2022).

A Listagem 4 representa a interface que contém os métodos que são comuns para todos os *services*. A Listagem 5, apresenta a classe abstrata que implementa esses métodos.

Listagem 5 - Classe abstrata contendo a implementação dos métodos padrões para os *services*

```
public abstract class GenericServiceImpl<T, ID, Y> implements
GenericService<ID, Y> {

    protected abstract GenericRepository<T, ID> getRepository();

    private final MapperService mapper;
    private final Class<T> entityClass;
    private final Class<Y> dtoClass;

    @Override
    @Transactional(readOnly = true)
```

```
    public List<Y> getAll() {
        List<T> list = getRepository() instanceof GenericUserRepository ?
(List<T>) findAllByUsuario(null) : getRepository().findAll();
        return mapEntityListToDto(list);
    }

    protected Iterable<T> findAllByUsuario(Pageable pageable) {
        var usuarioId = (ID) getLoggedUsuario().getId();
        var repository = (GenericUserRepository<T, ID>) getRepository();
        return pageable == null ?
repository.findAllByUsuarioId(usuarioId) :
repository.findAllByUsuarioId(usuarioId, pageable);
    }

    @Override
    @Transactional
    public Y save(Y dto) {
        return save(dto, null);
    }

    @Override
    @Transactional
    public Y update(ID id, Y dto) {
        findById(id);
        preUpdate(dto);
        return save(dto, id);
    }

    private Y save(Y dto, ID id) {
        var entity = mapDtoToEntity(dto);
        preSave(entity, id);
        entity = getRepository().save(entity);
        return mapEntityToDto(entity);
    }

    @Override
    @Transactional
    public Y getById(ID id) {
        var entity = findById(id);
        return mapEntityToDto(entity);
    }

    @Override
    @Transactional
    public void delete(ID id) {
        findById(id);
        preDelete(id);
        getRepository().deleteById(id);
    }

    protected Y mapEntityToDto(T entity) {
        return mapper.mapEntityToDto(entity, dtoClass);
    }

    protected T findById(ID id) {
        var byId = getRepository() instanceof GenericUserRepository ?
                ((GenericUserRepository<T, ID>)
getRepository()).findByIdAndUsuarioId(id, (ID)
getLoggedUsuario().getId()) :
                getRepository().findById(id);
        return byId.orElseThrow(() -> new NotFoundException((Long) id));
```

```
    }

    // demais métodos
}
```

Fonte: Autoria própria (2022).

Na Listagem 4, a classe abstrata GenericServiceImpl implementa os métodos que são declarados na interface GenericService. Para a maioria das operações, é chamado um método de um *repository* e o resultado dessa chamada é transformado para um *Data Transfer Object* (DTO). Na declaração de *generics* dessa classe existem os identificadores "T", "ID" e "Y", tal que "T" é o retorno do *repository*, sendo este a classe que representa a entidade do banco, "ID" é o identificador do tipo de valor que é usado para o "@Id" da classe e "Y" é a representação do DTO. Portanto, a classe retornada sempre é transformada em DTO antes de ser devolvida para o *controller*. Para fazer as conversões entre essas classes, é utilizado o ModelMapper. A anotação @Transactional é utilizada para realizar o controle transacional, geralmente utilizada quando há manipulação de dados do banco.

A Listagem 6 exibe a interface ProdutoService.

Listagem 6 - Interface contendo a declaração dos métodos necessários para o *service* do produto

```
public interface ProdutoService extends GenericService<Long, ProdutoDto>
{

    List<ProdutoDto> findAllByCategoriaId(Long id);

    List<ProdutoDto> findAllByIdIn(List<Long> ids);

    List<ProdutoDto> findAllByFilter(String nome, Long categoriaId,
BigDecimal precoMin, BigDecimal precoMax, Sort sort);

    void validateProdutosEstoque(List<PedidoItemDto> dtos);

    PrecoDto findMinAndMaxPreco();

    // relatórios
    List<VendaProdutos> relatorioVendaProdutos(String sort);
}
```

Fonte: Autoria própria (2022).

A interface ProdutoService herda a interface GenericService presente na Listagem 4. Ela é utilizada na classe ProdutoServiceImpl, exibida na Listagem 7.

Listagem 7 - Classe implementando os métodos declarados na interface de *service* do produto

```
@Service
@RequiredArgsConstructor
public class ProdutoServiceImpl extends GenericServiceImpl<Produto, Long,
ProdutoDto> implements ProdutoService {

    private final ProdutoRepository repository;
```

```
    private final PedidoItemRepository pedidoItemRepository;

    @Override
    protected GenericRepository<Produto, Long> getRepository() {
        return repository;
    }

    @Override
    protected void preSave(Produto entity, Long id) {
        super.preSave(entity, id);
        if (entity.getEmpresa() == null) {
            entity.setEmpresa(getLoggedEmpresa());
        }
    }

    @Override
        public List<ProdutoDto> findAllByCategoriaId(Long id) {
        var list = repository.findAllByCategoriaIdAndEmpresaId(id,
getLoggedEmpresa().getId());
        return mapEntityListToDto(list);
    }

    @Override
    public List<ProdutoDto> findAllByIdIn(List<Long> ids) {
        var list = repository.findAllByIdInAndEmpresaId(ids,
getLoggedEmpresa().getId());
        return mapEntityListToDto(list);
    }

    @Override
    public List<ProdutoDto> findAllByFilter(FiltroDto filtro) {
        var list = repository.findAllByFilter(getNome(filtro.getNome()),
filtro.getCategoriaId(),
                getPrecoMin(filtro.getPrecoMin()),
getPrecoMax(filtro.getPrecoMax()),
                getLoggedEmpresa().getId(),
filtro.getTipoOrdenacao().getSort());
        return mapEntityListToDto(list);
    }

    // demais métodos
}
```

Fonte: Autoria própria (2022).

A anotação @Service somente pode ser usada em classes e serve para indicar que essa classe é um provedor de serviços. Com essa anotação, o Spring conseguirá prover a instância dessa classe por meio da injeção de dependências.

A classe ProdutoServiceImpl herda a classe abstrata. O método getRepository() é sobrescrito, de acordo com a implementação do GenericRepository utilizada para essa classe. Por implementar a interface ProdutoService, ela obrigatoriamente precisa escrever as implementações dos métodos que foram declarados. É possível perceber na declaração da classe, a utilização dos *generics* com o DTO ao herdar a classe genérica, onde ela fica definida

como “GenericServiceImpl<Produto, Long, ProdutoDto>“, e sempre retornando o “ProdutoDto” quando é necessário o retorno da entidade Produto.

A Listagem 8 exibe a interface GenericRepository.

Listagem 8 - Repository genérico

```
@NoRepositoryBean
public interface GenericRepository<T, ID> extends JpaRepository<T, ID> {

}
```

Fonte: Autoria própria (2022).

Ela é uma interface que herda as declarações definidas na interface JpaRepository. A anotação @NoRepositoryBean indica que não será necessário criar uma instância desse repositório. Isso se torna necessário pois será usado uma instância já existente e ao tentar gerar outra resulta em um erro para subir a aplicação.

A interface JpaRepository fornece os métodos padrões para operações *Create*, *Read*, *Update*, *Delete* (CRUD), além de permitir a construção de consultas usando *Query Methods*, como o método findAllByCategoriaIdAndEmpresaId(Long categoriaId, Long empresaId), que é montada dinamicamente de acordo com os campos definidos na classe que representa a entidade do banco.

A Listagem 9 apresenta a classe Produto que representa a entidade Produto.

Listagem 9 - Entidade Produto

```
@Entity
@Getter
@Setter
@NoArgsConstructor
@EntityListeners(AuditingEntityListener.class)
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 100, nullable = false)
    private String nome;

    @Column(length = 2048, nullable = false)
    private String imagemUrl;

    @Column(length = 1024)
    private String descricao;

    @Column(nullable = false)
    private BigDecimal preco;

    @Column(name = "data_criacao", nullable = false, updatable = false)
```

```
    @CreatedDate
    private LocalDate dataCriacao;

    @Column(name = "data_atualizacao")
    @LastModifiedDate
    private LocalDate dataAtualizacao;

    @Column(nullable = false)
    private float estoque;

    @Column(nullable = false)
    @ColumnDefault("0")
    private int percentualDesconto;

    @Column(nullable = false)
    private float pesoPreco;


    @Column(nullable = false)
    private float pesoUnidade;

    @Column(nullable = false)
    private float quantidadeVendida;

    @ManyToOne
    @JoinColumn(name = "unidade_medida_id", nullable = false)
    private UnidadeMedida unidadeMedida;

    @ManyToOne
    @JoinColumn(name = "categoria_id", nullable = false)
    private Categoria categoria;

    @ManyToOne
    @JoinColumn(name = "empresa_id", nullable = false)
    @JsonIgnore
    private Empresa empresa;
}
```

Fonte: Autoria própria (2022).

A classe apresentada na Listagem 9 é responsável por representar a entidade de produto que é presente no banco de dados. A tabela produto será criada de acordo com as definições descritas nessa classe. As anotações @Getter e @Setter são da biblioteca Lombok e servem para reduzir o código *boilerplate*, evitando a necessidade de adicionar *getters* e *setters* manualmente para cada atributo da classe. A anotação @Entity indica que essa classe representa uma entidade do banco, e quando não definido a anotação @Table, é utilizado o nome da classe para indicar o nome da tabela.

A anotação @EntityListeners serve para especificar que essa classe estará escutando eventos das classes que forem especificadas no parâmetro da anotação, que nesse caso, é a classe AuditingEntityListener. Consequentemente, toda vez que há alguma alteração nessa entidade, é disparado um evento e, nesse caso, os campos com as anotações @CreatedDate e

@LastModifiedDate são alterados, de acordo com suas próprias regras, realizando assim a auditoria desses campos a cada criação/edição de um produto.

A anotação @Id indica que esse atributo será a chave primária da tabela. A anotação @Column serve para especificar como o atributo da classe será mapeado para a tabela correspondente. A anotação @JoinColumn(name=”name”) especifica a chave estrangeira do relacionamento com outra tabela. A anotação @JsonIgnore serve para que esse atributo não seja serializado ao montar os dados *JavaScript Object Notation* (JSON) que são retornados através do *controller*.

A Listagem 10 apresenta a classe ProdutoDto, que é a representação DTO da classe Produto.

Listagem 10 - Classe DTO da classe Produto

```
@Getter
@Setter
@NoArgsConstructor
public class ProdutoDto {

    private Long id;

    @NotBlank(message = "Campo obrigatório")
    private String nome;

    @NotBlank(message = "Campo obrigatório")
    private String imagemUrl;

    @Length(max = 512, message = "Descrição muito longa")
    private String descricao = "";

    @NotNull(message = "Campo obrigatório")
    @Min(value = 0, message = "Preço deve ser maior que 0")
    private BigDecimal preco;

    private LocalDate dataCriacao;

    private LocalDate dataAtualizacao;

    private float estoque = 0;

    @Min(value = 0, message = "Desconto não pode ser menor que 0%")
    @Max(value = 100, message = "Desconto não pode ser maior que 100%")
    private int percentualDesconto = 0;

    @Min(value = 0, message = "Preço do peso deve ser maior ou igual a
0")
    @Max(value = 1, message = "Preço do peso deve ser menor ou igual a
1")
    private float pesoPreco;

    @Min(value = 0, message = "Peso da unidade deve ser maior ou igual a
0")
    @Max(value = 1, message = "Peso da unidade deve ser menor ou igual a
1")
```

```
    private float pesoUnidade;

    private float quantidadeVendida = 0;

    @NotNull(message = "Campo obrigatório")
    private UnidadeMedida unidadeMedida;

    @NotNull(message = "Campo obrigatório")
    private Categoria categoria;

    @JsonIgnore
    private Empresa empresa;

    @JsonIgnore
    public Empresa getEmpresa() {
        return empresa;
    }

    @JsonProperty
    public void setEmpresa(Empresa empresa) {
        this.empresa = empresa;
    }
}
```

Fonte: Autoria própria (2022).

As anotações @NotBlank, @NotNull, @Max, @Min são usadas para fazer a validação dos dados durante a criação do objeto. A mensagem definida é a mensagem que é retornada na requisição, informando o usuário que tal campo é obrigatório ou que um valor inválido foi informado. A anotação @JsonProperty que serve para definir um *getter* ou *setter* para uma propriedade específica, nesse caso está sendo usado adicionar uma instância de empresa no produto, mas sem retorná-la serializá-la no JSON de resposta.

Na Listagem 11 é apresentado a classe NotFoundException, localizada no diretório *exceptions* do projeto.

Listagem 11 - Exceção lançada ao procurar por um registro cujo não existente

```
public class NotFoundException extends RuntimeException {

    public NotFoundException(Long id) {
        super("Não foi encontrado o registro com o id: " + id);
    }
}
```

Fonte: Autoria própria (2022).

Essa e outras exceções customizadas são utilizadas para expressar validações de erros específicas que podem ocorrer. A classe NotFoundException é lançada quando o *id* informado pelo usuário no *endpoint* da requisição, não é encontrado no banco de dados.

A Listagem 12 exibe a classe PrincipalUtils, localizada no diretório *utils*.

Listagem 12 - Classe auxiliar para pegar as informações do usuário autenticado

```
public class PrincipalUtils {

    private PrincipalUtils() {
    }

    public static Usuario getLoggedUsuario() {
        return getPrincipal().getUsuario();
    }

    public static Empresa getLoggedEmpresa() {
        return getPrincipal().getEmpresa();
    }

    private static PrincipalDto getPrincipal() {
        var principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        if (principal instanceof String) {
            throw new InvalidLoggedUserException();
        }
        return (PrincipalDto) principal;
    }
}
```

Fonte: Autoria própria (2022).

O construtor padrão dessa classe é privado, simbolizando que ela não pode ser instanciada e só poderá ser acessada com valores estáticos. Ela serve como uma classe utilitária, fazendo um *wrapping* no conteúdo salvo como "Principal". Esse conteúdo é armazenado na memória da aplicação para cada requisição realizada no servidor, de modo que ele representa o objeto de autenticação. No caso da Listagem 12, é armazenado o objeto PrincipalDto, que basicamente é uma classe incluindo as classes Usuario e Empresa, que são as informações solicitadas para se autenticar na aplicação. A validação verificando se é uma instância de String, serve para validar se existe algum usuário logado, pois quando não há usuário logado, o método getPrincipal() retorna uma String contendo o texto "anonymous".

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento, incluindo a análise de requisitos, modelagem do banco de dados e a implementação de um aplicativo para dispositivos móveis para a compra de produtos por clientes de supermercados cadastrados, além da parte de gestão para os próprios supermercados conseguirem vender seus produtos de forma *on-line*, por meio de uma aplicação *web* ou *desktop*. O projeto possui funcionalidades semelhantes a um *e-marketplace*, porém todas as operações são de acordo com a empresa selecionada durante a autenticação do cliente, além de também não possuir a funcionalidade de realizar o pagamento diretamente pelo aplicativo.

Para o desenvolvimento do servidor, foi utilizado o modelo de arquitetura REST, esse modelo introduz a ideia da troca de informações cliente-servidor sem a necessidade de manter salvo o estado atual do cliente. Por meio das requisições feitas pelo cliente são enviadas as informações necessárias para saber qual usuário e qual empresa estão autenticados na aplicação cliente. A troca de mensagens para essa aplicação ocorreu por meio do tipo de dados JSON. Foi utilizada a aplicação Postman para testar a API construída, de forma a validar as requisições antes de realizá-las diretamente pela aplicação cliente. Por meio do Postman, também foi possível gerar uma documentação contendo todos os *endpoints* cadastrados.

O desenvolvimento da API REST, foi realizado a partir do *framework* Spring, utilizando a linguagem de programação Java. O Spring facilitou bastante o desenvolvimento do *back-end*, pois possui projetos específicos bem definidos e consistentes para a construção desse tipo de API. O Spring Boot simplificou a criação do projeto e de algumas configurações. O Spring Data JPA permitiu a criação de todo o banco de dados através da criação de classes, técnica conhecida como ORM. O Spring Security permitiu criar e gerar as práticas de autorização e autenticação dos usuários na aplicação, conseguindo assim identificar qual usuário estava fazendo a requisição.

Para o banco de dados, foi utilizado o PostgreSQL que é robusto e de código aberto, e recebe atualizações de melhorias constantemente. Durante o projeto foi usado o tipo de dados embutido *ARRAY*, que representa uma coleção de dados homogêneos. O *ARRAY* permitiu salvar a lista de códigos dos produtos favoritos diretamente na entidade de usuário, sem a necessidade de criar uma nova tabela para isso, além de ter sido utilizado também para o armazenamento dos *banners* cadastrados para a empresa.

Para a aplicação *front-end*, foi utilizado o *framework* Flutter, utilizando a linguagem de programação Dart. Um dos pontos fortes da utilização do Flutter foi a possibilidade de construir

a aplicação para todas as plataformas usando uma mesma base de código. Por possuir um conjunto com vários *widgets*, foi possível produzir a aplicação quase toda sem a necessidade de utilizar componentes de terceiros. Para otimização de tempo, foram utilizados alguns componentes disponibilizados pela comunidade.

Uma das dificuldades encontradas durante o desenvolvimento do projeto, foi a inserção da entidade ‘Empresa’ no escopo. Como ela tem vínculos com pedidos, produtos e bairros, houve a necessidade de adicionar validações e filtros específicos para esses casos, além da necessidade de manter tanto no *token*, quanto no contexto do servidor, as informações de qual empresa o usuário escolheu ao se autenticar. Outra dificuldade, foi a adição do módulo *desktop* no mesmo projeto do módulo *mobile*, que acabou por consumir recursos desnecessários da aplicação.

Como trabalhos futuros, para complementar o que foi desenvolvido nesse trabalho, pretende-se implementar o pagamento do pedido via aplicativo, controle de estoque automático a partir dos softwares utilizados nas caixas registradoras dos mercados, cadastro de produtos por meio de código de barras, cadastro de marcas e subcategorias para os produtos, fazer *reviews* dos produtos adquiridos, da empresa e dos entregadores, módulo separado apenas para o funcionário que está entregando o pedido conseguir alterar o status do mesmo, criação de alertas de preço e estoque para os produtos, criação de conta e login através de redes sociais, remoção automática do *background* das imagens dos produtos, afim de ficar sempre com o fundo branco, gráficos com maior impacto estatístico para tentar ajudar no crescimento da empresa em questão, maior personalização por parte das empresas dentro da aplicação, adição de uma entidade ‘cidade’, possibilitando que empresas de outras cidades possam utilizar a aplicação, alterar para que as categorias sejam apresentadas por empresa e não como uma entidade comum para todas as empresas, adição de uma interface gráfica para cadastro dos supermercados por parte do administrador, introdução de uma tabela ‘Empresa_Usuario’ para melhor gerenciamento dos usuários que serão os funcionários das empresas.

REFERÊNCIAS

ABRASEL. (2018). Associação Brasileira de Bares e Restaurantes - **O Mercado de Delivery de Comida Fatura Mais de 10 bilhões no Brasil**. http://www.sp.abrasel.com.br/noticias/4410-27022018-mercado-de-delivery-de-alimentos-fatura-mais-de-r10-bi-no-brasil. Acesso em 14 nov. 2022.

DART Overview. Disponível em: https://dart.dev/overview. Acesso em 14 nov. 2022.

FLUTTER FAQ. Disponível em: https://docs.flutter.dev/resources/faq. Acesso em 14 nov. 2022.

FOGRO | Food & Grocery App UI Kit for Adobe XD. Disponível em: https://themeforest.net/item/fogro-food-grocery-app-ui-kit-for-adobe-xd/31303286. Acesso em 14 nov. 2022.

FRANÇA, L.; STEINHORST, C.; BOAVENTURA, R.; SIWERT, C.; SILVA, D. R. S.; CORRÊA, P.; CRISTOFOLINI, M.; DAMACENO, A. R.; OLIARI, D. E.; ANNUSECK, M.; MANFRINI, J. E. **Search Food BNU**. In: Sociedade Brasileira De Estudos Interdisciplinares Da Comunicação, 21., 2014. Anais... INTERCOM, 2014.

JORNAL DA USP, 2021. Disponível em: https://jornal.usp.br/atualidades/delivery-transformou-tendencia-em-necessidade-e-continua-em-crescimento. Acesso em: 14 de nov, 2022.

LEE, Valentino; SCHNEIDER, Heather; SCHELL, Robert. **Aplicações Móveis.** 1. ed. São Paulo: Makron Books, 2005.

MENIGHINI, Gustavo Verdi, OLIVEIRA, Jean Carlos Carvalho, SILVA, Vanessa de Cillos, PIACENTE, Fabrício José. **Impacto da pandemia na demanda por aplicativo de delivery de alimentação em Piracicaba/SP.** Research, Society and Development, v. 10, n. 6, e28310615945, 2021, disponível em <https://rsdjournal.org>. Acesso em 15 nov, 2022.

MONTENEGRO, M. R. (2020). **Do Capitalismo de Plataforma à Difusão dos Aplicativos: Apontamentos Sobre Novos Nexos Entre os Circuitos da Economia Urbana em Tempos de Covid-19**. São Paulo: Revista Brasileira de Geografia Econômica. Disponível em: https://doi.org/10.4000/espacoeconomia.17256. Acesso em 15 nov. 2022.

OLIVEIRA, T. C.; ABRANCHES, M. V.; LANA, R. M. **(In)Segurança alimentar no contexto da pandemia por SARS-CoV-2**. Cad. Saúde Pública, v. 36, n. 4, p. e00055220, 2020. Disponível em: https://doi.org/10.1590/0102-311X00055220. Acesso em 14 nov. 2022.

PEREGRINO, Fernanda. **87% dos brasileiros usam o smartphone para fazer compras pela internet**. Disponível em: https://cndl.org.br/varejosa/87-dos-brasileiros-usam-o-smartphone-para-fazer-compras-pela-internet/. Acesso em 14 nov. 2022.

POWARCZUK, Edgar. **Internet para pequenos negócios**: ferramentas para fazer bons negócios na internet / Edgar Powarczuk. -- Brasília: SEBRAE, 2012.

RONDINELLI, Júlia. **Vendas em marketplace crescem acima do total de e-commerce**. Disponível em: https://www.ecommercebrasil.com.br/noticias/vendas-em-marketplace-crescem-acima-do-total-de-e-commerce. Acesso em 14 nov. 2022.

SILVEIRA, André Felipe. Medeufome. **Software para gerenciamento de pedidos online e offline para restaurantes**. Disponível em: https://docplayer.com.br/17427091-Medeufomesoftware-para-o-gerenciamento-de-pedidos-online-e-offline-para-restaurantes.html. Acesso em: 15 nov. 2022

TERRA, Ana; PEREIRA, Frederico; RIBEIRO, Lucas; CARVALHO, Marcos. **Sistema mobile de delivery de comida online**. RE3C - Revista Eletrônica Científica de Ciência da Computação. Edição: v. 11 n. 1 (2016). Disponível em: https://revistas.unifenas.br/index.php/RE3C/article/view/165/0 Acesso em 14 nov. 2022.

TURBAN, Efraim; King, David; Lee, Jae Kiu; Liang, Ting-Peng; Turban, Deborrah C. **Electronic Commerce**: A Manegerial Perspective. New Jersey: PrenticeHall Inc., 2004.

HIGOR. **Introdução ao Modelo Cascata**. Disponível em https://www.devmedia.com.br/introducao-ao-modelo-cascata/29843. Acesso em 14 nov. 2022.