UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DOUGLAS LUIZ MONDSTOCK

APLICATIVO PARA DELIVERY EM SUPERMERCADOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2016

DOUGLAS LUIZ MONDSTOCK

APLICATIVO PARA DELIVERY EM SUPERMERCADOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

APLICATIVO PARA DELIVERY EM SUPERMERCADOS

por

DOUGLAS LUIZ MONDSTOCK

Este trabalho de conclusão de curso foi apresentado no dia 21 de junho de 2016, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Banca examinadora:

Profª. Drª. Beatriz Terezinha Borsoi
Orientadora

Profª. Me. Andreia Scariot Beulke

Prof. Me. Vinicius Pegorini

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª. Me. Soelaine Rodrigues Ascari
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

MONDSTOCK, Douglas Luiz. Aplicativo para *delivery* em supermercados. 2016. 72 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

O uso da Internet facilitou o processo de compras. A amplitude da cobertura geográfica provida por essa rede aos sistemas desenvolvidos em atendimento ao padrão *web* permitiram o acesso facilitado e bastante amplo para o comércio de bens e serviços por meio da Internet. Entre as vantagens das compras *online* estão a não necessidade de deslocamento físico até o estabelecimento comercial, a facilidade de pesquisa de preços, a comparação de produtos e a comodidade da entrega. Compras rotineiras como as de supermercado entram no rol das que se beneficiam da Internet. A entrega das mercadorias de supermercado exige algumas especificidades como o tempo. Pela alta probabilidade de compra de produtos *in natura* e de produtos que podem necessitar de refrigeração, o *delivery* em supermercados, deve ser rápido, no sentido de ser utilizado o menor tempo possível entre a saída dos produtos do supermercado e a chegada no cliente. Neste trabalho, é reportado o desenvolvimento de um aplicativo para *delivery* de supermercado com o objetivo de agilizar o processo de encaminhamento das compras, a entrega e as operações realizadas pelo cliente. O aplicativo permitirá a realização das compras e fará o encaminhamento para entrega. Dados do sistema de comércio eletrônico utilizado pelo supermercado são exportados para o aplicativo desenvolvido permitindo expor a página de produtos para a escolha do cliente compondo o carrinho de compras.

Palavras-chave: *Delivery* em supermercados. Comércio eletrônico. Compras *online*.

ABSTRACT

MONDSTOCK, Douglas Luiz. Application for delivery in supermarkets. 2016. 72 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

The use of the Internet has facilitated the purchasing process. The extent of geographical coverage provided by the network to the developed systems in in accordance with its web standard, allowed easy and plenty broad access to assets and services commerce through the Internet. Between the benefits of online shopping are no need for physical travel to the shop, facility to price searches, comparing products and the comfort of delivery. Routine shopping like supermarket purchases enter the list of those that benefit from the Internet. The delivery of supermarket materials requires some specifics as time. By the high probability of buying fresh products and products that may need refrigeration, the supermarket delivery, must be fast, in order to be used the shortest time between the exit of the supermarket products and the arrival at customer. In this work is reported the development of an application for supermarkets delivery aiming to streamline the process of routing purchasing, delivery and operations undertaken by the customer. The application will enable the realization of purchases and will forward to delivery. Data from the electronic trading system used by the supermarket will be exported to the developed application-allowing expose the product page for customer choice composing the shopping cart.

Keywords: *Delivery*. Ecommerce. *Online shopping*.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE SIGLAS

CPF	Cadastro de Pessoa Física
DAO	*Data Access Object*
DBA	*Data Base Administrator*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
XML	*eXtensible Markup Language*

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

A necessidade de ter, o mais rápido possível, os bens de consumo adquiridos é uma das consequências da falta de tempo da sociedade moderna, que ganhou ênfase com o desenvolvimento desenfreado da tecnologia. Realizar atividades diárias com facilidade, comodidade e agilidade tornaram-se parte do cotidiano do ser humano. E isso é proporcionado pelas tecnologias de informação e comunicação aplicadas a educação, entretenimento, saúde e operações comerciais, entre outros, dos mais diversos tipos e finalidades.

Com o passar do tempo, a acessibilidade tem deixado de ser uma escolha e passou a ser uma necessidade. As diversas atividades que as pessoas realizam diariamente e, muitas vezes, simultaneamente, incentiva que a tecnologia seja utilizada como forma de auxílio para desempenhar parte dessas atividades. Como exemplo, cita-se os aplicativos *web* e móveis que se destacam no uso de operações comerciais, pois são de fácil acesso se comparados aos aplicativos *desktops* e baseados em redes proprietárias.

O mercado eletrônico, de acordo com Albertin (1998), é obtido por meio da aplicação intensa de Tecnologia de Informação no mercado tradicional. Para esse autor, esse tipo de comércio é considerado uma realidade que está se consolidando rapidamente e que trará grandes benefícios para as organizações que o utilizam nas suas estratégias e poderá ser uma ameaça ainda maior para as que não utilizarem.

Dados comprovam a expansão do uso da Internet e do comércio eletrônico no mundo e no Brasil. De acordo com eCommerceOrg (2015), o Brasil, em 2012, tinha 45,6% da população com acesso à Internet e ocupava o 5º lugar no mundo em usuários de Internet. Ainda de acordo com essa fonte, o faturamento do país em comércio eletrônico em 2014 foi de 35,8 bilhões e para 2015 a estimativa é de 43 bilhões.

As empresas que não se adaptarem ao comércio eletrônico podem sofrer o risco de perder espaço com as novas possibilidades de fazer negócio e isso inclui o uso da Internet, seja para os aplicativos *web* ou para dispositivos móveis. E, ainda, a entrega a domicílio pela economia de tempo, dinheiro e praticidade proporcionada às pessoas.

Considerando esse contexto, a proposta deste trabalho é desenvolver um aplicativo que possibilite ao cliente escolher os produtos do supermercado diretamente de um *notebook*, *tablet* ou celular e recebê-los em sua residência, sem a necessidade de deslocar-se fisicamente ao supermercado. O aplicativo desenvolvido por meio deste trabalho integra *delivery* a um site de comércio eletrônico para o segmento de supermercado.

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos relacionados ao resultado pretendido com esse trabalho.

1.2.1 Objetivo Geral

Desenvolver um aplicativo web para *delivery* em supermercados para realização de pedidos com entregas a domicílio.

1.2.2 Objetivos Específicos

- Implementar uma ferramenta que possibilite ao cliente realizar suas compras de supermercado utilizando recursos da *web*.
- Desenvolver o aplicativo visando agilidade do usuário ao efetuar o pedido e o pagamento dos produtos.
- Desenvolver um aplicativo para realização de *delivey* em supermercados utilizando dados de base de dados do sistema já existente no

supermercado.

1.3 JUSTIFICATIVA

A necessidade de agilidade na realização das atividades que as pessoas têm e a praticidade provida pelos dispositivos móveis e pela Internet trazem a oportunidade de utilizar tecnologias para o desenvolvimento de aplicativos que visam conforto, praticidade e economia de tempo aos seus usuários.

A realização de compras em supermercado é uma atividade rotineira e comum que pode ser realizada de maneira *online* porque não há a necessidade de provar (no sentido de vestir para experimentar), como o caso de roupas e calçados, por exemplo. E, além disso, os itens adquiridos em um supermercado uma certa quantidade de itens, como os da cesta básica, são semelhantes entre compras distintas, facilitando, dessa forma, o processo de escolha. Assim, um sistema *web* de supermercado que tenha uma forma de escolha dos itens fácil de usar (no sentido de atender requisitos de usabilidade), que provenha forma de pagamento segura e entrega confiável pode ter um público considerável para uso.

A entrega das compras ao consumidor, o *delivery*, é importante por fornecer praticidade e economia de tempo aos clientes e é um diferencial para o supermercado que o implementa. Vários supermercados de grande porte no Brasil realizam serviço de *delivery* para compras *online*. Leal (2009) indica uma série desses supermercados e algumas especificidades dos serviços que eles oferecem como preços diferenciados para as compras *online,* taxa única de entrega independentemente do valor da compra e possibilidade de parcelamento do valor da compra quando de itens não alimentares.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico. No Capítulo 3 são apresentados os materiais e o método utilizados para o desenvolvimento do trabalho. No Capítulo 4 está o resultado da realização do trabalho que é a modelagem e a implementação do aplicativo.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho que se refere à comércio eletrônico e *delivey* em supermercados.

2.1 E-COMMERCE

Atualmente, a Internet tem se tornado a melhor forma de coletar e compartilhar informação e tem estado envolvida na logística da indústria tradicional (KISS; BICHLER, 2008). O uso da Internet para operações comerciais está desenvolvendo rapidamente em decorrência da sua localização geográfica que transcende fronteiras e da sua natureza de interoperabilidade. Essas características oferecem à Internet vantagens técnicas bem superiores em termos de competição com a indústria de comércio tradicional (HOU *et al.*, 2012).

Comércio eletrônico (*e-commerce*) é definido como a compra e venda de produtos ou serviços na Internet (LADAN, 2014). Esse autor destaca que tem havido um crescimento massivo no nível de comércio conduzido eletronicamente deste em decorrência da ampla disseminação da Internet.

2.2 DELIVERY EM SUPERMERCADOS

A pressão competitiva nos ambientes de supermercado a varejo é crescente devido aos requisitos dos clientes (MCCARTHY-BYRNE; MENTZER, 2011). Os clientes estão cada vez mais exigentes em termos de qualidade dos produtos e do atendimento. Isso implica rapidez, comodidade e segurança de compra, pagamento e entrega. Nesse contexto, os vendedores varejistas estão procurando novas maneiras de melhorar o planejamento da sua cadeia de suprimentos e os mecanismos de execução (STERNBECK; KUHN, 2014).

A abrangência de atuação do comércio eletrônico (*e-commerce*) tem transformado o mercado de varejo, desafiando a forma como os consumidores compram bens e serviços (POZZI, 2013). A Internet provê várias facilidades em relação ao comércio tradicional, dentre as quais Brynjolfsson, Yu e Smith (2003)

citam a facilidade de comparação de preços, a variedades de produtos. Pozzi (2013) acrescenta a não necessidade de o cliente deslocar-se até a loja física e a facilidade de receber suas compras em casa. Entregas a domicílio é particularmente significativo para bens comprados com regularidade. Para esses autores provendo acesso em massa para entregas a domicílio, o comércio eletrônico elimina os custos físicos das compras.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para a realização deste trabalho. Os materiais estão relacionados às tecnologias e ferramentas utilizadas e o método apresenta a sequência das principais atividades realizadas.

3.1 MATERIAIS

No Quadro 1 estão listados os materiais utilizados para a modelagem e a implementação do aplicativo que será obtido como resultado da realização deste trabalho.

Ferramenta	Versão	Referência	Finalidade
Visual Paradigm	10_0 sp1	http://www.visual-paradigm.com/	Modelagem dos requisitos, casos de uso e diagrama de entidades e relacionamentos do banco de dados, modelagem do banco de dados.
Eclipse Luna	Luna SR2	https://eclipse.org/downloads/	Ambiente de desenvolvimento.
Apache Tomcat	7.0.55	tomcat.apache.org/	Servidor *web*.
Postgres	9.3.5-1	http://www.postgresql.org/	Banco de dados.
PgAdmin	3	http://www.pgadmin.org/	Administrador do banco de dados.
Java	JDK 1.8	http://www.oracle.com/br/index.html	Linguagem de programação.
JSON		http://json.org/	Intercâmbio de dados.
Bootstrap	3.3.4	http://getbootstrap.com/	Ferramenta para desenvolvimento responsivo de sites.
XML		https://www.w3.org/XML/	Formatação de dados
Spring Boot	v1.1.8	http://projects.spring.io/spring-boot/	Para uso de convenções sobre configurações

Quadro 1 - Ferramentas e tecnologias

3.2 MÉTODO

Como método foi utilizado o processo sequencial linear de Pressman (2008). Esse modelo foi utilizado porque os requisitos são poucos, embora a aplicação tenha implementação relativamente complexa em decorrência das tecnologias e das regras de negócio envolvidas. A seguir estão expostas as fases desse processo.

Requisitos

Foi realizada uma pesquisa por aplicativos semelhantes ou que realizassem funcionalidades relacionadas à *delivery* de supermercados como, o aplicativo www.mercode.com.br/ que possui alguns aspectos semelhantes. Essa busca auxiliou a redefinir regras de negócio e o escopo do aplicativo. Uma dessas regras é que o banco de dados do supermercado exportaria os dados dos produtos por meio de um arquivo em formato *eXtensible Markup Language* (XML) que disponibilizaria para venda no site do aplicativo desenvolvido como resultado deste trabalho e o aplicativo disponibiliza para o supermercado outro arquivo XML com a movimentação desses produtos para que possa ser feita devidamente a saída no banco de dados do supermercado.

Análise e projeto

A modelagem foi realizada por meio de diagramas de casos de uso, de classes e de entidades e relacionamentos do banco de dados. Para a elaboração dos casos de uso os requisitos foram identificados e listados, organizados em funcionais e não funcionais.

Na fase de análise e projeto também foi definido o uso das tecnologias, no sentido da aplicação dos seus recursos.

Desenvolvimento

O desenvolvimento foi realizado utilizando a linguagem Java e demais tecnologias relacionadas que estão listadas no Quadro 1.

4 RESULTADO

Este capítulo apresenta o resultado da realização do trabalho que é a modelagem de um sistema para gerenciamento de *delivery* em supermercados.

4.1 ESCOPO DO SISTEMA

O aplicativo computacional implementado como resultado deste trabalho possibilita aos clientes de um supermercado a realização de suas compras rotineiras pela *web*. O administrador de banco de dados, *Data Base Administrator* (DBA), do supermercado será responsável por disponibilizar um *eXtensible Markup Language(*XML*)* com os produtos do supermercado. A partir da tabela de produtos será formada a tela inicial de compras do aplicativo com respectivos campos para cada produto.

O aplicativo disponibiliza uma listagem dos produtos com a imagem (foto) do mesmo, o valor unitário e um campo para o cliente inserir a quantidade desejada. Os itens escolhidos pelo cliente serão adicionados a um carrinho de compra, desde que haja a quantidade solicitada disponível em estoque.

Assim, após a escolha dos itens ao carrinho o cliente definirá a forma de pagamento pelo sistema e finalizará a compra. Ao finalizá-la será emitido um comprovante de pedido e um e-mail é enviado para o usuário avisando que seu pedido foi realizado.

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a listagem dos requisitos funcionais do sistema.

Identificação	Nome	Descrição
RF01	Importar produtos no banco de dados do supermercado	O sistema recebe um XML com os dados dos produtos do supermercado e este são utilizados para compor a página de apresentação de produtos do sistema.
RF02	Selecionar produto	Selecionar o produto desejado para que o cliente possa ver os detalhes do mesmo e adicioná-lo no carrinho de compras, indicando a quantidade que deseja comprar.
RF03	Gerenciar carrinho de	Adicionar e/ou remover itens do carrinho de

	compra	compras
RF04	Exportar compras dos clientes	Disponibilizar ao Administrador DBA um XML com os dados de movimentação dos produtos.
RF05	Escolher forma de pagamento	Oferecer ao cliente as opções disponíveis para pagamento ex: Boleto, cartão de crédito, etc.
RF06	Finalizar compra	Verificar se os procedimentos da compra do cliente estão corretos e enviar um e-mail de confirmação para o cliente com os dados da compra.
RF07	Listar produtos	Listar produtos por categorias pré-definidas no sistema.
RF08	Cadastrar cliente	O cliente deve efetuar o cadastro para efetuar compras no sistema.
RF09	Cadastrar usuário	Permitir ao gerente do supermercado cadastrar um atendente para administrar funções como relatórios do sistema e para emitir os pedidos.
RF10	Emitir pedido	Após confirmação de pagamento é realizada a emissão do pedido para o atendente separar os produtos relacionados ao pedido.

Quadro 2 - Requisitos funcionais

O Quadro 3 apresenta os requisitos não funcionais do sistema.

Identificação	Nome	Descrição
RNF 01	Validar Cadastro de Pessoa Física (CPF)	O sistema deve realizar a verificação de autenticidade do CPF.
RNF 02	Validar saldo de estoque	Só será possível ao cliente adicionar itens ao carrinho se os mesmos possuírem saldo positivo no estoque.
RNF03	Logar no sistema	É obrigatório estar logado para poder efetuar compra no sistema.
RNF 04	Enviar *email*	Após o pedido ser realizado com sucesso um *email* é enviado ao cliente para informá-lo da compra realizada.

Quadro 3 - Requisitos não funcionais

O diagrama de casos de uso, apresentado na Figura 1, contém as funcionalidades essenciais do sistema realizadas pelos seus atores, que são: cliente, administrador do sistema e atendente. O Administrador tem acesso a opções de relatórios e controle de usuários, assim como importar produtos do supermercado e exportar as movimentações dos produtos no sistema. O cliente irá realizar ações como buscar produtos, manter cliente, gerenciar carrinho de compras e realizar compras. O atendente é responsável por imprimir os pedidos realizados para que possam ser separados no supermercado.

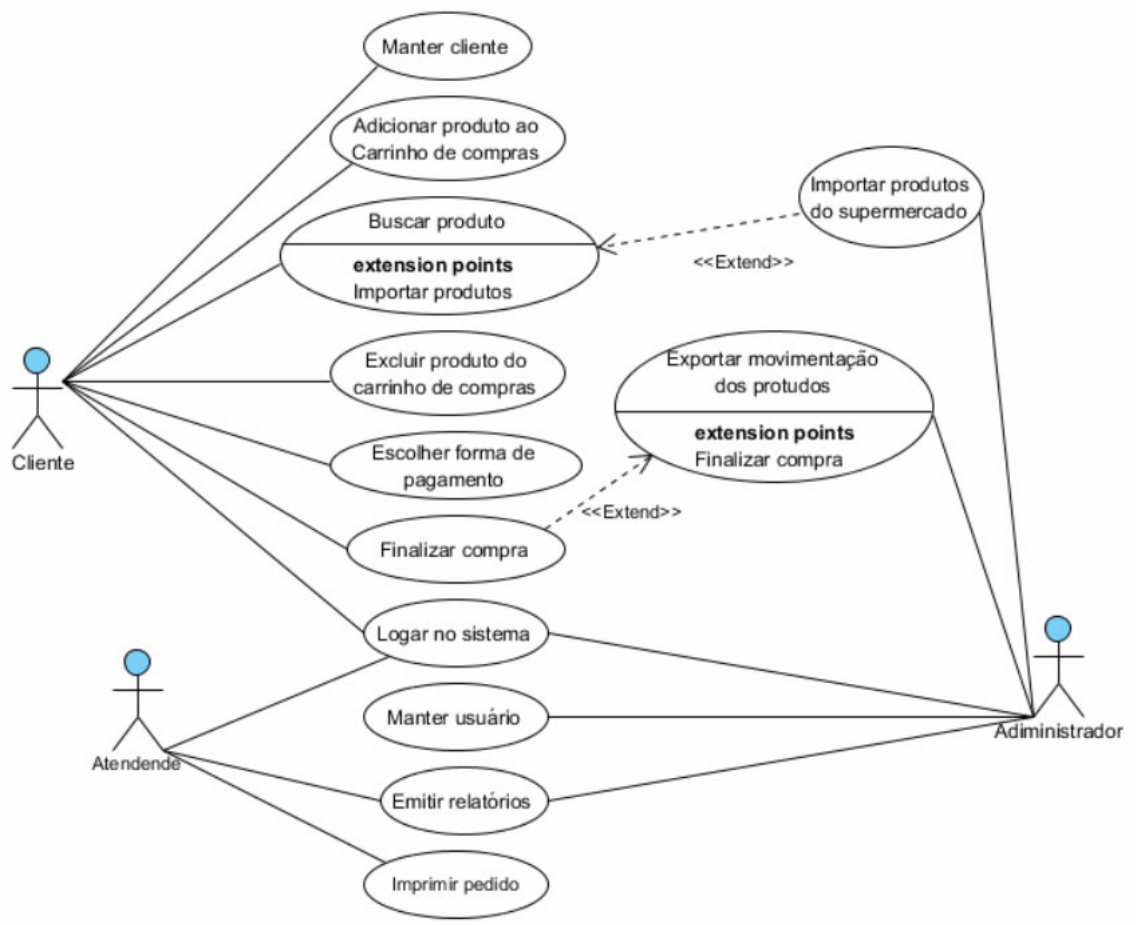

Figura 1 - Diagrama de casos de uso

No Quadro 4 está apresentado o caso de uso cadastrar cliente.

Caso de Uso: Manter Cliente
Atores: Cliente.
Precondições: Numero do CPF, endereço, *email*.
Pós-condições: Dados do cliente inseridos no banco de dados.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator informa os dados. **2.** O sistema verifica os dados
Tratamento de Exceções e Variantes: **Exceção 1a: Cpf, endereço ou *email* não existem.** 1a.1 O sistema informa ao ator que os dados não conferem. 1a.2 Retorna ao evento 1 do fluxo principal. Variante 1a.1: Finaliza o caso de uso.

Quadro 4 - Caso de uso cadastrar cliente

No Quadro 5 está apresentado o caso de uso buscar produto.

Caso de Uso: Buscar produto
Atores: Cliente**.**
Precondições: Nome do produto.
Pós-condições: Lista dos produtos.

Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator informa o produto que deseja buscar. **2.** O sistema verifica disponibilidade desse produto e retorna ao ator o produto com seus dados.
Tratamento de Exceções e Variantes: **Exceção 1a: Produto não encontrado.** 1a.1 O sistema informa ao ator que o produto não foi encontrado. 1a.2 O sistema retorna sugestões de produtos semelhantes ao ator. Variante 1a.1: Finaliza o caso de uso.

Quadro 5 - Caso de uso buscar produto

No Quadro 6 está apresentado o caso de uso logar no sistema.

Caso de Uso: Logar no sistema.
Atores: Atendente, Cliente.
Precondições: Possuir cadastro no sistema e digitar email e senha.
Pós-condições: Ator logado no sistema.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator informa o *email* e senha. 2. O sistema verifica se *email* e senha estão cadastrados **3.** Ator tem acesso a tela de cliente.
Tratamento de Exceções e Variantes: **Exceção 1a: endereço de *email* ou senha não encontrados.** 1a.1 O sistema informa ao ator que os dados não conferem. 1a.2 O sistema disponibiliza opção de alteração de senha ao ator 1a.3 Retorna ao caso de uso 1 do fluxo principal. Variante 1a.1: Finaliza o caso de uso.

Quadro 6 - Logar no sistema

No Quadro 7 está apresentado o caso de uso adicionar produto ao carrinho de compras.

Caso de Uso: Adicionar produto ao carrinho de compras
Atores: Cliente.
Precondições: Produto disponível em estoque.
Pós-condições: Produto adicionado ao carrinho de compras.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator seleciona o produto que deseja adicionar ao carrinho. 2. O sistema adiciona o produto ao carrinho e retorna uma mensagem ao cliente **3.** O sistema disponibiliza opção ao cliente para adicionar mais produtos.
Tratamento de Exceções e Variantes: **Exceção 1a: Produto indisponível.** 1a.1 O sistema informa ao ator que o produto está indisponível no momento e não disponibiliza opção de adicionar ao carrinho. Variante 1a.1: Finaliza o caso de uso.

Quadro 7 - Caso de uso adicionar produto ao carrinho de compras

No Quadro 8 está apresentado o caso de uso excluir produto do carrinho de compras.

Caso de Uso: Excluir produto do carrinho de compras
Atores: Cliente.
Precondições: Produto inserido no carrinho de compras.
Pós-condições: Produto excluído do carrinho de compras.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator seleciona o produto para excluir. **2.** O sistema exclui o produto e retorna uma mensagem ao ator.
Tratamento de Exceções e Variantes: **Exceção 1a: Nenhum produto adicionado no carrinho de compras.** 1a.1 O sistema informa que não existem produtos no carrinho para exclusão. Variante 1a.1: Finaliza o caso de uso.

Quadro 8 - Caso de uso excluir produto do carrinho de compras.

No Quadro 9 está apresentado o caso de uso escolher forma de pagamento.

Caso de Uso: Escolher forma de pagamento
Atores: Cliente.
Precondições: Produtos adicionados no carrinho de compras, ator logado.
Pós-condições: Forma de pagamento escolhida.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator informa a forma de pagamento. 2. O sistema verifica qual forma de pagamento escolhida. **3.** Se forma de pagamento escolhida for cartão de crédito, o sistema verifica o cartão de crédito.
Tratamento de Exceções e Variantes: **Exceção 1a: Cartão de crédito inválido.** 1a.1 O sistema informa ao ator que o cartão de crédito é inválido. 1a.2 Retorna ao fluxo principal do caso de uso. Variante 1a.1: Finaliza o caso de uso.

Quadro 9 - Caso de uso escolher forma de pagamento.

No Quadro 10 está apresentado o caso de uso finalizar compra.

Caso de Uso: Finalizar compra.
Atores: Cliente.
Precondições: Carrinho possui produtos adicionados, forma de pagamento foi escolhida, produto possui saldo em estoque.
Pós-condições: Compra finalizada.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator confere os dados da compra e finaliza. **2.** O sistema verifica os dados da compra e finaliza.
Tratamento de Exceções e Variantes: **Exceção 1a:** 1a.1 O sistema informa ao ator que os dados não conferem. 1a.2 Retorna ao fluxo principal. Variante 1a.1: Finaliza o caso de uso.

Quadro 10 - Caso de uso finalizar compra.

No Quadro 11 está apresentado o caso de uso cadastrar usuário.

Caso de Uso: Cadastrar usuário
Atores: Administrador, Atendente.
Precondições: Ator estar logado no sistema.
Pós-condições: Dados do usuário inseridos no banco de dados.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator informa os dados do usuário. **2.** O sistema verifica os dados.
Tratamento de Exceções e Variantes: **Exceção 1a: email ou senha inválidos.** 1a.1 O sistema informa ao ator que os dados são inválidos. 1a.2 Retorna ao caso de uso 1 do fluxo principal. Variante 1a.1: Finaliza o caso de uso.

Quadro 11 - Caso de uso cadastrar usuário

No Quadro 12 está apresentado o caso de uso emitir relatórios.

Caso de Uso: Emitir relatórios
Atores: Administrador, Atendente.
Precondições: Ator estar logado no sistema.
Pós-condições: relatório exibido ao ator.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O Ator seleciona o tipo de relatório que deseja. **2.** O sistema retorna o relatório ao ator.
Tratamento de Exceções e Variantes: **Exceção 1a: Não existem dados referentes ao relatório.** 1a.1 O sistema informa ao ator que não foi possível gerar o relatório. 1a.2 Retorna ao fluxo principal do caso de uso. Variante 1a.1: Finaliza o caso de uso.

Quadro 12 - Caso de uso emitir relatórios

No Quadro 13 está apresentado o caso de uso exportar produtos no estoque.

Caso de Uso: Importar produtos no estoque.
Atores: Administrador.
Precondições: Acesso a inclusão de produtos.
Pós-condições: Itens de estoque do banco de dados do supermercado importados de um arquivo XML.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator exporta os dados para o banco do sistema. **2.** O sistema retorna uma mensagem ao cliente
Tratamento de Exceções e Variantes: **Exceção 1a: nenhum dado a ser exportado.** 1a.1 O sistema informa ao ator que os dados não foram alterados. 1a.2 Retorna ao fluxo principal do caso de uso. Variante 1a.1: Finaliza o caso de uso.

Quadro 13 - Caso de uso exportar produtos no estoque

No Quadro 14 está apresentado o caso de uso importar compras do cliente.

Caso de Uso: Exportar movimentação dos produtos.
Atores: Administrador.
Precondições: Acesso a movimentação de produtos.
Pós-condições: Dados da movimentação de produtos exportados para o banco de dados do supermercado.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: 1. O ator exporta os dados para o banco do supermercado. **2.** O sistema retorna uma mensagem ao cliente.
Tratamento de Exceções e Variantes: **Exceção 1a: nenhum dado a ser exportado.** 1a.1 O sistema informa ao ator que os dados não foram importados. Variante 1a.1: Finaliza o caso de uso.

Quadro 14 - Caso de uso importar compras do cliente

No Quadro 15 está apresentado o caso de uso imprimir pedido.

Caso de Uso: Imprimir pedido
Atores: Atendente.
Precondições: Finalizar compra, Pedido finalizado.
Pós-condições: Separação dos produtos para entrega.
Sequência típica de eventos (Fluxo Principal): Esse caso de uso inicia quando: **1.** O ator imprime o pedido.
Tratamento de Exceções e Variantes: **Exceção 1a: não existem pedidos pendentes.** 1a.1 O sistema informa ao ator que não existem pedidos pendentes. Variante 1a.1: Finaliza o caso de uso.

Quadro 15 - Caso de uso imprimir pedido

A Figura 2 apresenta o diagrama de classes de análise do sistema.

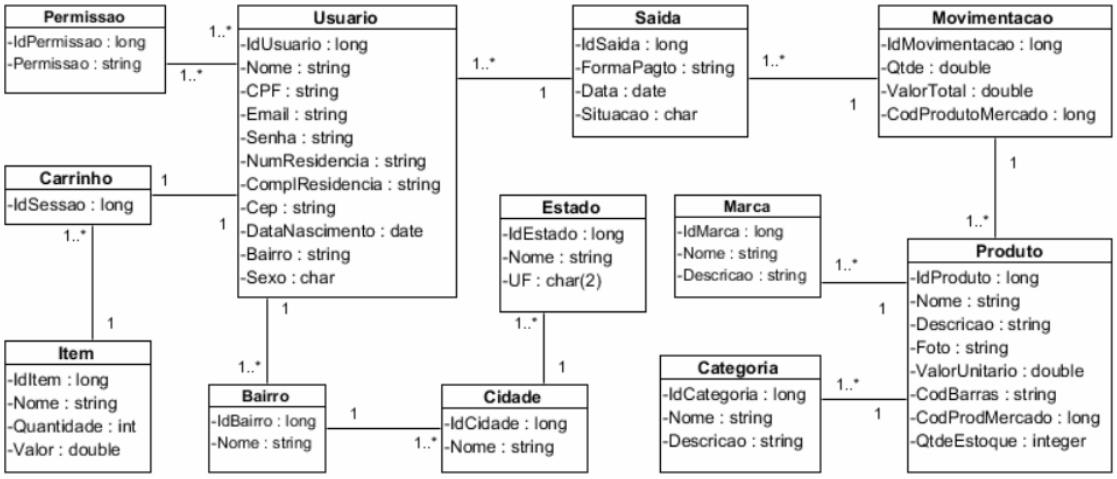

Figura 2 - Diagrama de classes de análise do sistema

As classes apresentadas no diagrama da Figura 2 estão documentadas a seguir.

O Quadro 16 apresenta a classe usuário. Essa classe tem diversas anotações do Java Persistence API (JPA), como a @Entity para indicar que a classe é uma entidade do banco de dados, a @Id para indicar que o campo id é o identificador da entidade, a @GeneratedValue que indica que o valor será gerado no banco de dados como sendo chave primária ou usado em conjunto com a anotação @Id e a @Column para definir alguns parâmetros das colunas como o nome, como, por exemplo, se ela pode receber valores nulos ou se o valor da coluna é único.

Identificação	Usuario
Descrição	Usuarios do sistema, utilizarão da mesma classe apenas com constructors diferentes, Cliente e Atendente.
Requisitos	RF10
Atributos	@Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idusuario; @Column(length=50, nullable = false) private String nome; @Column(name = "datanascimento", nullable = false) @Temporal(TemporalType.DATE) @DateTimeFormat(pattern = "dd/MM/yyyy") private Date dataNascimento; @Column(length=1, nullable=false) private String sexo; @Column(name = "username", length = 200, nullable = false) private String username; @Column(name = "password", length = 50) private String password; @Column(name = "bairro", length = 200, nullable = false) private String bairro; @Column(name = "rua", length = 200, nullable = false) private String rua; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "idbairro", referencedColumnName = " idbairro ") private Bairro bairro; @Column(name = "cpf", length = 15, nullable = false, unique = true) private String cpf; @Column(name = "numResidencia", length = 15, nullable = false) private String numResidencia; @Column(name = "complResidencia", length = 50)private String complResidencia; @Column(name = "cep", length = 10)private String cep;

Quadro 16 - Descrição da classe Usuario

No Quadro 17 está a apresentação da classe bairro.

Identificação	Bairro
Descrição	Bairro do cliente.
Requisitos	RF10
Atributos	@Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idbairro; @Column(name = "nome", length = 100, nullable = false)

	private String nome; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "idcidade", referencedColumnName="idcidade") private Cidade cidade;

Quadro 17 - Descrição da classe Bairro

No Quadro 18 está a apresentação da classe cidade.

Identificação	Cidade
Descrição	Cidade do cliente.
Requisitos	RF10
Atributos	@Entity @Table(name = "cidade") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idcidade; @Column(name = "nome", length = 100, nullable = false) private String nome; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name= "idestado", referencedColumnName="idestado") private Estado estado;

Quadro 18 - Descrição da classe Cidade

No Quadro 19 está a apresentação da classe estado.

Identificação	Estado
Descrição	Estado do cliente.
Requisitos	RF10
Atributos	@Entity @Table(name = "estado") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idestado; @Column(name = "nome", length = 50) private String nome; @Column(name = "uf", length = 2) private String uf;

Quadro 19 - Descrição da classe Estado

No Quadro 20 está a apresentação da classe item.

Identificação	Item
Descrição	Itens do carrinho de compras.
Requisitos	RF03
Atributos	Long id; String nome; Integer quantidade; Double valor; String formaPagamento; String foto;

Quadro 20 - Descrição da classe Item

No Quadro 21 está a apresentação da classe carrinho.

Identificação	Carrinho
Descrição	Carrinho de compras do sistema.
Requisitos	RF03
Atributos	List<Item> (Itens): Lista de itens no carrinho;
Métodos	void adicionar(Item item) void remover(Item item) List<Item> getItens() JSONArray getJsonItens()

Quadro 21 - Descrição da classe Carrinho

No Quadro 22 está a apresentação da classe saída.

Identificação	Saida
Descrição	Saida de estoque das compras que os clientes efetuam.
Requisitos	RF13
Atributos	@Entity @Table(name = "saida") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idsaida; @ManyToOne @JoinColumn(name = "idusuario", referencedColumnName = "idusuario") private Usuario usuario; @Column(name = "datasaida", nullable = false) @Temporal(TemporalType.DATE) private Date datasaida; @OneToMany(mappedBy = "saida", cascade = CascadeType.PERSIST, fetch = FetchType.EAGER) private List<Movimentacao> movimentacao; @Column(length=1, name = "formaPgto", nullable = false) private String formaPgto;

Quadro 22 - Descrição da classe Saida

No Quadro 23 está a apresentação da classe movimentação.

Identificação	Movimentacao
Descrição	Gerencia as movimentações de estoque.
Requisitos	RF07
Atributos	@Entity @Table(name = "movimentacao") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idMovimentacao; private Integer quantidade; private Double valor; private Long codProdutoMercado; @ManyToOne @JoinColumn(name = "idsaida", referencedColumnName = "idsaida") private Saida saida; @ManyToOne @JoinColumn(name = "idproduto", referencedColumnName =

	"idproduto") private Produto produto;

Quadro 23 - Descrição da classe Movimentacao

No Quadro 24 está a apresentação da classe produto.

Identificação	Produto
Descrição	Produtos disponíveis no supermercado.
Requisitos	RF01
Atributos	@Entity @Table(name = "produto") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) @Column(name = "idproduto") private Long idproduto; @Column(name = "nome", length = 100, nullable = false) private String nome; @Column(name = "descricao", length = 200) private String descricao; @Column(name = "foto", length = 100) private String foto; @Column(name="qtdeEstoque", nullable = false) private Integer qtdeEstoque; @Column(name = "valorUnitario", length = 30, nullable = false) private double valorUnitario; @Column(name = "codBarras", length = 30, nullable = false) private String codBarras; @Column(name = "codProdutoMercado", nullable = false) private Long codProdutoMercado; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "idcategoria", referencedColumnName = "idcategoria") private Categoria categoria; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "idmarca", referencedColumnName = "idmarca") private Marca marca;

Quadro 24 - Descrição da classe Produto

No Quadro 25 está a apresentação da classe categoria.

Identificação	Categoria
Descrição	Categoria dos produtos .Cada produto possui uma categoria.
Requisitos	RF01
Atributos	@Entity @Table(name = "categoria") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) @Column(name = "idcategoria") private Long idcategoria; @Column(name = "descricao", length = 50, nullable = false) private String descricao;

Quadro 25 - Descrição da classe Categoria

No Quadro 26 está a apresentação da classe marca.

Identificação	Marca
Descrição	Marca dos produtos Cada produto possui uma marca.
Requisitos	RF01
Atributos	@Entity @Table(name = "marca") @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private Long idmarca; @Column(name = "nome", length = 100, nullable = false) private String nome; @Column(name = "descricao", length = 200, nullable = false) private String descricao;

Quadro 26 - Descrição da classe Marca

A Figura 3 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação.

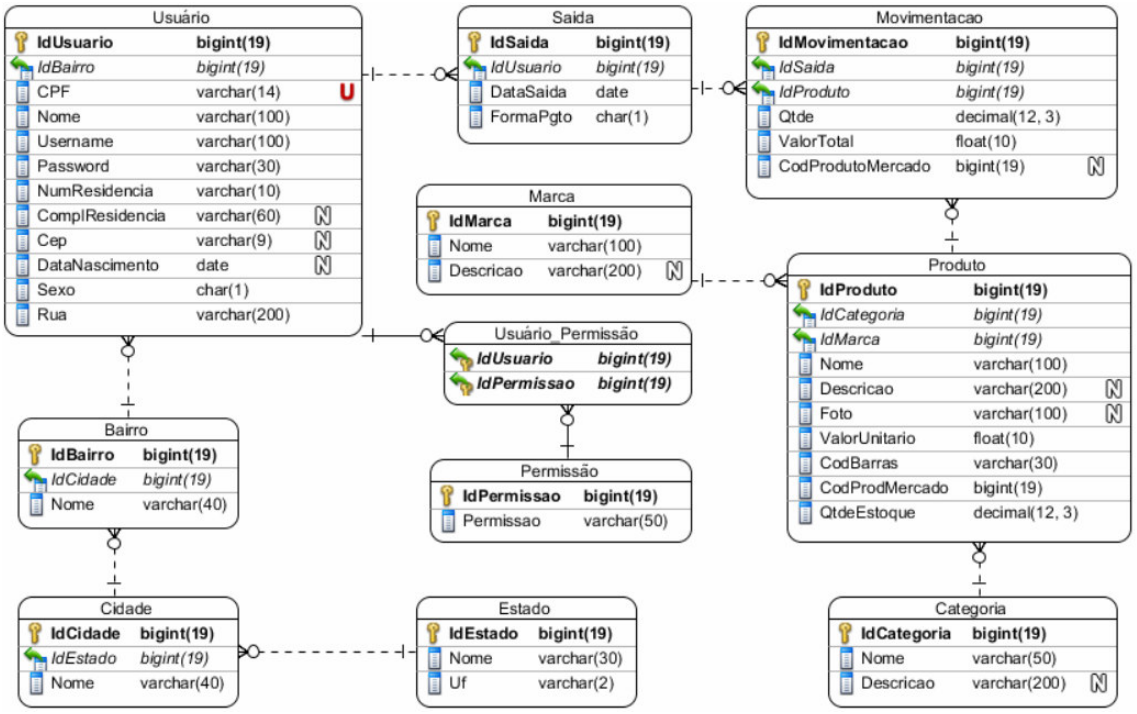

Figura 3 - Diagrama de entidade e relacionamentos

No Quadro 26 estão os campos da tabela usuário.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdUsuario	Integer	Sim	Não	Não	
IdBairro	Integer	Não	Sim	Não	Da tabela Bairro
CPF	Varchar	Não	Não	Não	
Nome	Varchar	Não	Não	Não	
Email	Varchar	Não	Não	Não	
Senha	Varchar	Não	Não	Não	

NumResidencia	Varchar	Não	Não	Não	
ComplResidencia	Varchar	Não	Não	Não	
Cep	Varchar	Não	Não	Não	
DataNascimento	Date	Não	Não	Não	
Bairro	Varchar	Não	Não	Não	
Sexo	Char	Não	Não	Não	

Quadro 27 - Campos da tabela Usuário

No Quadro 28 estão descritos os campos da tabela bairro.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdBairro	Integer	Sim	Não	Não	
IdCidade	Integer	Não	Sim	Não	Da tabela Cidade
Nome	Varchar	Não	Não	Não	

Quadro 28 - Campos da tabela Bairro

No Quadro 29 estão descritos os campos da tabela cidade.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdCidade	Integer	Sim	Não	Não	
IdEstado	Integer	Não	Sim	Não	Da tabela Estado
Nome	Varchar	Não	Não	Não	

Quadro 29 - Campos da tabela Cidade

No quadro 30 estão descritos os campos da tabela estado

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdEstado	Integer	Sim	Não	Não	
Nome	Varchar	Não	Não	Não	
Uf	Varchar	Não	Não	Não	

Quadro 30 - Campos da tabela Estado

No Quadro 31 estão descritos os campos da tabela saída.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdSaida	Integer	Sim	Não	Não	
IdUsuario	Integer	Não	Sim	Não	Da tabela Usuário
FormaPgto	Char	Não	Não	Não	
Data	Date	Não	Não	Não	

Quadro 31 - Campos da tabela Saida

No Quadro 32 estão descritos os campos da tabela movimentação.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdMovimentacao	Integer	Sim	Não	Não	
IdSaida	Integer	Não	Sim	Não	Da tabela Saida
IdProduto	Integer	Não	Sim	Não	Da tabela Produto
Qtde	Decimal	Não	Não	Não	
Valor Total	Float	Não	Não	Não	
CodProdutoMercado	Integer	Não	Não	Não	

Quadro 32 - Campos da tabela Movimentacao

No Quadro 33 estão descritos os campos da tabela produto.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdProduto	Integer	Sim	Não	Não	
IdCategoria	Integer	Não	Sim	Não	Da tabela Categoria
IdMarca	Integer	Não	Sim	Não	Da tabela Marca
Nome	Varchar	Não	Não	Não	
Descricao	Varchar	Não	Não	Não	
Foto	Varchar	Não	Não	Não	
ValorUnitario	Float	Não	Não	Não	
CodBarras	Varchar	Não	Não	Não	
CodProdMercado	Integer	Não	Não	Não	
QtdeEstoque	Integer	Não	Não	Não	

Quadro 33 - Campos da tabela Produto

No Quadro 34 estão descritos os campos da tabela categoria.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdCategoria	Integer	Sim	Não	Não	
Nome	Varchar	Não	Não	Não	
Descricao	Varchar	Não	Não	Sim	

Quadro 34 - Campos da tabela Categoria

No Quadro 35 estão descritos os campos da tabela marca.

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdMarca	Integer	Sim	Não	Não	
Nome	Varchar	Não	Não	Não	
Descricao	Varchar	Não	Não	Sim	

Quadro 35 - Campos da tabela Marca

No Quadro 36 estão descritos os campos da tabela permissao

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdPermissao	Integer	Sim	Não	Não	
Permissão	Varchar	Não	Não	Não	

Quadro 36 - Campos da tabela permissão

No Quadro 37 estão descritos os campos da tabela usuario_permissao

Campo	Tipo	Chave Primária	Chave Estrangeira	Nulo	Observações
IdUsuario	Integer	Sim	Não	Não	Da tabela Usuario
IdPermissao	Varchar	Não	Não	Não	Da tabela Permissao

Quadro 37 - Campos da tabela usuario_permissão

4.3 APRESENTAÇÃO DO SISTEMA

A tela inicial do sistema é a de produtos (Figura 4). Por meio dessa tela os usuários podem adicionar itens ao carrinho, escolhendo a quantidade de cada item e buscar produtos pela barra de pesquisa.

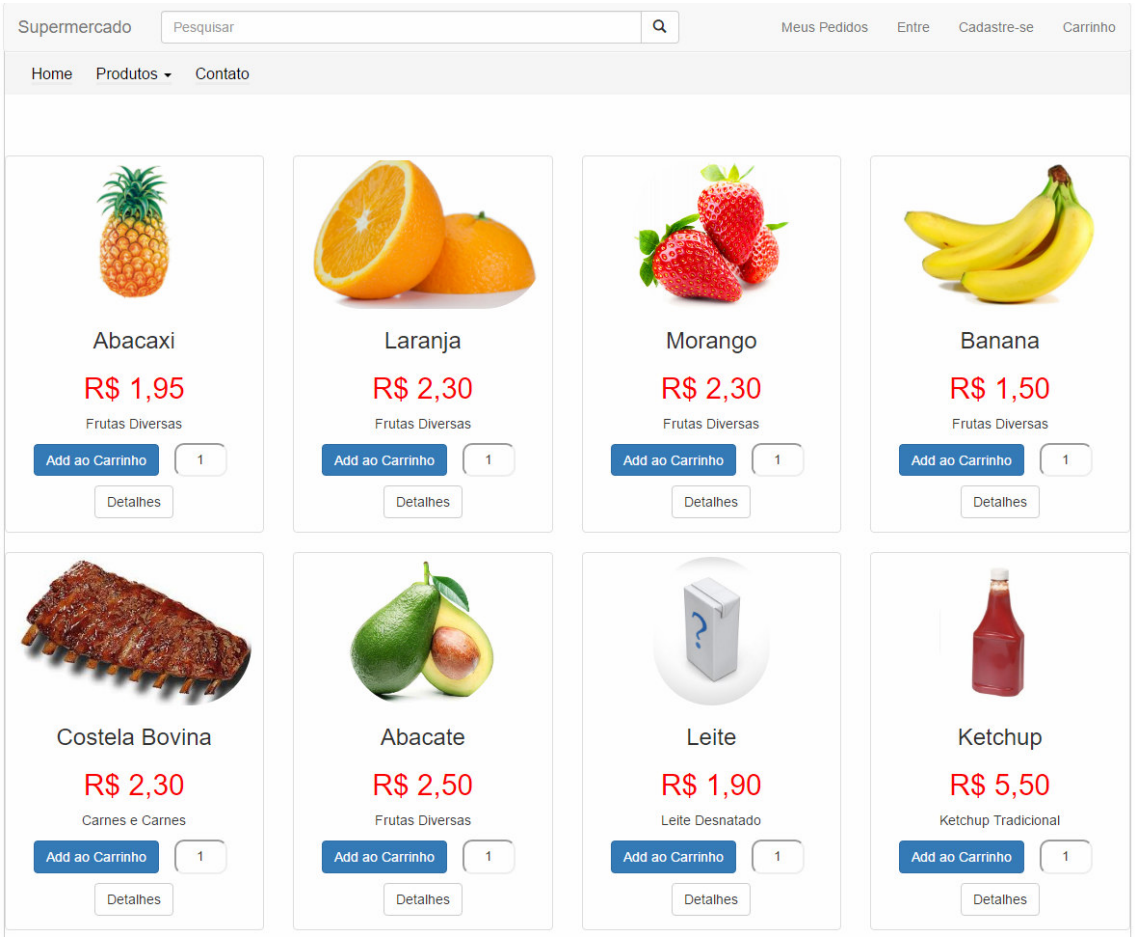

Figura 4 - Tela inicial

Ao adicionar os produtos no carrinho de compras, uma mensagem é apresentada informado o usuário que o produto foi adicionado. A área destacada na Figura 5 apresenta esta mensagem.

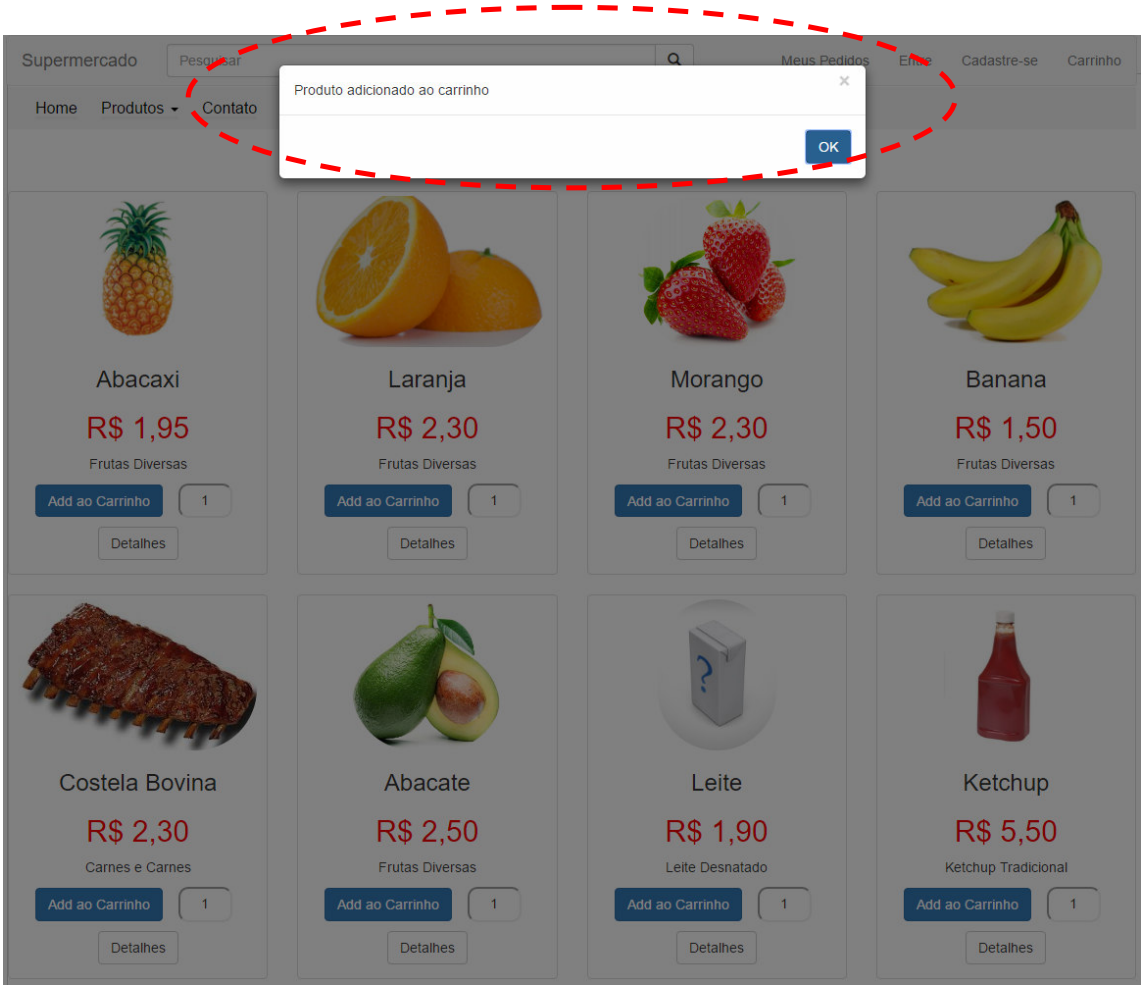

Figura 5 - Adicionando item ao carrinho

É possível visualizar os itens do carrinho de compras clicando no *link* “Carrinho” no canto superior direito da tela inicial e será redirecionada para a página apresentada na Figura 6.

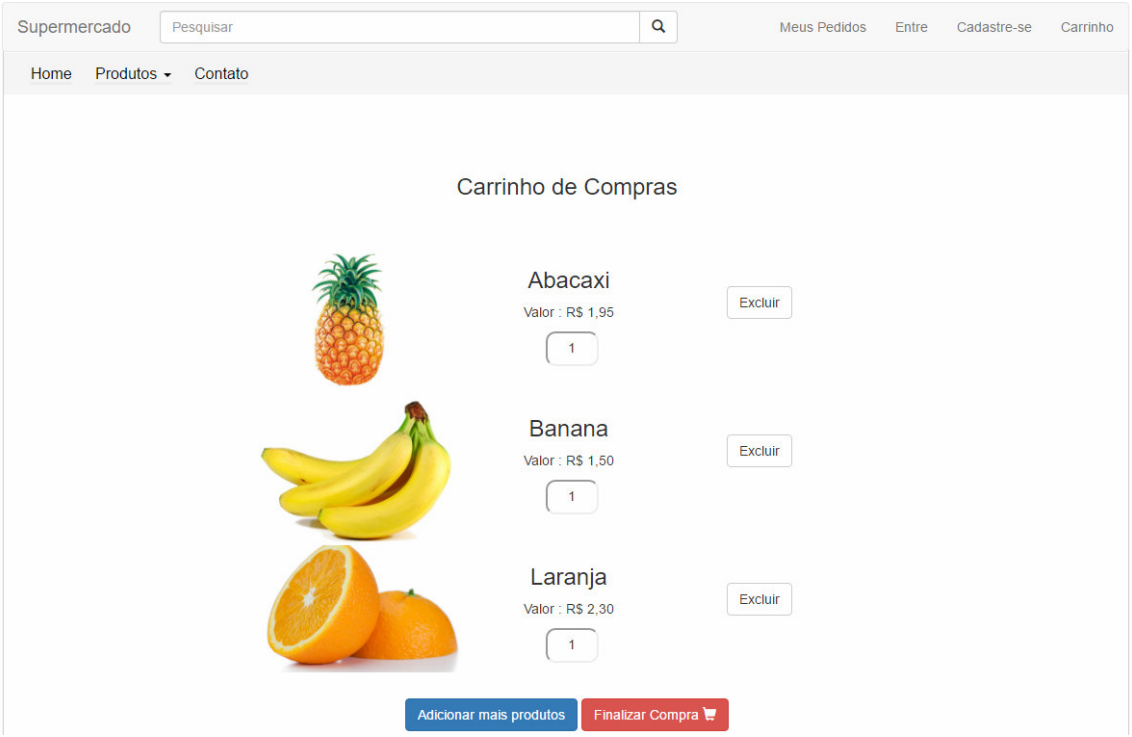

Figura 6 - Carrinho de compras

No carrinho de compras é possível excluir itens e, também, alterar a quantidade de cada item adicionado. Ao clicar no botão excluir uma mensagem para confirmar a exclusão é exibida ao usuário.

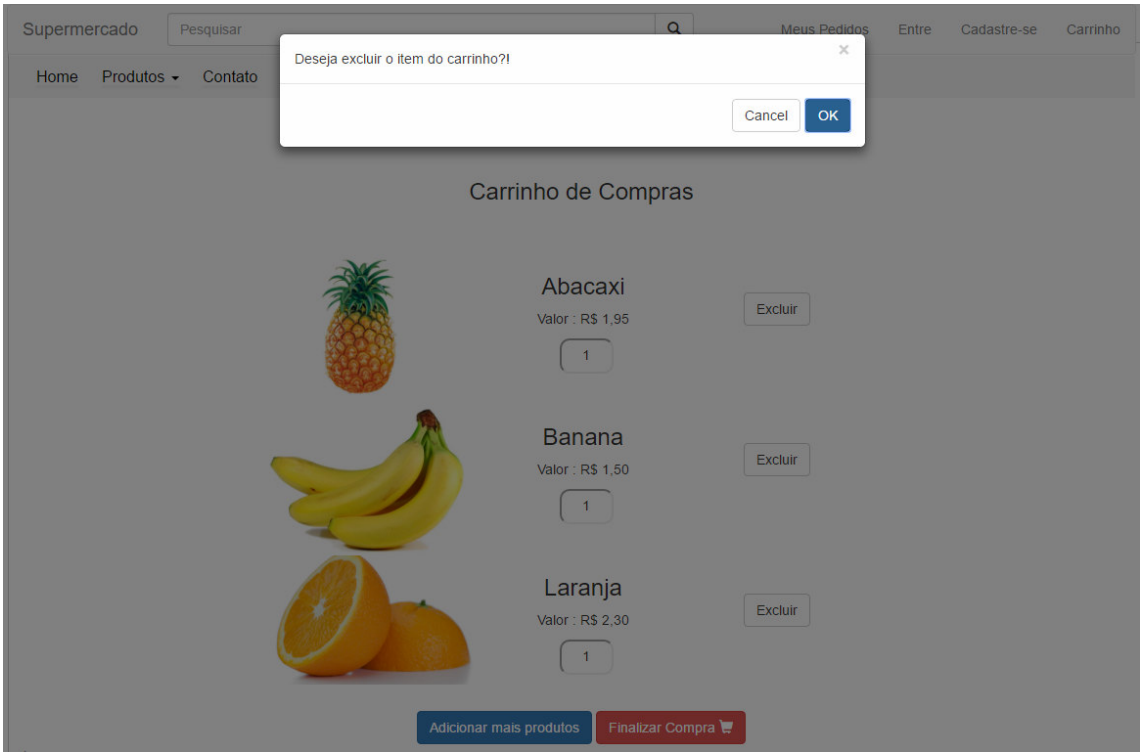

Figura 7 - Confirmar exclusão de item no carrinho

Ao confirmar a exclusão do item é apresenta uma mensagem (Figura 8) para que o usuário seja informado da ação que realizou.

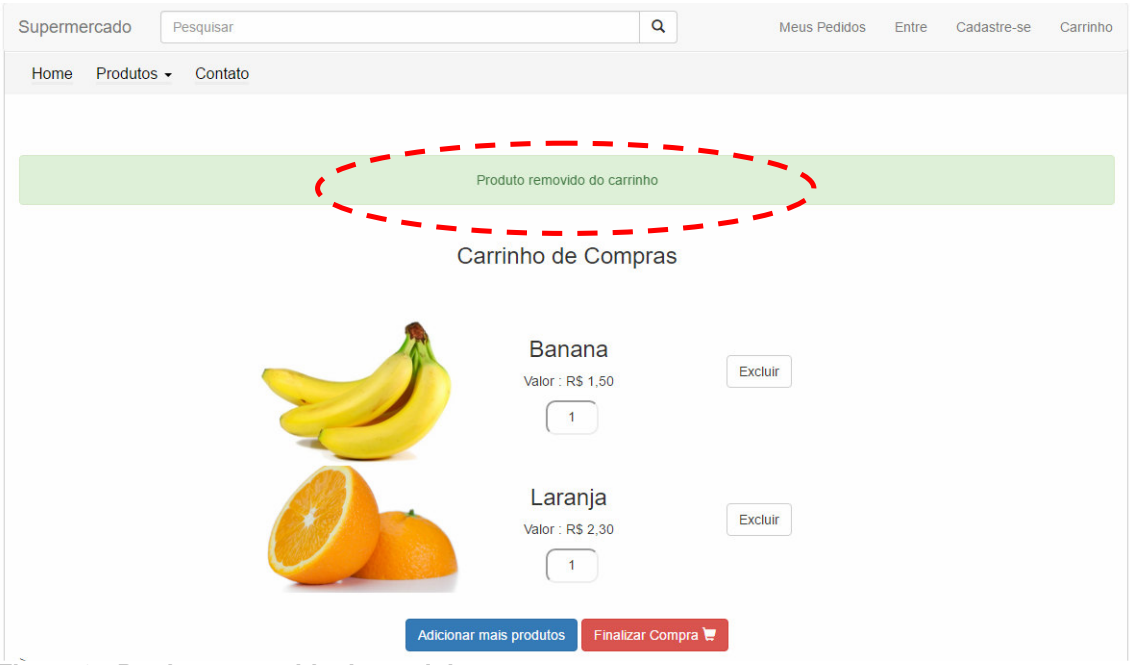

Figura 8 - Produto removido do carrinho

O botão “Adicionar mais produtos” retorna para a tela inicial e o botão “Finalizar Compra”, redireciona o usuário para a tela de *checkout*. Caso o usuário não esteja logado ele é redirecionado para a tela de *login* como mostra a Figura 9.

Figura 9 - Tela de login

Ao tentar efetuar o acesso, é realizada uma validação do usuário e da senha informados. Caso essas informações sejam inválidas, o usuário é informado por

meio de uma mensagem apresentada na tela, como mostra a Figura 10. O *link* “Cadastre-se aqui!” Redireciona para a tela de cadastro de usuários.

Figura 10 - Login inválido

Ao efetuar o acesso com sucesso (*login* e senha válidos), o usuário é encaminhado para a tela de *checkout*. Nessa tela é possível selecionar a forma de pagamento cartão de crédito ou dinheiro, como mostra a Figura 11.

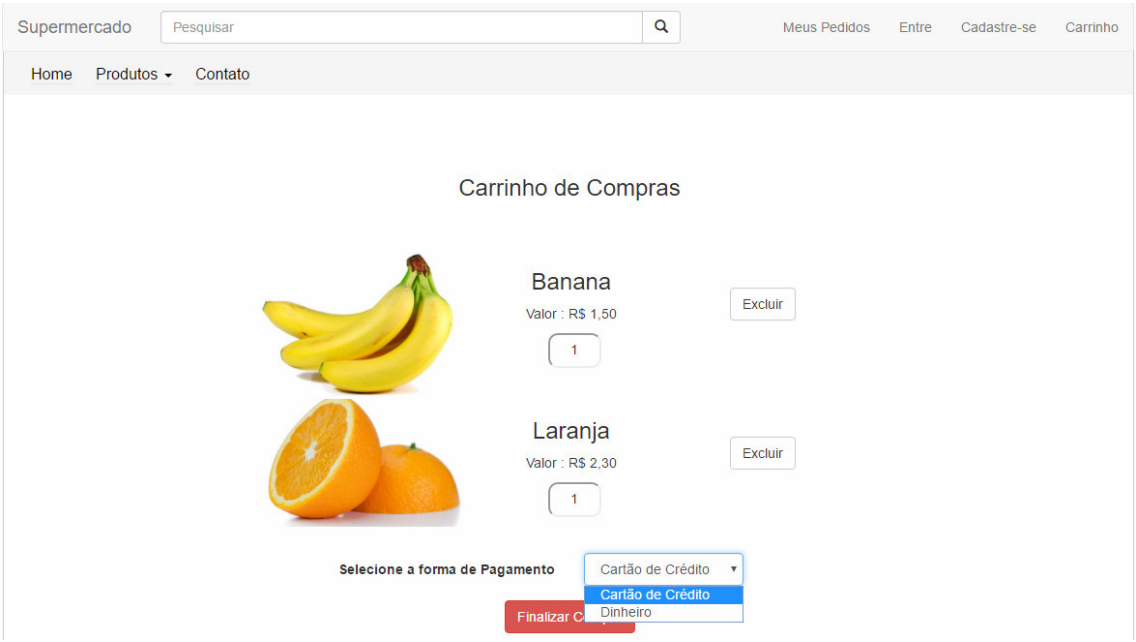

Figura 11 - Finalizar Compra

Clicando no botão “Finalizar Compra” um aviso é emitido para o usuário informando-o que o pedido foi efetuado com sucesso, como mostra a Figura 12.

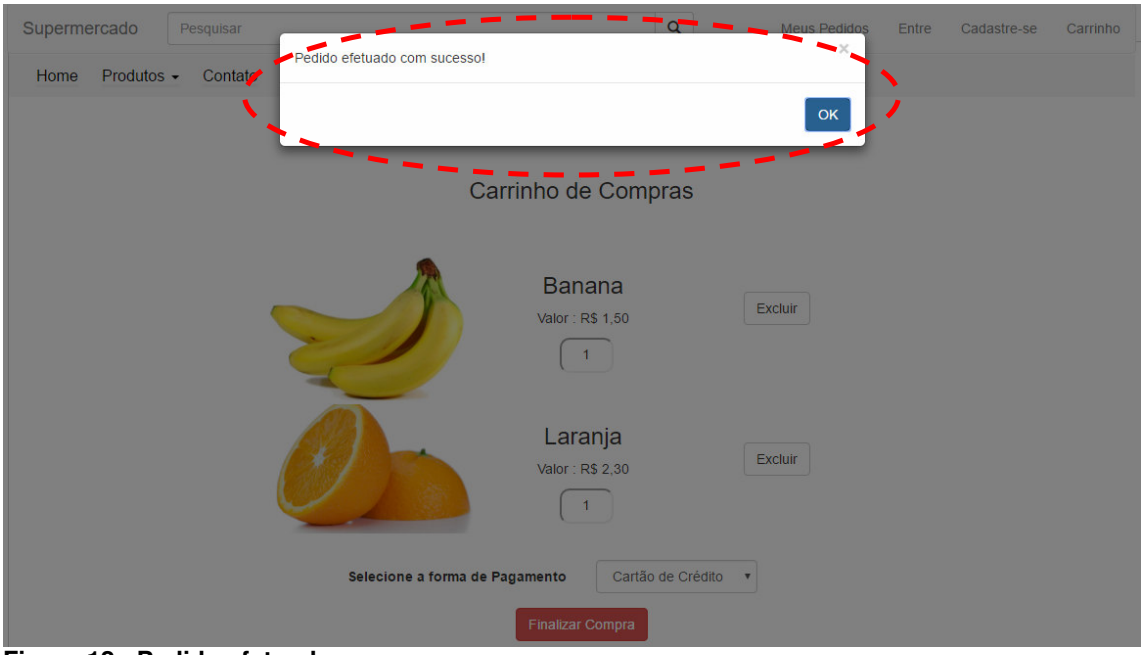

Figura 12 - Pedido efetuado com sucesso

Após o pedido ser efetuado com sucesso, o usuário é redirecionado para página de pedidos, como mostra a Figura 13, e um *email* é enviado informado-o do pedido.

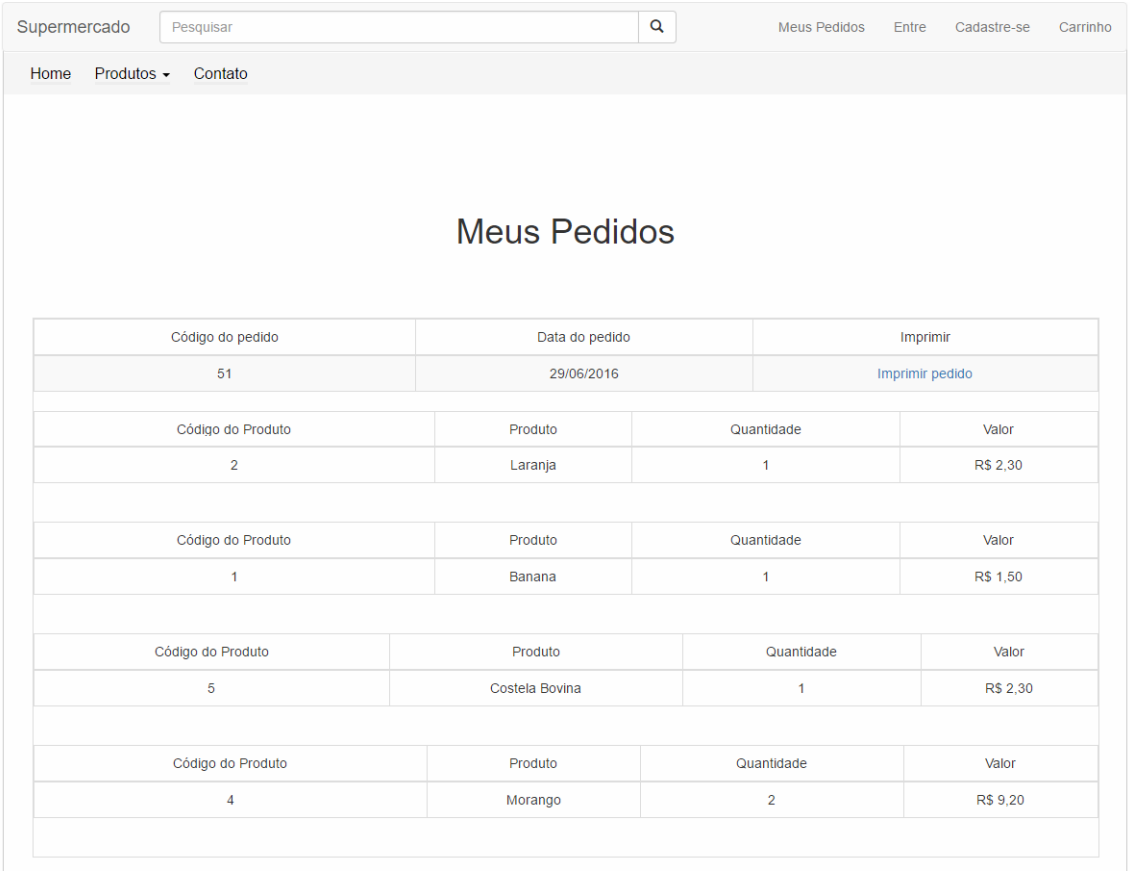

Figura 13 - Tela de pedidos

Ao clicar no link “Cadastre-se”, como mostra a Figura 14. o usuário é redirecionado para tela de Cadastro, como mostra a Figura 15.

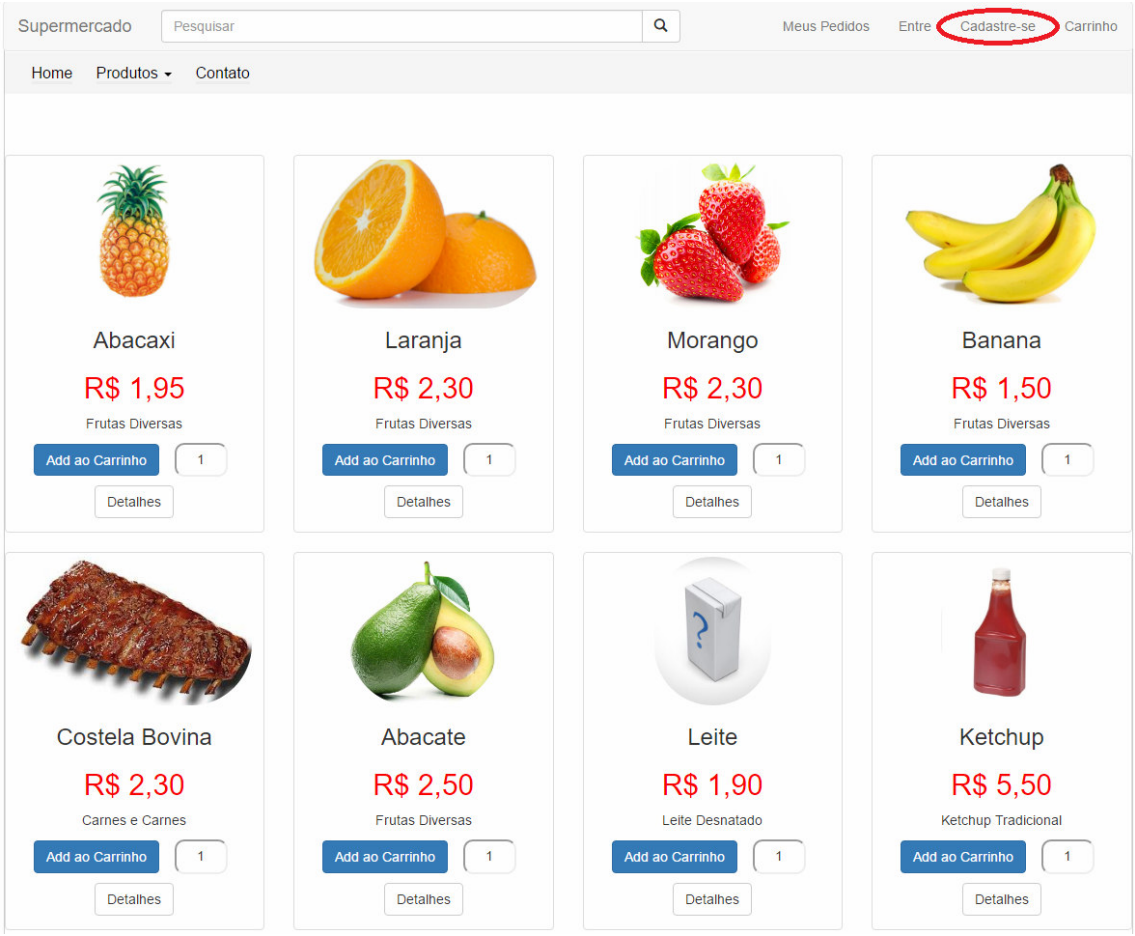

Figura 14 - Link Cadastre-se

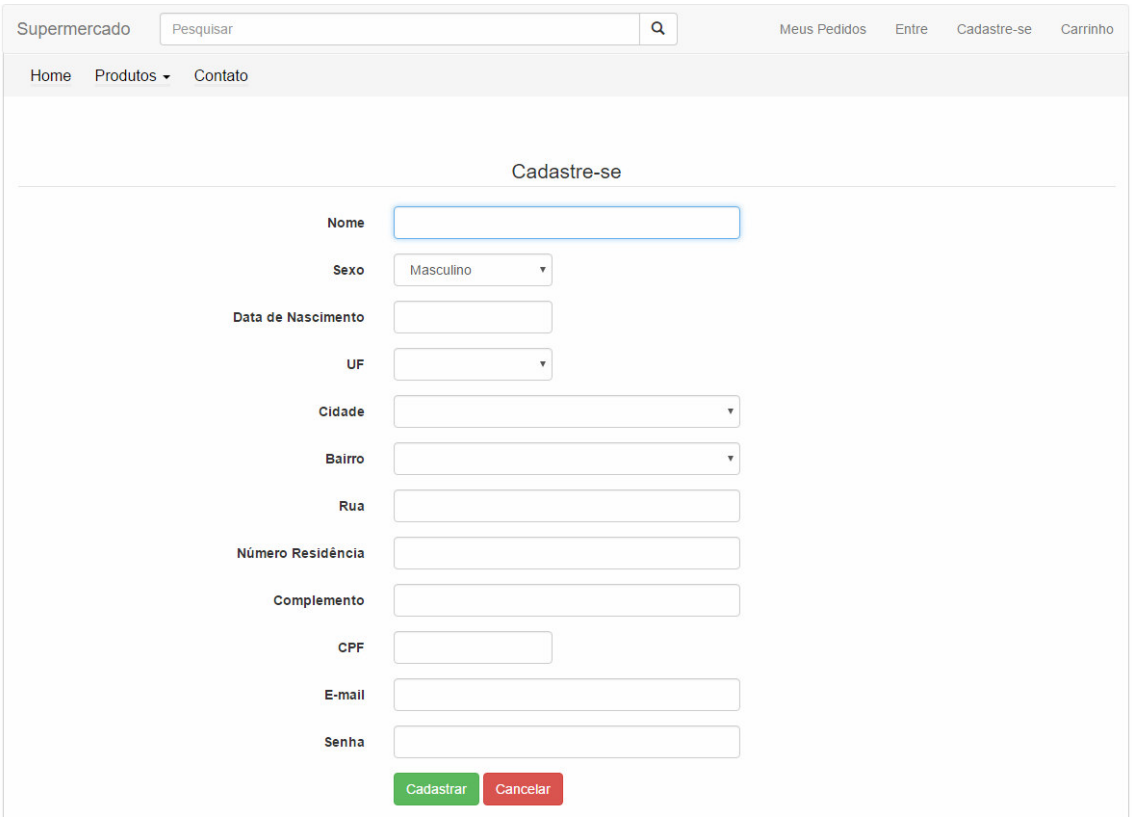

Figura 15 - Tela de cadastro de usuário

Se o usuário clicar no botão “Cadastrar” sem informar os dados, o sistema realiza uma validação para os campos obrigatórios e uma mensagem é exibida ao usuário, como é apresentado na região destacada da Figura 16.

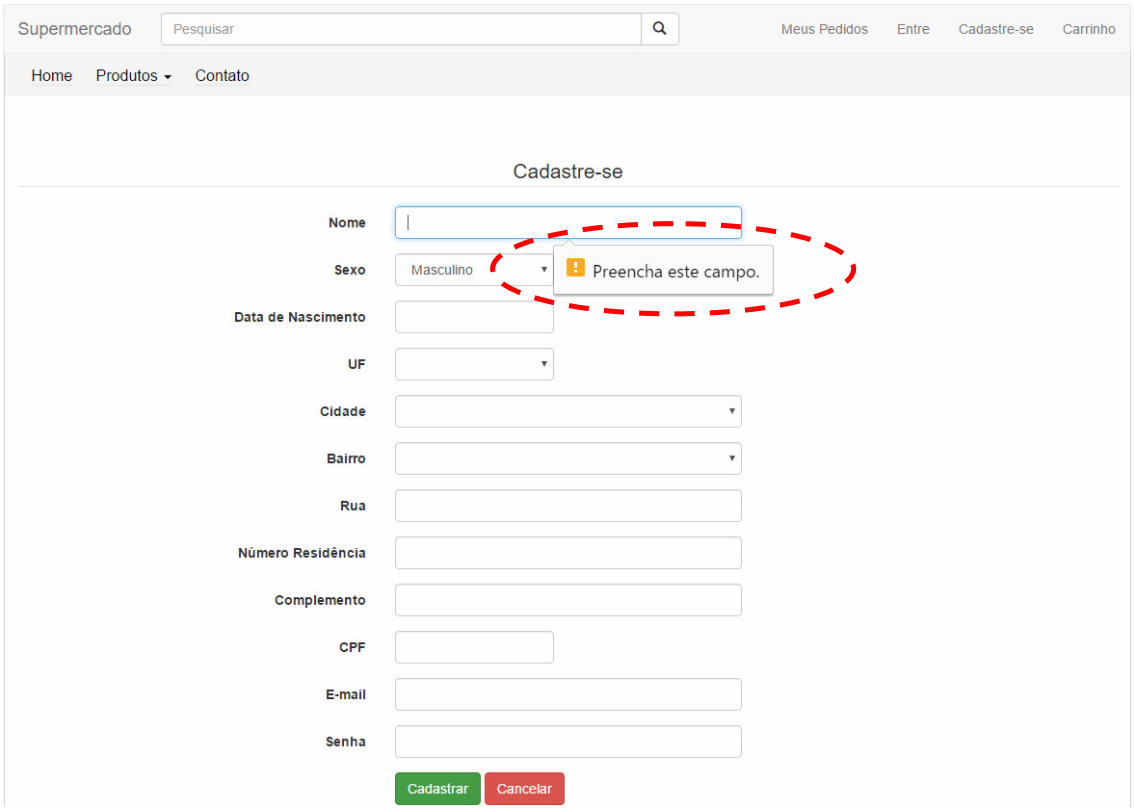

Figura 16 - Validação de campos obrigatórios

Essa validação ocorre para todos os campos que são obrigatórios no formulário. Se o usuário informar dados que são únicos no sistema, como CPF e *email*, o sistema realiza uma validação informando que o CPF ou email já foram cadastrados. E se dados incorretos, não válidos para a forma de validação utilizada para CPF e *email*, por exemplo, uma mensagem é apresentada na tela, conforme ilustra a região circulada da Figura 17.

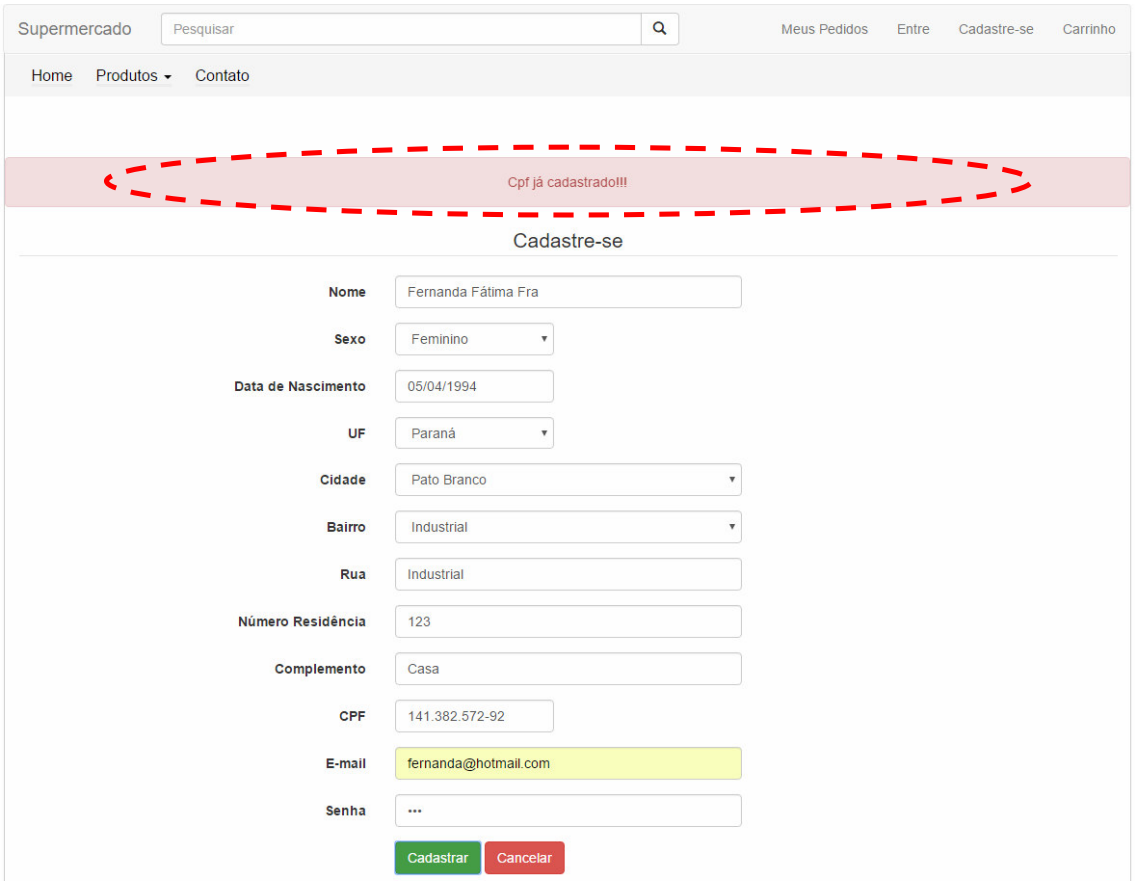

Figura 17 - Validação de campos únicos

Ao efetuar o cadastro com sucesso (dados válidos), uma mensagem é exibida ao usuário, como a apresentada na região destacada da Figura 18.

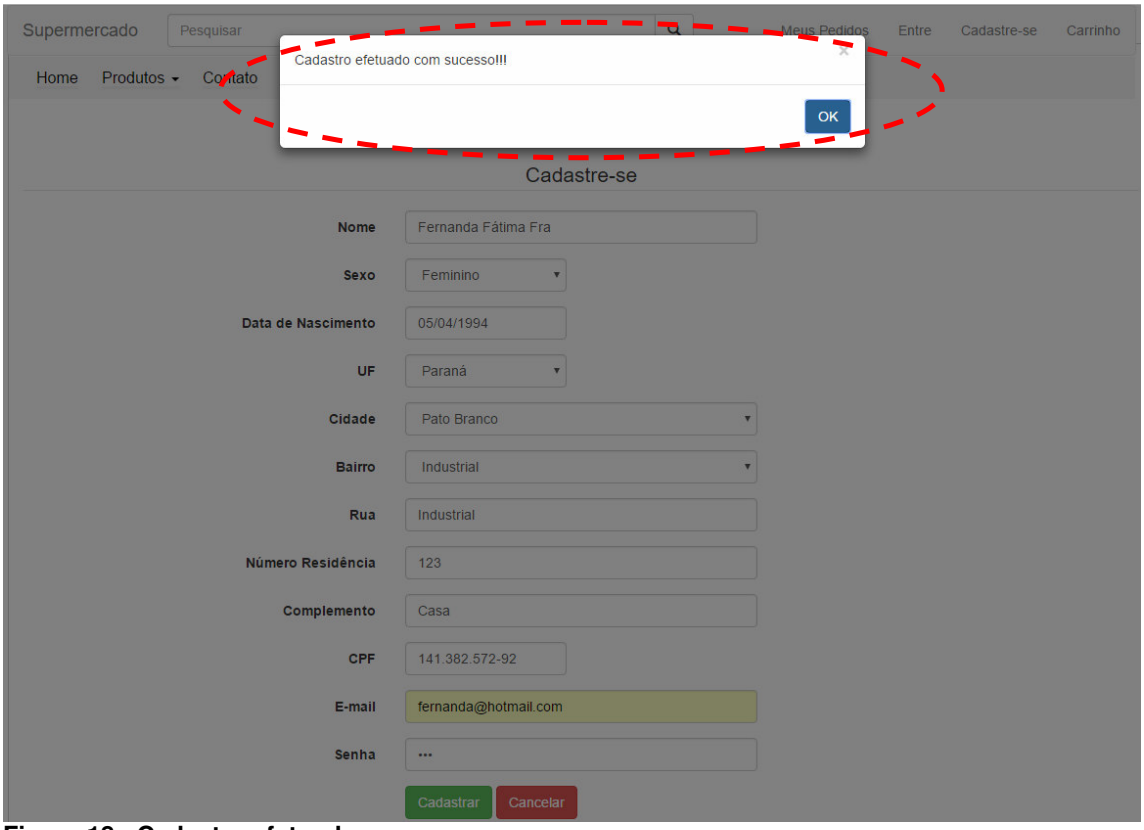

Figura 18 - Cadastro efetuado com sucesso

Em seguida, o usuário é redirecionado para tela de perfil (Figura 19). Se o usuário não estiver logado no sistema, ele é redirecionado novamente para tela de *login*.

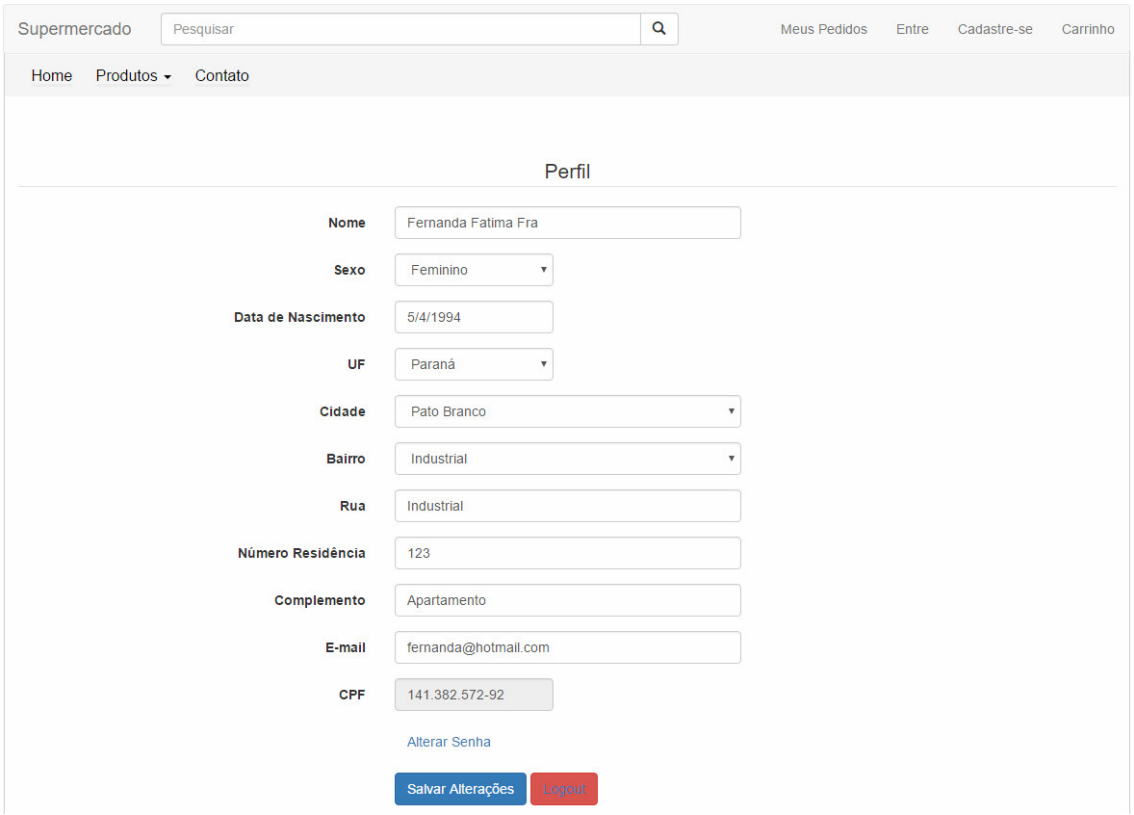

Figura 19 - Perfil do usuário

Na tela de perfil (Figura 20), o usuário pode editar a maioria dos campos com exceção do campo de CPF pois é um campo único e é validado na hora do cadastro assim o cliente não consegue digitá-lo incorretamente. Ao clicar no botão “Salvar” as alterações são realizadas e uma mensagem é exibida ao usuário indicando que os dados foram alterados com sucesso. Ao clicar no botão *logout,* o sistema é finalizado.

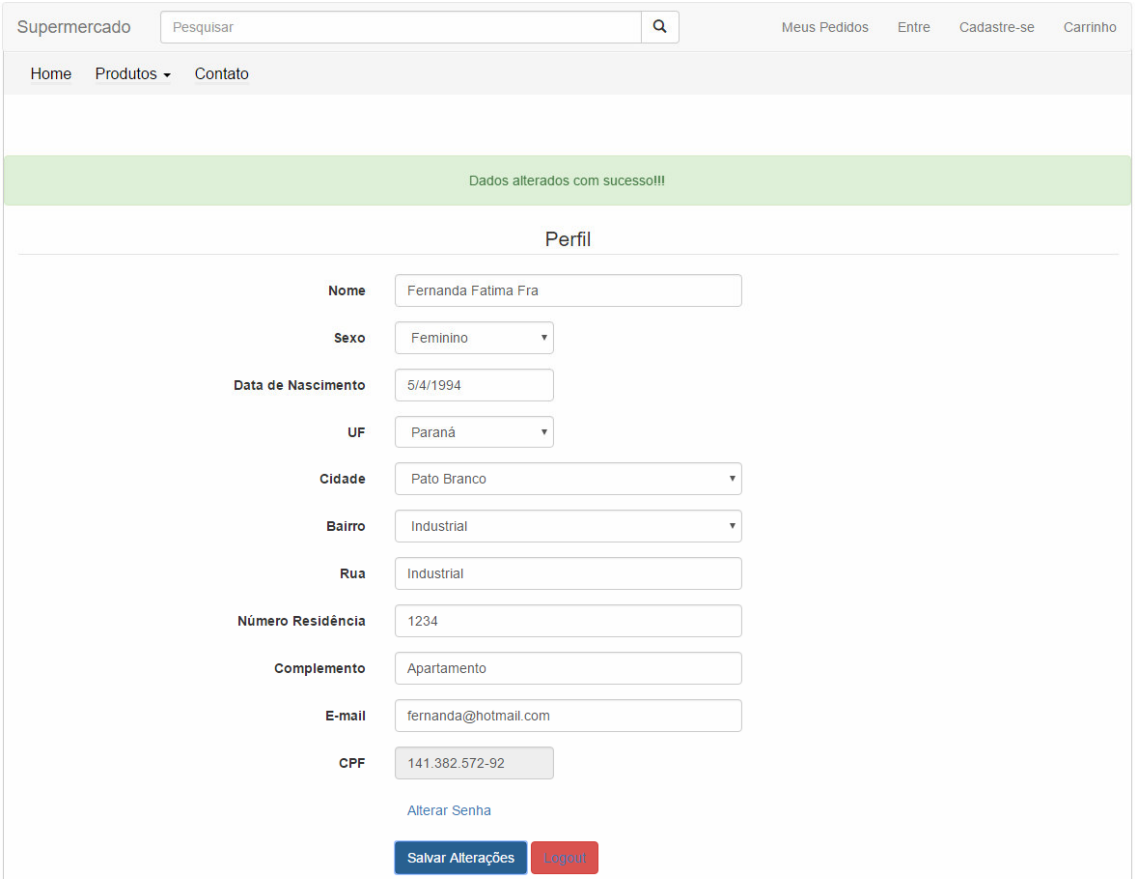

Figura 20 - Dados alterados

Ao clicar no *link* “Alterar senha” o usuário é redirecionado para a tela da Figura 21.

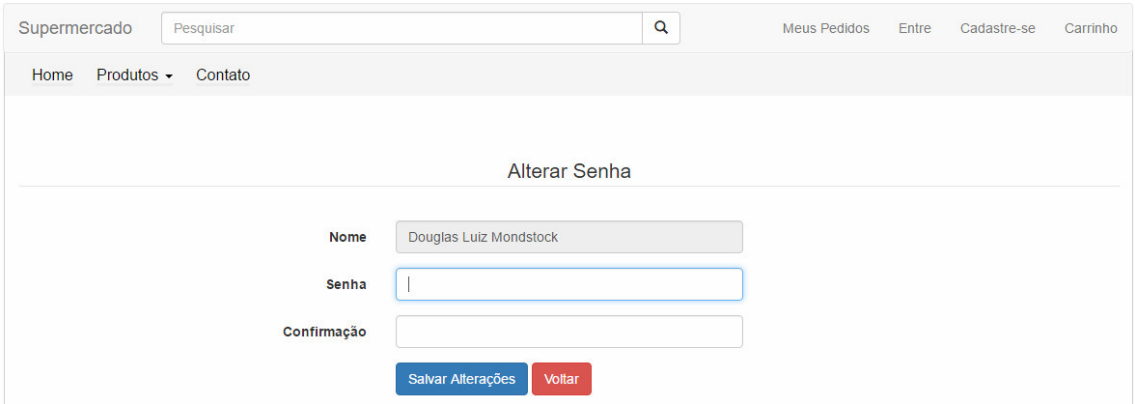

Figura 21 - Alterar senha

O sistema permite que a senha seja alterada com sucesso somente se a confirmação e senha forem compatíveis (iguais). Nesse caso, uma mensagem é exibida ao usuário informando-o de que a senha foi alterada com sucesso, conforme ilustrado na Figura 22 e uma confirmação de *login* é requisitada novamente.

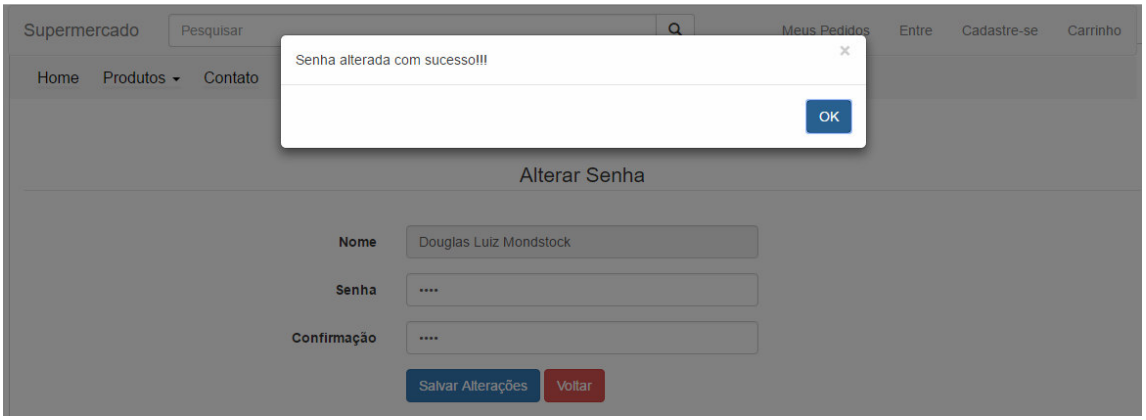

Figura 22 - Senha alterada com sucesso

As funcionalidades relacionadas à administração do sistema são acessadas somente por meio do *link* /admin/ digitado diretamente no navegador. Essa implementação foi realizada dessa forma pela segurança, pois não mostra o caminho para os clientes e também é necessário o *login* de usuário ou de administrador para acessar a tela inicial.

Figura 23 - Tela inicial administrativo

Ao clicar no *link* “Cadastros/Usuários” o usuário ou administrador é redirecionado para tela de Cadastros de usuário como mostra a Figura 24.

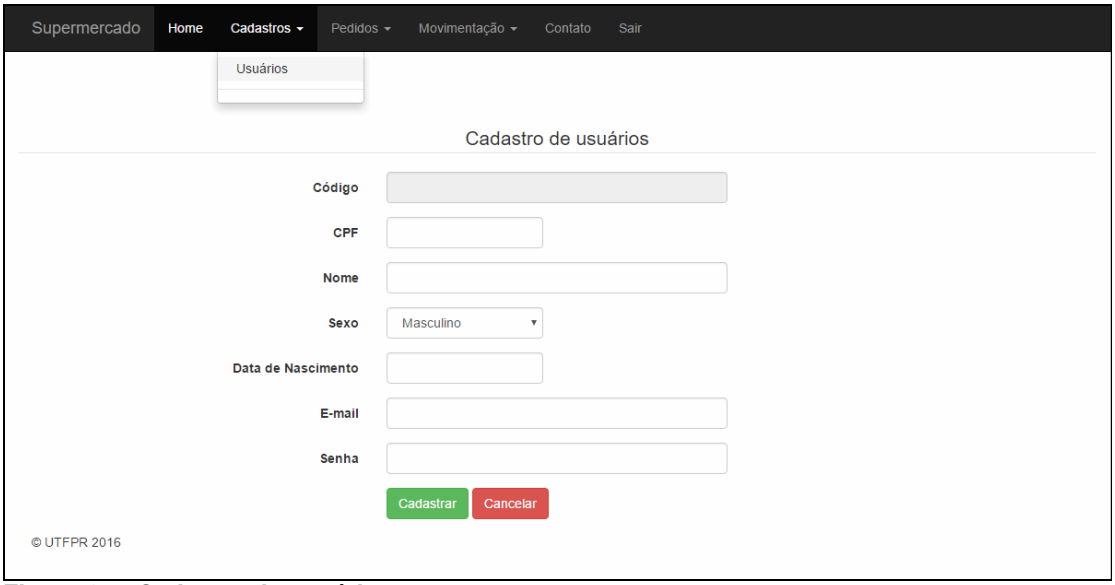

Figura 24 - Cadastro de usuários

As validações do cadastro de usuários são as mesmas realizadas do cadastro de clientes. Ao efetuar o cadastro com sucesso (dados válidos), uma mensagem é exibida ao usuário, como mostra a Figura 25.

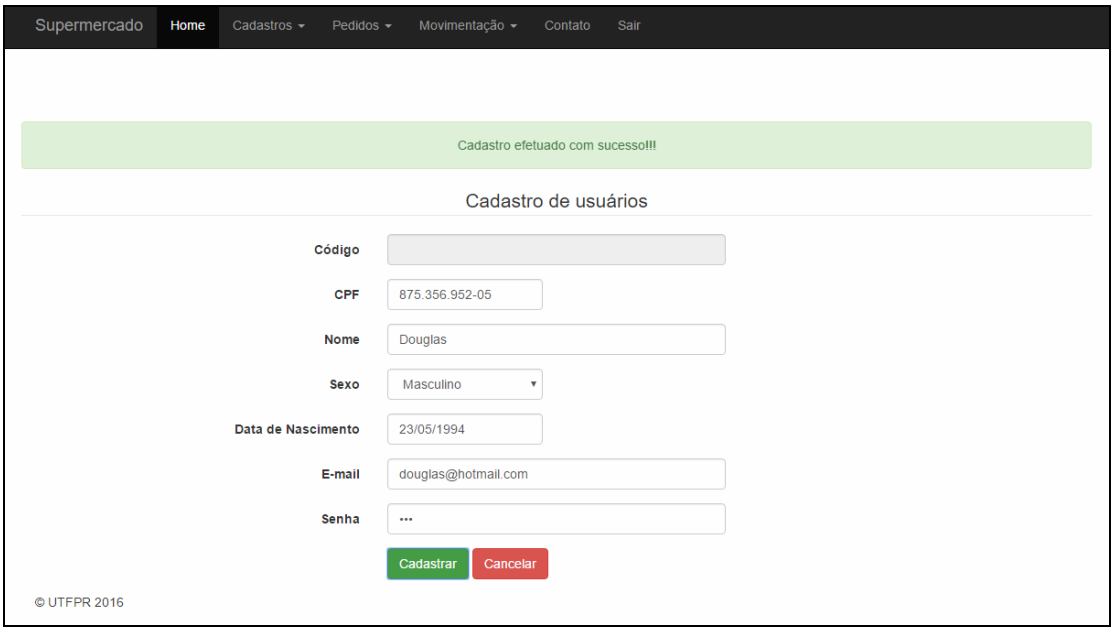

Figura 25 - Cadastro efetuado com sucesso

Clicando no *link* “Pedidos/Relatório” são listados os pedidos, conforme apresenta a Figura 26.

Figura 26 - Relatório de pedidos

Clicando no botão “Imprimir pedido” a janela de impressão é aberta e é possível imprimir o pedido (Figura 27).

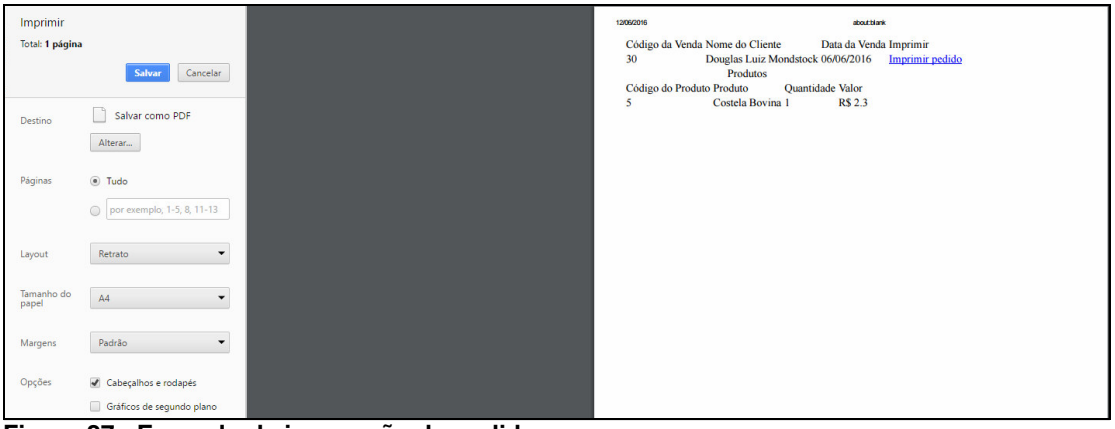

Figura 27 - Exemplo de impressão do pedido

Clicando no *link* “Movimentação/Exportar Movimentação” o usuário é redirecionado para a tela da Figura 28.

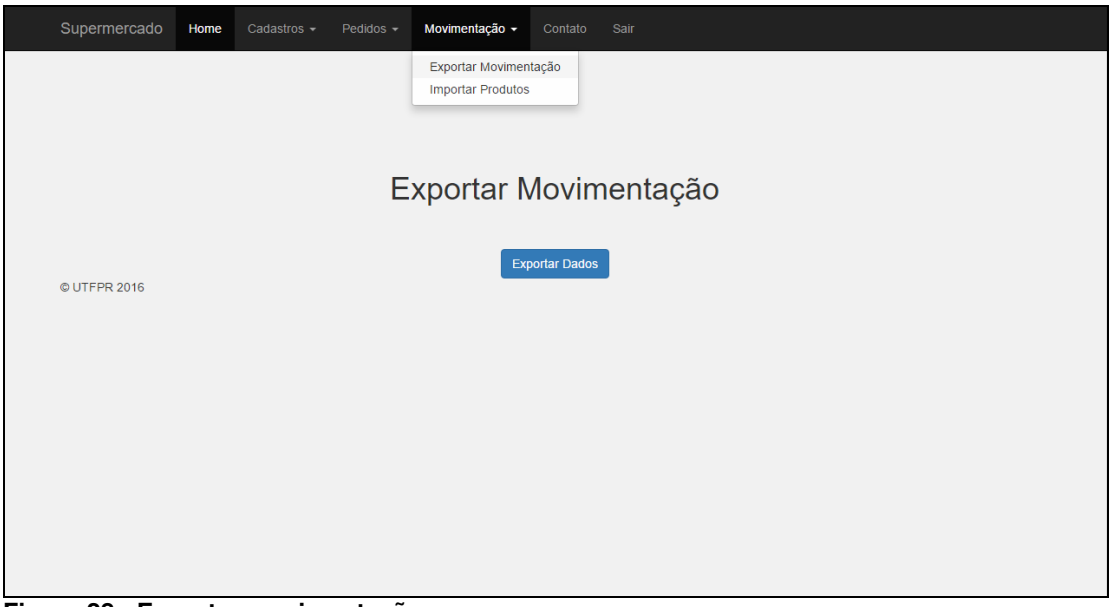

Figura 28 - Exportar movimentação

Ao clicar no botão “Exportar Dados” os dados de movimentação são exportados para um XML no caminho supermercado\src\main\webapp\WEB-INF\xml (Figura 29).

Figura 29 - Movimentação exportada com sucesso

Clicando no *link* “Movimentação/Importar produtos” o usuário é redirecionado para a tela da Figura 30.

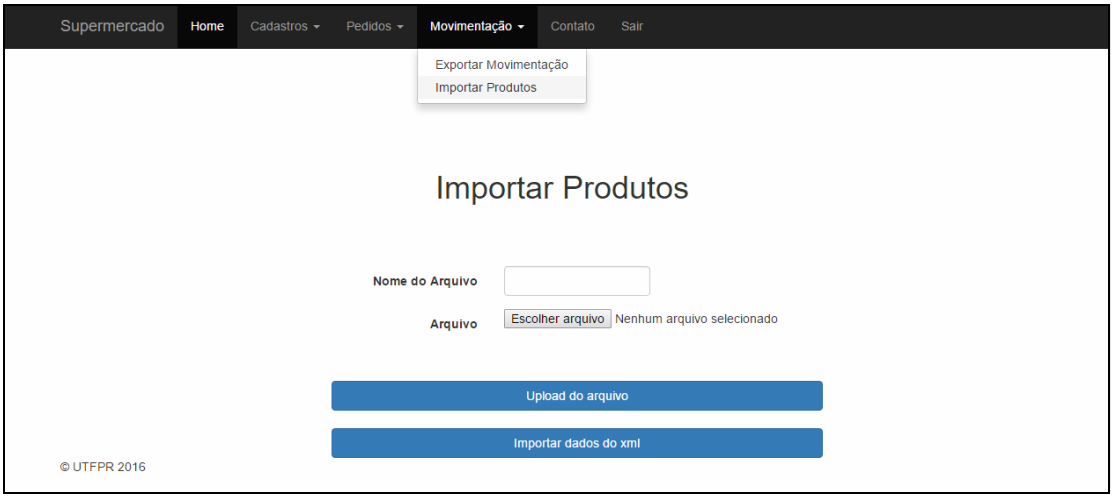

Figura 30 - Importar produtos

O usuário informa o nome do arquivo e escolhe o arquivo clicando no botão “Upload do arquivo”. Com essa ação é realizado o *upload* do arquivo para o sistema (Figura 31).

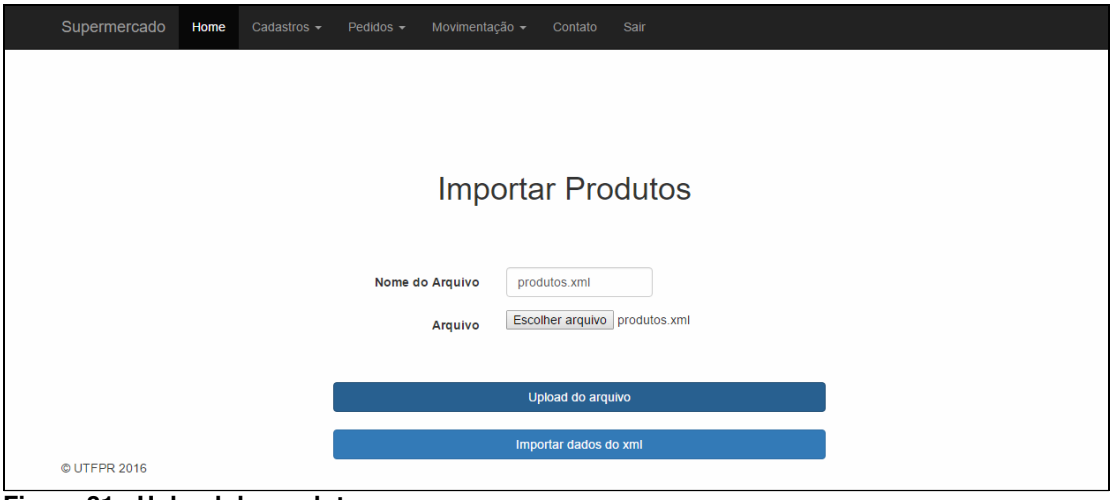

Figura 31 - Upload de produtos

Clicando no botão “Importar dados do xml”, os dados do arquivo salvo são importados para o banco do sistema (Figura 32).

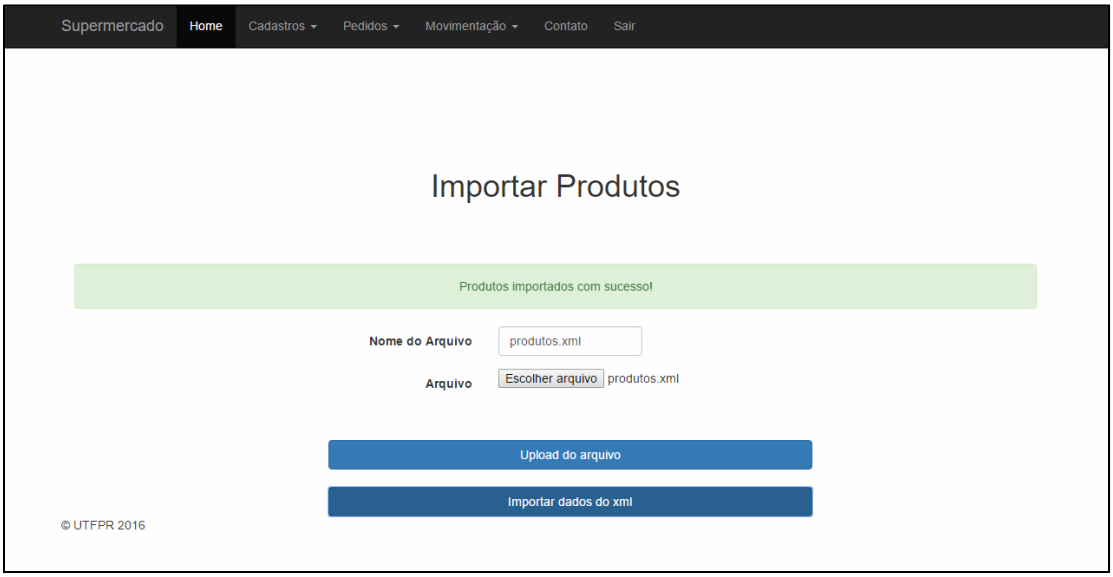

Figura 32 - Produtos importados com sucesso

4.4 IMPLEMENTAÇÃO DO SISTEMA

A classe MainServer é responsável por executar a aplicação e com a segurança do WebSecurityconfig uma das funcionalidades do *framework* Spring Security. A classe MainServer é apresentada na Listagem 1.

```
@Configurable
@EnableAutoConfiguration
@ComponentScan
public class MainServer extends SpringBootServletInitializer{

	public static void main(String[] args) {
		SpringApplication.run(MainServer.class, args);
	}

	@Bean
	public WebSecurityConfigurerAdapter webSecurityConfigurerAdapter(){
		return new WebSecurityConfig();
	}

}
```

Listagem 1 - MainServer

A Classe WebSecurityConfig (Listagem 2) é responsável pela segurança do sistema e nela é possível indicar qual usuário poderá acessar determinada pasta ou arquivo do sistema.

```
@Override
	protected void configure(HttpSecurity http) throws Exception {
		// desabilitar configuração externa e caso acesso negado ir para
"erro403"

	http.csrf().disable().exceptionHandling().accessDeniedPage("/erro403")
		.and()
		.formLogin().loginPage("/login").defaultSuccessUrl("/index/")
		.failureUrl("/login?error=bad_credentials").permitAll()
		.and()
		.authorizeRequests()
			.antMatchers("/cadastro/**").permitAll()
			.antMatchers("/index/**").permitAll()
			.antMatchers("/produtoDetalhes/**").permitAll()
			.antMatchers("/usuario/**").hasRole("ADMIN")
			.antMatchers("/checkout/**").hasRole("CLIENTE")
			.antMatchers("/categoria/**").hasRole("ADMIN")
			.antMatchers("/produto/**").hasRole("ADMIN")
			.antMatchers("/admin/**").hasAnyRole(("ADMIN"), ("USUARIO"))
			.antMatchers("/exportar/**").hasAnyRole(("ADMIN"), ("USUARIO"))
			.antMatchers("/importar/**").hasAnyRole(("ADMIN"), ("USUARIO"))
			.antMatchers("/cliente/**").hasRole("CLIENTE")
			.antMatchers("/pedido/**").hasRole("CLIENTE")
			.antMatchers("/listaproduto/**").permitAll()
			.antMatchers("/**").permitAll();//hasRole("USER");
	}
```

Listagem 2 - Classe WebSecurityConfig

Para cada usuário é cadastrada uma *role* no banco de dados que vai determinar quais permissões ele possui, definindo, assim, as funcionalidades do sistema que ele pode acessar.

O leiaute do sistema é feito por duas páginas de *template*. Um *template* com o

leiaute das páginas que envolvem o usuário cliente e o outro para as páginas administrativas. Na Listagem 3, a nave em “div class="nave navbar navbar-default nav-position" é um .class que foi cirado para alteração em cores.

```
<%@ tag pageEncoding="UTF-8"%>

<%@ attribute name="cssEspecificos" fragment="true" required="false"%>
<%@ attribute name="scriptsEspecificos" fragment="true" required="false"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
<%@ taglib prefix="layout" tagdir="/WEB-INF/tags/layoutclient"%>

<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="utf-8" />
<meta content="text/html;charset=utf-8" http-equiv="Content-Type">
<title>Supermercado</title>
<meta name="description" content="" />
<meta name="viewport"
      content="width=device-width, initial-scale=1.0, maximum-scale=1.0" />

<!-- Bootstrap -->
<link href="<c:url value="/static/css/bootstrap.min.css"/>"
      rel="stylesheet" />
<link href="<c:url value="/static/css/jquery-ui.min.css"/>"
      rel="stylesheet" />
      <link href="<c:url value="/static/css/geral.css"/>"
      rel="stylesheet" />

<jsp:invoke fragment="cssEspecificos"></jsp:invoke>

</head>
<body class="bd container">
      <div class="nave navbar navbar-default nav-position">
            <div class="navbar-header">
           <button     type="button"     data-target=".navbar-collapse"     data-
toggle="collapse" class="navbar-toggle">
               <span class="sr-only">Toggle navigation</span>
               <span class="icon-bar"></span>
               <span class="icon-bar"></span>
               <span class="icon-bar"></span>
           </button>
           <a class="home navbar-brand homelink" href="/index/">Supermercado</a>
       </div>
       <div class="collapse navbar-collapse">
                  <form   action="/listaproduto/b"   method="GET"   id="frmBusca"
name="frmBusca" class="navbar-form navbar-left" role="search">
                        <div class="input-group "style="width:255%;">
                              <input      type="text"      class="form-control"
placeholder="Pesquisar" name="buscaProduto">
                              <div class="input-group-btn">
                                    <button      name="buscar"      id="buscar"
class="btn btn-default" type="submit">
                                          <i     class="glyphicon     glyphicon-
search"></i>
```

```
                                    </button>
                                </div>
                        </div>
                    </form>
                    <ul class="nav navbar-nav navbar-right">
                        <li><a href="/pedido/">Meus Pedidos</a></li>
                        <li><a href="/cliente/p">Entre</a></li>
                        <li><a href="/cadastro/">Cadastre-se</a></li>
                        <li                                    class="login-click"><a
href="/carrinho/">Carrinho</a></li>
                        <li>
                    </ul>
            </div>
            <!--/.nav-collapse -->
            </div>
      <!-- Page Content -->
      <br />
            <div class="menu dropdown">
                    <ul>
                        <li><span><a href="/index/">Home</a></span></li>
                        <li class="dropdown"><a href="#" class="dropdown-toggle"
                                                data-toggle="dropdown">Produtos    <b
class="caret"></b></a>
                                                <ul class="dropdown-menu">
                                                <li><a href="#">BlaBla</a></li>
                                            </ul>
                        </li>
                        <li><a href="#">Contato</a></li>
                    </ul>
            </div>
            <div class="spacer">
                    <br /> <br />
            </div>
            <div class="row" id="divmain">
                    <jsp:doBody />
            </div>

      <script src="<c:url value="/static/js/jquery-2.1.1.min.js"/>"></script>
      <script src="<c:url value="/static/js/bootstrap.min.js"/>"></script>
      <script src="<c:url value="/static/js/jquery-ui.min.js"/>"></script>
      <script src="<c:url value="/static/js/bootbox.js"/>"></script>
      <script src="<c:url value="/static/js/geral.js"/>"></script>

      <jsp:invoke fragment="scriptsEspecificos"></jsp:invoke>>

<footer class="row text-center fixed">
                    <div class="bs-footer footer">
                        <div class="col-md-6">
                            <h5>Copyright &copy; Douglas Luiz Mondstock 2016
</h5>
                    </div>
                    <div class="col-md-6">
                        <h5>Copyright &copy; Douglas Luiz Mondstock 2016</h5>
                    </div>
                    </div>
</footer>
</body>
</html>
```

Listagem 3 - Código para o template do leiaute do sistema

A página que lista os produtos da página inicial é o index.jsp. Nessa página é carregado um Json com os dados dos produtos e montado na tela por meio do código JavaScript apresentado na Listagem 4.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib tagdir="/WEB-INF/tags/layoutclient" prefix="layout" %>

<layout:template>
      <jsp:attribute name="cssEspecificos">
            <link href="<c:url value="/static/css/geral.css" />"
            rel="stylesheet" />
            <link href="<c:url value="/static/css/bootstrap.min.css" />"
            rel="stylesheet" />
            <link href="<c:url value="/static/css/jquery-ui.min.css"/>"
            rel="stylesheet" />
      </jsp:attribute>
      <jsp:attribute name="scriptsEspecificos">
      <script src="<c:url value="/static/js/bootbox.js"/>"></script>

      <script type="text/javascript">
                  $(document).ready(function(){
                        carregarDados();
                  });
function carregarDados(){
                        $.getJSON("/listaproduto/dados", function(data){
                              var div;
                              //var quantidade = 1;

                              $.each(data.data, function(){
                                    var valor = numeroParaMoeda(this[3]);
                                    div = '<div class="col-md-3 col-sm-6" >'
+
                                 '<div class="thumb thumbnail" style="min-
height: 315px;max-height: 396px;">' +
                                       '<img class="list-img img-responsive"
style="max-height: 154px; border-radius: 63px;"src="<c:url
value="/static/img/'+ this[6] + '.jpg"/>"' +
                                       '<div class="caption border-radius:
63px;">' +
                                       '<input type="hidden" id="nomeProduto'+
this[0] +'" value="'+this[1]+'" />' +
                                             '<h3>' + this[1] + '</h3>' +
                                             '<input type="hidden"
id="valor'+this[0]+'" value="'+this[3]+'" />' +
                                             '<div class="valor"><h2>R$ '+
valor+'</h2></div>'+
                                             '<p>'+ this[4]+'</p>'+

                                             '<a href="#"
onclick="adicionarProduto('+this[0]+')" class="btn btn-primary">Add ao
Carrinho</a>' +
                                             '<input type="text"
class="quantidade" name="quantidade" id="quantidade'+this[0]+'" value="1"
/>'+
                                          '<a href="/listaproduto/' + this[0]
+'" class="btn btn-default" id="detalhes">Detalhes</a>' +
```

```
                                        '</p>' +
                                    '</div>' +

                                '</div>' +
                            '</div>';
                                $("#divmain1").append(div);
                            });//Fim each

                        });//Fim getJSON
                    } //Fim carregarDados
            </script>
      </jsp:attribute>

      <jsp:body>
            <div class="row text-center" id="divmain1">
                        </div>
            <div class="spacer">
                  <br /> <br /> <br /> <br />
            </div>
      </jsp:body>
</layout:template>
```

Listagem 4 - JavaScritp do index

A página *index* e todas as outros que utilizam esse template são injetadas no leiaute por meio da *tag <layout:template></layout:template>*

Com o *framework* Spring toda classe que possui a *annotation* @Controller pode ser acessada por uma URL como a classe de ListaProdutoController que contém o método listarDados que é responsável por montar o Json dos produtos da tela index.jsp, o método busca que retorna o produto digitado na barra de pesquisa do *template* e o método detalhesProtudo que retorna o Json para a tela de produtoDetalhes.jsp. O código da classe ListaProdutoController é apresentado na Listagem 5.

```
@Controller
@RequestMapping("/listaproduto")
public class ListaProdutoController {
      @Autowired
      ProdutoRepository produtoRepository;
      @RequestMapping(value = "/dados", produces = {"application/json";
charset=utf-8})
      @ResponseBody
      public String listarDados() {
            List<Produto> lista = produtoRepository.findAll();
            JSONObject retorno = new JSONObject();
            JSONArray array = new JSONArray();
            JSONArray ja;
            for (Produto produto : lista) {
                  ja = new JSONArray();
                  ja.put(produto.getIdproduto());
                  ja.put(produto.getNome());
                  ja.put(produto.getDescricao());
                  ja.put(produto.getValorUnitario());
                  ja.put(produto.getMarca().getNome());
```

```
                    ja.put(produto.getCategoria().getDescricao());
                    ja.put(produto.getFoto());
                    array.put(ja);
                }
                retorno.put("data", array);
                return retorno.toString();
        }
        @RequestMapping(value = "/b", method = RequestMethod.GET)
        public String buscaProduto(@RequestParam("buscaProduto") String
buscaProduto, Model model) {
                model.addAttribute("buscaProduto", buscaProduto);
                return "busca";
        }
        @RequestMapping(value = "/busca/{buscaProduto}", method = RequestMethod.GET,
produces = {"application/json"; charset=utf-8})
        @ResponseBody
        public String busca(@PathVariable String buscaProduto ) {
                JSONObject retorno = new JSONObject();
            if(buscaProduto != null){
                    List<Produto> lista =
produtoRepository.findByNomeStartingWith(buscaProduto);
                    JSONArray array = new JSONArray();
                    JSONArray ja;
                    for (Produto produto : lista) {
                            ja = new JSONArray();
                            ja.put(produto.getIdproduto());
                            ja.put(produto.getNome());
                            ja.put(produto.getDescricao());
                            ja.put(produto.getValorUnitario());
                            ja.put(produto.getMarca().getNome());
                            ja.put(produto.getCategoria().getDescricao());
                            ja.put(produto.getFoto());
                            array.put(ja);
                    }
                    retorno.put("data", array);
            }
            return retorno.toString();
        }
        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public String editar(@PathVariable Long id, Model model) {
                model.addAttribute("id", id);
                return "produtoDetalhes";
        }
        @RequestMapping(value = "/detalhes/{codigo}", method = RequestMethod.GET,
produces = {"application/json"; charset=utf-8})
        @ResponseBody
        public String detalhesProduto(@PathVariable Long codigo) {
                JSONObject retorno = new JSONObject();
                Produto produto = produtoRepository.findOne(codigo);
                retorno.put("idproduto", produto.getIdproduto());
                retorno.put("nome", produto.getNome());
                retorno.put("foto", produto.getFoto());
                retorno.put("descricao", produto.getDescricao());
                retorno.put("valor", produto.getValorUnitario());
                retorno.put("marca", produto.getMarca().getNome());
                return retorno.toString();
        }
```

Listagem 5 - classe ListaProdutoController

Um produto é adicionado ao carrinho por meio da Classe CarrinhoController e

do método adicionar() que é apresentado na Listagem 6.

```
@RequestMapping(value = "/adicionar", method=RequestMethod.POST, produces =
={"application/json"; charset=utf-8}
	@ResponseBody
	public String adicionar(HttpServletRequest  request){
		Item item = new Item();
		item.setId(Long.parseLong(request.getParameter("idproduto")));
		item.setNome(request.getParameter("nomeProduto"));
	item.setQuantidade(Integer.parseInt(request.getParameter("quantidade")));
		item.setValor(Double.parseDouble(request.getParameter("valor")) *
Integer.parseInt(request.getParameter("quantidade")));
		item.setFoto(produtoRepository.getOne(item.getId()).getFoto());
		Carrinho carrinho;
		//Caso exita um carrinho na sessão ele é recuperado
		if(request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA) !=
null){
			carrinho = (Carrinho)
request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA);
		}else{
			carrinho = new Carrinho();
		}
		carrinho.adicionar(item);
		request.getSession().setAttribute(Carrinho.CARRINHO_COMPRA,
carrinho);
		JSONObject retorno = new JSONObject();
		retorno.put("itens", carrinho.getJsonItens());
		retorno.put("situacao", "OK");
		retorno.put("mensagem", "Produto adicionado ao carrinho");
		return retorno.toString();
	}
```

Listagem 6 - Método adicionar() da classe CarrinhoController

Os itens adicionados ao carrinho da sessão são listados na tela carrinho.jsp por meio de um JavaScript, cujo código está na Listagem 7)..

```
function carregarDados(){
			$("#carrinho").html("");
			$.getJSON("/carrinho/dados", function(data){
				$.each(data.data, function(){
	var valor = numeroParaMoeda(this[3]);
					div ='<div class="row">' +
						'<div class="col-md-4 col-md-offset-1 col-
sm-6" >' +
						'<img class="list-img img-responsive"
style="max-height: 154px;"src="<c:url value="/static/img/'+
						 this[5].toLowerCase() + '.jpg"/>"' +
						'</div>' +
						'</div>' +
					   '<div class="col-md-3 col-sm-6" >' +
					'<h3>' + this[1] + '</h3>' +
					'<p>Valor : R$ '+ valor +'</p>'+
					'<input type="text" class="quantidade"
name="quantidade" id="quantidade" value="'+this[2]+'" />'+
					'</div>' +
					  '<a href="#" onclick="excluirRegistro(' +
this[0] +')" class="excluir btn btn-default"' +
					  'id="excluir">Excluir</a>' +
```

```
                                        '</div>' +
                                    '</div>' +
                                    '</div>' +
                                '</div>';
                                $(div).appendTo("#carrinho");

                            });//Fim each
                        });//Fim getJSON
                } //Fim carregarDados
```

Listagem 7 - Carrinho.jsp

Por meio do JSON é montado pelo método listar().

```
@RequestMapping(value = "/dados", method=RequestMethod.GET, produces =
{"application/json"; charset=utf-8})
	@ResponseBody
	public String listar(HttpServletRequest request){
		Carrinho carrinho;
		//Caso exita um carrinho na sessão ele é recuperado
		if(request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA) !=
null){
			carrinho = (Carrinho)
request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA);
		}else{
			carrinho = new Carrinho();
		}
		JSONObject retorno = new JSONObject();
		JSONArray array = new JSONArray();
		JSONArray itens;
		for (Item item : carrinho.getItens()) {
			itens =  new JSONArray();
			itens.put(item.getId());
			itens.put(item.getNome());
			itens.put(item.getQuantidade());
			itens.put(item.getValor());
			itens.put(item.getNome());
			itens.put(item.getFoto());
			array.put(itens);
		}
		retorno.put("data", array);
		return retorno.toString();
	}
```

Listagem 8 - Método listar()

Para excluir itens é utilizado outro JavaScript na página carrinho.jsp. O código para a exclusão de registros é apresentado na Listagem 9.

```
function excluirRegistro(id){
	bootbox.confirm('Deseja excluir o item do carrinho?!',
		function(resultado){
			if (resultado){
			$.ajax({
			type: 'POST',
			url : '/carrinho/remover/' + id,
			success: function(dados){
				if (dados.situacao == "OK"){
					exibirMensagem(dados.situacao, dados.mensagem);
```

```
                                carregarDados();
                            }
                        }//FIM success
                    });//FIM Ajax
                }//FIM if
        }); //FIM bootbox
} // FIM Excluir
```

Listagem 9 - Exclusão de registros do carrinho de compras

A exclusão de registros do carrinho de compras utiliza de o método remover da Classe CarrinhoController.

```
@RequestMapping(value = "/remover/{codigo}", method=RequestMethod.POST, produces =
{"application/json"; charset=utf-8})
        @ResponseBody
        public String remover(HttpServletRequest request, @PathVariable Long
codigo){
                Item item = new Item();
                item.setId(codigo);
                Carrinho carrinho = new Carrinho();
                //Caso exista um carrinho na sessão ele é recuperado
                if(request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA) !=
null){
                        carrinho = (Carrinho)
request.getSession().getAttribute(Carrinho.CARRINHO_COMPRA);
                }
                carrinho.remover(item);

                request.getSession().setAttribute(Carrinho.CARRINHO_COMPRA,
carrinho);
                JSONObject retorno = new JSONObject();
                retorno.put("itens", carrinho.getJsonItens());
                retorno.put("situacao", "OK");
                retorno.put("mensagem", "Produto removido do carrinho");
                return retorno.toString();
        }
```

Listagem 10 - Método remover da Classe CarrinhoController

O arquivo geral.js contém JavaScripts que são importantes para o sistema. O código desse arquivo está na Listagem 11.

```
function exibirMensagem(tipo, mensagem) {
        if (tipo == 'OK') {
                $('#divSucesso').html(mensagem);
                $('#divSucesso').fadeIn('fast');
                $('#divSucesso').fadeOut(3000);
        } else {
                $('#divErro').html(mensagem);
                $('#divErro').fadeIn('fast');
                $('#divErro').fadeOut(2000);
        }
}
function redirecionar(destino) {
        window.location = destino;
}
function adicionarProduto(idproduto) {
```

```
	var nomeProduto = $('#nomeProduto'+idproduto).val();
	var valor = $('#valor'+idproduto).val();
	var quantidade = $('#quantidade'+idproduto).val();
	$.ajax({
		type : 'POST',
		url : '/carrinho/adicionar?idproduto=' + idproduto
				+ '&nomeProduto=' + nomeProduto
				+ '&quantidade=' +quantidade + '&valor='
				+ valor,
		success : function(dados) {
			bootbox.alert(dados.mensagem, function(resultado) {
			});
		}// FIM success
	});// FIM Ajax
}
function finalizarCompra(){
	var elem = document.getElementById("formaPgto").value;
	$.ajax({
		type: 'POST',
		url : '/checkout/salvar/' + elem,
		success : function(dados){
			//exibirMensagem(dados.situacao,dados.mensagem);
			if(dados.situacao == "OK"){
				bootbox.alert(dados.mensagem, function(resultado) {
					redirecionar("/pedido/");
				});
				enviarEmail();
			}
		}
	});//FIM Ajax
}function enviarEmail(){
	$.ajax({
		type: 'GET',
		url : '/checkout/emailConfirmacao'
	});//FIM Ajax
}
function imprimirPedido(){
	var conteudo;
	$.each(document.getElementById('pedido'),function() {
		conteudo = document.getElementById('pedido').innerHTML +
document.getElementById('produto').innerHTML;
	});
	tela_impressao = window.open('about:blank');
	tela_impressao.document.write(conteudo);
	tela_impressao.window.print();
	tela_impressao.window.close();
}
function numeroParaMoeda(n, c, d, t)
{
    c = isNaN(c = Math.abs(c)) ? 2 : c, d = d == undefined ? "," : d, t = t
== undefined ? "." : t, s = n < 0 ? "-" : "", i = parseInt(n = Math.abs(+n
|| 0).toFixed(c)) + "", j = (j = i.length) > 3 ? j % 3 : 0;
    return s + (j ? i.substr(0, j) + t : "") +
i.substr(j).replace(/(\d{3})(?=\d)/g, "$1" + t) + (c ? d + Math.abs(n -
i).toFixed(c).slice(2) : "");
}
```

Listagem 11 - Funções do arquivo geral.js

O método finalizarCompra() chama o método salvar() da Classe

SaidaController. Esse método é responsável por salvar cada saída com seus determinados itens. O código do método finalizarCompra() é apresentado na Listagem 12.

```
@RequestMapping(value = "/salvar/{elem}", method = RequestMethod.POST,
produces = {"application/json; charset=utf-8"})
@ResponseBody
public String salvar(HttpServletRequest request, @PathVariable Long elem) {

	Saida saida = new Saida();
	saida.setDatasaida((new Date()));
	saida.setFormaPgto(String.valueOf(elem));
	Usuario usuario = (Usuario) SecurityContextHolder.getContext()
			.getAuthentication().getPrincipal();
	saida.setUsuario(usuario);
	List<Movimentacao> lista = new ArrayList<>();

	Carrinho carrinho = (Carrinho) request.getSession().getAttribute(
			Carrinho.CARRINHO_COMPRA);

	Movimentacao mv;
	for (Item item : carrinho.getItens()) {
		int qtdeEstoque = 0;
		int estoque = 0;
		mv = new Movimentacao();
		Produto produto = produtoRepository.findOne(item.getId());
		qtdeEstoque = produto.getQtdeEstoque();

		if (qtdeEstoque >=  item.getQuantidade()){
		estoque = qtdeEstoque - item.getQuantidade();
		produto.setQtdeEstoque(qtdeEstoque);;
		mv.setSaida(saida);
		mv.setProduto(produtoRepository.findOne(item.getId()));
		mv.setQuantidade(item.getQuantidade());
		mv.setValor(item.getValor() * item.getQuantidade());
	mv.setCodProdutoMercado(mv.getProduto().getCodProdutoMercado());
		lista.add(mv);
		}
	}
	saida.setMovimentacao(lista);
	JSONObject retorno = new JSONObject();
	try {
	saidaRepository.save(saida);
	request.getSession().removeAttribute(Carrinho.CARRINHO_COMPRA);
		retorno.put("situacao", "OK");
		retorno.put("mensagem", "Pedido efetuado com sucesso!");
	request.getSession().removeAttribute(Carrinho.CARRINHO_COMPRA);
	} catch (Exception ex) {
		retorno.put("situacao", "ERRO");
		retorno.put("mensagem", "Erro na compra!");
	}
	return retorno.toString();
}
```

Listagem 12 - Método finalizarCompra()

A classe SaidaController também possui o método emailConfirmacao() (Listagem 13) que é responsável por enviar um *email* ao cliente em cada compra

realizada com sucesso. Esse método é chamado no finalizarCompra() apresentado na Listagem 12 do arquivo geral.js

```
@RequestMapping(value = "/emailConfirmacao", method = RequestMethod.GET, produces
= {"application/json; charset=utf-8"})
	@ResponseBody
	public void emailConfirmacao(HttpServletRequest request) {

		JSONObject retorno = new JSONObject();
		Usuario usuario = (Usuario) SecurityContextHolder.getContext()
				.getAuthentication().getPrincipal();
		try {
			SendEmailController1 email = new SendEmailController1(
					new String[] { usuario.getUsername() },
					"Situação da sua Compra",
					"<h1>Seu pedido foi realizado com sucesso em breve
será entregue em sua casa!!!</h1>");
			email.sendEmail();
			System.out.println("Email enviado com sucesso");
		} catch (Exception ex) {
			System.out.println("Não foi possível enviar o email");
		}
	}
```

Listagem 13 - Método emailConfirmacao

O código da classe SendEmailController é apresentado na Listagem 14.

```
private String[] from;
	private String html;
	private String subject;
	private String EMAIL = "douglasmondstock@gmail.com";
	private String SENHA_EMAIL = "**********";
	public SendEmailController1() {
		// TODO Auto-generated constructor stub
	}
	public SendEmailController1(String[] from,String subject, String html) {
		this.from = from;
		this.html = html;
		this.subject = subject;
	}
	private Properties getProperties() {
		Properties props = new Properties();
		props.put("mail.smtp.host", "smtp.gmail.com");
		props.put("mail.smtp.socketFactory.port", "465");
		props.put("mail.smtp.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");
		props.put("mail.smtp.auth", "true");
		props.put("mail.smtp.port", "465");
		return props;
	}

	public void sendEmail(String[] from, String subject, String html) {
		this.from = from;
		this.html = html;
		this.subject = subject;
		executeSendEmail();
	}
```

```
	public void sendEmail() {
		executeSendEmail();
	}

	private void executeSendEmail(){

		Session session = Session.getInstance(getProperties(), new
javax.mail.Authenticator() {
		    protected PasswordAuthentication getPasswordAuthentication() {
		        return new PasswordAuthentication(EMAIL, SENHA_EMAIL);
		    }
		});

		/** Ativa Debug para sessão */
		session.setDebug(true);

		try {

			MimeMessage message = new MimeMessage(session);
			message.setFrom(new InternetAddress(this.EMAIL)); // Remetente

			Address[] toUser = InternetAddress // Destinatário(s)
					.parse(configureEmailsSend(this.from));

			message.addRecipients(Message.RecipientType.CC, toUser);
			message.setSubject(this.subject);// Assunto
			message.setText(this.html, "utf-8", "html");
			/** Método para enviar a mensagem criada */
			Transport.send(message);

			System.out.println("Feito!!!");

		} catch (MessagingException e) {
			throw new RuntimeException(e);
		}
	}

	private String configureEmailsSend(String[] from) {
		StringBuilder retorno = new StringBuilder();
		for (int i = 0; i < from.length; i++) {
			retorno.append(from[i]);
			retorno.append(",");
		}

		return retorno.toString();
	}

	public String[] getFrom() {
		return from;
	}

	public void setFrom(String[] from) {
		this.from = from;
	}

	public String getHtml() {
		return html;
	}
```

```
	public void setHtml(String html) {
		this.html = html;
	}

	public String getSubject() {
		return subject;
	}

	public void setSubject(String subject) {
		this.subject = subject;
	}
```

Listagem 14 - Classe SendEmailController

O Json da tela de pedidos dos clientes é montado por meio da classe PedidoClienteController no método listar() como mostra o código da Listagem14. Por meio desse método, são listados cada saída e seus respectivos produtos.

```
@RequestMapping(value = "/listar", method=RequestMethod.GET, produces =
{"application/json; charset=utf-8"})
	@ResponseBody
	public String listar() {
		Usuario usuario = (Usuario)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
		System.out.println(usuario);
		List<Saida> lista = saidaRepository.findByUsuario(usuario);
		JSONArray array = new JSONArray();
		for (Saida saida : lista) {
			JSONObject saidaJSON = new JSONObject();
			saidaJSON.put("idsaida", saida.getIdsaida());
			saidaJSON.put("nomecliente", saida.getUsuario().getNome());
			SimpleDateFormat dataF = new SimpleDateFormat("dd/MM/yyyy");
			saidaJSON.put("data", dataF.format(saida.getDatasaida()));
			JSONArray produtos = new JSONArray();
			for (Movimentacao produto : saida.getMovimentacao()) {
				JSONObject produtoJSON = new JSONObject();
				produtoJSON.put("idproduto", produto.getProduto()
						.getIdproduto());
				produtoJSON.put("nome", produto.getProduto().getNome());
				produtoJSON.put("quantidade", produto.getQuantidade());
				produtoJSON.put("valor", produto.getValor());

				produtos.put(produtoJSON);
			}
			saidaJSON.put("produtos", produtos);
			array.put(saidaJSON);
		}
		System.out.println(array.toString());
		return array.toString();
	}
```

Listagem 15 - Classe PedidoClienteController

O JavaScript que lista os pedidos na tela pedidos.jsp é apresentado na Listagem 16.

```
function carregarDados(){
```

```
	$.getJSON("/pedido/listar", function(datainicio){
		var i = 0;
		$.each(datainicio, function(data){
		tabela = '<div id="pedido">'+
			'<table class="table table-striped table-bordered display">' +
				'<thead>' +
				'<tr>' +
					'<td>Código do pedido</td>'+
					'<td>Data do pedido</td>'+
					'<td>Imprimir</td>'+
					'</tr>'+
					'</thead>'+
					'<tbody>'+
					'<td>' + this.idsaida+ '</td>'+
					'<td>'+ this.data +'</td>'+
					'<td><a href="#"
onclick="imprimirPedido()">Imprimir pedido</a></td>' +
					'</tr>'+
					'</tbody>'+
					'</table>'+
				'</div>'
				$("#table").append(tabela);
				i++;
				var j = 0;
				$.each(this.produtos, function(dataproduto){
				var valor = numeroParaMoeda(this.valor);
					tabela = '<div id="produto">'+
					'<table class="table display table-bordered">' +
					'<thead>' +
					'<tr>' +
						'<td>Código do  Produto</td>'+
						'<td>Produto</td>'+
						' <td>Quantidade</td>'+
						'<td>Valor</td>'+
							'</tr>'+
							'</thead>'+
							'<tbody>'+
							'<tr>' +
							'<td>'+ this.idproduto+ '</td>'+
							'<td>'+ this.nome+ '</td>'+
							'<td>' + this.quantidade + '</td>'+
							'<td>R$ '+ valor +'</td>'+
								'</tr>'+
							'</tbody>'+
						'</table>'+
						'<div class="spacer"><br /></div>'+
						'</div>'
						$("#table").append(tabela)
						j++;
							});
						});//Fim each
					});//Fim getJSON
		} //Fim carregarDados
		</script>
```

Listagem 16 - Função para carregar dados

O *login* de usuários é gerenciado pela classe LoginController e login.jsp por

meio do Spring Security como apresentado no código da Listagem 17.

```
@Controller
public class LoginController {
	@RequestMapping(value = "/login", method = RequestMethod.GET)
	public String login(@RequestParam(value = "error", required=false)
		String error, Model model){
		if (error != null){
			model.addAttribute("error", "<h3>Usuário ou senha inválidos!!!
Tente outra vez.</h3>");
		}
		return "login";
	}
	@RequestMapping("/logout")
	public String logout(){
		return "/";
	}
}

<form action="login" method="POST" class="login-form">
		<div class="form-group">
			<label class="sr-only" for="username">Usuário</label> <input
				type="text" name="username" id="username"
	class="form-signin form-username form-control" placeholder="Usuário"
required autofocus>
		</div>
		<div class="form-group">
			<label class="sr-only" for="password">Senha</label> <input
			type="password" name="password" id="password"
			class="form-signin form-password form-control"
placeholder="Senha" required>
		</div>
		<button type="submit" id="submit" name="submit" class="bt btn btn-lg
btn-primary">Entrar</button>
		<button id="Cancelar" name="Cancelar" class="bt btn btn-lg  btn-
danger" onClick="history.go(-1)">Voltar</button>
</form>
${error}
```

Listagem 17 - Classe classe LoginController

O cadastro de clientes é feito pelo cad-cliente.jsp e pela classe CadastroClienteController utilizando um formulário com método post e ação a url do método na classe e do JavaScript (Listagem 18).

```
<form action="/cliente/salvar" method="POST" id="frm" name="frm"  class="form-
horizontal">
$("#frm").submit(function(){
	$.ajax({
	type : $("#frm").attr('method'),
	url : '/cadastro/salvar',
	data: $("#frm").serialize(),
	success: function(dados){
		if(dados.situacao == "OK"){
			bootbox.alert(dados.mensagem, function(resultado) {
				redirecionar("/usuario/p");
				});
		}else if(dados.situacao == "ERRO"){
```

```
                    exibirMensagem(dados.situacao, dados.mensagem);
                    }
                    }//Fim success
              }); //Fim Ajax
              return false;
       }); //Fim form submit

@RequestMapping(value = "/salvar", method = RequestMethod.POST, produces =
{"application/json"; charset=utf-8})
       @ResponseBody
       public String salvar(Usuario cliente, BindingResult erros, Model model){

              JSONObject retorno = new JSONObject();


              if(usuarioRepository.findByCpf(cliente.getCpf()) != null){
                     retorno.put("situacao", "ERRO");
                     retorno.put("mensagem", "Cpf ja cadastrado!!!");
              }else if(usuarioRepository.findByUsername(cliente.getUsername()) !=
null){
                     retorno.put("situacao", "ERRO");
                     retorno.put("mensagem", "Email ja cadastrado!!! Por gentileza
informe outro endereço de email");

              }else{
                     String password = cliente.getPassword();
                     String encodedPassord = cliente.getEncodedPassword(password);
                     cliente.setPassword(encodedPassord);

                     cliente.addPermissao( getPermissao() );

                     usuarioRepository.save(cliente);
                     retorno.put("situacao", "OK");
                     retorno.put("mensagem", "Cadastro efetuado com sucesso!!!");
              }

              return retorno.toString();
       }
```

Listagem 18 - Cad-cliente.jsp

Na parte administrativa do sistema, o *upload* de arquivos é feito pelo código da classe ImportarProdutosController (Listagem 19)

```
RequestMapping(value = "/uploadFile", method = RequestMethod.POST, produces =
{"application/json"; charset=utf-8})
       @ResponseBody
       public String uploadFileHandler(@RequestParam("name") String
name,@RequestParam("file") MultipartFile file) {
              JSONObject retorno = new JSONObject();

              if (!file.isEmpty()) {
                     try {
                            byte[] bytes = file.getBytes();

                            // Create the file on server
                            File serverFile = new
File("D:\\Facul_UTFPR\\TCC\\supermercado\\src\\main\\webapp\\WEB-INF\\xml\\"+
```

```
name);
                        BufferedOutputStream stream = new BufferedOutputStream(
                                    new FileOutputStream(serverFile));
                        stream.write(bytes);
                        stream.close();

                        Logger.info("Server File Location="    +
serverFile.getAbsolutePath());

                        retorno.put("situacao", "OK");
                        retorno.put("mensagem", "Você fez o upload com sucesso
do arquivo=" + name);
                  } catch (Exception e) {
                        retorno.put("situacao", "ERRO");
                        retorno.put("mensagem", "Não foi possível fazer o
upload=" + name);
                  }
            } else {
                  retorno.put("situacao", "ERRO");
                  retorno.put("mensagem", "Não foi possível fazer o upload pois o
arquivo está vazio=" + name);
            }
            return retorno.toString();
      }
```

Listagem 19 - Classe ImportarProdutosController

O *upload* de arquivos é também chamado na tela pelo código da página importarProdutos.jsp (Listagem 20).

```
<form method="POST" action="/importar/uploadFile" id="upload" name="upload"
      enctype="multipart/form-data" class="form-horizontal">
      <div class="form-group">
            <label class="col-md-4 control-label" for="textinput">Nome do
Arquivo</label>
      <div class="col-md-4">
             <input type="text" name="name" class="form-control input-md">
      </div>
      </div>
      <div class="form-group">
            <label class="col-md-4 control-label" for="textinput">Arquivo</label>
      <div class="col-md-4">
             <input type="file" class="input-file" name="file">
      </div>
      </div>
      <div class="spacer"><br /><br /></div>
            <button id="upload" name="upload" class="btn btn-primary btn-block"
            type="submit">Upload do arquivo</button>
</form>

$("#upload").submit(function(){
            $.ajax({
                        type : $("#frm").attr('method'),
                         url : '/importar/uploadFile' + name + file,
                         data: $("#frm").serialize(),
                        success: function(data){
                              exibirMensagem(data.situacao, data.mensagem);
                        },
                        error: function(){
```

```
                                exibirMensagem(data.situacao, data.mensagem);
                            }//Fim success
                        }); //Fim Ajax
                        return false;
                }); //Fim form submit
```

Listagem 20 - ImportarProdutos.jsp

A importação do XML é chamada pelo formulário que está na classe importarProdutos.jsp e pelo método apresentado na Listagem 21.

```
<form action="/importar/importarXML" method="POST" id="frm" name="frm"
class="form-horizontal" >
                <button id="salvar" name="salvar" class="btn btn-primary btn-block"
                        type="submit">Importar dados do xml</button>
                        </form>
@RequestMapping(value = "/importarXML", method = RequestMethod.POST, produces =
{"application/json"; charset=utf-8})
        @ResponseBody
        public String importarXML(){
          XStream xstream= new XStream();
          JSONObject retorno = new JSONObject();
               try {
                        FileInputStream fis = new FileInputStream(new
File("D:\\Facul_UTFPR\\TCC\\supermercado\\src\\main\\webapp\\WEB-
INF\\xml\\produtos.xml"));
                        @SuppressWarnings("unchecked")
                        List<Produto> produtos = (List<Produto>)xstream.fromXML(fis);
                        Produto produtoBanco;
                        for (Produto produto : produtos) {
                                produtoBanco =
produtoRepository.findByCodProdutoMercado(produto.getCodProdutoMercado());
                                if(produtoBanco != null){
                                        produto.setIdproduto(produtoBanco.getIdproduto());
                                }
                                produtoRepository.save(produto);
                                marcaRepository.save(produto.getMarca());
                                categoriaRepository.save(produto.getCategoria());
                        }
                        retorno.put("situacao", "OK");
                        retorno.put("mensagem", "Produtos importados com sucesso!");

                } catch (IOException e) {
                        retorno.put("situacao", "ERRO");
                        retorno.put("mensagem", "Não foi possível importar os
produtos!");
                        e.printStackTrace();
                }
                return retorno.toString();
        }
```

Listagem 21 - Formulário para importar XML

Por fim, para exportar a movimentação é utilizado o método gravarXML() da classe ExportarMovimentacaoController chamada na tela exportarMovimentacao.jsp pelo código da Listagem 22.

```
@RequestMapping(value = "/gravarXML", method = RequestMethod.POST, produces =
{"application/json"; charset=utf-8})
	@ResponseBody
	public String gravarXML(){
		JSONObject retorno = new JSONObject();
	  XStream xstream= new XStream();
		try {
				List<Movimentacao> movimentacoes =
movimentacaoRepository.findAll();
				Date data =  new Date();
				SimpleDateFormat dataF = new SimpleDateFormat("dd-MM-
yyyy-hh-mm-ss");

				FileWriter arquivo = new
FileWriter("D:\\Facul_UTFPR\\TCC\\supermercado\\src\\main\\webapp\\WEB-
INF\\xml\\movimentacao"+
				dataF.format(data).toString() +".xml");
				arquivo.append(xstream.toXML(movimentacoes));
				arquivo.flush();
				arquivo.close();
				retorno.put("situacao", "OK");
				retorno.put("mensagem", "Movimentação exportada com
sucesso!");
		} catch (IOException e) {
			// TODO Auto-generated catch block
			retorno.put("situacao", "ERRO");
			retorno.put("mensagem", "Erro ao exportar!");
			e.printStackTrace();
		}
		return retorno.toString();
	}

form action="/exportar/gravarXML" method="POST" id="frm" name="frm" class="text-
center form-horizontal" >
		<button id="salvar" name="salvar" class="btn btn-primary"
			type="submit">Exportar Dados</button>
		</form>

$("#frm").submit(function(){
			$.ajax({
				type : $("#frm").attr('method'),
				 url : '/exportar/gravarXML',
				 data: $("#frm").serialize(),
				success: function(data){
					exibirMensagem(data.situacao, data.mensagem);
				},
				error: function(){
					exibirMensagem(data.situacao, data.mensagem);;
			    }//Fim success
			}); //Fim Ajax
			return false;
		}); //Fim form submit
		</script>
```

Listagem 22 - Método gravarXML

CONCLUSÃO

Comércio eletrônico é utilizado por uma parcela significativa da população e as operações realizadas são as mais diversas: de comércio de bens intangíveis como músicas, livros digitais, a acesso a portais de conteúdo, passando por bens duráveis como móveis, eletrodomésticos e produtos de vestuário, chegando a bens perecíveis como frutas e verduras.

Além das facilidades que os sites de comércio eletrônico provêm aos usuários, que é a compra de produtos sem a necessidade de deslocar-se fisicamente ao estabelecimento, eles possibilitam mais facilmente comparar produtos e preços. Esse tipo de comércio também pode fornecer serviços agregados. Entre esses serviços está a entrega, que por razões diversas, como econômicas, pode ser realizada por empresas terceirizadas. Nesses casos, o sistema de comércio eletrônico e o sistema da empresa responsável pelo *delivery* podem trocar dados. Recursos tecnológicos têm procurado garantir que essa troca ocorra de forma segura.

O objetivo deste trabalho foi implementar um sistema para realizar compras de um supermercado *delivery*. A aplicação foi desenvolvida para a plataforma *web*, pela facilidade de acesso e manutenção. Para esse desenvolvimento, foram utilizadas diversas ferramentas e tecnologias.

Uma dessas ferramentas utilizadas é o Spring que é um *framework open-source* criado para simplificar o desenvolvimento. Todos os componentes do sistema são gerenciados pelo Spring utilizando inversão de controle e injeção de dependência. A Inversão de controle possibilita menos acoplamento entre as dependências, delegando para uma outra classe, interface, componente, serviço, etc.

O Spring possui um módulo *Model-View-Controller*, fornece integração com vários *frameworks* e foi projetado em torno de um DispatcherServer que despacha pedidos para os controladores configuráveis baseado em duas anotações básicas *@Controller* () e *@RequestMapping*.

Também foi utilizado o Spring-boot que é um projeto da Pivot para simplificar o início de aplicações Spring. Esse projeto elimina os arquivos XML e as

configurações básicas de uma aplicação Spring, configurando todas as bibliotecas necessárias e disponibiliza um servidor Tomcat ou um Jetty.

Na utilização das tecnologias para implementar o sistema desenvolvido como resultado deste trabalho não foram encontradas dificuldades consideráveis, além das pertinentes ao uso de tecnologias novas para um desenvolvedor. Essas dificuldades foram superadas com a pesquisa em materiais técnicos e ajudas em fóruns especializados.

Como trabalhos futuros, ressaltam-se a implementação de funcionalidades, como alterar a situação do pedido, calcular frete e informar a data prevista para a entrega.

REFERÊNCIAS

ALBERTIN, Alberto Luiz. Comércio eletrônico: benefícios e aspectos de sua aplicação. Administração da Produção e Sistemas de Informação. **RAE - Revista de Administração de Empresas**. São Paulo, v. 38, n. 1, p. 52-63 Jan./Mar. 1998.

BRYNJOLFSSON, Erik; YU, Jeffrey H.; SMITH, Michael D. **Consumer surplus in the digital economy: estimating the value of increased product variety at online booksellers**. Management Science, v. 49, n. 11, p. 1580-1596, 2003.

ECOMMERCEORG. **Evolução da Internet e do e-commerce**. Disponível em: <http://www.e-commerce.org.br/stats.php>. Acesso em: 23 fev. 2015.

HOU, Jiachen; XIONG, Gang; LIU, Xiwei; FAN, Dong; LIU, Shen. **A kind of online power materials supermarket system**. IEEE International Conference on Automation and Logistics, 2012, p. 57-62.

KISS, Christine; BICHLER, Martin. **Identification of influencers measuring influence in customer networks**. Decision Support Systems, *v.* 46, n. 1, p. 233-253, 2008.

LADAN, Mohamad I. **E-commerce security issues**. 2014 International Conference on Future Internet of Things e Cloud, 2014, p. 197-201.

LEAL, Fabiana**. Internet muda o hábito de ir ao supermercado**. Disponível em: < http://tecnologia.terra.com.br/internet/internet-muda-o-habito-de-ir-ao-supermercado,d1d8887dc5aea310VgnCLD200000bbcceb0aRCRD.html>. Acesso em: 23 abr. 2015.

MCCARTHY-BYRNE, Teresa M.; MENTZER, John T. **Integrating supply chain infrastructure and process to create joint value**. International Journal of Physical Distribution & Logistics Management, v. 41, n. 2, 2011, p. 135-161.

POZZI, Andrea. **E-commerce as a stockpiling technology: Implications for consumer savings**. International Journal of Industrial Organization, v. 31, 2013, p. 677-689.

PRESSMAN, Roger. **Engenharia de software**. Rio de Janeiro: McGraw-Hill, 2008.

STERNBECK, Michael G.; KUHN, Heinrich **Grocery retail operations and automotive logistics: a functional cross-industry comparison**, Benchmarking: an International Journal, v. 21, n. 5, p. 814-834, 2014.