UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

VINICIUS FERRARINI

E-COMMERCE DE PRODUTOS DE INFORMÁTICA E JOGOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

VINICIUS FERRARINI

E-COMMERCE DE PRODUTOS DE INFORMÁTICA E JOGOS

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, campus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

E-COMMERCE DE PRODUTOS DE INFORMÁTICA E JOGOS
por

VINICIUS FERRARINI

Este trabalho de conclusão de curso foi apresentado em 10 de novembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Beatriz Terezinha Borsoi e Vinicius Pegorini. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Andreia Scariot Beulke
Prof. Orientador (UTFPR)

Beatriz Terezinha Borsoi
Banca (UTFPR)

Vinicius Pegorini
Banca (UTFPR)

Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

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

AGRADECIMENTOS

Agradeço primeiramente a Deus por me fornecer foco e força para o desenvolvimento do trabalho do início ao fim.

Agradeço a professora Andreia Scariot Beulke por toda ajuda e atenção prestada, sendo ela essencial para a conclusão do mesmo, muito obrigado professora.

Agradeço a todos os professores que me proporcionaram aprendizado, repassando seus conhecimentos durante toda a especialização.

Agradeço a meus pais, que sempre me apoiaram, ajudaram e incentivaram para iniciar e também concluir a especialização.

Agradeço a minha namorada Alana que esteve presente durante todo o desenvolvimento deste trabalho, pelo apoio e incentivo a concluir o mesmo.

Agradeço também aos meus amigos João Paulo Merlin e Gaspar Barancelli pelas sugestões dadas durante o desenvolvimento do trabalho.

RESUMO

FERRARINI, Vinicius. E-commerce de produtos e informática e jogos. 2017. 59f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

O comércio eletrônico é uma área que se expandiu e ganhou visibilidade no comércio que envolvendo serviços como compra e venda de mercadorias. Isso ocorre pela possibilidade de as empresas divulgarem e comercializarem seus produtos de forma mais abrangente, atingindo uma gama maior de clientes, pois as transações comerciais são realizadas por meio da Internet. Assim, o objetivo desse trabalho foi o desenvolvimento de um *e-commerce* de produtos de hardware, periféricos e software de jogos. O sistema possui a parte administrativa que permite a gestão desses produtos por meio do gerenciamento e da manutenção dos produtos e a parte da loja virtual que é visualizada e utilizada pelo consumidor final para comprar os produtos e realizar pagamentos. O desenvolvimento desse sistema permitiu a integração de diversas tecnologias. Dentre essas tecnologias destacam-se o *framework* Angular para trabalhar com o *front-end*, a linguagem Java *e o framework Spring* para a implementação *back-end*. Como resultado obteve-se um sistema de fácil manutenção com a combinação dessas tecnologias. Além disso, foi utilizado o conceito de microsserviços implementados com o *framework* Spring para receber as requisições via REST do sistema administrativo desenvolvido com a tecnologia Angular e para a estrutura da loja virtual e foram *controllers* das interfaces HTML.

Palavras-chave: E-commerce. Framework Angular. Microsserviço.

ABSTRACT

FERRARINI, Vinicius. E-commerce to sell computer products and games. 2017. 59. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

The electronic commerce is an area that has expanded and gained visibility when it involves services as buying and selling of goods. This process occurs due to the possibility of companies publicize and sell their products reaching a wider range of customers. Thus, the objective of this work was the development of an e-commerce to sell hardware, peripherals and software games. The system has an administrative area that allows the management of the products catalog and the virtual store area that is viewed and used by final consumers to buy products and make a payment. The development of this system allowed the integration of various technologies. Among these technologies the Angular framework was used to develop the front-end, the Java language and the Spring framework were used to build the back-end. As a result we obtained a system of easy maintenance with the combination of these technologies. In addition, it was used the concept of micro services deployed with the Spring framework to receive the requests with REST from the administrative system developed with the technology Angular.

Keywords: E-commerce. Framework Angular. Microservices.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
AOT	*Ahdead of Time*
B2B	*Business to Business*
B2C	*Business to Consumer*
C2C	*Consumer to Consumer*
CEP	Código de Endereçamento Postal
DOM	*Document Object Model*
ES6	ECMAScript 6
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
MVC	*Model, View* e *Controller*
MVW	*Model, View, Whatever*
REST	*Representational State Transfer*
SPA	*Single Page Application*
SQL	*Structured Query Language*
TI	Tecnologia da Informação
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

A *Web* é um ambiente que integra informações que podem ser acessadas por meio de diversas plataformas. A forma padrão dessas informações é o hipertexto que permite a ligação entre diversos documentos. Esses documentos fazem parte das aplicações *web* que podem ser estáticas, dinâmicas, loja *on-line*, portal, aplicativos com gerenciamento de conteúdo, entre outras. Pressman e Lowe (2009) afirmam que a *Web* se tornou um recurso indispensável para diversos segmentos do mercado, como, por exemplo, comércio, negócios, engenharia, educação, saúde, entre outros.

No que tange às lojas virtuais, também chamadas de *e-commerce* ou comércio eletrônico, a Internet se tornou um grande mecanismo de negócios. Isso porque o *e-commerce* pode ser compreendido como uma transação comercial que é realizada com o auxílio de um equipamento eletrônico com acesso à Internet. Turban (*et al.,* 2010) afirmam que um comércio eletrônico pode ser definido como um processo de compra, venda e troca de produtos, serviços ou informações com o auxílio de um computador. Napierala (2016) corrobora com esses autores, afirmando, ainda, que o *e-commerce* compreende transações interorganizacionais, utilizando sistemas de comunicação em rede. Essas transações podem ser realizadas entre empresas e pessoas (*Business to Consumer* - B2C), empresas e empresas (*Business to Business – B2B*) ou entre consumidores finais (*Consumer to Consumer* – C2C) (NAPIERALA, 2016).

De acordo com Napierala (2016), a Tecnologia da Informação (TI), especialmente, os sistemas de *e-commerce* trazem uma série de conceitos relacionados à forma de organização da produção e transformações no ambiente social e produtivo. Para Prado e Souza (2014), as empresas precisam adequar-se às mudanças que a TI proporciona para manterem-se no mercado. Essas mudanças estão relacionadas à configuração da economia e a maneira de as empresas realizarem seus negócios.

Laudon (2004), afirma que para as empresas de pequeno e médio porte, o *e-commerce* é uma oportunidade de equilibrar as chances de entrada no mercado, pois pode inicializar as vendas com custos mais baixos e viabilizar o contato *on-line* com os consumidores. Napierala (2016) atesta que o comércio tradicional, muitas vezes, acaba bloqueando a entrada de pequenas e médias empresas porque possuem suas próprias redes de comércio, canais de distribuição e o amplo mercado consumidor. Assim, o *e-commerce* proporciona maior inserção dessas empresas, pois podem ser vendidos produtos de diversos produtores e podem oferecer um conjunto mais amplo de produtos e serviços.

Nesse cenário, este trabalho tem por objetivo desenvolver um *e-commerce* de produtos de hardware, periféricos e software de jogos. O *e-commerce* proposto permitirá a gestão desses produtos garantindo o controle de todo o processo que vai desde o cadastro de um novo produto, até a venda ao consumidor final.

2 MATERIAIS E PROCEDIMENTOS

Esse capítulo tem por objetivo elencar os materiais utilizados na modelagem e na implementação do aplicativo para o desenvolvimento do trabalho proposto.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo.

Ferramenta / Tecnologia	**Versão**	**Disponível em**	**Aplicação**
Angular	4.0	https://angular.io/	*Framework* TypeScript
Angular CLI	1.4.9	https://cli.angular.io/	*Framework* TypeScript
PrimeNG	4.1.0	https://www.primefaces.org/primeng/#/	*Framework* de componentes
Astah Community	7.2.0	http://astah.net/editions/community	Modelagem do sistema: desenvolvimento do diagrama de casos de uso
Bootstrap	3.3.7	https://getbootstrap.com/docs/3.3/	*Framework front-end*
Intellij IDEA	2017.2.5	https://www.jetbrains.com/idea/	Ambiente de desenvolvimento para o sistema
WebStorm	2017.2.5	https://www.jetbrains.com/*web*storm/	Ambiente de desenvolvimento para o sistema em angular
Apache Freemarker	1.5.7	http://freemarker.org/	Modelo de *templates* para renderização de texto e HTML em desenvolvimento *web* em Java
Java	8	http://www.java.com/pt_BR/download/win10.jsp	Linguagem de programação
MySQL	5.7	https://www.mysql.com/	Banco de dados
MySQL Workbench	6.3	https://www.mysql.com/products/workbench/	Ferramenta para administração do banco de dados.
Spring Boot	1.5.4	https://projects.spring.io/spring-boot/	Uso de convenções sobre configurações.
Spring Data		https://projects.spring.io/spring-data-jpa/	*Framework* de implementação de repositórios

	2.0.0		
Spring Security	4.2.3	https://projects.spring.io/spring-security/	*Framework* de autenticação e segurança da aplicação.
OAuth2	2.0	https://oauth.net/2/	Protocolo de autorização para APIs *web*.

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

2.1.1 Angular

Angular é um *framework* TypeScript mantido e atualizado pela Google. Seu objetivo é simplificar o desenvolvimento de aplicações *web*. Para Pereira (2014), o Angular apresenta características relacionadas à produtividade, desempenho, fácil customização, implementa o conceito de diretivas e *Two-way Data Binding*, suporta o desenvolvimento de módulos e tem fácil integração com diversos *frameworks* e ferramentas JavaScript.

A versão 2.0 foi totalmente reescrita e remodelada com relação à versão AngularJS (ou 1.0), pois foram inseridos vários conceitos de desenvolvimento. Guedes (2017) afirma que a principal mudança com relação ao desempenho, pois o AngularJS é muito mais lento em relação a outros *frameworks front-end*. A versão 2.0 deixa a resposta mais rápida e a usabilidade mais dinâmica. Além disso, o Angular 2.0 apresenta a linguagem *TypeScript* e implementa funcionalidades do ECMAScript 6 (ES6) com tipagens nas variáveis e uma sintaxe mais clara, aproximando-se das linguagens C# e Java.

Ao discorrer sobre as versões do Angular, Guedes (2017) explica que desde o lançamento da versão 2, muitas outras versões surgiram, como, por exemplo, o Angular 2.1 que tinha melhorias nos roteamentos, a versão 2.2 que deixou o sistema compatível com a *Ahdead of Time* (AOT) e a versão 2.3 que apresentou melhorias nas mensagens de erros e avisos. Guedes (2017), comenta que a mudança de número da versão 2 para a versão 4 deve-se à atualização do TypeScript, pois a versão 2 do Angular suporta a versão 1.8 do TypeScript e, também, porque todos os pacotes do *core* se encontravam na versão 2.4, mas somente o pacote de roteamento estava na versão 3.4. Sendo assim, não poderia ter uma versão 2 usando pacotes de versão 3.4. Portanto, a versão 3.0 não chegou a ser implantada devido ao número de atualizações correlacionadas às rotas da versão 4. Além disso, também, foram criadas novas *features* e alterada a forma de compilação para a que versão final do projeto desenvolvido seja menor e mais leve.

Angular utiliza o conceito de componentes. Conforme explica Guedes (2017), o conjunto de um *template* e uma classe forma um componente. Um *template* é criado com trechos de código *Hypertext Markup Language* (HTML) com *tags* customizadas e classes que são marcadas com *@component* para gerenciar o conteúdo que será exibido no *template*.

Angular utiliza o padrão de desenvolvimento *Single Page Application* (SPA) que são aplicações desenvolvidas em JavaScript e executam do lado cliente em forma de *template* (GUEDES, 2017). Em SPA, a página principal é carregada apenas uma vez e possui recursos para carregar, dinamicamente, conteúdos requisitados por meio de *Uniform Resource Locator* (URL) (ALMEIDA, 2015). Conforme explica Almeida (2015), uma aplicação que utiliza o padrão SPA proporciona melhor experiência para o usuário, apresenta melhor desempenho e diminui a carga de trabalho no servidor. Isso porque, basicamente, a interação ocorre no cliente e o servidor é encarregado de fazer a comunicação com o banco de dados que, normalmente, traz esses dados para o cliente no formato *JavaScript Object Notation* (JSON).

Outra característica do Angular é o padrão *Model*, *View* e *Controller* (MVC) que permite a distribuição da aplicação em camadas. Pereira (2014) afirma que o Angular é conhecido como um *framework Model, View, Whatever* (MVW), ou seja, representa as camadas de modelo, visão e qualquer coisa que seja necessária, pois *model* e *view* podem ser sincronizadas sem a necessidade obrigatória de um *controller*. Assim, é importante que as regras de negócio não sejam de responsabilidade do cliente, devendo ser implementadas no *back-end*, sendo possível distribui-las para qualquer aplicação *front-end* (PEREIRA, 2014).

Guedes (2017) afirma que o Angular trabalha com o conceito de diretivas que são uma forma de interação com o *template*, ampliando a capacidade do HTML, pois dependendo de qual for a regra de negócio, as diretivas modificam ou interagem com o HTML. Para Gudes (2017) tudo o que modifica a estrutura do *Document Object Model* (DOM) pode ser considerado uma diretiva.

Para enviar ou sincronizar os dados de uma classe de componente e o *template* o Angular usa o conceito de *data binding*, sendo classificados em quatro tipos (GUEDES, 2017):

a) *Interpolation*: usado quando há a necessidade de passar dados que estão na classe do componente para o *template*. Isso é realizado por meio de chaves duplas e dentro delas há a variável que está na classe do componente.

b) *Property binding*: usado quando há a necessidade de passar informações da classe do componente para alguma propriedade de *tag* do *template*.

c) *Event binding*: quando os dados são passados do *template* para a classe do componente. É utilizando quando há interação com o usuário e a cada ação executada, o evento atualiza ou manipula algo dentro da classe do componente.

d) *Two-way data binding*: é uma junção do *property binding* com o evento *binding*, ou seja, atualiza o *template* e a classe do componente quando a variável declarada for alterada.

Além disso, o Angular permite trabalhar com serviços e injetor de dependências. Um serviço pode ser configurado de forma global ou dentro do componente. Guedes (2017) explica que em uma aplicação desenvolvida em Angular há duas árvores hierárquicas, uma é a de componentes e a outra de injetores. A árvore hierárquica serve para mostrar qual é o componente raiz e seus dependentes e, nesse caso, utiliza o conceito de herança. Para tanto, é necessário configurar uma classe de serviço no componente pai, sendo possível usar um serviço dentro do componente e, se o injetor de serviço for configurado no componente raiz, todos os dependentes terão acesso a esse serviço.

O Angular possui outras características que não serão todas abordadas nesse item, pois apresenta um vasto conjunto de itens e conceitos. Antes de iniciar uma aplicação desenvolvida com essa tecnologia, é importante compreender como funciona a arquitetura do sistema e componentes no Angular.

2.1 PROCEDIMENTOS E CONFIGURAÇÕES

A aplicação Angular foi desenvolvida a partir do *framework* Angular CLI, que facilita e torna o desenvolvimento mais rápido. Na Listagem 1 é possível visualizar como o desenvolvimento do projeto foi iniciado utilizando o Angular CLI. Os comandos foram executados por meio do terminal de comandos do sistema operacional em uso.

```
- npm install -g @angular/cli
- ng new nome-do-projeto
```

Listagem 1 - Criar projeto utilizando Angular CLI

Após criar o projeto foi necessário acessar a sua pasta por meio do terminal de comando do sistema operacional. O Angular CLI cria uma estrutura de projeto pronto para ser executado por meio do comando exibido na Listagem 2.

```
- ng serve
```

Listagem 2 - Comando para subir a aplicação utilizando Angular CLI.

O Angular CLI fornece diversos comandos para facilitar a criação de classes, componentes, módulos entre outros, que podem ser visualizados na Listagem 3.

Arquivo	Comando
Component	ng g component nome-do-componente
Directive	ng g directive nome-da-diretiva
Pipe	ng g pipe nome-do-pipe
Service	ng g service nome-do-service
Class	ng g class nome-da-classe
Guard	ng g guard nome-do-guard
Interface	ng g interface nome-da-interface
Enum	ng g enum nome-do-enum
Module	ng g module nome-do-modulo

Listagem 3 - Arquivos e comandos disponíveis utilizando Angular CLI

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema desenvolvido.

3.1 ESCOPO DO SISTEMA

O sistema desenvolvido, como resultado deste trabalho, tem por objetivo permitir a comercialização e a gestão de um *e-commerce* de produtos de hardware, periféricos e software da linha *gamer* (jogos), garantindo o controle de todo o processo, desde o cadastro de um novo produto, até a venda ao consumidor final. O sistema permite que um produto adquirido seja cadastrado, que o estoque seja atualizado e informações sejam inseridas e atualizadas, caso o produto já tenha sido cadastrado.

O *e-commerce* está organizado em departamentos, para permitir que o consumidor possa encontrar os produtos com mais facilidade. O consumidor poderá manter seus produtos em um carrinho de compras, que será armazenado em quanto o navegador estiver aberto, podendo sair do sistema e posteriormente retornar para finalizar a sua compra.

Ao concluir seu pedido, o cliente deverá logar-se preenchendo os campos de usuário e senha para realizar o pagamento. Em seu cadastro, o cliente também poderá atualizar informações pessoais, acompanhar a situação de seus pedidos e manter seu histórico de pedidos. Caso o consumidor ainda não possua cadastro, ele deverá realizá-lo, informando seus dados pessoais e de endereço de entrega. Após realizar *login* ou cadastro, o cliente é direcionado para a interface de pagamento, na qual poderá finalizar sua compra realizando a simulação de pagamento por meio de boleto bancário ou cartão de crédito.

O administrador poderá lançar cada etapa dos pedidos em aberto, separação no estoque, envio a transportadora e entrega finalizada.

3.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a descrição dos requisitos funcionais e o Quadro 3 os requisitos não funcionais do sistema.

Identificação	Nome	Descrição

RF 01	Manter usuário	O cadastro dos usuários do sistema, contendo os dados necessários para identificação, *login* e a definição das prioridades de acesso. Basicamente, os usuários do sistema são: administrador, cliente e sistema.
RF 02	Manter categoria	As categorias referem-se ao agrupamento dos produtos por categoria.
RF 03	Manter subcategoria	As subcategorias referem-se ao agrupamento dos produtos por categoria e também por sub categoria.
RF 04	Manter produto	Realizar o cadastro de produtos que deve estar associado a uma categoria.
RF 05	Manter cliente	Para efetuar uma compra e interagir com o carrinho o cliente realiza seu cadastro e mantém seus dados atualizados.
RF 06	Manter carrinho	O cliente poderá gerenciar seu carrinho de compras por meio das operações de inclusão, exclusão, alteração e consulta dos itens do carrinho.
RF 07	Manter endereços	Ao finalizar a compra o cliente poderá selecionar um endereço já cadastrado ou incluir um novo endereço para entrega do pedido. Além disso, poderá optar por retirar o produto no local.
RF 08	Realizar pagamento	O usuário poderá escolher a forma de pagamento.
RF 09	Emitir relatórios	Permite a emissão de relatórios de vendas, produtos e clientes.
RF 10	Gerenciar pedidos	Permite o cancelamento de pedidos, visualização de histórico de pedidos, alteração de *status* do pedido, consultar o *status* do pedido e visualizar o pedido.
RF 11	Finalizar compra	Uma compra será finalizada após a seleção do endereço de entrega e a forma de pagamento.
RF 12	Consultar frete	O cliente poderá consultar o valor do frete mediante número do Código de Endereçamento Postal (CEP) para entrega. O sistema deve permitir que possam ser realizadas consultas de valor de frete no *web-service* dos correios.
RF 13	Recuperar senha	O cliente poderá recuperar sua senha caso tenha esquecido a mesma.
RF 14	Manter galeria de fotos	O administrador poderá cadastrar uma ou mais fotos para cada produto.

Quadro 2 – Requisitos Funcionais

Identificação	Nome	Descrição
RNF 01	Acesso ao sistema	O acesso ao sistema será realizado por meio de *login* e senha.
RNF 02	Restrições de formato	Os campos de *e-mail* e senha devem ser validados de acordo com as restrições de formato.
RNF 03	Campos de verificação de	Os campos para redigitar *e-mail* e senha devem ser idênticos aos valores preenchidos nos campos *e-mail*

	dados	e senha.
RNF 04	Campos de preenchimento obrigatórios	Os campos que são de preenchimento obrigatório serão validados por meio de uma função do sistema.
RNF 05	Campos com máscaras de entrada	Os campos que possuem caracteres especiais e, que não serão armazenados no banco de dados, como, por exemplo, “()”, “.”, “-”, para os campos RG, CPF, telefone, CEP, entre outros, serão validados por meio de máscaras de entrada.
RNF 06	Finalizar pedido	O cliente poderá finalizar o pedido somente se selecionar o endereço de entrega e a forma de pagamento.
RNF 07	Gerenciar pagamento	O usuário poderá escolher a forma de pagamento no cartão de crédito ou boleto bancário.
RNF 08	Vínculo	Dados relacionados devem estar vinculados. Por exemplo, uma categoria deve estar vinculada a um departamento e um produto deve estar vinculado a uma categoria.

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: administrador, clientes e sistema. O administrador é responsável pelos cadastros de produtos, departamentos, categorias de produtos e emissão de relatórios. O cliente é quem realizada as compras por meio do sistema. Para isso, o cliente precisa se cadastrar informando um *login* e senha para acesso sua conta, manter seu carrinho (adicionando ou removendo itens) e efetuar o pagamento. O cliente também poderá realizar operações de consulta dos pedidos. O sistema é responsável por enviar *e-mail* para o usuário informando o *status* da compra.

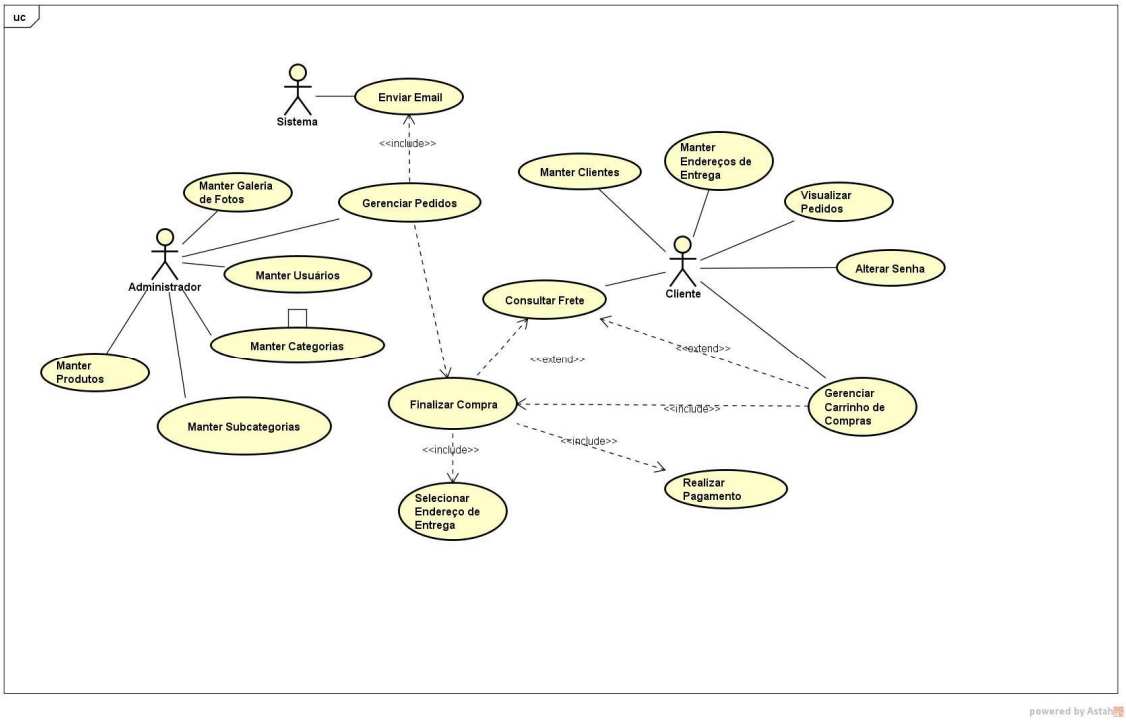

Figura 1 – Diagrama de casos de uso

Os Quadros 4 a 7 apresentam a descrição dos casos de uso que envolvem as operações de inclusão, alteração, exclusão e consulta. Esses casos de usos identificados como "manter" no diagrama de casos de uso apresentado na Figura 1.

Caso de uso: Incluir (refere-se à operação de inclusão de todos os casos de usos identificados como "manter"). **Descrição:** Inclusão dos dados cadastrais no sistema. **Evento Iniciador:** Ator solicita inclusão de um registro no sistema. **Atores:** Administrador ou cliente, de acordo com suas funções definidas no caso de uso. **Pré-condição:** Não há. **Sequência de Eventos:** 1. Ator acessa a tela para realizar o cadastro inserindo as informações necessárias. 2. O sistema insere as informações no banco de dados e informa ao usuário o *status* do procedimento. **Pós-Condição:** Registro inserido no banco de dados.	
Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. O Administrador ou Cliente deixa de informar os dados obrigatórios. 1.2. O Sistema desabilita o botão de salvar até que todos os dados obrigatórios sejam informados. 1.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.
2. Campos informados em formato incorreto.	2.1. O cliente informa dados em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.

Quadro 4 - Operação de incluir dos casos de uso de cadastro

Caso de uso:
Alterar (refere-se à operação de alteração de todos os casos de usos identificados como “manter”).
Descrição:
Alteração dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita alteração de um registro no sistema.
Atores:
Administrador ou Cliente de acordo com suas funções definidas no caso de uso.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O sistema apresenta o registro selecionado para alteração.
3. Usuário altera os dados do registro.
4. O sistema altera as informações no banco de dados e informa ao usuário o *status* do procedimento.

Pós-Condição:
Registro alterado no banco de dados.

Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. O Administrador ou Cliente exclui dados obrigatórios. 1.2. Sistema desabilita botão de salvar até que todos os dados obrigatórios sejam informados. 1.3. O sistema permanece na tela de edição mantendo as alterações realizadas.
2. Campos informados em formato incorreto.	2.1. O cliente altera os dados deixando em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de edição mantendo as alterações realizadas.

Quadro 5 - Operação de alterar dos casos de uso de cadastro

Caso de uso:
Excluir (refere-se à operação de exclusão de todos os casos de usos identificados como “manter”).
Descrição:
Exclusão dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita exclusão de um registro no sistema.
Atores:
Administrador ou Cliente de acordo com suas funções definidas no caso de uso.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para exclusão do registro.
2. O sistema exclui as informações no banco de dados e informa ao usuário o *status* do procedimento.

Pós-Condição:
Registro excluído no banco de dados.
Extensões:
Registro possuir vínculo com outros cadastros

Nome do fluxo alternativo (extensão)	Descrição
1. Exclusão de registro com vínculos no sistema	1.1. Cliente clica em excluir um registro que possui vínculos no sistema. 1.2. O sistema verifica que o registro tem vínculos, não exclui o respectivo registro e exibe mensagem de alerta ao usuário.

Quadro 6 - Operação de excluir dos casos de uso de cadastro

Caso de uso:
Consultar (refere-se à operação de consulta de todos os casos de usos identificados como “manter”).
Descrição:
Consulta dos dados cadastrais dos registros do sistema.
Evento Iniciador:
Ator solicita consulta de um registro no sistema.
Atores:
Administrador ou Cliente, de acordo com suas funções definidas no caso de uso.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O ator indica os filtros desejados para consulta.
3. O sistema apresenta os dados da consulta ao usuário.

Pós-Condição:
Dados da consulta apresentados ao usuário.

Quadro 7 - Operação de buscar dos casos de uso de cadastro

O Quadro 8 representa o caso de uso “visualizar pedidos”.

Caso de uso:
Visualizar pedidos
Descrição:
Visualização do histórico de pedidos realizados na loja, os produtos, valores e endereço de entrega do mesmo.
Evento Iniciador:
Cliente acessa a interface de pedidos.
Atores:
Cliente.
Pré-condição:
Não há.
Sequência de Eventos:
1. Cliente realiza *login* no sistema e acessa sua área restrita
2. Cliente clica no botão meus pedidos.

Pós-Condição:
1. Histórico de pedidos é exibido ao cliente

Quadro 8 - Expansão do caso de uso para visualizar pedidos

O Quadro 9 representa a expansão do caso de uso “manter carrinho”.

Caso de uso:
Manter carrinho
Descrição:
Gerenciamento do carrinho de compras, remoção de itens, alteração de quantidade e consulta de frete.
Evento Iniciador:
Cliente remove ou altera a quantidade dos produtos no carrinho de compras.
Atores:
Cliente.
Pré-condição:
Possuir produtos adicionados no carrinho de compras.
Sequência de Eventos:
1. Cliente acessa o seu carrinho de compras
2. Cliente altera a quantidade ou remove o produto do carrinho de compras.

Pós-Condição:
1. Informações do carrinho salvas.

Extensões
1. Cliente poderá consultar o frete referente à entrega do seu pedido

Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não	1.1. O Cliente deixa de informar a quantidade de produtos.

informados.	1.2. O Sistema informa a quantidade mínima de 1 unidade automaticamente.

Quadro 9 - Expansão do caso de uso para manter carrinho de compras

O Quadro 10 representa a expansão do caso de uso “selecionar endereço de entrega”.

Caso de uso: Selecionar endereço de entrega **Descrição:** Processo de seleção do endereço de entrega. **Evento Iniciador:** Cliente seleciona o endereço onde deseja que seu pedido seja entregue. **Atores:** Cliente. **Pré-condição:** Finalizar o carrinho de compras **Sequência de Eventos:** 1. Cliente acessa a interface de seleção de endereço de entrega 2. Cliente seleciona o endereço onde deseja que seja entregue seu pedido. **Pós-Condição:** 1. Endereço de entrega vinculado ao pedido 2. Interface de realização de pagamento disponível	
Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. O Cliente deixa de selecionar o endereço de entrega. 1.2. O Sistema desabilita o botão de prosseguir da interface de pagamento até que o cliente informe o endereço de entrega.

Quadro 10 - Expansão do caso de uso para selecionar endereço de entrega

O Quadro 11 representa a expansão do caso de uso “realizar pagamento”.

Caso de uso: Realizar pagamento **Descrição:** Processo de realização de pagamento. **Evento Iniciador:** Cliente seleciona a forma de pagamento dentre as disponíveis pela aplicação. **Atores:** Cliente. **Pré-condição:** Finalizar o carrinho de compras e selecionar o endereço de entrega. **Sequência de Eventos:** 1. Cliente acessa a interface de pagamento. 2. Cliente seleciona a forma de pagamento. 3. Cliente informa os dados do cartão de crédito ou imprime o boleto. 4. Cliente clica no botão realizar pagamento. **Pós-Condição:** 1. Interface de finalização de compra é exibida.	
Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. O Cliente deixa de selecionar a forma de pagamento. 1.2. O Sistema desabilita o botão realizar pagamento até que os dados obrigatórios de pagamento sejam informados.
2. Campos informados em formato incorreto.	2.1. Retorno da instituição financeira informando que o pagamento não foi realizado.

Quadro 11 - Expansão do caso de uso para realizar pagamento

O Quadro 12 representa a expansão do caso de uso “finalizar compra”.

Caso de uso:
Finalizar compra
Descrição:
Processo de finalização de compra.
Evento Iniciador:
Cliente verifica os dados de seu pedido como, produtos, quantidade, valores, endereço de entrega e dados de pagamento.
Atores:
Cliente.
Pré-condição:
Finalizar o carrinho de compras e selecionar o endereço de entrega e realizar pagamento.
Sequência de Eventos:
1. Cliente acessa a interface de finalização de compra.
2. Cliente visualiza as informações do seu pedido.
3. Cliente clica no botão finalizar compra.

Pós-Condição:
1. E-mail com todas as informações do pedido é encaminhado ao cliente e administrador.
2. Exibir a interface com os pedidos do cliente.

Inclusões:
1. Cliente terá que finalizar seu carrinho de compras
2. Cliente terá que selecionar o endereço de entrega
3. Cliente terá que realizar o pagamento

Quadro 12 - Expansão do caso de uso para finalizar compra

O Quando 13 representa a expansão do caso de uso “gerenciar pedidos”

Caso de uso:
Gerenciar pedidos
Descrição:
Processo de gerenciamento dos pedidos realizados.
Evento Iniciador:
Pedido necessita de alteração de status.
Atores:
Administrador.
Pré-condição:
Não há
Sequência de Eventos:
1. Administrador acessa a interface de pedidos.
2. Administrador clica no botão alterar no pedido que deseja atualizar.
3. Administrador altera os dados necessários e clica no botão salvar.

Pós-Condição:
1. Mensagem de sucesso ou erro é retornada pelo sistema.

Inclusões:
1. E-mail é enviado ao cliente quando o *status* do pedido é alterado.

Nome do fluxo alternativo (extensão)	Descrição
1. Campos obrigatórios não informados.	1.1. Administrador deixa de informar dados obrigatórios. 1.2. Sistema desabilita o botão de salvar até que os dados obrigatórios sejam informados.
2. Campos informados em formato incorreto.	2.1. O administrador informa dados em um formato incorreto e clica em salvar. 2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro. 2.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.

Quadro 13 - Expansão do caso de uso para gerenciar pedidos

O Quando 14 representa a expansão do caso de uso “recuperar senha”

<table>
<tr><td colspan="2">Caso de uso:
Recuperar Senha
Descrição:
Processo de recuperação de senha do usuário.
Evento Iniciador:
Usuário esqueceu sua senha e precisa acessar o sistema
Atores:
Administrador e Cliente
Pré-condição:
Não há
Sequência de Eventos:
1. Usuário acessa interface de login
2. Usuário clica no botão “esqueceu sua senha?”
3. Usuário informa seu e-mail de cadastro e clica no botão “enviar”
Pós-Condição:
2. Mensagem de sucesso ou erro, é retornada pelo sistema.
Inclusões:
4. E-mail é enviado ao cliente com as instruções para recuperação de senha</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não informados.</td><td>1.1. Usuário não informa o e-mail.
1.2. Sistema informa ao usuário que campo de e-mail é obrigatório</td></tr>
<tr><td>2. Campos informados em formato incorreto.</td><td>2.1. Usuário informa um e-mail inválido ou que não está cadastrado.
2.2. Sistema retorna uma mensagem informando ao usuário que o e-mail é inválido.</td></tr>
</table>

Quadro 14 - Expansão do caso de uso recuperar senha

3.2.1 Diagrama de Entidade e Relacionamento (DER)

Na Figura 2 são exibidos os relacionamentos entre as entidades do sistema, os tipos e os tamanhos de cada atributo. Dentre todas as tabelas, as entidades Product, Buy e User são as mais importantes possuem a maioria dos campos e relações. A entidade *Product* é a que possui mais relacionamentos, devido as suas características, divisões em categoria, subcategoria, marca, modelo e galeria de fotos. A entidade *Buy* é responsável pelo armazenamento de todas as compras feitas na loja. As entidades relacionadas a ela contêm os dados referentes a pagamento, endereço de entrega, informações de frete e aos produtos inclusos na compra. A entidade *User*, armazena as informações dos clientes e usuários do sistema, seus endereços de entrega cadastrados e *token*s para alteração de senha.

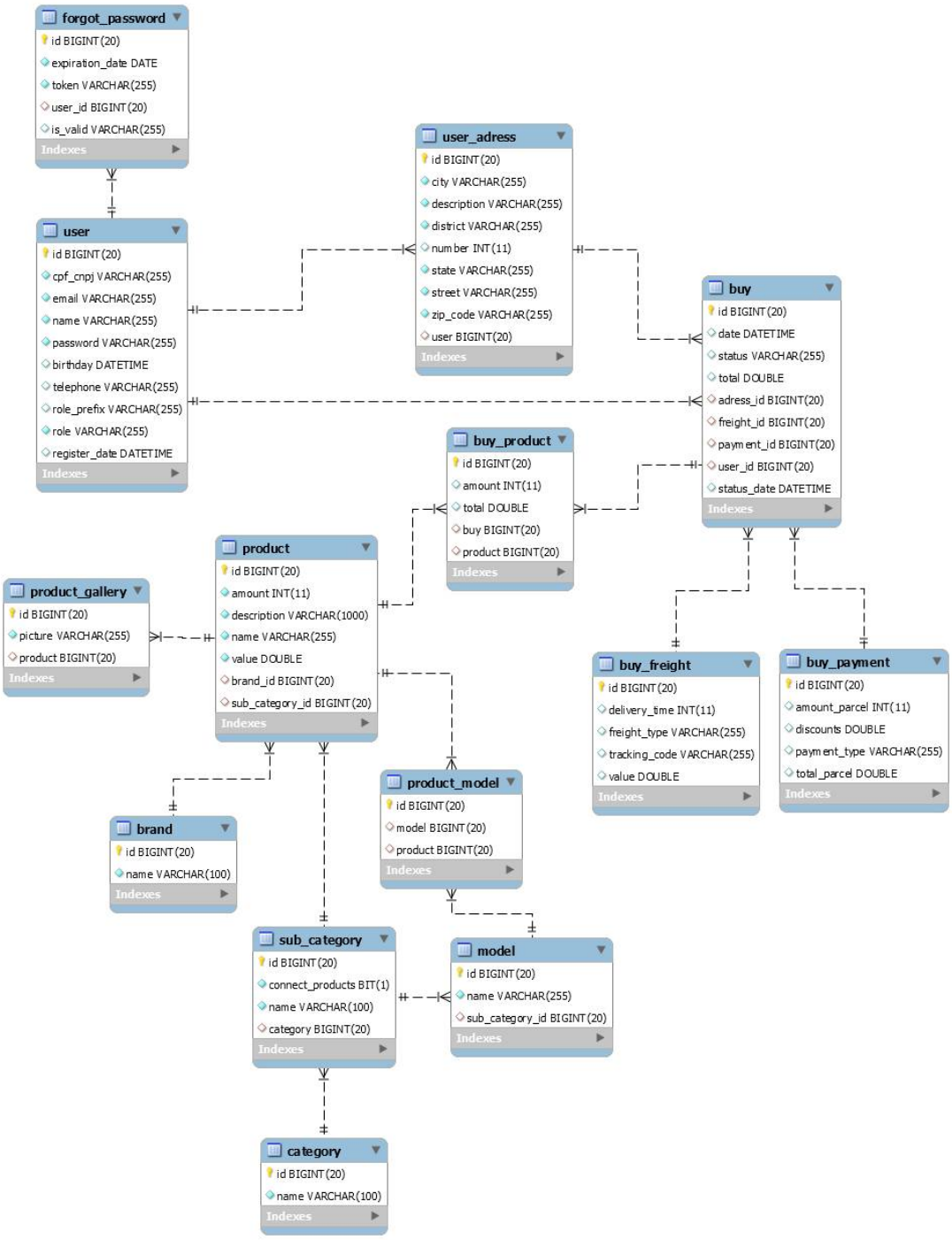

Figura 2 - Diagrama de entidade e relacionamento

3.4 APRESENTAÇÃO DO SISTEMA

Essa seção apresenta a descrição das principais funcionalidades que poderão ser visualizadas por meio de *prints* das telas do sistema. Nas Figuras de 3 a 16 é apresentado o sistema administrativo que é responsável por permitir os cadastros dos conteúdos (produtos,

categorias, usuários, fotos e pagamento) e gerenciamento das compras realizadas pelos consumidores.

A Figura 3 exibe a tela de login do sistema administrativo.

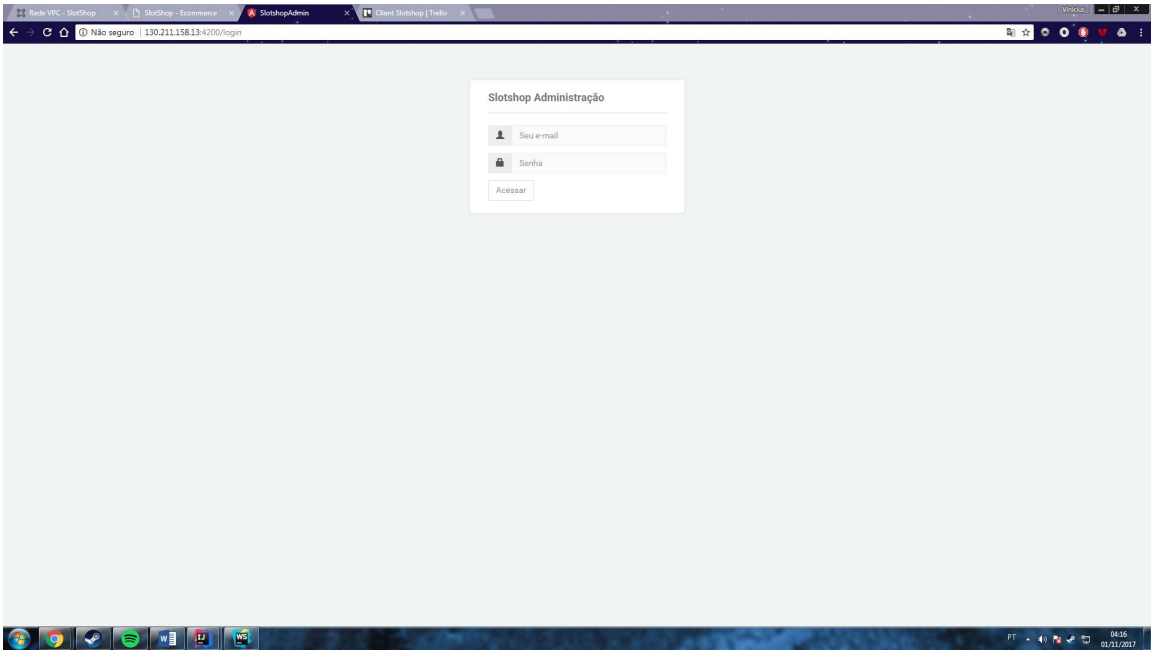

Figura 3 - Interface de login do sistema administrativo

Após realizar o *login* o usuário será redirecionado para a página inicial do sistema administrativo, que exibe um *dashboard*, com as informações das últimas compras realizadas na loja, novos usuários e valores faturados em vendas, como mostra a Figura 4.

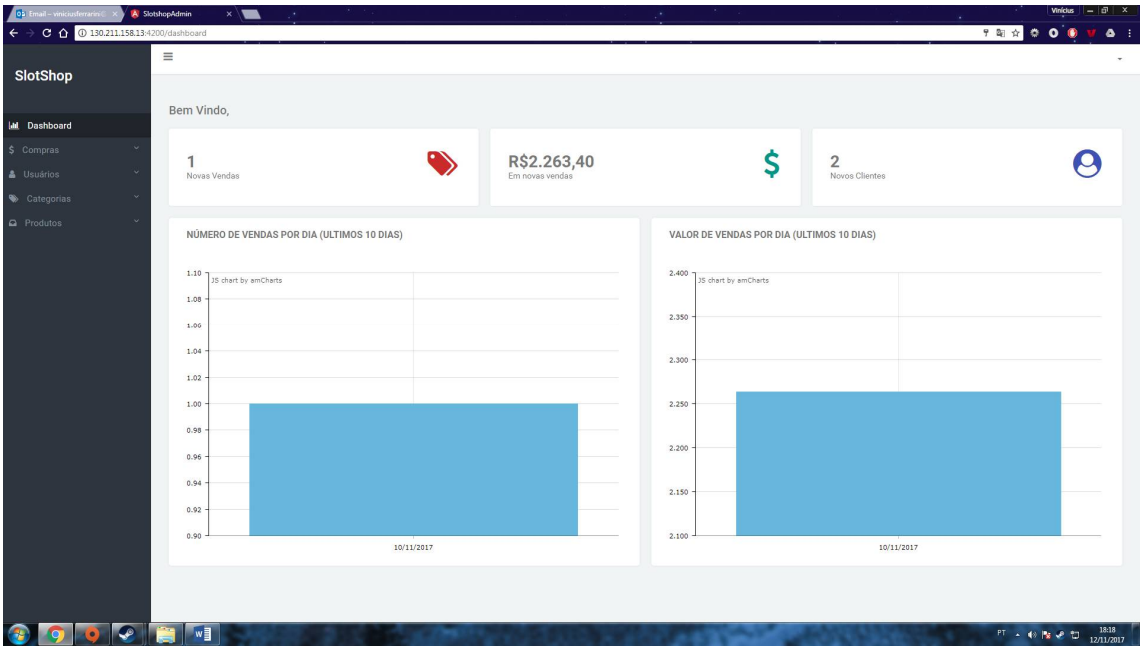

Figura 4 - Interface Dashboard

O menu responsável pela navegação entre os cadastros do sistema administrativo se encontra na lateral esquerda (Figura 5). Por meio desse menu é possível acessar a interface de categorias. As categorias e subcategorias são responsáveis pela organização dos produtos, como mostra a Figura 5.

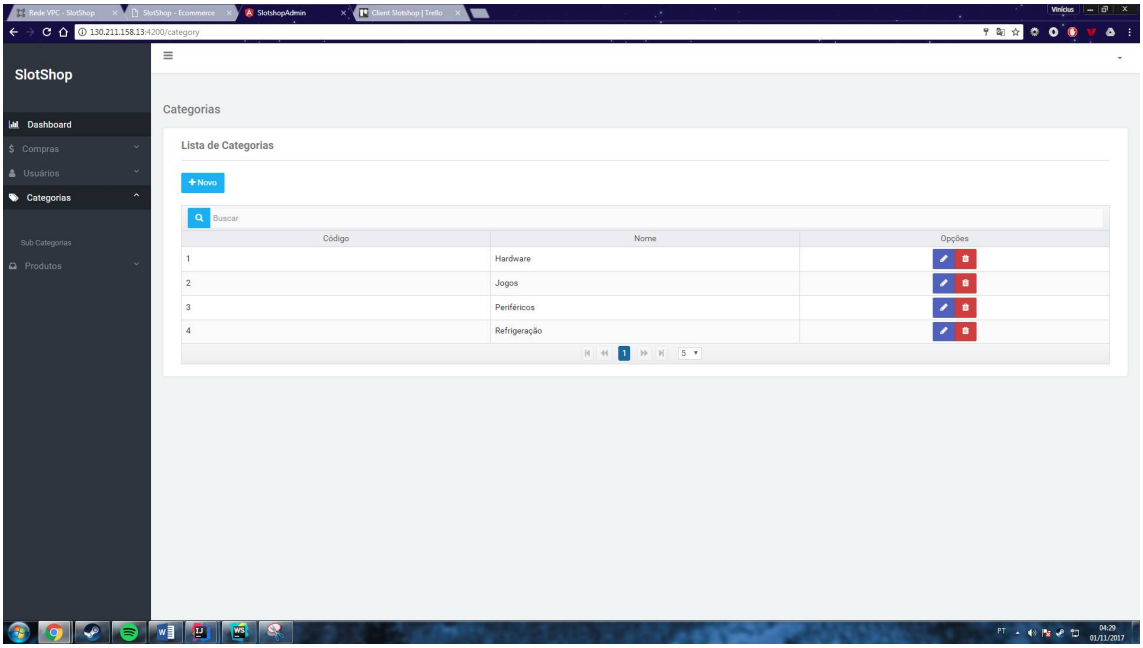

Figura 5 - Interface de listagem de categorias

Ao clicar no botão “Novo” é exibida uma janela *modal* (Figura 6), para incluir novos registros. Na coluna de opções há dois ícones que representam, respectivamente, as operações de editar e excluir os registros no banco de dados. As telas que permitem as operações de incluir, alterar e excluir são padronizadas para todos os registros.

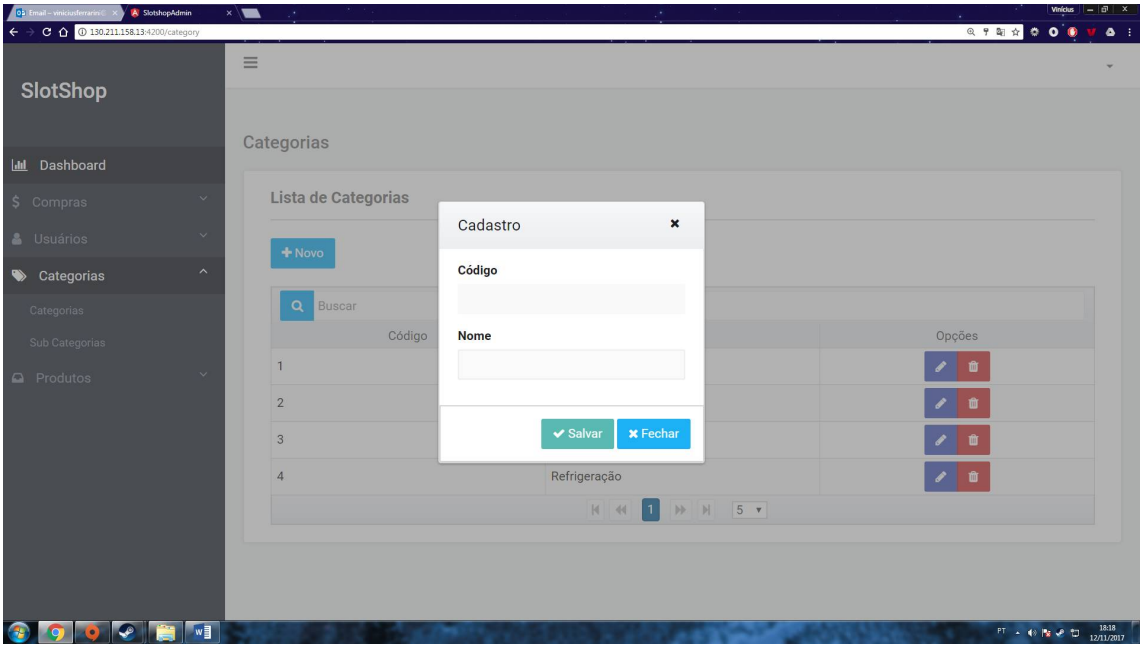

Figura 6 - Interface de cadastro e alteração de categorias

Após preencher ou alterar as informações do registro, o usuário poderá inclui-lo ou atualizá-lo clicando no botão Salvar. Ao clicar no botão Fechar a interface de cadastro será fechada sem salvar os dados informados na base de dados. Essas funcionalidades são padronizadas para todos os cadastros.

No menu categorias, é possível acessar o cadastro das subcategorias, como mostra a Figura 7.

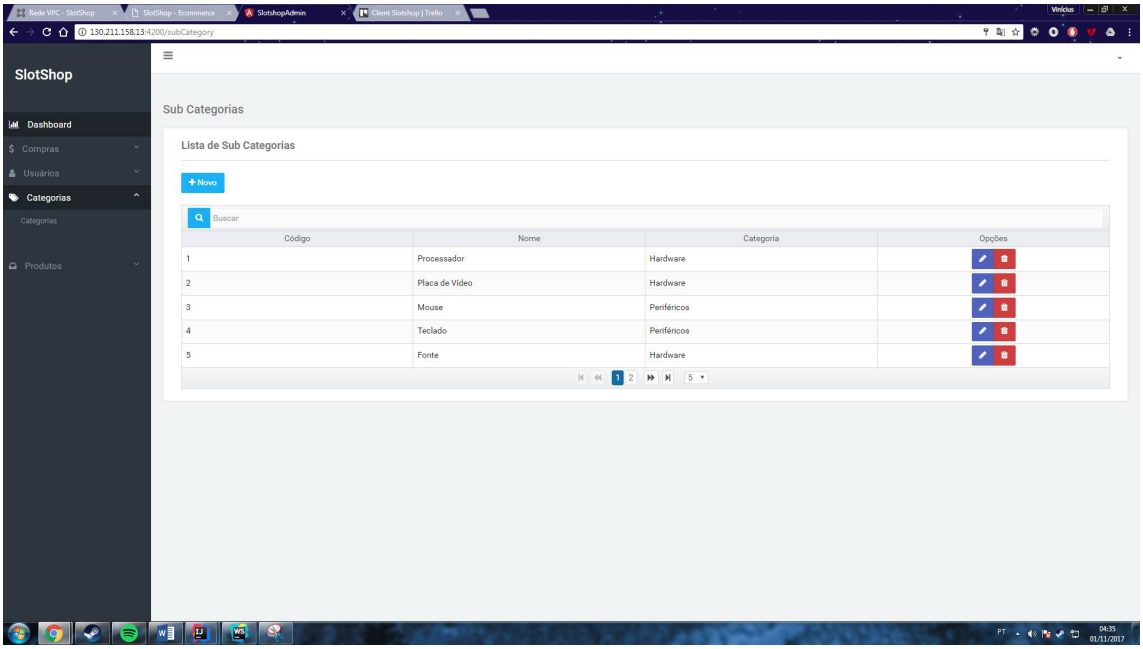

Figura 7 - Interface de listagem de sub categorias

A Figura 8 exibe a tela de inclusão de uma nova subcategoria.

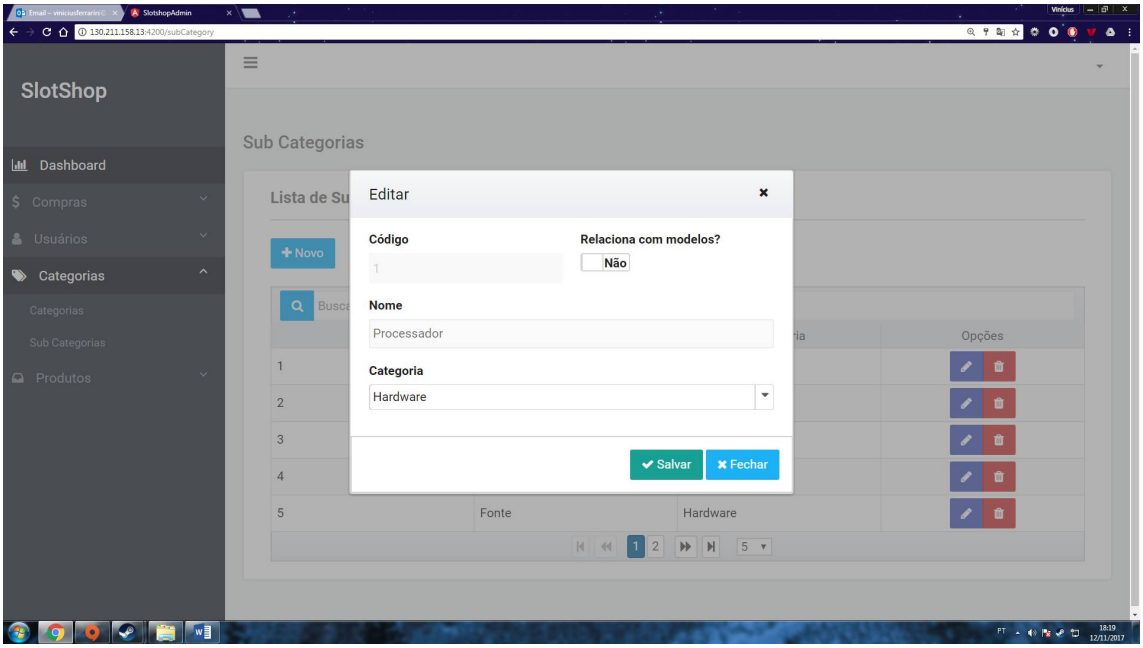

Figura 8 - Interface de cadastro e alteração de sub categorias

O menu produtos possui três submenus: produtos, marcas e modelos. A opção de cadastrar as marcas é responsável pela manutenção das marcas dos produtos, como mostra a Figura 9.

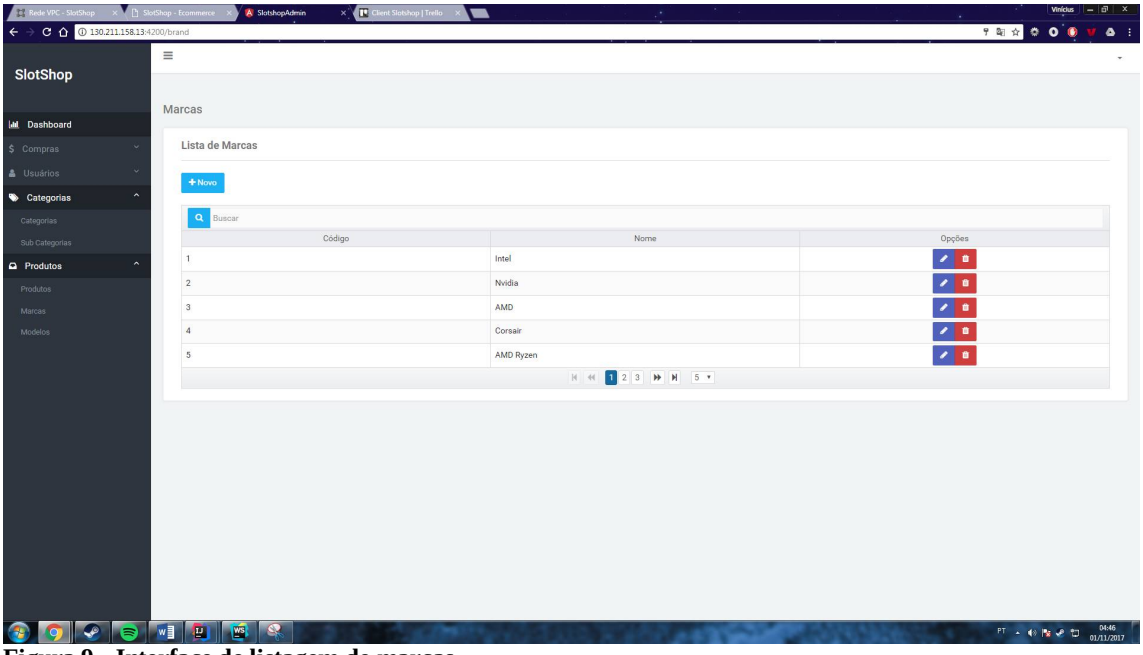

Figura 9 - Interface de listagem de marcas

O submenu produtos exibe a interface de manutenção de produtos, como mostra a Figura 10.

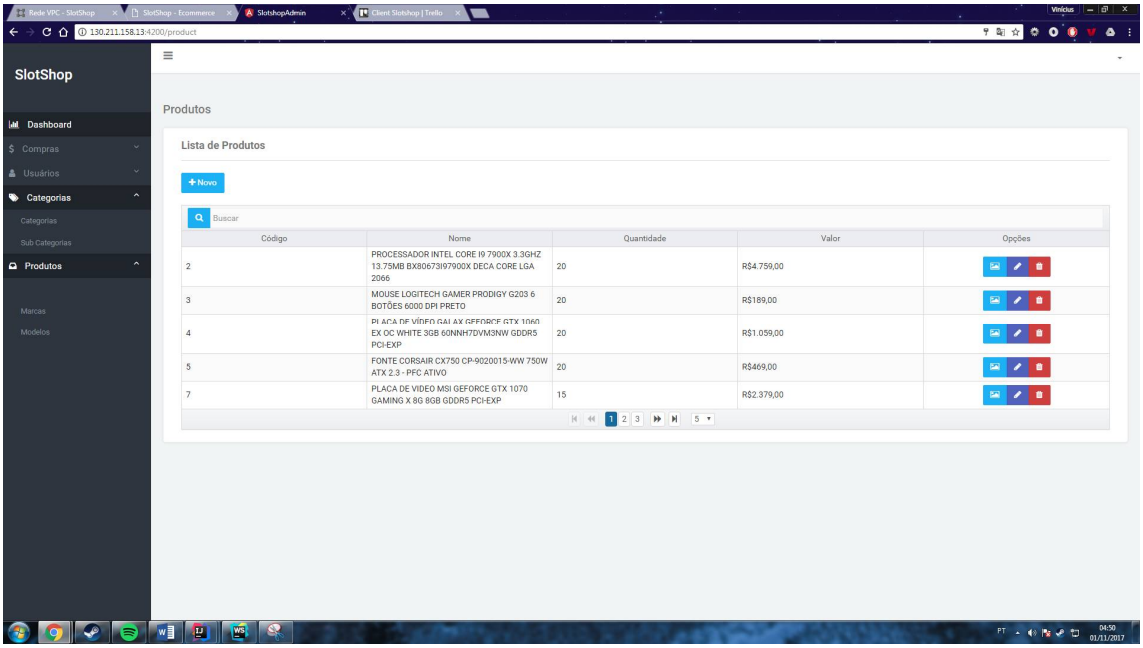

Figura 10 - Interface de listagem de produtos

A Figura 11 exibe a tela de inserção de um novo produto. Os campos de marca e subcategoria exibem os dados inseridos no banco de dados nas respectivas tabelas.

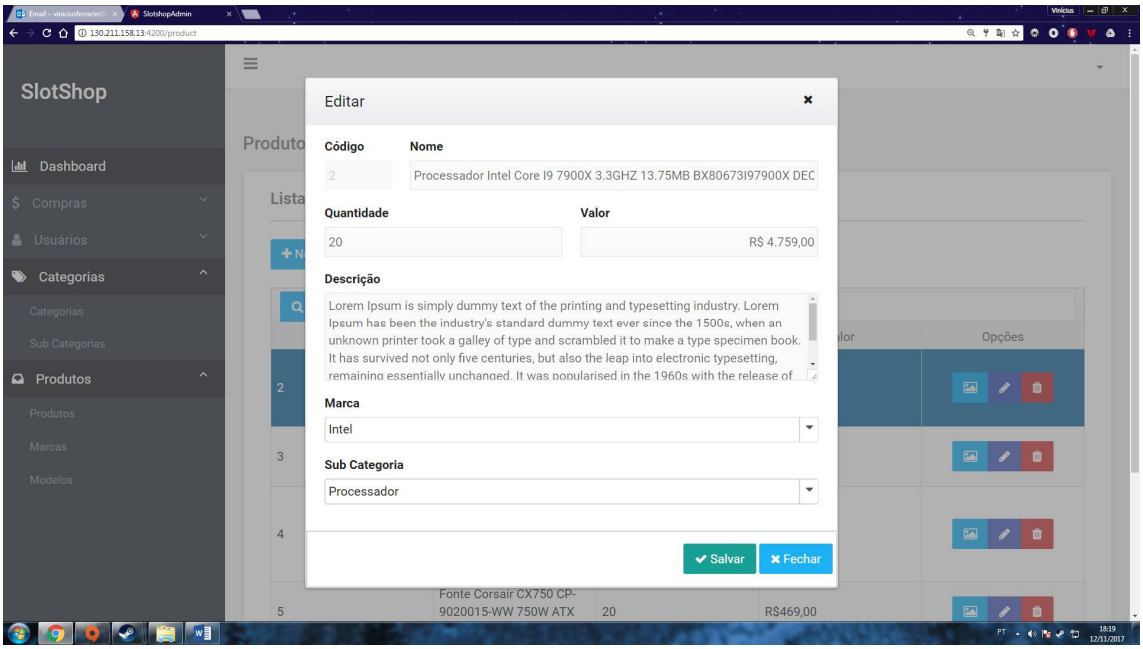

Figura 11 - Interface de cadastro e alteração de produtos

Na listagem de produtos, ao clicar no botão de galeria de fotos será carregada a interface responsável pelo *upload* das imagens dos produtos (Figura 12). Para inserir as imagens, o usuário poderá arrastá-las para o painel de *upload* ou clicar no botão *Choose* para selecioná-las e clicar no botão *upload* para que sejam enviadas ao servidor. Para remover as

imagens é necessário clicar no ícone (vermelho) que corresponde ao botão remover no canto superior direito nas miniaturas das imagens cadastradas, conforme exibido na Figura 12.

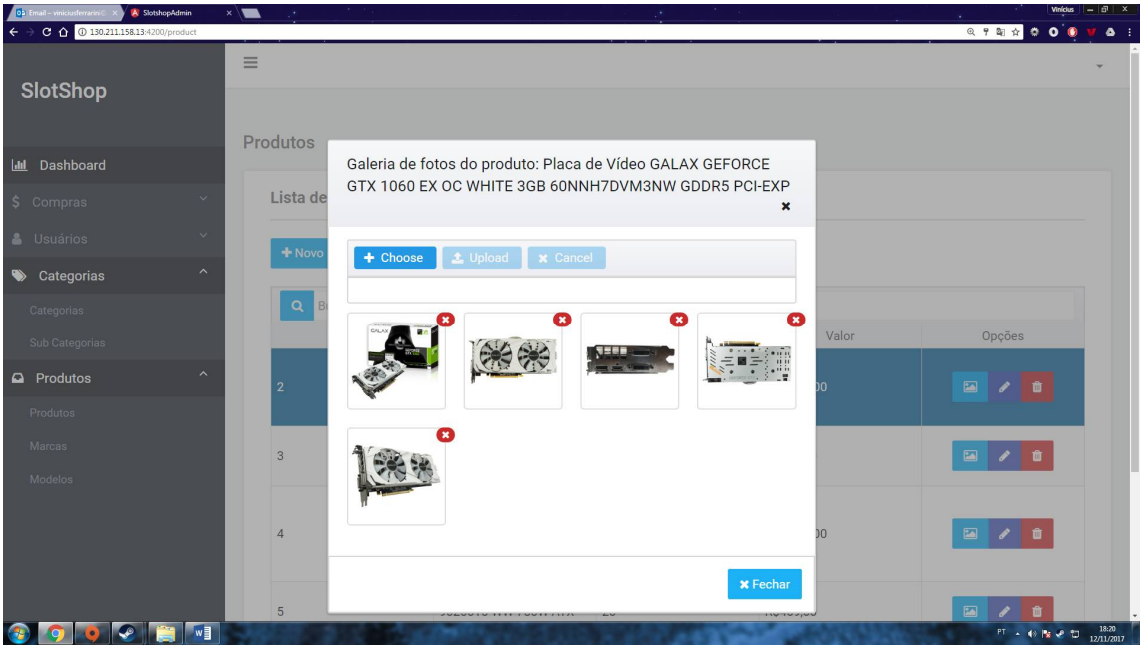

Figura 12 - Interface galeria de fotos

Ao acessar o menu usuários é exibida a interface com a listagem de usuários, como apresentado na Figura 13.

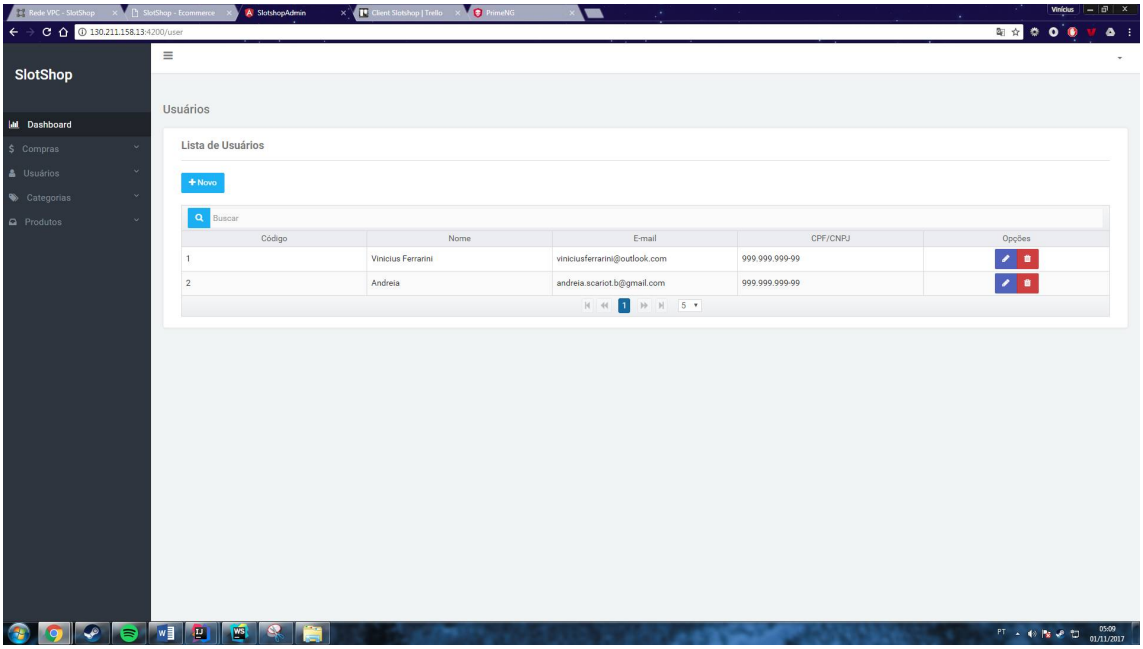

Figura 13 - Interface de listagem de usuários

A Figura 14 exibe a listagem de compras que são realizadas somente pelos clientes.

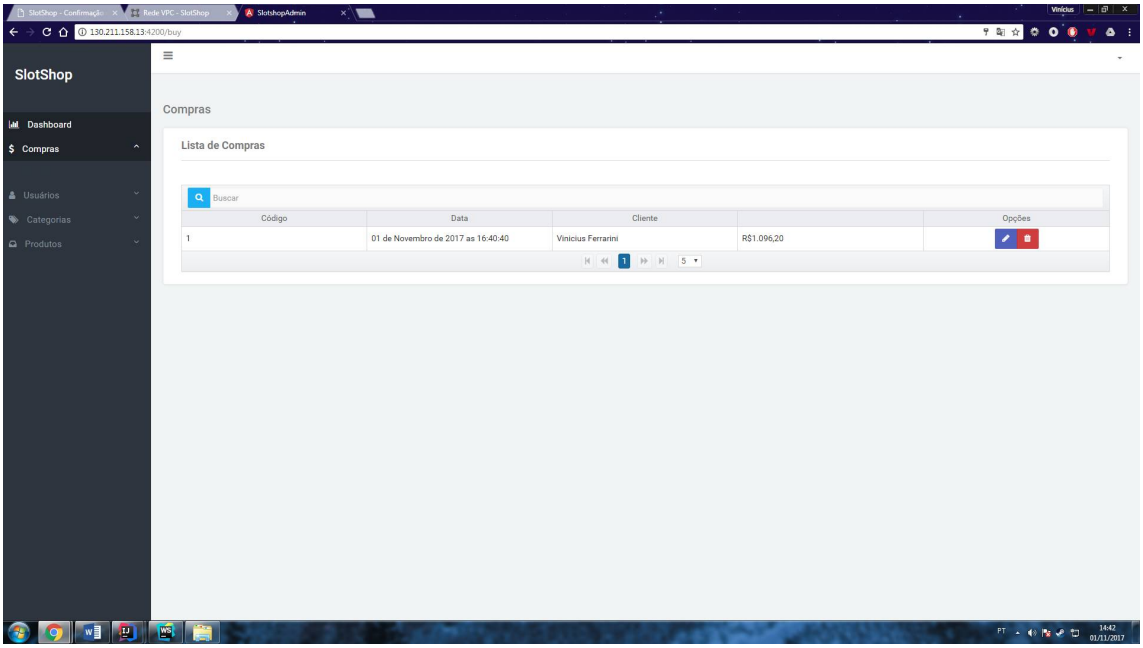

Figura 14 - Interface de listagem de compras

Ao clicar no botão ícone que representa a opção de editar, é exibida uma nova interface com todas as informações da compra (Figura 15). Nessa tela é possível alterar o *status* da compra.

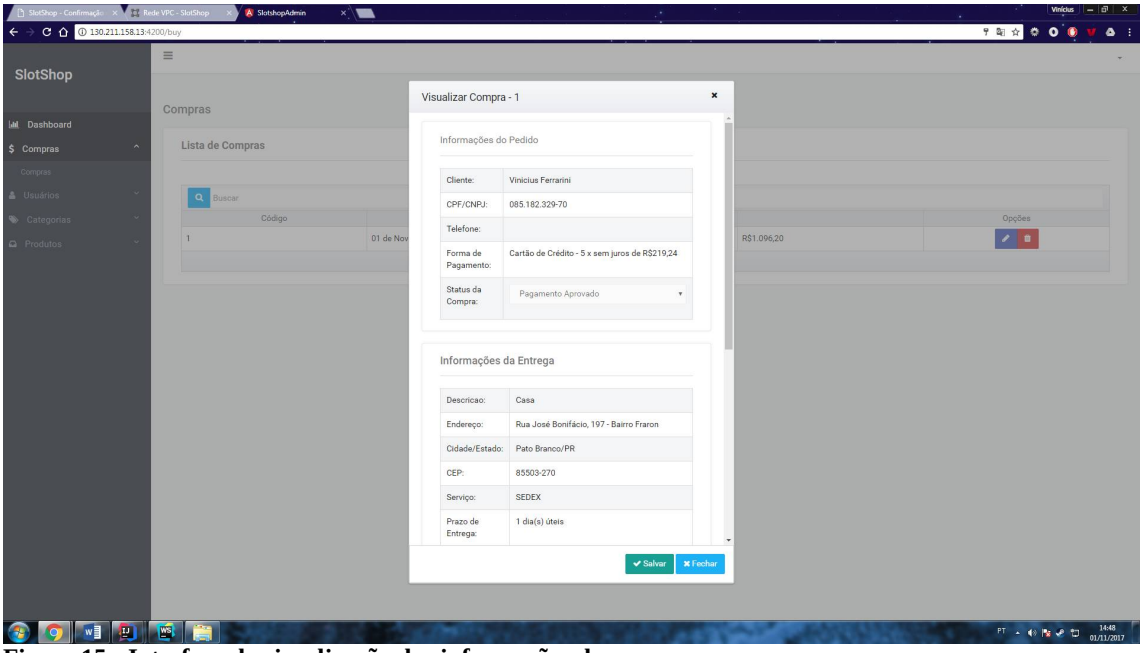

Figura 15 - Interface de visualização das informações de compra

Quando o *status* for alterado para “enviado”, o campo de código de rastreamento será habilitado, para que o administrador possa informar o código de rastreamento da transportadora, conforme apresentado na Figura 16.

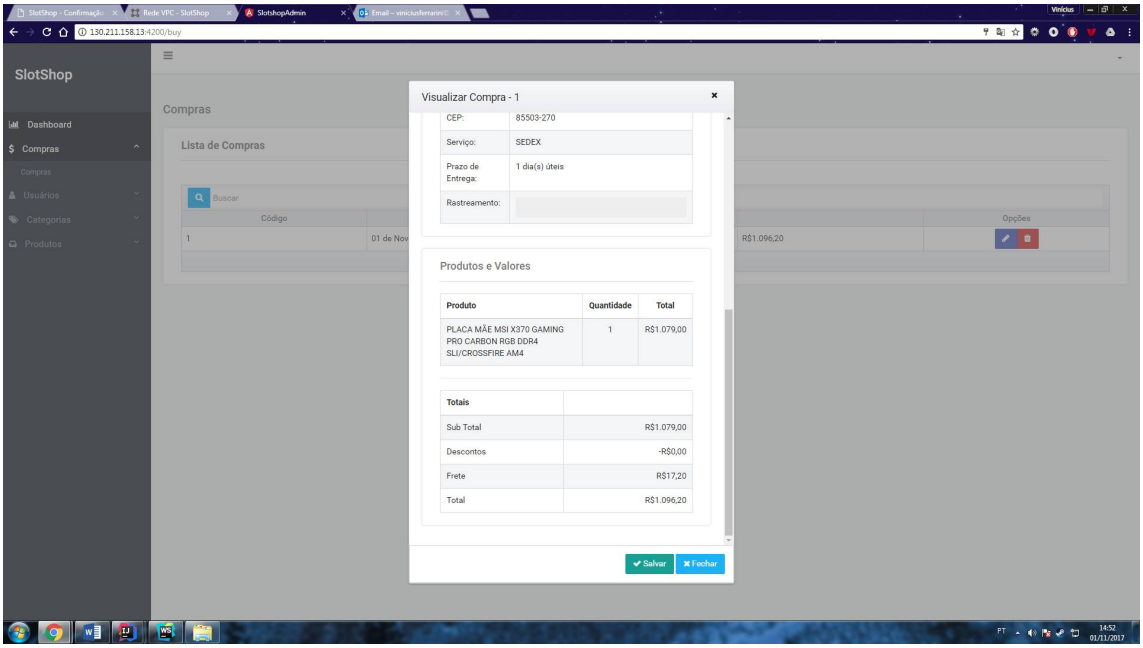

Figura 16 - Interface de visualização de informações de compra

A Figura 17 apresenta a página inicial da loja virtual que exibe os dados últimos produtos cadastrados, o menu de categorias e de informações da loja.

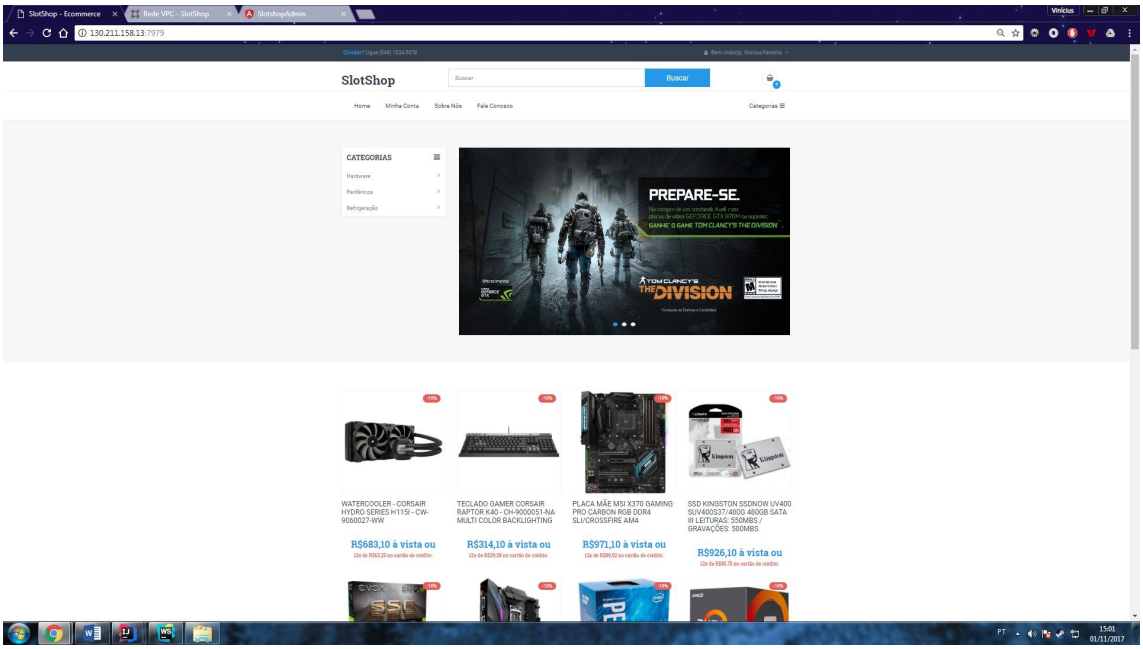

Figura 17 - Interface pagina inicial da loja

Ao clicar sobre um produto o usuário será redirecionado para a interface de detalhes desse produto, com informações adicionais e imagens relacionadas ao produto selecionado. Também é possível adicionar o produto no carrinho de compras informando a quantidade desejada, conforme exibe a Figura 18.

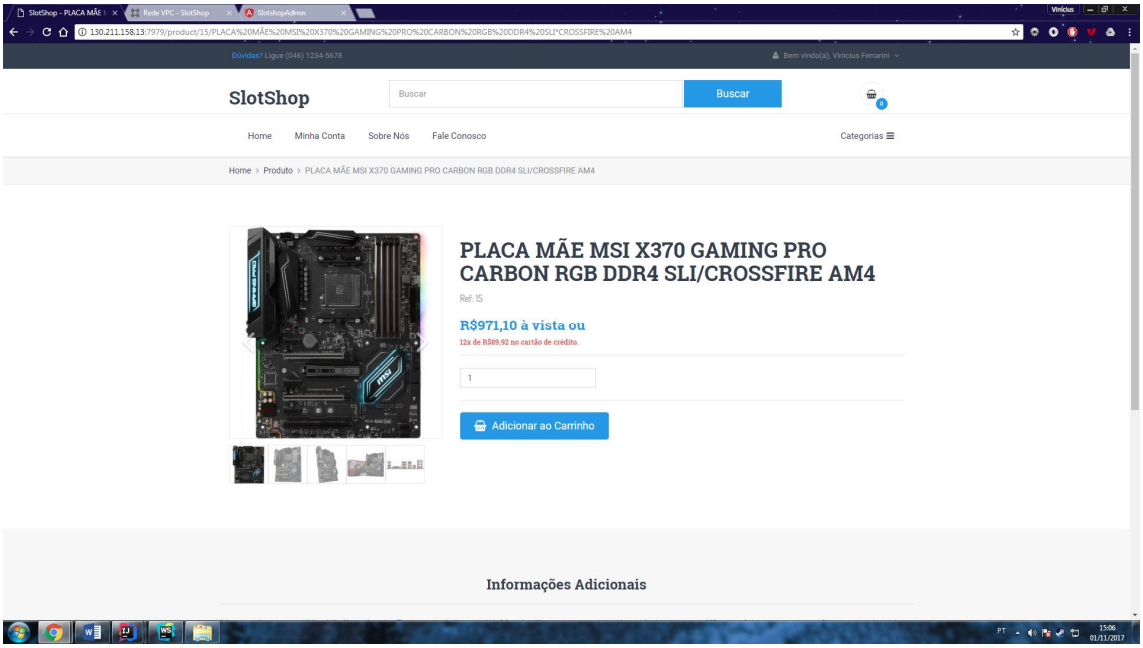

Figura 18 - Interface de detalhes do produto

A Figura 19 exibe a tela do carrinho de compras com um resumo da compra, sendo possível alterar a quantidade dos produtos, calcular o frete para um endereço de entrega e visualizar o valor total de sua compra.

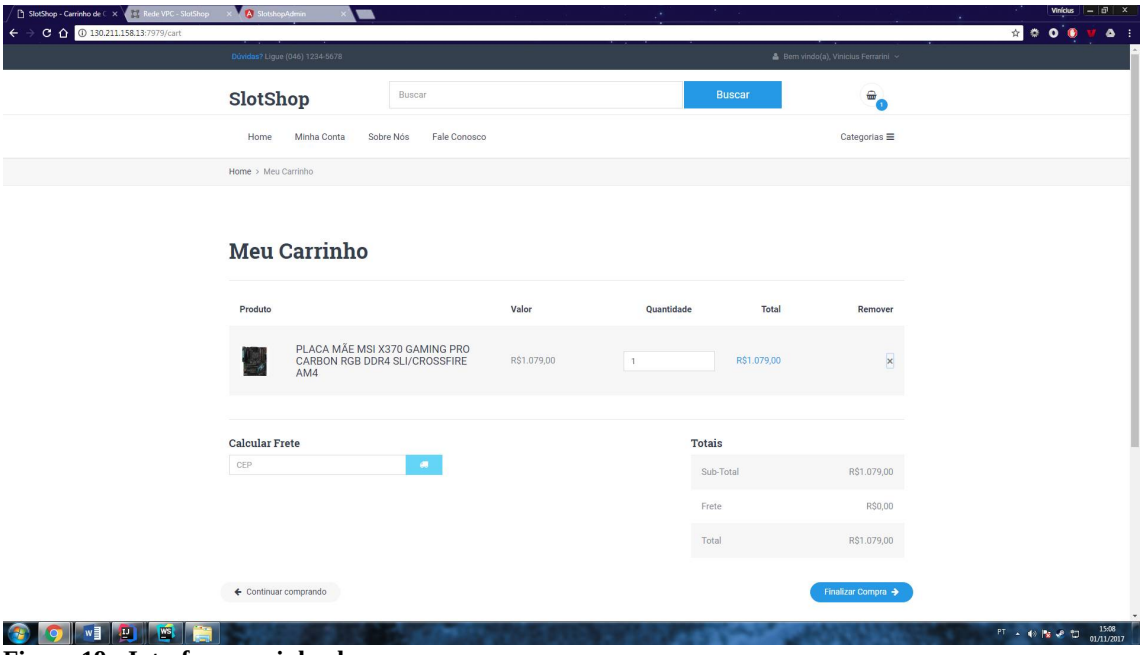

Figura 19 - Interface carrinho de compras

Ao informar o CEP e clicar no botão calcular, é realizada ao *web service* dos correios uma requisição que retorna os valores e prazos de entrega para o CEP informado, conforme

apresentado na Figura 20. Após escolher a forma de entrega, a compra deverá ser finalizada por meio do botão “Finalizar Compra”.

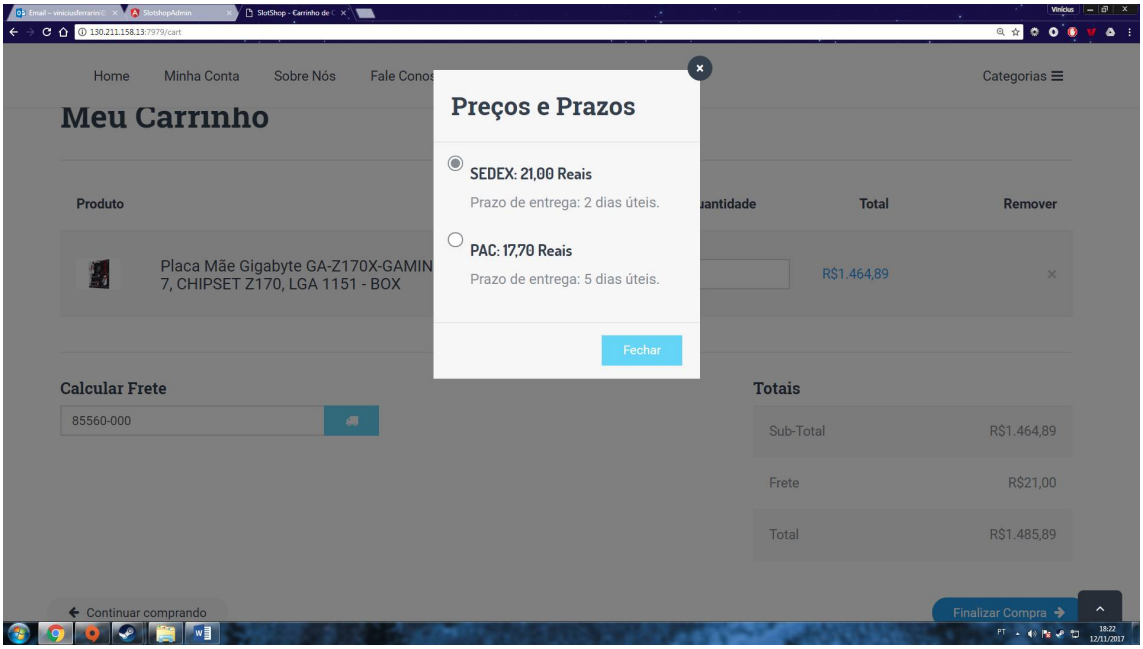

Figura 20 - Interface de cálculo de frete

Para acessar a interface de finalização de compra o usuário deverá estar autenticado no sistema, caso não esteja será redirecionado para a tela de autenticação, onde poderá realizar seu *login* ou se cadastrar (Figura 21).

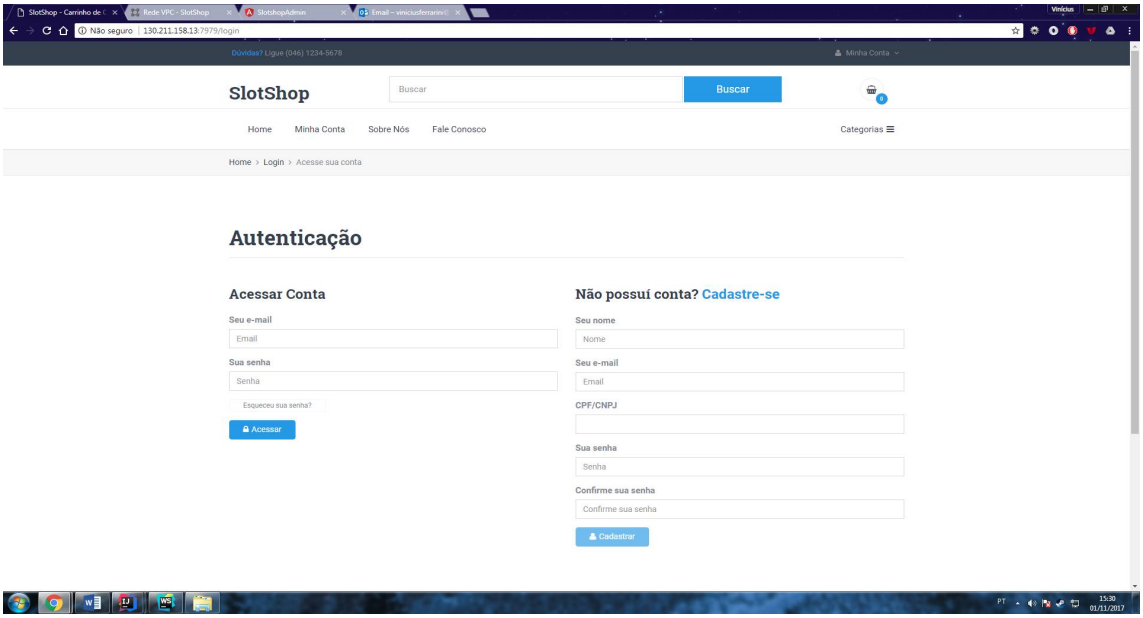

Figura 21 - Interface de autenticação e cadastro

A Figura 22 mostra a tela de finalização da compra que será exibida após o usuário realizar seu cadastro ou *login.* Na primeira aba dessa tela são exibidos os produtos e as respectivas quantidades que o cliente está adquirindo.

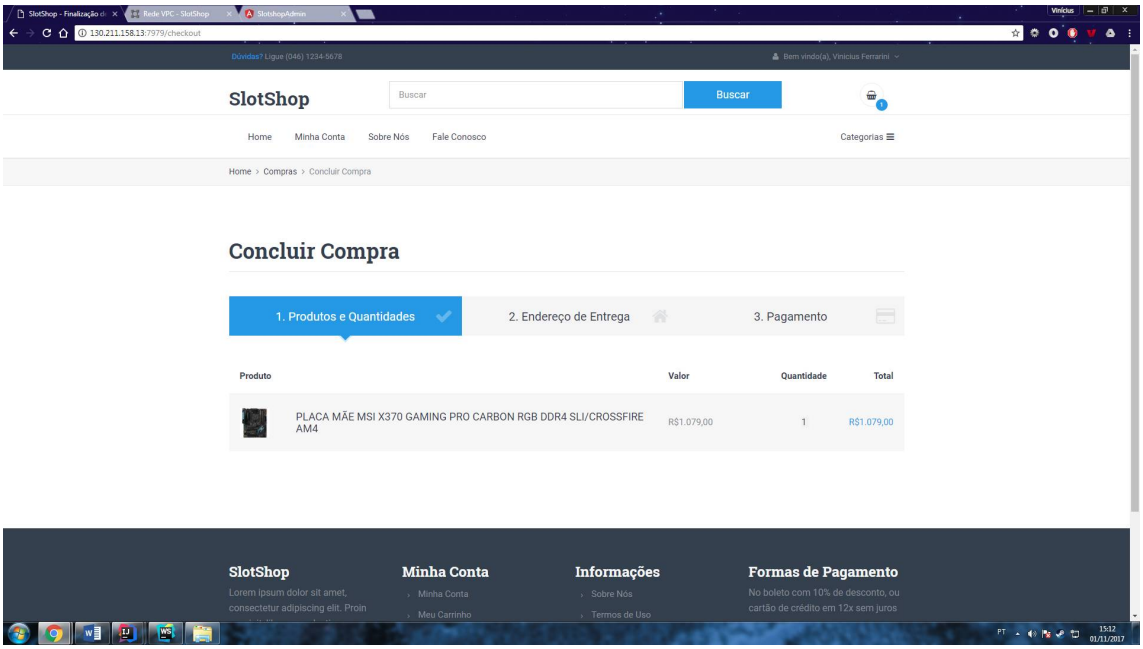

Figura 22 - Interface de finalização de compra - Produtos e quantidades

Clicando na aba endereço de entrega, é listada a interface de seleção de endereço de entrega (Figura 23). Caso o usuário tenha calculado o frete na interface do carrinho de compras e possuir um endereço cadastrado com o mesmo CEP, o mesmo será pré-selecionado e poderá ser alterado.

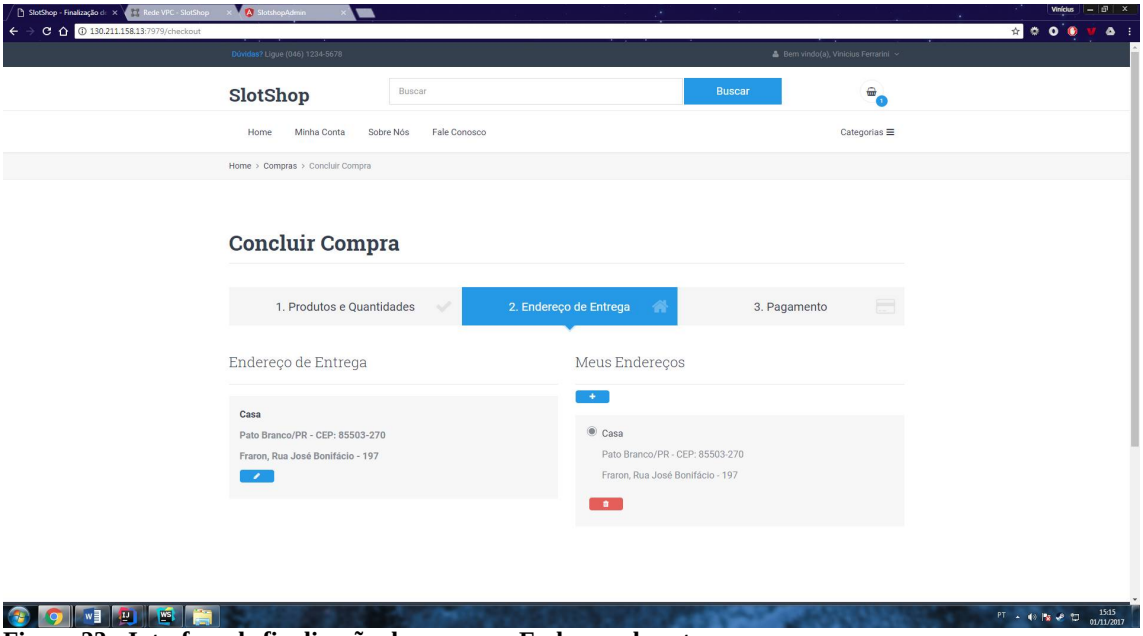

Figura 23 - Interface de finalização de compra - Endereço de entrega

A tela da Figura 23 também possibilita ao usuário incluir novos endereços, alterar ou excluir os registros cadastrados. Ao clicar sobre o botão incluir (representando pelo ícone com o sinal de +) é exibida a interface de cadastro e alteração de endereços, conforme mostra a Figura 24.

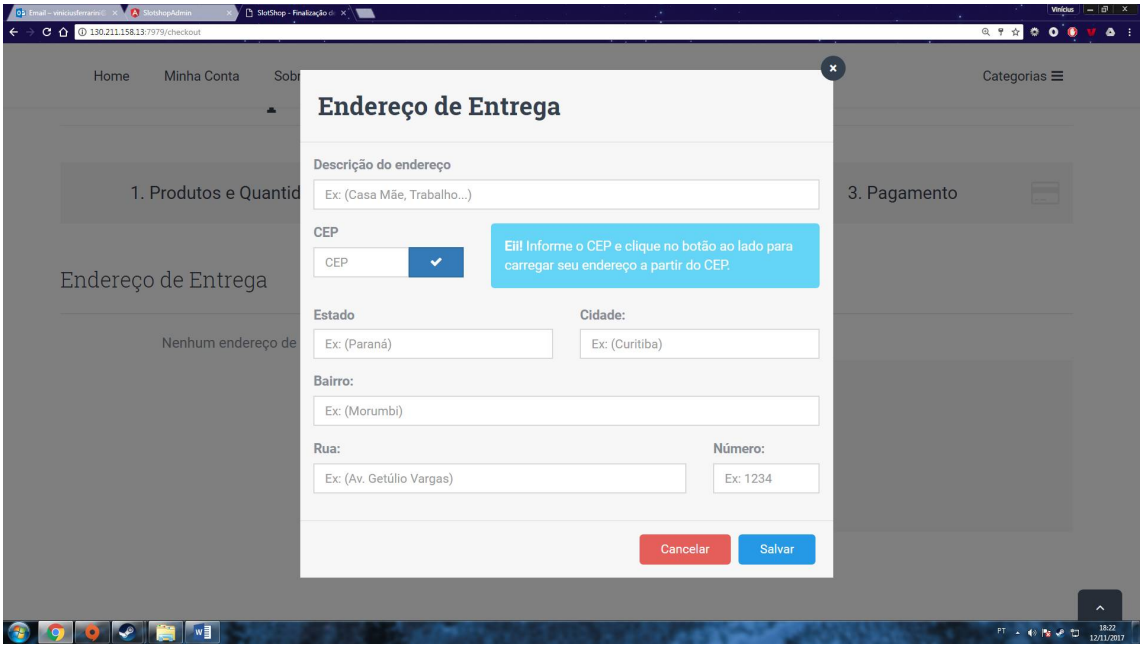

Figura 24 - Interface de finalização de compra - Cadastro e alteração de endereço de entrega

Após selecionar o endereço de entrega desejado, o usuário poderá avançar para a aba de Pagamento, apresentada na Figura 25. Nesta interface é possível optar por uma forma de

pagamento, podendo ser por boleto ou cartão de crédito e, também, visualizar os totais de sua compra.

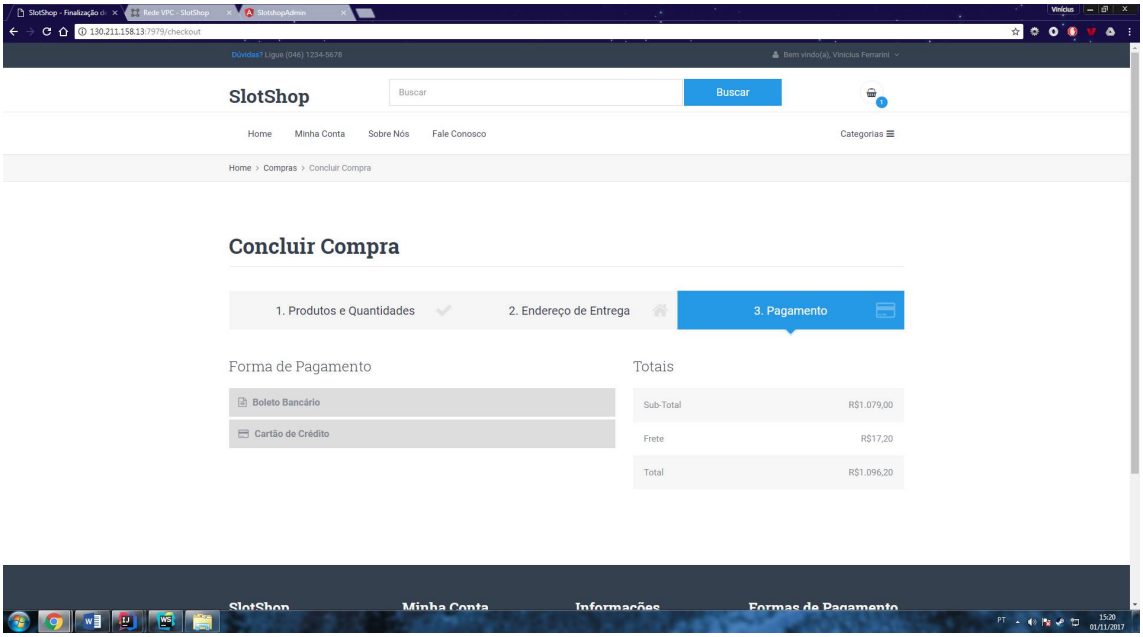

Figura 25 - Interface de finalização de compra - Pagamento

Após informar os dados de pagamento, usuário deve clicar no botão pagar. Caso todas as informações estejam informadas corretamente a interface de aprovação de pagamento seja exibida, conforme apresentando na Figura 26.

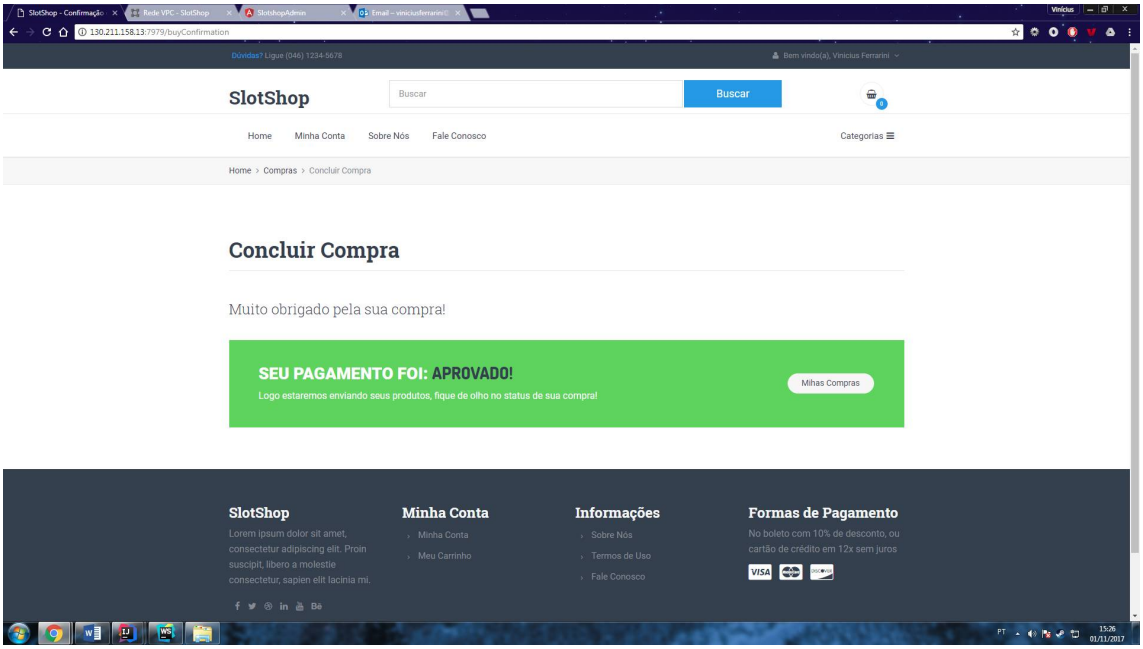

Figura 26 - Interface de finalização de compra - Aprovação de Pagamento

Ao clicar no botão minhas compras (Figura 26), ou passar o ponteiro do mouse sobre o seu nome no canto superior direito da loja, o usuário poderá acessar a interface de compras realizadas, apresentada na Figura 27.

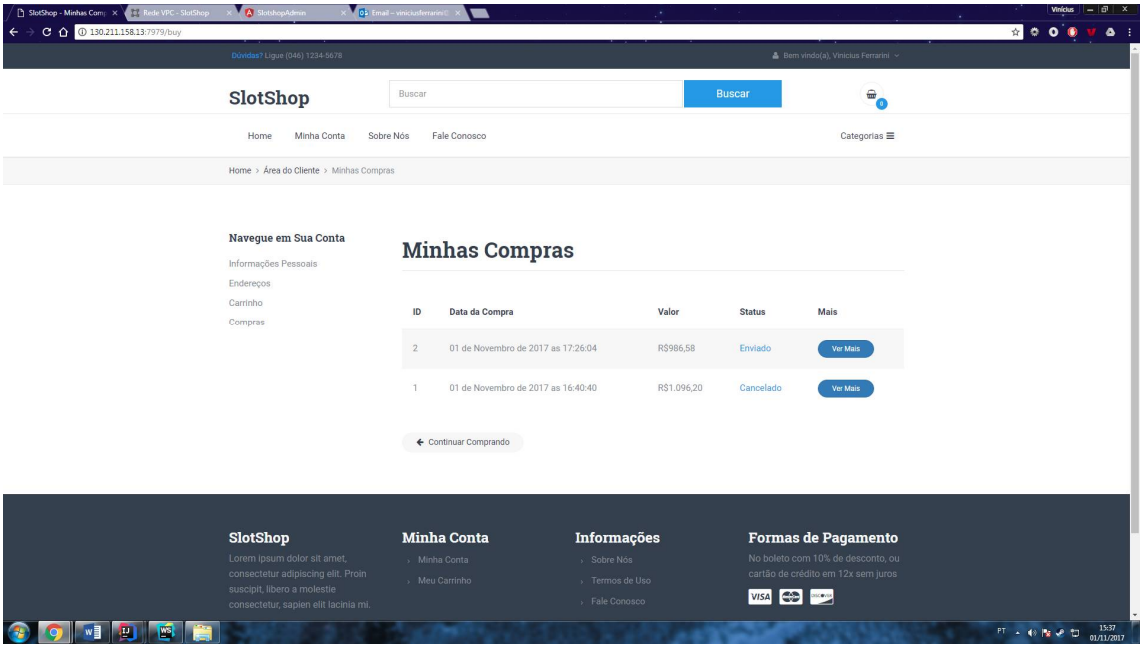

Figura 27 - Interface de compras realizadas

Aqui são listadas todas as compras realizadas pelo usuário e clicando sobre o botão “Ver Mais” (Figura 27) é possível visualizar todas as informações da compra (Figura 28).

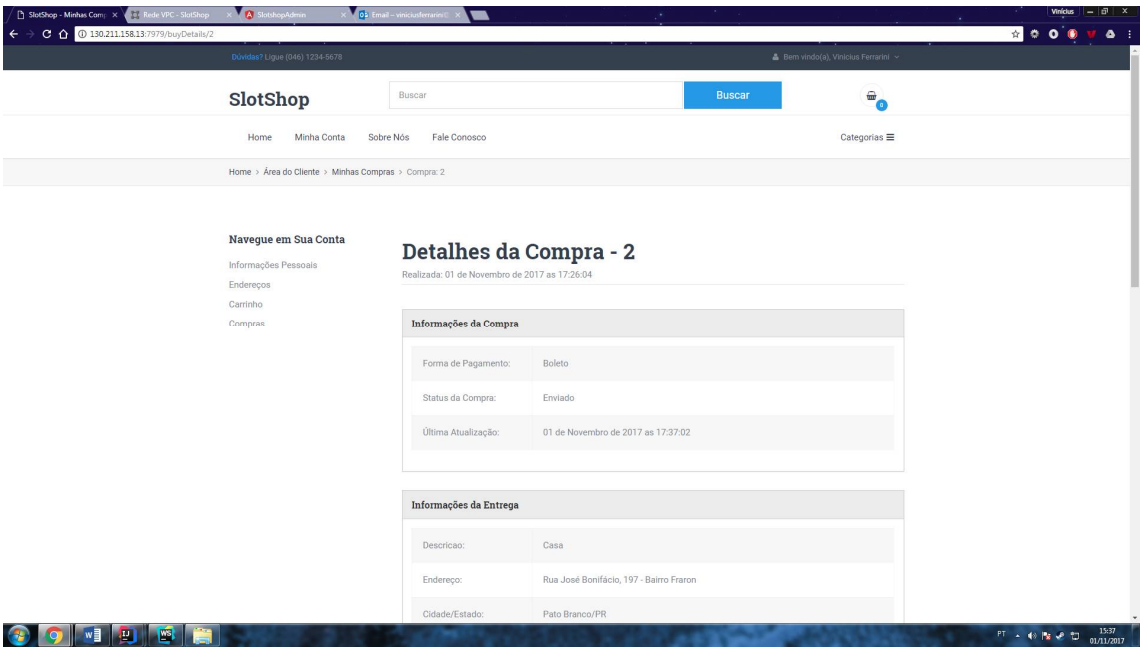

Figura 28 - Interface de detalhes da compra 1

Caso a compra já tenha sido enviada, o usuário poderá rastreá-la clicando botão rastrear sendo exibidos os dados de rastreamento de objetos, conforme mostra a Figura 29.

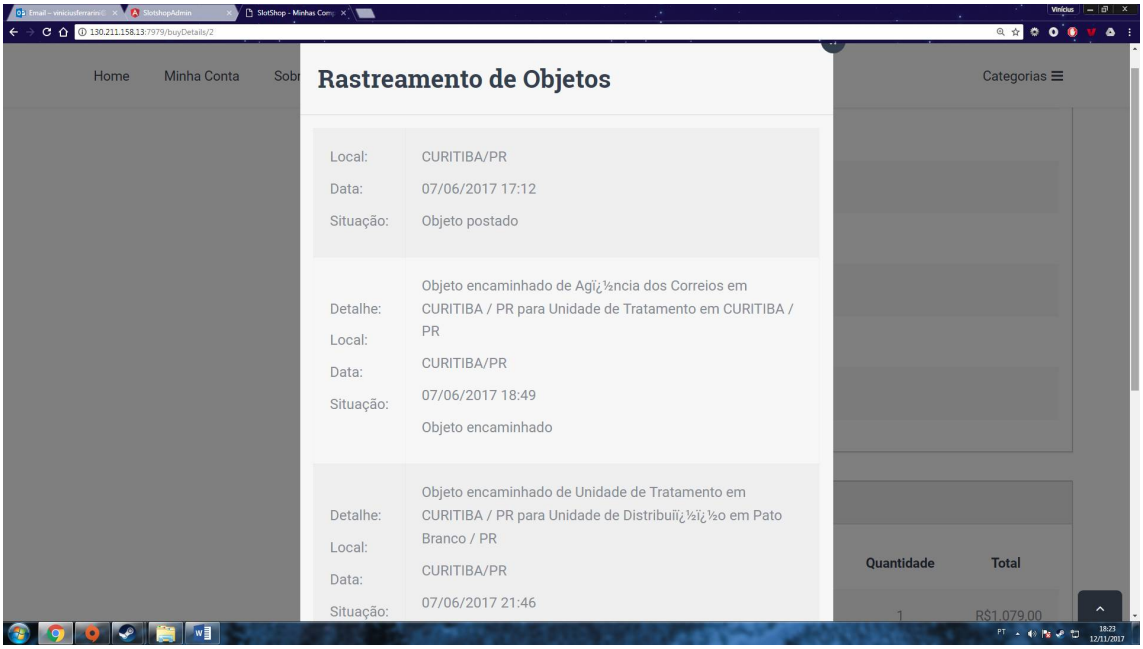

Figura 29 - Interface de rastreamento de objetos

No menu lateral a interface de detalhes de compras, o usuário também poderá visualizar ou manter seus endereços cadastrados e suas informações pessoais e sua senha, conforme mostra a Figura 30.

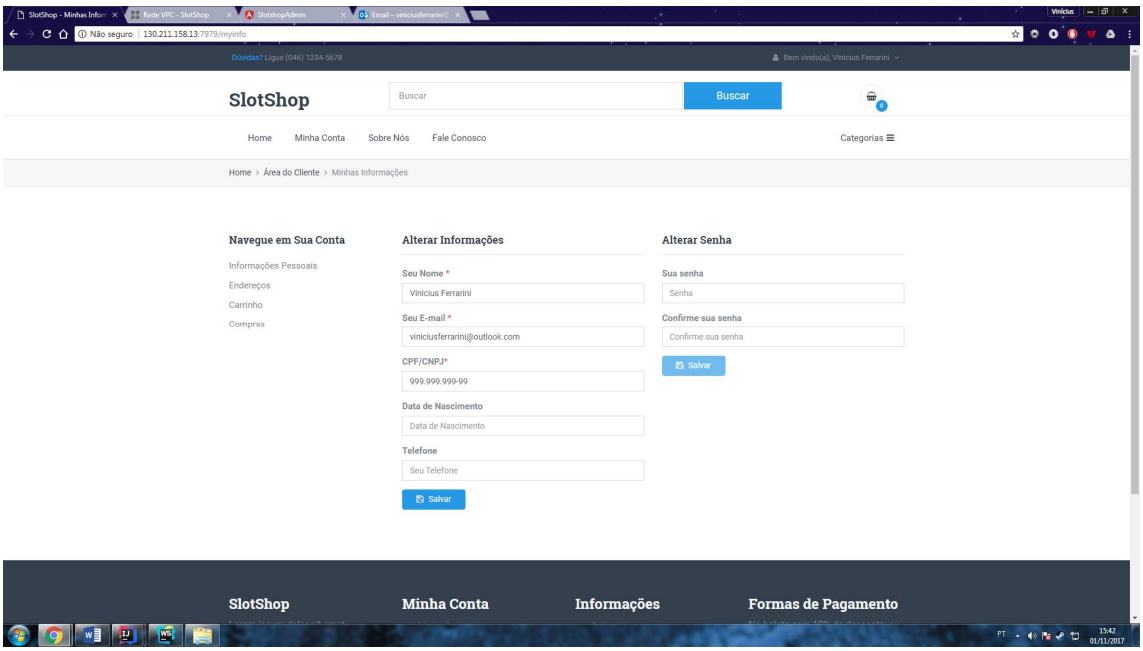

Figura 30 - Interface de informações pessoais

3.5 IMPLEMENTAÇÃO DO SISTEMA

Para a conexão e autenticação da aplicação Angular com o *back-end* foi necessário implementar alguns arquivos de configuração. O primeiro foi criar um novo atributo no objeto *enviroment.prod.ts* para definir o endereço do servidor de *back-end* onde serão realizadas as requisições via *Representational State Transfer* (REST), conforme mostra a Listagem 4..

```
export const environment = {
  production: true,
  proxy: "http://130.211.158.13:7990",
  showLogRequestTime: true
};
```

Listagem 4 - Configuração do endereço do servidor de backend

Como os cadastros possuem em sua maior parte as mesmas ações, foram implementadas classes abstratas com todos os métodos necessários para serem utilizados nos componentes. A classe abstrata *CrudService*, exibida na Listagem 5, possui todos os métodos *Hypertext Transfer Protocol* (HTTP) para busca e envio de informações ao servidor.

```
@Injectable()
export abstract class CrudService <T extends CrudEntity<ID>, ID>  {

  private proxyUrl: string = environment.proxy;

    constructor(private http: HttpClient) { }

  get<T>(): Observable<T> {
    return this.http.get<T>(this.proxyUrl + this.getUrl());
  }

  post<T>(body: string): Observable<T> {
    return this.http.post<T>(this.proxyUrl + this.getUrl(), body);
  }

  put<T>(body: string): Observable<T> {
    return this.http.put<T>(this.proxyUrl + this.getUrl(), body);
  }

  delete<T>(body: string): Observable<T> {
    return this.http.delete<T>(this.proxyUrl + this.getUrl() + "/" +
body);
  }

  patch<T>(body: string): Observable<T> {
    return this.http.patch<T>(this.proxyUrl + this.getUrl(), body);
  }

  protected abstract getUrl(): string;
```

```
}
```

Listagem 5 - Classe CrudService responsável pelos métodos de comunicação com o servidor

Cada componente possui um *service*, que ele estende a classe *CrudService*, automaticamente podendo acessar cada um desses métodos, como pode ser visualizado na Listagem 6.

```
import {Injectable} from '@angular/core';
import {CrudService} from "../service/crud.service";
import {Brand} from "./brand";
import {HttpClient} from "@angular/common/http";

@Injectable()
export class BrandService extends CrudService<Brand, number>{

  constructor(httpClient: HttpClient) {
    super(httpClient);
  }

  protected getUrl(): string {
    return "/brand";
  }
}
```

Listagem 6 - Classe BrandService estendendo a Classe CrudService para ter acesso aos métodos de comunicação com o servidor

Para tornar ainda mais fácil a implementação dos componentes, foi implementada uma classe abstrata responsável pelo gerenciamento dos cadastros. Essa classe possui todos os métodos utilizados pela interface de cadastro, como pode ser visualizado na Listagem 7.

```
import {OnInit, ViewContainerRef} from "@angular/core";
import {CrudEntity} from "./crud.entity";
import {CrudService} from "./crud.service";
import {ToastsManager} from "ng2-toastr";

export abstract class CrudController<T extends CrudEntity<ID>, ID>
implements OnInit {

  lista: T[] = [];
  objeto: T = new this.type;
  displayEdit: Boolean = false;
  acao: string;

  constructor(protected toastr: ToastsManager,
              protected vcr: ViewContainerRef,
              public crudService: CrudService<T, ID>,
              private type: any) {
    this.toastr.setRootViewContainerRef(this.vcr);
  };

  ngOnInit(): void {
    this.getTable();
  };
```

```
  getTable() {
    this.crudService
      .get<any[]>()
      .subscribe((data: any[]) => this.lista = data,
        error => () => {
          this.errorMessages(error);
        });
  }

  persist() {
    if (this.objeto.id != null) {
      this.crudService.post<any>(JSON.stringify(this.objeto))
        .subscribe(
          res => {
            this.toastr.success("Cadastro atualizado com sucesso!",
"Sucesso!");
            this.displayEdit = false;
          },
          err => {
            this.errorMessages(err);
          }
        );
    } else {
      this.crudService.post<any>(JSON.stringify(this.objeto))
        .subscribe(
          res => {
            this.toastr.success("Cadastro realizado com sucesso!",
"Sucesso!");
            const listaTemp = [...this.lista];
            this.objeto = res;
            listaTemp.push(res);
            this.lista = listaTemp;
            this.displayEdit = false;
          },
          err => {
            this.errorMessages(err);
          }
        );
    }
  };

  private errorMessages(err) {
    if (err.error.errors.length > 0) {
      err.error.errors.forEach(e => {
        this.toastr.error(e.defaultMessage, "Erro!");
      });
    }
  }

  new() {
    this.objeto = new this.type;
    this.displayEdit = true;
    this.acao = "Cadastro";
  };

  remove(item) {
    this.crudService.delete<any>(JSON.stringify(item.id))
      .subscribe(res => {
        this.toastr.success("Registro removido com sucesso!", "Sucesso!");
        const listTemp = [...this.lista];
        const index = listTemp.indexOf(this.objeto);
```

```
        listTemp.splice(index, 1);
        this.lista = listTemp;
        this.displayEdit = false;
      }, err => {
        this.errorMessages(err);
      });
  };

  onRowSelect(item) {
    this.objeto = item;
    this.displayEdit = true;
    this.acao = "Editar ";
  };

}
```

Listagem 7 - Classe abstrata CrudController, responsável pela manipulação das interfaces dos cadastros

Todos os componentes de cadastro estendem a classe *CrudController*, como mostra a Listagem 8.

```
import {Component, OnInit, ViewContainerRef} from '@angular/core';
import {Brand} from "./brand";
import {CrudController} from "../service/crud.controller";
import {BrandService} from "./brand.service";
import {ToastsManager} from "ng2-toastr";
import {FormControl, FormGroup, Validators} from "@angular/forms";

@Component({
  selector: 'app-brand',
  templateUrl: './brand.component.html',
  styleUrls: ['./brand.component.css']
})
export class BrandComponent extends CrudController<Brand, number>
implements OnInit {

  constructor(protected toastr: ToastsManager,
              protected vcr: ViewContainerRef,
              brandService: BrandService) {
    super(toastr, vcr, brandService, Brand);
  }
}
```

Listagem 8 - Componente BrandComponent extendendo a classe CrudController

Para que os usuários realizem *login* na aplicação, foi necessário implementar um serviço responsável pela autenticação e validação de *token*, como pode ser visto na Listagem 9.

```
import {Injectable} from "@angular/core";
import "rxjs/add/operator/toPromise";
import {environment} from "../../environments/environment";
import {Subject} from "rxjs/Subject";
import {ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot}
from "@angular/router";
import {Headers, Http, RequestOptions} from "@angular/http";
import {Observable} from "rxjs/Observable";
```

```
@Injectable()
export class LoginService implements CanActivate {

  public isLoggedIn = new Subject<Boolean>();

  constructor(private router: Router,
              private http: Http) {
    this.isLoggedIn = new Subject<Boolean>();
  }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> | Promise<boolean> | boolean {
    return this.isAuthorized();
  }

  observableIsLoggedIn() {
    return this.isLoggedIn.asObservable();
  }

  login(email, password): void {

    const params = new URLSearchParams();
    params.append('username', email);
    params.append('password', password);
    params.append('grant_type', 'password');

    const headers = new Headers({ 'Content-type': 'application/x-www-form-
urlencoded', 'Authorization': 'Basic YXBwOmFwcA==' });
    const options = new RequestOptions({headers: headers});

    const url = `${environment.proxy}/oauth/token`;
    this.http.post(url, params.toString(), options)
      .subscribe(res => {
        this.setLoggedIn(res.json());
      }, error => {
        if (JSON.parse(error._body).error === "invalid_grant") {
          window.alert("E-mail ou senha inválido!");
        }
        this.logout();
      });
  }

  setLoggedIn(res) {
    this.isLoggedIn.next(true);
    localStorage.setItem('access_token', res.access_token);
    this.router.navigate(['/dashboard']);
  }

  logout() {
    localStorage.removeItem('access_token');
    this.isLoggedIn.next(false);
    this.router.navigate(['/login']);
  }

  isAuthorized() {
    if (localStorage.getItem('access_token') !== null) {
      return true;
    }
    this.logout();
    return false;
  }
```

```
}
```

Listagem 9 - Classe LoginService responsável pela autenticação do sitema

Como todas as páginas da aplicação necessitam que o usuário esteja autenticado com o servidor, foi necessário criar um interceptor que identifica todas as requisições do tipo HTTP, verificando se a mesma contém um *token* válido, gerado pelo servidor no momento do *login*. Na Listagem 10 é possível visualizar a implementação deste interceptor.

```
import {Injectable} from '@angular/core';
import 'rxjs/add/operator/do';
import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest,
HttpResponse} from '@angular/common/http';
import {Observable} from "rxjs/Observable";
import {LoginService} from "../login/login.service";
import {environment} from "../../environments/environment";

@Injectable()
export class HttpLoginInterceptor implements HttpInterceptor {

  constructor(private loginService: LoginService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {

    if (!this.loginService.isAuthorized()) {
      return next.handle(req);
    }

    const authReq = req.clone({
      headers: req.headers.set('Content-Type',
'application/json').append('Authorization', 'bearer ' +
localStorage.getItem("access_token"))
    });

    const started = Date.now();
    return next.handle(authReq).do(event => {
      if (environment.showLogRequestTime) {
        if (event instanceof HttpResponse) {
          this.loginService.isLoggedIn.next(true);
          const elapsed = Date.now() - started;
          console.log(`">>>>>>>>>>>>>>>>>>>>>> Request for
${req.urlWithParams} took ${elapsed} ms.`);
        }
      }
    }, err => {
      if ( err.status === 401 ) {
        this.loginService.logout();
      }
    });
  }
}
```

Listagem 10 - Interceptor HttpLoginInterceptor responsável pela verificação das requisições HTTP

Aplicações desenvolvidas em Angular utilizam o conceito de SPA, portanto, para que seja possível que o usuário navegue entre as páginas do sistema foi necessário implementar

um arquivo de configuração de rotas, no qual foi definida a *Uniform Resource Locator* (URL) das páginas e qual componente será responsável pela mesma, como pode ser visualizado na Listagem 11.

```
import {RouterModule, Routes} from "@angular/router";
import {LoginComponent} from "../login/login.component";
import {HomeComponent} from "../home/home.component";
import {DashboardComponent} from "../dashboard/dashboard.component";
import {ProductComponent} from "../product/product.component";
import {CategoryComponent} from "../category/category.component";
import {ModelComponent} from "../model/model.component";
import {SubCategoryComponent} from "../sub-category/sub-
category.component";
import {BrandComponent} from "../brand/brand.component";
import {LoginService} from "../login/login.service";
import {UserComponent} from "../user/user.component";
import {BuyComponent} from "../buy/buy.component";

const routes: Routes = [
  {path: '', redirectTo: '/login', pathMatch: 'full'},
  {path: 'login', component: LoginComponent},
  {path: 'home', component: HomeComponent, canActivate: [LoginService]},
  {path: 'dashboard', component: DashboardComponent, canActivate:
[LoginService]},
  {path: 'product', component: ProductComponent, canActivate:
[LoginService]},
  {path: 'category', component: CategoryComponent, canActivate:
[LoginService]},
  {path: 'subCategory', component: SubCategoryComponent, canActivate:
[LoginService]},
  {path: 'model', component: ModelComponent, canActivate: [LoginService]},
  {path: 'brand', component: BrandComponent, canActivate: [LoginService]},
  {path: 'user', component: UserComponent, canActivate: [LoginService]},
  {path: 'buy', component: BuyComponent, canActivate: [LoginService]}
];

export const AppRoutingModule = RouterModule.forRoot(routes);
```

Listagem 11 - Arquivo responsável pelas rotas da aplicação

O desenvolvimento do sistema foi divido em dois microsserviços utilizando o Spring Boot. O primeiro microsserviço denominado *slotshop-server* contém todas as entidades que persistem informações no banco dedados. Nele, também estão implementadas as interfaces *service*, *repository* e *controllers* de cada entidade. Esta aplicação foi implementada para receber as requisições via REST do sistema administrativo em Angular.

O outro microsserviço denominado *slotshop-client*, contém toda a estrutura da loja, e *controllers* das interfaces HTML. Para que não fosse necessária a duplicidade de código nos dois microsserviços e a loja pudesse acessar o banco de dados sem realizar requisições via *rest template* ao microsserviço *slotshop-server*, foi implementado por meio de um recurso do Maven chamado *plugin*, que exporta diretórios ou arquivos em especifico para serem utilizados em outras aplicações. Na Listagem 12 é possível visualizar a configuração no arquivo pom.xml do microsserviço *slothsop-server* para que os arquivos fossem compartilhados.

```
<build>
   <finalName>slotshop-server</finalName>
   <plugins>
      <plugin>
         <groupId>org.spring*framework*.boot</groupId>
         <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-jar-plugin</artifactId>
         <executions>
            <execution>
               <id>package-client-model</id>
               <phase>package</phase>
               <goals>
                  <goal>jar</goal>
               </goals>
               <configuration>
                  <classifier>client-model</classifier>
                  <includes>
                     <include>**/model/**</include>
                     <include>**/service/**</include>
                              <include>**/repository/**</include>
                              <include>**/util/**</include>
                     <include>**/enumeration/**</include>
                  </includes>
               </configuration>
            </execution>
         </executions>
      </plugin>
   </plugins>
</build>
```

Listagem 12 – Configuração de exportação de arquivos no pom.xml do micro-serviço slotshop-server

A Listagem 13 apresenta um exemplo de utilização de entidades e *services* sendo utilizados no microsserviço *slotshop-client*.

```
@Controller
@RequestMapping("buy")
public class BuyController {

    @Autowired private BuyClientService buyClientService;

    @Autowired private BuyService buyService;

    @Autowired private UserService userService;

    @GetMapping
    public ModelAndView get(){
        return new ModelAndView("/buy");
    }

    @PostMapping
    public @ResponseBody Buy buy(@Valid @RequestBody Checkout checkout,
HttpSession session){
        return buyClientService.buy(checkout, session);
    }

    @GetMapping("/findAll")
    public @ResponseBody List<Buy> findAllByUser(){
        return buyService.findByUser(userService.getLoggedUser());
    }

}
```

Listagem 13 - Controller utilizando entidades e serviços implementados no micro-serviço slotshop-server

3.5.1 Implementação microsserviço *slotshop-server*

O microsserviço, denominado, *slotshop-server*, é responsável por receber as requisições via REST do sistema administrativo que foi desenvolvido em Angular. Para isso, é necessário realizar algumas configurações para que o projeto receba esse tipo de requisição. A primeira configuração, exibida na Listagem 14, mostra a dependência incluída no arquivo pom.xml para tornar a aplicação *web*.

```
<dependency>
   <groupId>org.spring*framework*.boot</groupId>
   <artifactId>spring-boot-starter-*web*</artifactId>
</dependency>
```

Listagem 14 - Dependência spring boot starter *web*

Cada requisição recebida via REST acessa uma URL disponível no sistema. Para que isso seja possível foi necessário implementar *controllers* com algumas anotações como é possível visualizar na Listagem 15. A anotação *@RestController* define que todos os métodos disponíveis vão retornar as requisições via JSON e a anotação *@RequestMapping(“/product”)* define a URL em que as requisições serão atendidas.

```
@RestController
@RequestMapping("/product")
public class ProductController extends RestCrudController<Product, Long> {

    @Autowired private ProductService productService;

    @Override
    protected CrudService<Product, Long> getService() {
        return productService;
    }

}
```

Listagem 15 - Controller responsável pelas requisições de produtos

Os dados retornados por meio dos *controllers*, estão armazenados em uma base de dados MySql, que também necessita de configurações para que possa ser acessada. Na Listagem 16, é possível visualizar a dependência incluída por meio do Maven, do *driver* necessário para conexão com a base de dados, e na Listagem 17 as configurações incluídas no arquivo *application.properties* de acesso ao banco de dados.

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Listagem 16 - Dependência do driver MySql

```
spring.datasource.dialect= org.hibernate.dialect.MySQLDialect
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/slotshop
spring.datasource.username=root
spring.datasource.password=*******
```

Listagem 17 - Propriedades para conexão com a base de dados

Para buscar as informações na base dados, foi utilizado um *framework* do Spring chamado Spring Data JPA. Para que fosse possível a sua utilização foi necessário incluir uma dependência por meio do Maven, conforme exibe a Listagem 18.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Listagem 18 - Dependência Spring Data

Nos repositórios referentes a cada entidade, foi necessário estender uma interface do Spring Data que possui métodos padrões para serem utilizados sem a necessidade de implementação, como, por exemplo, o método *save*. Na Listagem 19 é possível visualizar um *repository* estendendo essa interface e, também, contendo métodos personalizados de busca que não existem na interface *JpaRepository*. A utilização do Spring Data faz com que não seja necessário implementar consultas *Structured Query Language* (SQL), sendo necessário

apenas descrever por meio do nome do método o que deseja buscar e quais filtros deseja utilizar, como o método findByNameContainingOrDescriptionContainingIgnoreCase, que busca os dados contendo o parâmetro no atributo *name* ou *description* ignorando se letras estão em maiúsculas ou minúsculas.

```
@Repository
public interface ProductData extends JpaRepository<Product, Long> {

    List<Product> findTop12ByOrderByIdDesc();

    Page<Product>
findByNameContainingOrDescriptionContainingIgnoreCase(String searchName,
String searchDescription, Pageable pageable);

    Page<Product> findBySubCategory(SubCategory subCategory, Pageable
pageable);

    Page<Product> findBySubCategoryCategory(Category category, Pageable
pageable);
}
```

Listagem 19 - ProductData utilizando Spring Data para retornar informações da base de dados

Todas as requisições recebidas neste microsserviço devem ser autenticadas. Assim, foi necessário configurar um *framework* de autenticação baseado em *token*. Para isso, foi utilizado o OAuth2, que também necessita que sejam incluídas dependências por meio do Maven, como pode ser visualizado na Listagem 20, e também um arquivo de configuração exibido na Listagem 21.

```
<dependency>
   <groupId>org.spring*framework*.cloud</groupId>
   <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
```

Listagem 20 - Dependência OAuth2

```
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthorizationConfig extends
AuthorizationServerConfigurerAdapter {

    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws
Exception {
        clients.inMemory()
                .withClient("app")
                .secret("app")
                .autoApprove(true)
                .authorizedGrantTypes("authorization_code", "refresh_token",
"password", "client_credentials")
                .scopes("openid");
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
throws Exception {
        endpoints
                .authenticationManager(authenticationManager)
                .userDetailsService(userDetailsService);
    }

    @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer)
throws Exception {

oauthServer.tokenKeyAccess("permitAll()").checkTokenAccess("isAuthenticated()");
    }

}
```

Listagem 21 - Classe de configuração OAuth2

3.5.2 Implementação microsserviço *slotshop-client*

Algumas interfaces do sistema necessitam de autenticação. Para isso, foi necessário implementar um arquivo de configuração para configurar quais URLs e tipos de requisição podem ser acessados sem estar autenticado no sistema, como apresentado na Listagem 22.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired private UserDetailsService userDetailsService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                    .antMatchers("/error/**").permitAll()
                    .antMatchers(HttpMethod.GET,  "/css/**", "/js/**",
"/img/**", "/fonts/**").permitAll()
                    .antMatchers(HttpMethod.GET, "/", "/findFirst10",
"/picture", "/search/findBySearchTermAndPage").permitAll()
                    .antMatchers(HttpMethod.GET, "/navbar/**",
"/product/**", "/cart/**", "/category", "/user", "/about", "/contact",
"/login/forgotPassword", "/forgotPassword",
"/forgotPassword/changePassword").permitAll()
                    .antMatchers(HttpMethod.POST, "/cart/**",
"/register/newUser", "/search/findBySearchTerm").permitAll()
                .anyRequest().authenticated()
                .and()
                    .formLogin()
                    .loginPage("/login")
                    .defaultSuccessUrl("/buy").permitAll()
                .and()
                    .logout()
                    .logoutRequestMatcher(new
AntPathRequestMatcher("/logout")).logoutSuccessUrl("/login");
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(new
BCryptPasswordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

Listagem 22 - Arquivo de configuração de configuração de acesso as urls

Para que os arquivos HTML fiquem organizados, de fácil manutenção e sem duplicidade de código, foi implementada na aplicação o *framework* Freemarker que possibilita que pedaços de código que se repetem em várias páginas sejam separados em um arquivo e incluído por meio de uma *tag* em qualquer uma das interfaces HTML. Para isso, foi necessário incluir a dependência do *framework* por meio do Maven como pode ser visualizado na Listagem 23 e, também, a implementação de uma classe de configuração exibida na Listagem 24.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

Listagem 23 - Dependência Freemarker

```
@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {

    @Bean
    public ViewResolver viewResolver() {
        FreeMarkerViewResolver resolver = new FreeMarkerViewResolver();
        resolver.setCache(true);
        resolver.setPrefix("");
        resolver.setSuffix(".html");
        resolver.setContentType("text/html; charset=UTF-8");
        return resolver;
    }

    @Bean
    public FreeMarkerConfigurer freemarkerConfig() throws IOException,
TemplateException {
        FreeMarkerConfigurationFactory factory = new
FreeMarkerConfigurationFactory();
        factory.setTemplateLoaderPaths("classpath:templates",
"src/main/resources/templates");
        factory.setDefaultEncoding("UTF-8");
        FreeMarkerConfigurer result = new FreeMarkerConfigurer();
        result.setConfiguration(factory.createConfiguration());
        return result;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        VersionResourceResolver versionResourceResolver = new
VersionResourceResolver()
                .addVersionStrategy(new ContentVersionStrategy(), "/**");

        registry.addResourceHandler("/resource/**")
                .addResourceLocations("classpath:/static/")
                .setCachePeriod(60 * 60 * 24 * 365)
                .resourceChain(true)
                .addResolver(versionResourceResolver);
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

}
```

Listagem 24 - Classe de configuração freemarker

Na Listagem 25 é possível visualizar o *template navbar* que e responsável pela listagem do menu de categorias na loja, conforme exibido na Listagem 25.

```
<div class="navbar-vertical" id="navbar">
    <ul class="nav nav-stacked">
        <li class="header">
            <h6 class="text-uppercase">Categorias <i class="fa fa-navicon
pull-right"></i></h6>
        </li>
        <li v-for="nav in navList">
            <a class="dropdown-toggle" data-toggle="dropdown"
:href="'/category?id=' + nav.category.id + '&page=0'">
                {{nav.category.name}}<i class="fa fa-angle-right pull-
right"></i>
            </a>
            <ul v-if="nav.subCategory.length > 0" class="dropdown-menu">
                <li v-for="sub in nav.subCategory"><a
:href="'/category?id=' + sub.id + '&page=0'">{{sub.name}}</a></li>
            </ul>
        </li>
    </ul>
</div>
```

Listagem 25 - Template navbar

A Listagem 26 apresenta por meio de um trecho de código do arquivo index.html, como é realizada a chamada do *template navbar* nos demais arquivos HTML.

```
<!-- start section -->
<section class="section light-background">
    <div class="container">
        <div class="row">
            <div class="col-sm-4 col-md-3">
                <#include "/navbar.html" />
            </div><!-- end col -->
            <div class="col-sm-8 col-md-9">
                <div class="row">
                    <div class="col-sm-12">
                        <div class="owl-carousel slider owl-theme">
                            <div class="item">
                                <figure>
                                    <a href="javascript:void(0);">
                                        <img src="/img/slider/nvidia.jpg"
alt=""/>
                                    </a>
                                </figure>
                            </div><!-- end item -->
                            <div class="item">
                                <figure>
                                    <a href="javascript:void(0);">
                                        <img src="/img/slider/asus.jpg"
alt=""/>
                                    </a>
                                </figure>
                            </div><!-- end item -->
                            <div class="item">
                                <figure>
                                    <a href="javascript:void(0);">
                                        <img src="/img/slider/corsair.jpg"
alt=""/>
                                    </a>
                                </figure>
                            </div><!-- end item -->
                        </div><!-- end owl carousel -->
```

```
                    </div><!-- end col -->
                </div><!-- end row -->
            </div><!-- end col -->
        </div><!-- end row -->
    </div><!-- end container -->
</section>
<!-- end section -->
```

Listagem 26 - Chamada template navbar no arquivo index.html

4 CONCLUSÃO

O objetivo deste trabalho foi realizar o desenvolvimento de um *e-commerce* de produtos de hardware, periféricos e software de jogos. Diante das tecnologias disponíveis para desenvolvimento web, as utilizadas neste trabalho (como Angular e Spring), possibilitaram a integração entre o Angular e o microsserviço utilizando Spring e também a autenticação usando OAuth2. A junção dessas tecnologias possibilitou o desenvolvimento de um sistema de fácil manutenção e desenvolvimento ágil devido à estrutura implementada.

Durante o processo de desenvolvimento foram encontradas algumas dificuldades devido à falta de experiência com algumas tecnologias utilizadas. Para superar esses desafios foram consultadas bibliografias, tutoriais e fóruns para buscar informações e experiências vivenciadas por outros desenvolvedores, além dos conhecimentos adquiridos em sala de aula. Assim, a prática do desenvolvimento deste trabalho proporcionou conhecimento em novas tecnologias, como Angular, Angular CLI, PrimeNG, OAuth2 e Apache Freemarker.

Como trabalho futuro pretende-se aplicar a estrutura em *Cloud* utilizando os recursos do Spring e Netflix, pois possibilitam maior escalabilidade dos serviços caso o sistema tenha um grande volume de acessos. Além disso, pretende-se disponibilizar a funcionalidade para que o usuário possa compor o seu computador a partir de componentes de hardware disponíveis no sistema e, assim, adquirir uma máquina elaborada e montada de acordo as suas necessidades de configuração do usuário. Para tornar possível realizar pagamentos em ambiente de produção, como trabalho futuro também aplica-se a integração do sistema com um provedor de pagamentos.

REFERÊNCIAS

ALMEIDA, Flávio. **Mean**: Full stack JavaScript para aplicações *web* com MongoDB, Express, Angular e Node. São Paulo: Casa do Código, 2015

GUEDES, THIAGO. **Crie aplicações com Angular**: o novo *framework* do Google. São Paulo: Casa do Código, 2017.

LAUDON, Kenneth C. Sistemas de informações gerenciais: administração a empresa digital. Trad. **Arlete Simille Marques.** 5. ed. São Paulo: Pearson, 2004.

NAPIERALA Hieronim. As vantagens competitivas do comércio eletrônico para empresas de pequeno e médio porte. **FAE**, v.19, n1, p-68-79, Curitiba, 2016.

PEREIRA, Michael Henrique R. **AngularJS**: uma abordagem prática e objetiva. São Paulo: Novatec, 2014.

PRADO, Edmir Parada Vasques; SOUZA, César Alexandre. **Fundamentos de sistemas de informação**. Rio de Janeiro: Elsevier, 2014.

PRESSMAN, Roger S.; LOWE, David. Engenharia *Web*. Rio de Janeiro: LTC, 2009.

TURBAN, Efraim.; LEIDNER, Dorothy.; MCLEAN, Ephraim.; WETHERBE, James. **Tecnologia da Informação para gestão**. 6. Ed Porto Alegre: Bookman, 2010.