**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**
**CÂMPUS PATO BRANCO**
**CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**ANDRÉ LUIZ DONATTI COSTIM**

# SISTEMA WEB PARA COMERCIALIZAÇÃO DE PEÇAS AUTOMOTIVAS USADAS

**TRABALHO DE CONCLUSÃO DE CURSO**

**PATO BRANCO - PR**
**2017**

**ANDRÉ LUIZ DONATTI COSTIM**

# SISTEMA WEB PARA COMERCIALIZAÇÃO DE PEÇAS AUTOMOTIVAS USADAS

Trabalho de Conclusão de Curso de Graduação, apresentado à disciplina de Trabalho de Conclusão de Curso II, do Curso 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.

Orientadora: Profa. Beatriz Terezinha Borsoi

**PATO BRANCO - PR**
**2017**

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

## SISTEMA WEB PARA COMERCIALIZAÇÃO DE PEÇAS AUTOMOTIVAS USADAS

**POR**

**ANDRÉ LUIZ DONATTI COSTIM**

Este trabalho de conclusão de curso foi apresentado no dia 05 de dezembro de 2017, 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 MSc Andreia Scariot Beulke

Prof. MSc. Vinicius Pegorini

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

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

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

## RESUMO

COSTIM, André Luiz Donatti. Sistema *web* para comercialização de peças automotivas usadas. 85f. 2017. Monografia de Trabalho de Conclusão de Curso - Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

É comum que as peças utilizadas na manutenção de veículos sejam provenientes de reúso, ou seja, que elas venham de outros veículos. Esses podem ser veículos danificados em acidentes que não serão consertados ou veículos retirados de circulação, mas que possuem peças e/ou partes que podem ser reutilizadas. Considerando a diversidade de montadoras e de marcas de veículos e a quantidade de estabelecimentos que realizam manutenção de veículos, localizar a peça adequada ou com a melhor relação custo x benefício pode ser difícil. Os recursos oferecidos pelas tecnologias de informação e comunicação podem auxiliar na localização de peças e, assim, consertos podem ser realizados de maneira mais rápida, barata e efetiva. Um sistema *web* que permita que fornecedores anunciem peças e compradores possam pesquisar essas peças facilita a compra e a venda, agiliza a localização da peça necessária, possibilita comparação de preço, qualidade e prazo, entre e outros e permite obter a melhor relação custo x benefício. Neste trabalho de conclusão de curso é apresentada a modelagem e o desenvolvimento de um sistema *web* que visa auxiliar no processo de compra e venda de peças automotivas usadas. O objetivo é que compradores e fornecedores dessas peças tenham um local virtual (um sistema *web*) para realizar transações desse tipo de comércio: anunciar peças, localizar peças, negociar e realizar a transação de compra e venda.

**Palavras-chave**: Sistema web. Comércio de peças usadas. Rich Internet Application

## ABSTRACT

COSTIM, André Luiz Donatti. Web system for exchanging of used vehicles parts. 85f. 2017. Monografia de Trabalho de Conclusão de Curso - Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

It is very common that in the maintenance of the parts being repaired be originated from reuse and with that said, they have come from vehicles in disuse. This include vehicles damaged in accidents that will not be fixed and vehicles taken from circulation but that still has parts that can be reused in repair of other vehicles. Having in mind that different vehicles manufacturers exist, the vehicles brands and the diverse conditions that parts coming from reuse can be found, finding the parts that are more appropriate or with the better cost vs benefit condition can be tough. Anyhow, the resources offered with information technologies and communication can provide help in the search for parts to use in repairs, in a fast and more effective way to repairs be done. A web system that allows different providers to advertise parts and to different buyers be able to search for parts make easier the process of buy and sell and decrease the time spent to find the needed part. In this text is presented the modeling and development of a web system that has the intention to help the process of buy and sell of used parts of vehicles. The objective is that buyers and sellers of vehicle parts have a virtual place to start and conclude transactions by themselves.



**Keywords**: Web System. Exchange of used vehicles parts. Rich Internet Application.

## LISTA DE SIGLAS

| | |
|---|---|
| CNPJ | Cadastro Nacional de Pessoa Jurídica |
| CSS | Cascading Style Sheets |
| CTB | Código de Trânsito Brasileiro |
| DENATRAN | Departamento Nacional de Trânsito |
| DER | Diagrama de Entidade e Relacionamento |
| HTML | HyperText Markup Language |
| IBGE | Instituto Brasileiro de Geografia e Estatística |
| IDE | Integrated Development Environment |
| RIA | Rich Internet Application |
| RF | Requisito Funcional |
| RNF | Requisito não Funcional |
| UML | Unified Modeling Language |

## LISTA DE FIGURAS

Figura 1 – Diagrama de casos de uso....23
Figura 2 – Diagrama de classes de análise do sistema....29
Figura 3 – Diagrama de entidades e relacionamentos do banco de dados....32
Figura 4 – Página de cadastro de usuário....37
Figura 5 – Página de cadastro da empresa....38
Figura 6 – Página de informações do usuário....38
Figura 7 – Página de login....39
Figura 8 – Página de busca....40
Figura 9 – Página de resultado de busca....41
Figura 10 – Página de visualização de peça da busca....41
Figura 11 – Página de estoque....42
Figura 12 – Página de relatório de peças....43
Figura 13 – Página de visualização de peça do estoque....43
Figura 14 – Página de edição de peça....44
Figura 15 – Página de cadastro de peça....45
Figura 16 – Página do chat....45
Figura 17 – Página de informações da empresa....46
Figura 18 – Código da estrutura HTML do cabeçalho....47
Figura 19 – Código da lógica do componente do cabeçalho....47
Figura 20 – Código da lógica do componente do cabeçalho....48
Figura 21 – Código da lógica do componente do cabeçalho....49
Figura 22 – Código da estrutura HTML do componente do rodapé....49
Figura 23 – Código da lógica do componente do rodapé....50
Figura 24 – Código da estrutura HTML da tela de login....51
Figura 25 – Código da lógica do componente de login....52
Figura 26 – Código da lógica do componente de login....52
Figura 27 – Código da estrutura HTML da tela de cadastro de usuário....53
Figura 28 – Código da estrutura HTML da tela de cadastro de usuário....53
Figura 29 – Código da estrutura HTML da tela de cadastro de usuário....53
Figura 30 – Código da estrutura HTML da tela de cadastro de usuário....54
Figura 31 – Código da lógica do componente de cadastro de usuário....54
Figura 32 – Código da lógica do componente de cadastro de usuário....54
Figura 33 – Código da lógica do componente de cadastro de usuário....55
Figura 34 – Código da lógica do componente de cadastro de usuário....55
Figura 35 – Código da estrutura HTML da tela de Chat....56
Figura 36 – Código da estrutura HTML da tela de Chat....56
Figura 37 – Código da estrutura HTML da tela de Chat....57
Figura 38 – Código da lógica do componente de Chat....57
Figura 39 – Código da lógica do componente de Chat....58
Figura 40 – Código da lógica do componente de Chat....58
Figura 41 – Código da lógica do componente de Chat....59
Figura 42 – Código da lógica do componente de Chat....60
Figura 43 – Código da lógica do componente de Chat....60
Figura 44 – Código da estrutura HTML da tela Home....61
Figura 45 – Código da estrutura HTML da tela Home....61
Figura 46 – Código da lógica do componente da Home....62

Figura 47 – Código da lógica do componente da Home ..... 62
Figura 48 – Código da lógica do Serviço de Autenticação ..... 63
Figura 49 – Código da lógica do Serviço de Autenticação ..... 63
Figura 50 – Código da lógica do Serviço de Autenticação ..... 63
Figura 51 – Código da lógica do Serviço de Autenticação ..... 64
Figura 52 – Código da lógica do Serviço de Autenticação ..... 64
Figura 53 – Código da lógica do Serviço de Autenticação ..... 65
Figura 54 – Código da lógica do Serviço de Autenticação ..... 65
Figura 55 – Código da lógica do Serviço de Autenticação ..... 66
Figura 56 – Código da lógica do Serviço de Autenticação ..... 67
Figura 57 – Código da lógica do Serviço de Autenticação ..... 67
Figura 58 – Código da lógica do Serviço de Autenticação ..... 68
Figura 59 – Template de envio do Código Validador ..... 69
Figura 60 – Código da lógica do Serviço de Mensagens ..... 69
Figura 61 – Código da lógica do serviço de mensagens ..... 70
Figura 62 – Código da lógica do serviço de mensagens ..... 70
Figura 63 – Código da lógica do serviço de mensagens ..... 71
Figura 64 – Código da lógica do serviço de peças ..... 71
Figura 65 – Código da lógica do serviço de peças ..... 72
Figura 66 – Código da lógica do serviço de peças ..... 72
Figura 67 – Código da lógica do serviço de peças ..... 73
Figura 68 – Código da lógica do serviço de peças ..... 74
Figura 69 – Código da lógica do Serviço de Peças ..... 75
Figura 70 – Código da lógica do Serviço de Peças ..... 76
Figura 71 – Código da lógica do Serviço de Peças ..... 77
Figura 72 – Código da lógica do Serviço de Peças ..... 77
Figura 73 – Código da lógica do Módulo de Rotas ..... 78
Figura 74 – Código da lógica do Módulo de Rotas ..... 78
Figura 75 – Código da lógica do Módulo de Rotas ..... 79
Figura 76 – Código da lógica do Módulo de Principal ..... 79
Figura 77 – Código da lógica do Módulo de Principal ..... 80
Figura 78 – Código da lógica do Módulo de Principal ..... 80
Figura 79 – Código da lógica do Módulo de Principal ..... 80
Figura 80 – Código da lógica do Módulo de Principal ..... 81
Figura 81 – Código da lógica do Módulo de Principal ..... 81
Figura 82 – Código da estrutura lógica do componente principal ..... 82
Figura 83 – Código da estrutura HTML do componente principal ..... 82
Figura 84 – Código da estrutura HTML da página index ..... 82

## LISTA DE TABELAS E QUADROS

Tabela 1 – Quantidade de veículos da frota brasileira em 2015 ..... 10

Quadro 1 - Ferramentas e tecnologias utilizadas ..... 16
Quadro 4 – Caso de uso realizar cadastro ..... 24
Quadro 5 – Caso de uso realizar login ..... 25
Quadro 6 – Caso de uso manter peça ..... 25
Quadro 7 – Caso de uso realizar busca ..... 26
Quadro 8 – Caso de uso enviar mensagens ..... 26
Quadro 9 – Caso de uso visualizar estoque ..... 27
Quadro 10 – Caso de uso visualizar peça ..... 27
Quadro 11 – Caso de uso enviar sugestões ..... 28
Quadro 12 – Caso de uso gerar relatórios ..... 28
Quadro 13 – Descrição da classe Fone ..... 29
Quadro 14 – Descrição da classe Empresa ..... 30
Quadro 15 – Descrição da classe Endereço ..... 30
Quadro 16 – Descrição da classe Usuário ..... 30
Quadro 17 – Descrição da classe Peça ..... 31
Quadro 18 – Descrição da classe Marca ..... 31
Quadro 19 – Descrição da classe Estado ..... 31
Quadro 20 – Descrição da classe Mensagem ..... 32
Quadro 21 – Descrição da classe Sugestão ..... 32
Quadro 23 – Campos da tabela Contato ..... 33
Quadro 24 – Campos da tabela Empresa ..... 33
Quadro 25 – Campos da tabela Endereço ..... 34
Quadro 26 – Campos da tabela Usuário ..... 34
Quadro 27 – Campos da tabela Peça ..... 34
Quadro 28 – Campos da tabela Marca ..... 35
Quadro 29 – Campos da tabela Estado ..... 35
Quadro 30 – Campos da tabela Mensagem ..... 35
Quadro 31 – Campos da tabela Sugestão ..... 36
Quadro 32 – Campos da tabela Código Validador ..... 36

**SUMÁRIO**

**1 INTRODUÇÃO** ..... **10**
1.1 CONSIDERAÇÕES INICIAIS ..... 10
1.2 OBJETIVOS ..... 12
1.2.1 Objetivo Geral ..... 12
1.2.2 Objetivos Específicos ..... 12
1.3 JUSTIFICATIVA ..... 12
1.4 ESTRUTURA DO TEXTO ..... 13
**2 RICH INTERNET APPLICATION** ..... **14**
**3 MATERIAIS E MÉTODO** ..... **16**
3.2 MÉTODO ..... 18
**4 RESULTADO** ..... **21**
4.1 ESCOPO DO SISTEMA ..... 21
4.2 MODELAGEM DO SISTEMA ..... 22
4.3 APRESENTAÇÃO DO SISTEMA ..... 36
4.4 IMPLEMENTAÇÃO DO SISTEMA ..... 46
**5 CONCLUSÃO** ..... **83**
**REFERÊNCIAS** ..... **84**

# 1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, o objetivo e a justificativa de realização deste trabalho.

## 1.1 CONSIDERAÇÕES INICIAIS

A manutenção de veículos é um tipo de negócio que, tradicionalmente, faz reúso de peças. É comum que oficinas mecânicas, chapeações, autopeças e outros do ramo de manutenção de veículos façam uso de peças provenientes, por exemplo, de veículos que sofreram acidentes e que foram descartados da possibilidade de conserto pela severidade ou pela quantidade de danos, por opção da seguradora ou do proprietário, entre outros.

A Tabela 1 apresenta dados do Ministério das Cidades, Departamento Nacional de Trânsito (DENATRAN) de 2015, divulgados pelo Instituto Brasileiro de Geografia e Estatística (IBGE) em 2016 sobre a frota brasileira de veículos, fornecendo sustentação à afirmação da grande quantidade de veículos existente no País e consequentemente do volume de peças necessárias e disponíveis para reposição.

**Tabela 1 – Quantidade de veículos da frota brasileira em 2015**

| Tipo de veículo | Quantidade |
|---|---|
| Automóveis | 49.822.708 |
| Caminhões | 2.645.992 |
| Caminhões-trator | 593.892 |
| Caminhonetes | 6.588.813 |
| Camionetas | 2.908.233 |
| Micro-ônibus | 375.274 |
| Motocicletas | 20.216.193 |
| Motonetas | 3.833.159 |
| Ônibus | 590.657 |
| Tratores | 30.371 |
| Utilitários | 637.211 |

Fonte: IBGE (2016, p. 1).

A categorização e respectiva descrição ou definição para os tipos de veículos da Tabela 1 é definida pelo Código de Trânsito Brasileiro (CTB).

Com uma frota bastante expressiva, como indicam os dados da Tabela 1, com idade média de 8 anos e 8 meses, de acordo com dados de 2014 apresentados

em Leite (2015), e considerando a quantidade de acidentes que ocorrem, o mercado de peças para reposição e de veículos para conserto é bastante expressivo. Dados das Organizações das Nações Unidas no Brasil (ONUBR) colocam o País em primeiro lugar na América do Sul no número de acidentes de trânsito com morte por habitante (ORGANIZAÇÕES..., 2015). Paralelamente, os dados divulgados no relatório do mercado de reposição de peças, elaborados com base em informações oferecidas por empresas associadas ao Sindicato Nacional da Indústria de Componentes para Veículos Automotores (Sindipeças), apontaram crescimento de 2,19% do faturamento desse mercado em 2016 em relação a 2015 (SINDICATO..., 2017).

Em decorrência da quantidade de veículos existente e envolvida em acidentes de trânsito e da diversidade de tipos, marcas e montadoras de veículos, localizar uma determinada peça usada que é necessária em um conserto pode ser trabalhoso.

Considerando esse contexto, do número de veículos que representam a frota brasileira e da quantidade de veículos envolvidos em acidentes, alimentando um amplo mercado de reposição de peças, um sistema *web* para cadastro de peças que possa ser utilizado por várias empresas é uma forma de disponibilizar peças usadas para venda e facilitar a localização de peças para compra. É nesse contexto que foi implementado, como resultado deste trabalho, um aplicativo para *web* para que chapeações, auto-peças, mecânicas e negócios do ramo possam comercializar peças usadas.

No sistema desenvolvido, as peças são cadastradas e os dados do cadastro podem ser alterados ou o cadastro da respectiva peça excluído somente por quem realizou o cadastro. A consulta é realizada em todos os registros de peças da base de dados, mas somente por usuários cadastrados no sistema.

Uma primeira versão da modelagem do sistema implementado como resultado da realização deste trabalho foi desenvolvida como trabalho de estágio curricular supervisionado obrigatório pelo autor deste texto. Neste trabalho de conclusão de curso, essa modelagem foi revista e complementada e o sistema foi implementado.

## 1.2 OBJETIVOS

A seguir são apresentados os objetivos deste trabalho de conclusão de curso.

### 1.2.1 Objetivo Geral

Implementar um sistema *web* para compra e venda de peças de veículos, realizadas entre os usuários cadastrados.

### 1.2.2 Objetivos Específicos

A solução proposta para o sistema terá como finalidades:

- Fornecer filtros de busca que facilitem a consulta das peças cadastradas no sistema.
- Possibilitar que usuários cadastrados possam enviar mensagens para outros usuários, podendo negociar sobre compra e venda de peças.
- Gerar relatórios baseados em informações do sistema filtradas para o usuário ter um melhor controle de seu estoque.

## 1.3 JUSTIFICATIVA

Os dados de quantidade de veículos e de movimentação do mercado de peças permitem deduzir que é bastante expressivo o número de peças de automóveis que são reutilizadas no conserto e no reparo de veículos. Utilizar uma aplicação *web* para divulgar peças cadastradas e possibilitar a busca nessa base é uma forma de contribuir para o comércio de peças usadas. Essa aplicação proverá uma maneira de promover o reuso dessas peças e de agilizar a sua localização.

Um aplicativo que permita que peças sejam cadastradas e que apenas quem as cadastrou possa alterar ou excluir dados do cadastro e que usuários cadastrados

possam realizar buscas na base de dados, visa auxiliar na localização da peça necessária ou as mais adequadas para o conserto a ser realizado.

O sistema proposto como resultado deste trabalho tem como propósito possibilitar o cadastro de peças de veículos, em princípio usadas, e facilitar a sua busca. O objetivo é favorecer o comércio de peças usadas, mas não será impedido o cadastro de peças novas. Um proprietário de veículo ou uma pessoa pode ter adquirido uma peça nova e ela não ter sido utilizada, por algum motivo, e ele deseja vendê-la.

## 1.4 ESTRUTURA DO TEXTO

Este texto está organizado em capítulos. Este é o primeiro e apresenta as considerações iniciais com o contexto do sistema desenvolvido, os seus objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico centrado em aplicações *web*. No Capítulo 3 estão as ferramentas e as tecnologias utilizadas para a modelagem e a implementação do sistema. No Capítulo 4 é apresentado o resultado da realização do trabalho, ou seja, a modelagem e a implementação de um sistema *web* para comercialização de peças de veículos. Por fim, estão as considerações finais seguidas das referências utilizadas no texto.

# 2 RICH INTERNET APPLICATION

O termo *Rich Internet Application* (RIA) foi proposto em um White Paper da Macromedia em 2002, que apresentou o Flash MX da Macromedia (JEREMY, 2002). Para Hooshmand et al. (2014) as RIAs têm se tornado a norma para aplicações *web* modernas. Eles citam como exemplo a Google que tem desenvolvido a maioria dos seus principais produtos (Gmail, Google Groups, GoogleMaps e etc.) usando tecnologias como Ajax e JavaScript que caracterizam as RIAs.

RIAs são qualitativamente caracterizadas como aplicações *web* que visam prover características e funcionalidades das aplicações *desktop* tradicionais (HOOSHMAND et al., 2014). Elas empenham-se em fornecer aplicações mais responsivas, com capacidade de interação e interface melhoradas, visando oferecer uma experiência mais rica para o usuário (CASTELEYN; GARRIGOS; MAZÓN, 2014). Meliá et al. (2010) ressaltam que as RIAs combinam os melhores benefícios de distribuição e de manutenção providos pelas aplicações *web* enquanto suportam uma interface com o cliente mais rica, à semelhanças das aplicações *desktop.* Essas aplicações apresentam funcionalidades para o usuário como arrastar e soltar e oferecem elementos de interface e efeitos bastante diferenciados se comprados às aplicações *web* tradicionais, as baseadas em hipertexto, *links* e formulários simples. As RIAS agregam o melhor das aplicações *web* e *desktop*.

Para desenvolvimento as RIAs contam com várias tecnologias no lado cliente combinadas com comunicação assíncrona com o servidor (CASTELEYN; GARRIGOS; MAZÓN, 2014). As RIAs transferem a maior parte da carga de processamento da interface com o usuário para o cliente enquanto a maior parte dos dados (dos controles e de manutenção dos dados de negócio) permanecem no servidor da aplicação (MARTÍNEZ-RUIZ; ARTEAGA; VANDERDONCKT; GONZÁLEZ-CALLEROS, 2006).

Cliente *web* rico são RIAs que utilizam tecnologias que permitem forte interatividade do cliente para prover aos usuários uma melhor experiência no uso das aplicações web (BI-FENG, 2011). Essa interatividade torna a acessibilidade uma grande preocupação da *web*, considerando que todos deveriam ser capazes de acessar e interagir com uma página *web* (FERNANDES, et al, 2012).

As tecnologias para implementar clientes ricos incluem Ajax, JavaFX, Silverlight e a plataforma Adobe Flash (LABRIOLA; TAPPER; BOLES, 2011). As três últimas tecnologias são baseadas em *plugins* que devem ser instalados para desenvolver e executar a aplicação. Por exemplo: Flash nas páginas *web* precisa do Flash Player para executar, mas a Adobe parou de fornecer suporte para Flash Player para dispositivos móveis. E as aplicações com clientes ricos baseadas em *plugins* apresentam pouca compatibilidade com diferentes tipos de terminais. Além disso, o uso de *plugins* afeta o tempo de carga da página (LI-LI; ZHENG-LONG, 2012).

Ajax, por sua vez, é baseada em *HyperText Markup Language* (HTML) e não necessita de instalação de software extra para desenvolver ou executar a aplicação *web*. Por meio de comunicação assíncrona com o servidor interage com o cliente, mas a conexão com o servidor necessita de recursos de rede e de hardware, então quanto mais o cliente é atualizado, mais a taxa de resposta é afetada. Além disso, Ajax não pode trabalhar no modo *off-line* (LI-LI; ZHENG-LONG, 2012).

Como Ajax, HTML é um tipo de tecnologia cliente não baseada em *plugin*, mas isso pode não somente melhorar a interatividade com o usuário pela redução de dados extra transmitidos em cada requisição, mas, também, permite armazenar dados no cliente e reduzir o tempo de conexão entre o cliente e o servidor (LI-LI; ZHENG-LONG, 2012).

Meliá et al. (2010) destacam que as RIAs introduzem novas características arquiteturais no campo das aplicações *web* tradicionais. Esses autores ressaltam que os desenvolvedores de RIAs devem tomar diversas decisões arquiteturais. E que o desafio real desses desenvolvedores reside na escolha das melhores alternativas entre as variações de arquiteturas e de tecnologias existentes para as RIAs visando prover a melhor solução que atenda todos os requisitos do cliente.

# 3 MATERIAIS E MÉTODO

Este Capítulo apresenta as tecnologias e as ferramentas utilizadas para a modelagem e a implementação do aplicativo desenvolvido como resultado da realização deste trabalho. Neste Capítulo também são apresentadas as principais atividades para realizar essa modelagem e implementação.

No Quadro 1 as ferramentas e as tecnologias utilizadas para realizar a análise e o desenvolvimento da aplicação.

| **Ferramenta / Tecnologia** | **Referência (site)** | **Finalidade** |
|---|---|---|
| Angular 2 | https://angular.io/ | *Framework front-end*. |
| Font Awesome | http://fontawesome.io/ | Ícones para o site. |
| HTML5 | http://www.w3schools.com/html/ | Linguagem para criação da estrutura das páginas do site. |
| Firebase | https://firebase.google.com/ | Banco de dados. |
| AngularFire 2 | https://www.firebase.com/docs/web/libraries/angular/api.html | *Framework backend* para conectar dados do Angular com o banco Firebase. |
| Visual Studio Code | https://code.visualstudio.com | *Integrated Development Environment* de desenvolvimento. |
| Visual Paradigm | https://www.visual-paradigm.com | Diagramação do projeto: modelagem do sistema. |
| Axure RP | https://www.axure.com/ | Para prototipagem das telas. |
| Bootstrap 3 | https://getbootstrap.com/ | Estilização de componentes. |

**Quadro 1 - Ferramentas e tecnologias utilizadas**

A seguir está uma breve descrição das tecnologias apresentadas no Quadro 1.

**a) Angular 2**

O Angular 2 é um *framework front-end* que possibilita o desenvolvimento de aplicações *web* e *mobile*. Ele é desenvolvido e mantido por uma equipe específica da Google e também por meio de contribuições de sua comunidade.

De acordo com Booth (2017, p. 17), o Angular 2 auxilia obter o máximo proveito dos recentes desenvolvimentos dos navegadores *web* para que aplicações melhores sejam desenvolvidas.

O Angular 2 ganha destaque se comparado a outros *frameworks front-end* e ao seu predecessor o AngularJS, por ter uma construção modular de seu código e ser codificado em TypeScript. Com TypeScript é possível escrever código utilizando uma estrutura fortemente tipada e ter esse código compilado para JavaScript puro. O código é escrito no padrão ECMAScript 5 que é suportado por todos os navegadores *web* recentes.

**b) Font Awesome**

O Font Awesome é um conjunto de ferramentas de fontes e ícones (FONT AWESOME, 2017). Esse *kit* auxilia a gerar uma visão mais profissional do site, com fontes, estilos em *Cascading Style Sheets* (CSS) e ícones que trazem destaque e maior interação entre o sistema e o usuário.

**c) Firebase**

O Firebase é um *framework backend* gratuito para aplicações *web* e *mobile*. Firebase inclui banco de dados em tempo real, hospedagem de site, funções na nuvem, possibilidade de realizar análises sobre dados (FIREBASE, 2017).

**d) Visual Paradigm**

O Visual Paradigm facilita a realização da análise de sistemas por meio padrão de modelagem *Unified Modeling Language* (UML). Para ALHIR (1999), a UML é uma linguagem de modelagem utilizada para especificar, visualizar, construir e documentar artefatos de um sistema.

No desenvolvimento da modelagem deste trabalho, o Visual Paradigm foi utilizado para construir os diagramas de casos de usos, de Entidade e Relacionamentos (DER) e de classes.

**e) Visual Code Studio**

O Visual Studio Code é um *Integrated Development Environment* (IDE) ou plataforma de desenvolvimento da Microsoft. Esse editor foi lançado pela Microsoft em 2015 e alguns meses depois anunciado como *open source* (DEVMEDIA, 2017).

O Visual Code Studio possui suporte a TypeScript e contém um terminal integrado, como se fosse o *prompt* de comando do Windows, facilitando o uso de

comandos para instalar dependências necessárias e testar a execução do aplicativo em desenvolvimento.

**f) AngularFire 2**

O AngularFire 2 é uma biblioteca de funções criada para facilitar a interação entre o *framework* Angular 2 e a base de dados Firebase, utilizando a biblioteca RxJS (THINKSTER, 2017). Com o uso dessa biblioteca, a aplicação sendo construída reage às mudanças, como eventos de clique, dados sendo acessados e alterações na base de dados, entre outras (GITHUB, 2017).

**g) HTML5**

*HyperText Markup Language* (HTML) é um modo de descrever documentos interligados por *links*. Os elementos do HTML 5 possuem semântica, ou seja, significam algo no contexto de uso. Exemplos: parágrafo, cabeçalho nível 1, lista não ordenada (CROWTHER, 2013).

**h) Axure RP**

A Axure RP é uma ferramenta de prototipação que reúne um conjunto de funcionalidades que possibilitam uma rápida criação de protótipos, *wireframes* e diagramas (AXURE, 2017).

**i) Bootstrap 3**

O Bootstrap 3 reúne um conjunto de classes de estilização que pode ser aplicado aos componentes HMTL provendo ao sistema uma imagem mais profissional e visando melhorar a experiência do usuário (BOOTSTRAP, 2017).

## 3.2 MÉTODO

Os passos para realizar o trabalho estão apresentados a seguir.

**a) Levantamento de requisitos**

O levantamento de requisitos foi realizado a partir de conversas informais com um empresário e seu gerente da área de conserto de veículos e martelinho de ouro. Por meio dessas conversas foi identificada a necessidade de um sistema com base no seguinte contexto: muitas vezes é realizado o conserto em um veículo e nesse conserto sobram peças não danificadas, mas a seguradora ou o proprietário do

veículo requisita a troca. Visualizou-se, assim, a possibilidade de reusar essas peças em outros consertos ou vendê-las para outras oficinas.

Com a realização de diversas reuniões foi discutido como essa ideia poderia evoluir para um sistema. A partir disso foram definidos os requisitos e criados os protótipos de telas que estão descritos na Seção 4.3. O processo fundamental de negócio envolvido é de um sistema *on-line* em um site, para que peças sejam cadastradas e fiquem disponíveis para consulta por outros usuários do sistema. Estabeleceu-se que o sistema deveria ser de uso fácil e funcional. Nessas reuniões foram documentadas que a negociação de venda seria realizada entre o vendedor (quem está oferecendo a peça) e o comprador interessado, ou seja, o sistema não faria automaticamente a venda, como ocorre em um comércio eletrônico.

**c) Análise e projeto do sistema**

Após realizada a coleta de informações, gerando o levantamento de requisitos, foram utilizados conceitos da UML para desenvolver os diagramas de casos de uso, de classes e o DER, a expansão dos casos de uso e a documentação das classes e das tabelas. Buscando, assim, um melhor entendimento das necessidades do sistema e do seu comportamento após finalizado o desenvolvimento.

No diagrama de casos de uso são apresentadas as funcionalidades que o usuário terá acesso ao utilizar o sistema e na expansão de cada caso de uso está descrita como será a interação do usuário com o sistema e como cada funcionalidade se comportará. Após a elaboração do diagrama de classes foi realizada a expansão de cada classe com a listagem de seus atributos e a descrição de cada um dos seus métodos. O DER apresenta a estrutura do banco de dados do sistema, em seguida é apresentada cada tabela com seus campos e seus relacionamentos com outras tabelas. Os diagramas e suas expansões construídas a partir dos requisitos estão documentados na Seção 4.3.

**d) Implementação**

Completadas as fases de levantamento de requisitos e de análise e projeto do sistema foi realizada a fase de desenvolvimento utilizando as informações levantadas bem como os protótipos das telas definidos e os casos de uso e os diagramas gerados.

O desenvolvimento do sistema foi totalmente baseado em componentes feitos em Angular2, sendo que cada componente possui a sua parte lógica e o seu leiaute que é visível ao usuário. Este leiaute das telas foi desenvolvido com as linguagens HTML5 para a estruturação da página e CSS3 e Bootstrap para a sua estilização.

Com exceção das telas de *login* e de cadastro do usuário, todas as outras telas compartilham de um único arquivo de estilização. Visando, assim, manter uma melhor organização e padronização do *design* das telas.

O processamento e a busca de dados foram implementados em serviços do Angular2. Os componentes poderão importar esses serviços e declarar uma variável para ter acesso aos métodos implementados no serviço, como, por exemplo, realizar o *login* de um usuário, sendo que tanto os componentes como os serviços foram escritos nas linguagens Typescritpt e JavaScript.

Os dados são armazenados na base de dados Firebase com o uso do *framework* AngularFire2. Esse *framework* possui os métodos para a comunicação entre a aplicação e a base de dados para a inserção, a edição e a consulta de informações.

**e) Testes**

A fase de testes ocorreram em paralelo com o desenvolvimento, testes foram realizados paralelamente ao desenvolvimento do sistema, visando verificar se as funcionalidades estavam sendo implementadas de acordo com o especificado na análise realizada a partir dos requisitos definidos.

Para os testes de uso, usuários que não possuíam conhecimento de programação utilizaram o sistema, visando verificar como é a interação com o sistema e para avaliar se a interface é de fácil utilização. O sistema também foi utilizado por usuários com conhecimento de programação para encontrar possíveis *bugs*.

# 4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho que é o desenvolvimento de um sistema *web* para comercialização de peças de veículos usadas.

## 4.1 ESCOPO DO SISTEMA

O sistema automatizará o processo de negócio utilizado para fazer a busca de peças usadas por chapeações, autopeças, mecânicas e negócios do ramo. A solução proposta considera o contexto apresentado a seguir.

Para utilizar o sistema para cadastrar peças e enviar mensagens para outros usuários é necessário possuir um *login* de acesso. Ao realizar seu cadastro, o usuário será direcionado para a tela de cadastro da empresa.

O usuário autenticado poderá realizar o cadastro de peças, sendo que ele terá acesso para edição e exclusão apenas das peças por ele cadastradas. Desse modo, peças que tenham sido cadastradas por outros usuários, serão disponibilizadas apenas para consulta e visualização de informações.

O acesso às peças de cada usuário será realizado por uma tela de estoque. Para visualizar as peças cadastradas por outras pessoas é necessário realizar uma busca utilizando palavras-chaves que descrevam a peça desejada. A busca poderá ser filtrada por marcas e localização (Estado) do fornecedor da peça.

Após realizar uma busca e uma peça ser selecionada, uma tela com as informações da referida peça é apresentada. E o usuário terá a opção de entrar em contato com o usuário que cadastrou a peça para ter mais informações e para negociar a compra, se for o caso.

O sistema possibilitará que os usuários enviem mensagens para o suporte do sistema para reportar erros ou indicar melhorias que futuramente poderão ser implementadas.

## 4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta os requisitos funcionais identificados para o sistema. Nesse quadro RF significa Requisito Funcional.

| Identificação | Nome | Descrição |
|---|---|---|
| RF 01 | Cadastrar empresa | Realizar o cadastro da empresa com dados que permitam que seja criado o usuário para autenticação no sistema. |
| RF 02 | Cadastrar peça | Cadastrar as peças que serão disponibilizadas para venda, contendo, além de dados de cadastro, fotos para a exibição. |
| RF 03 | Enviar mensagens | O usuário autenticado poderá entrar em contato com outro usuário por meio de mensagens enviadas pelo site. |
| RF 04 | Visualizar estoque | O usuário autenticado poderá visualizar o seu estoque de peças cadastradas, sendo ativas e inativas. |
| RF 05 | Visualizar peça | O usuário, após a busca, poderá selecionar uma peça do resultado da busca e ter acesso às informações cadastradas da referida peça. |
| RF06 | Enviar sugestões | O usuário poderá enviar *email* para o administrador do sistema para enviar reclamações, reportar erros do sistema ou realizar sugestões para melhorias. |
| RF 07 | Gerar relatórios | O usuário terá a possibilidade de realizar a emissão de relatórios para um melhor gerenciamento das peças que ele cadastrou no sistema. |
| RF 09 | Editar empresa | O usuário autenticado poderá realizar a alteração de dados cadastrais da empresa vinculada ao seu cadastro. |
| RF 10 | Realizar *login* | O usuário criado para a empresa poderá realizar o *login* no sistema. |
| RF 11 | Realizar busca | O usuário autenticado realizará buscar por peças, recebendo uma lista com os resultados de sua busca. |
| RF 12 | Editar peça | O usuário somente poderá realizar a edição dos dados de cadastro de uma peça que ele tenha cadastrado. |
| RF 13 | Excluir peça | O usuário somente poderá realizar a exclusão de uma peça que ele tenha cadastrado. |

**Quadro 2 – Requisitos funcionais**

Os requisitos não funcionais identificados estão listados no Quadro 3. Nesse quadro RNF significa Requisito não Funcional.

| Identificação | Nome | Descrição |
|---|---|---|
| RNF 01 | Acesso ao sistema | O acesso ao sistema será realizado por meio de *login* e senha. |
| RNF 02 | Alterações | Cada usuário terá acesso à edição e à exclusão apenas das peças cadastradas por ele. |
| RNF 03 | Verificar *status* | Na busca, o sistema deve listar apenas peças que estejam com o *status* ‘ativo’. |
| RNF 04 | Filtros de busca | O sistema deve respeitar os filtros que o usuário utilizar na sua |

| | | |
|---|---|---|
| | | busca, retornando somente os resultados que coincidam com os critérios da busca. |
| RNF 05 | Visualização de estoque | Ao visualizar o seu estoque, o usuário terá acesso somente às peças por ele cadastradas podendo visualizar peças cadastradas por outros usuários apenas pela utilização da busca. |
| RNF 06 | Cadastro da empresa | No cadastro da empresa, ao ser indicado o Cadastro Nacional de Pessoa Jurídica (CNPJ) deve ser verificado se ele é válido e retornar as informações sobre a empresa preenchendo automaticamente os campos existentes. |
| RNF 07 | Visualização de peça | Ao visualizar uma peça por meio de uma busca, o usuário deve ter acesso apenas às informações (consulta). As operações de edição e exclusão de peças devem ser realizadas pela funcionalidade de estoque do sistema. |
| RNF 08 | Segurança | O sistema deverá impedir que uma nova empresa seja cadastrada com um CNPJ que já esteja cadastrado na base de dados, permitindo apenas a alteração desses dados somente com usuário autenticado no sistema. |

**Quadro 3 – Requisitos não funcionais**

O diagrama de casos de uso apresentado na Figura 1 contém as principais funcionalidades do sistema. Os usuários definidos como comprador/vendedor são os responsáveis por realizar o seu cadastro, fazer *login*, manter as peças, realizar buscas, enviar mensagens ou sugestões, visualizar estoque ou dados das peças e gerar relatórios.

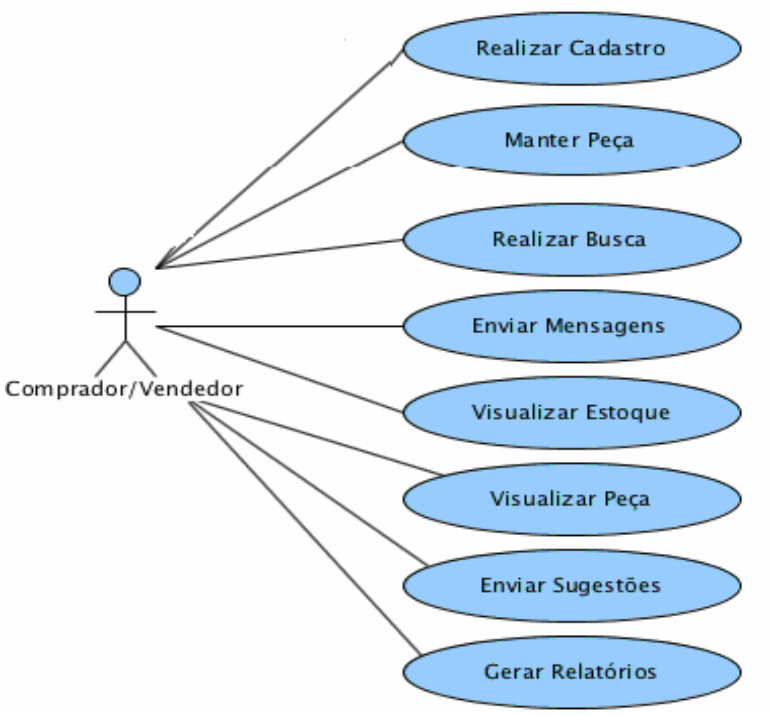


**Figura 1 – Diagrama de casos de uso**

No Quadro 4 está a descrição do caso de uso Realizar Cadastro.

**Caso de uso:**
Realizar Cadastro.
**Descrição**:
Para o usuário poder utilizar o sistema ele necessitará criar *login* e senha. Para o cadastro é necessário o preenchimento de dados de empresa.
**Evento Iniciador**:
Necessidade de cadastro no site.
**Ator**:
Comprador/Vendedor.
**Pré-condição**:
O usuário não possuir cadastro no site ou no aplicativo.
**Sequência de Eventos**:
1. Ator acessa a página inicial do sistema para realizar o cadastro do usuário.
2. Sistema apresenta os campos necessários para cadastro.
3. Ator preenche os dados e envia para o sistema realizar validação.
4. Sistema valida os dados e retorna para o ator a tela com os campos necessários para o cadastro da empresa.
5. Ator preenche os dados de sua empresa e envia para o sistema realizar validação.
6. Sistema valida os dados e os insere no banco de dados, redirecionando o ator para a tela inicial do sistema, com o seu *login* já realizado.
**Pós-Condição**:
Dados dos cadastros inseridos no banco de dados.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| Linha 4: Usuário quer alterar os dados do cadastro da empresa | 4.1 Ator pode retornar para a edição dos dados da empresa.<br>4.2 Retorna ao passo 6. |
| Linha 3: Dados não são válidos. | 3.1 No momento de salvar, o sistema realiza a validação dos dados e constata que existem dados inválidos. Sistema exibe mensagem ao usuário.<br>3.2 Retorna para o formulário de edição dos dados – passo 3. |
| Linha 5: Dados não são válidos | 5.1 No momento de salvar, o sistema realiza a validação dos dados e constata que existem dados inválidos. Sistema apresenta mensagem ao usuário.<br>5.2 Retorna para o formulário de edição dos dados – passo 4. |

**Quadro 4 – Caso de uso realizar cadastro**

No Quadro 5 está a descrição do caso de uso Realizar Login.

**Caso de uso:**
Realizar *Login*.
**Descrição**:
O usuário realiza o *login* no sistema.
**Evento Iniciador**:
Usuário acessa a tela de *login* do sistema.
**Ator**:
Comprador/Vendedor

**Pré-condição**:
Possuir um cadastro no sistema.
**Sequência de Eventos**:
1. Usuário acessa a tela principal do sistema.
2. Sistema apresenta formulário com campos de usuário e senha.
3. Usuário informa seu *login* e senha e envia os dados para validação.
4. Sistema verifica os dados no banco de dados como válidos e redireciona o usuário autenticado para a tela inicial do sistema.
**Pós-Condição**:
Usuário autenticado e com acesso a todas as funcionalidades do sistema pertinentes à consulta e cadastro de peças, edição e exclusão somente das peças por ele cadastradas.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| Linha 3: Usuário ou senha inválidos | 3.1 Ao serem apresentados os dados e o processo de verificação constatar que há dados inválidos, o sistema exibe mensagem requisitando que o usuário os informe novamente.<br>3.2 Retorna ao passo 3. A sequência do caso de uso prossegue com fluxo normal. |

**Quadro 5 – Caso de uso realizar login**

No Quadro 6 está a descrição do caso de uso Manter Peça.

**Caso de uso:**
Manter Peça.
**Descrição**:
Este caso de uso se refere às operações de inclusão, exclusão, edição e consulta de peças cadastradas pelo usuário no sistema.
**Evento Iniciador**:
Necessidade de cadastrar, alterar, excluir ou consultar peça.
**Ator**:
Comprador/Vendedor
**Pré-condição**:
Dados necessários disponíveis.
**Sequência de Eventos**:
1. Usuário acessa a tela de listagem de peças por ele cadastradas.
2. Sistema apresenta as peças que o usuário possui cadastradas.
3. Usuário realiza a operação desejada: cadastrar, excluir ou alterar.
4. Sistema verifica se os dados para a operação são válidos e realiza a operação.
**Pós-Condição**:
Operação de cadastro, edição ou exclusão realizada.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| Linha 3: Dados não são válidos | 3.1 No momento de salvar a operação, o sistema verifica que o formulário contém dados inválidos. O sistema apresenta mensagem e retorna o formulário para o usuário realizar as alterações necessárias.<br>3.2 Retorna ao passo 3. A sequência do caso de uso prossegue com fluxo normal. |

**Quadro 6 – Caso de uso manter peça**

No Quadro 7 está a descrição do caso de uso Realizar Busca.

| **Caso de uso:** |
|---|

Realizar Busca

**Descrição**:

O usuário pode realizar uma busca para visualizar peças cadastradas por outros usuários utilizando o campo de busca na tela principal do sistema.

**Evento Iniciador**:

Necessidade de conferir se determinada peça é disponibilizada por algum usuário do sistema.

**Ator**:

Comprador/Vendedor.

**Pré-condição**:

Estar autenticado no sistema e possuir dados sobre a peça.

**Sequência de Eventos**:

1. Usuário seleciona o campo de busca e insere informações sobre a peça.
2. Sistema realiza uma busca no banco de dados e retorna o resultado para o usuário baseado nos dados informados.

**Pós-Condição**:

Usuário visualiza os resultados apresentados pela busca.

**Quadro 7 – Caso de uso realizar busca**

No Quadro 8 está a descrição do caso de uso Enviar Mensagens.

**Caso de uso:**

Enviar Mensagens.

**Descrição**:

Os usuários têm a possibilidade de trocar mensagens entre si pelo próprio sistema para conversar sobre as peças disponíveis.

**Evento Iniciador**:

Interesse em ter contato com outro usuário.

**Ator**:

Comprador/Vendedor.

**Pré-condição**:

Usuário possui interesse em uma peça e em entrar em contato com o usuário que a disponibilizou.

**Sequência de Eventos**:

1. Usuário 1 seleciona a opção de enviar mensagem para o perfil do usuário 2.
2. Sistema exibe tela com campos para preenchimento da mensagem.
3. Usuário 1 preenche a mensagem e a envia.
4. Sistema entrega a mensagem do usuário 1 para o usuário 2.

**Pós-Condição**:

Usuário 1 realiza contato com usuário 2.

**Quadro 8 – Caso de uso enviar mensagens**

No Quadro 9 está a descrição do caso de uso Visualizar Estoque.

**Caso de uso:**

Visualizar Estoque

**Descrição**:

Este caso de uso se refere à possibilidade que o usuário tem de visualizar o seu estoque verificando que peças ele possui atualmente cadastradas no sistema.

**Evento Iniciador**:

Interesse em verificar peças cadastradas.

**Ator**:

Comprador/Vendedor.

**Pré-condição**:
Possuir peças cadastradas.
**Sequência de Eventos**:
1. Usuário seleciona a opção de visualizar o seu estoque.
2. Sistema apresenta ao usuário o seu estoque.
**Pós-Condição**:
Usuário visualiza o estoque com as suas peças cadastradas.

**Quadro 9 – Caso de uso visualizar estoque**

No Quadro 10 está a descrição do caso de uso Visualizar Peça.

**Caso de uso:**
Visualizar Peça.
**Descrição**:
Este caso de uso se refere à possibilidade de o usuário visualizar uma peça e informações cadastradas de forma mais detalhada, seja por meio da busca ou ao visualizar os dados do seu estoque.
**Evento Iniciador**:
Selecionar uma peça.
**Ator**:
Comprador/Vendedor.
**Pré-condição**:
Estar autenticado no sistema e possuir uma peça no seu estoque para visualizar ou realizar uma busca de peça no sistema.
**Sequência de Eventos**:
1. Usuário seleciona uma peça no seu estoque para verificar as suas informações.
2. Sistema exibe a peça com dados detalhados para o usuário.
**Pós-Condição**:
Usuário tem acesso aos dados da peça selecionada e aos seus detalhes.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| Linha 1: Usuário seleciona uma peça para visualizar seus dados de cadastro. | 1.1 Retorna ao passo 2. A sequência do caso de uso segue o fluxo normal. |

**Quadro 10 – Caso de uso visualizar peça**

No Quadro 11 está a descrição do caso de uso Enviar Sugestões.

**Caso de uso:**
Enviar Sugestões.
**Descrição**:
Neste caso de uso o usuário tem a possibilidade de enviar sugestões de melhorias ou correções de problemas encontrados no sistema.
**Evento Iniciador**:
Necessidade ou interesse de contato com o administrador do sistema.
**Ator**:
Comprador/Vendedor.
**Pré-condição**:
Melhoria sugerida ou problema encontrado.
**Sequência de Eventos**:
1. Usuário seleciona a opção para enviar sugestões.
2. Sistema apresenta a tela para o preenchimento e o envio da mensagem.
3. Usuário preenche a mensagem e a envia.
4. Sistema faz a entrega da mensagem ao administrador do sistema.

| |
|---|
| **Pós-Condição**:<br>Administrador tem acesso à mensagem do usuário. |

**Quadro 11 – Caso de uso enviar sugestões**

No Quadro 12 está a descrição do caso de uso Gerar Relatórios.

| |
|---|
| **Caso de uso:**<br>Gerar Relatórios.<br>**Descrição**:<br>O usuário tem a opção de gerar relatórios para a conferência das peças que cadastrou.<br>**Evento Iniciador**:<br>Usuário solicita ao sistema o relatório das suas peças cadastradas.<br>**Ator**:<br>Comprador/Vendedor<br>**Pré-condição**:<br>Possuir ao menos uma peça cadastrada.<br>**Sequência de Eventos**:<br>1. Usuário acessa tela de geração de relatório e realiza a configuração do relatório.<br>2. Sistema recebe as configurações e gera o relatório.<br>**Pós-Condição**:<br>Usuário visualiza relatório. |

**Quadro 12 – Caso de uso gerar relatórios**

Na Figura 2 está o diagrama de classes de análise do sistema.

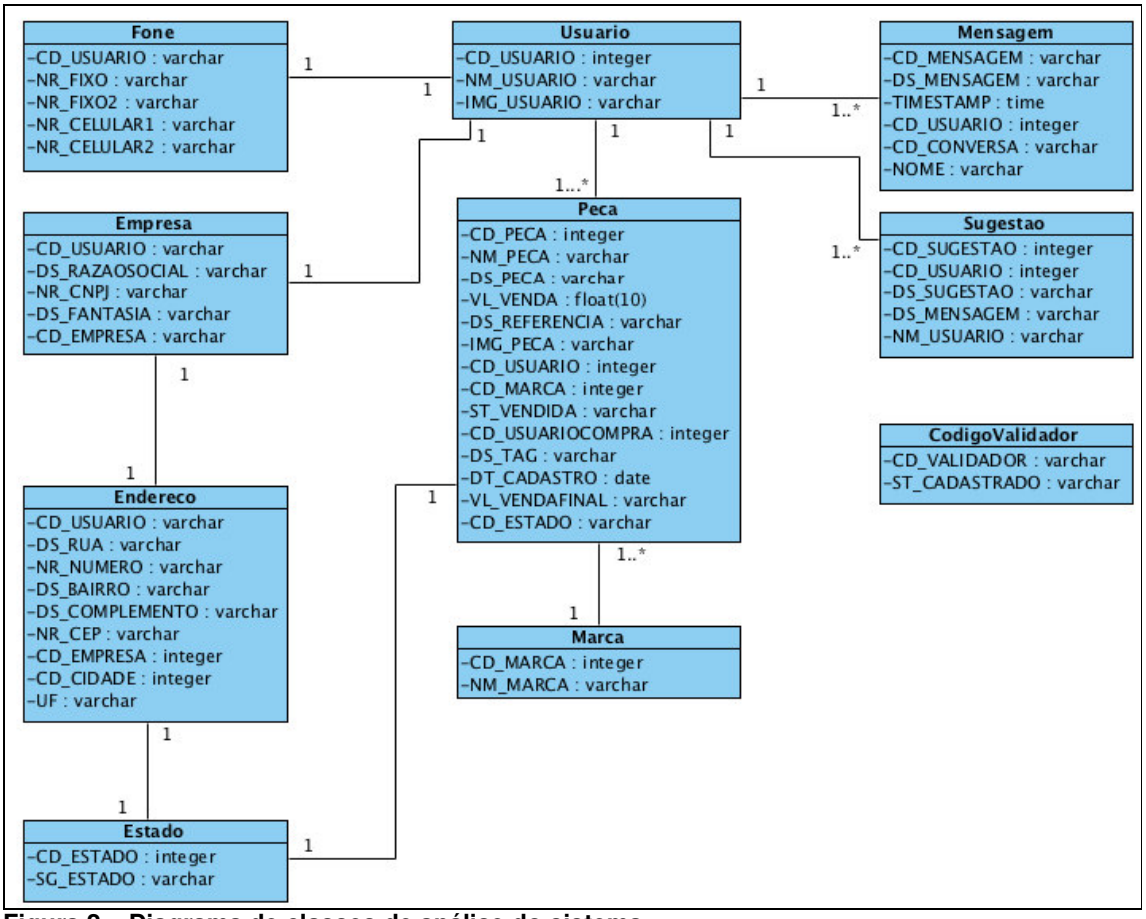


**Figura 2 – Diagrama de classes de análise do sistema**

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

No Quadro 13 está a apresentação da classe Fone.

| | |
|---|---|
| **Identificação:** | Fone |
| **Descrição:** | Cadastro de telefones da empresa no sistema. |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_USUARIO (varchar): código do registro<br>NR_FIXO (varchar): número de telefone fixo<br>NR_FIXO2 (varchar): número de telefone fixo 2<br>NR_CELULAR1 (varchar): número de celular 1<br>NR_CELULAR2 (varchar): número de celular 2 |
| **Métodos:** | Void add(dados : Fone);<br>Void remover(cod : integer);<br>Void atualizar(dados : Fone); |

**Quadro 13 – Descrição da classe Fone**

No Quadro 14 está a apresentação da classe Empresa.

| | |
|---|---|
| **Identificação:** | Empresa |

| | |
|---|---|
| **Descrição:** | Empresas que os usuários cadastram no sistema. |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_USUARIO (varchar): código do registro<br>DS_RAZAOSOCIAL (varchar): descrição da razão social<br>NR_CNPJ (varchar): número do CNPJ<br>DS_FANTASIA (varchar): descrição do nome fantasia<br>CD_EMPRESA(varchar): código da empresa |
| **Métodos:** | Void add (dados: Empresa);<br>Void remover (cod: integer);<br>Void atualizar (dados: Empresa); |

**Quadro 14 – Descrição da classe Empresa**

No Quadro 15 está a apresentação da classe Endereço.

| | |
|---|---|
| **Identificação:** | Endereço |
| **Descrição:** | Endereços vinculado à empresa |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_USUARIO (varchar): código do registro<br>DS_RUA (varchar): nome da rua<br>NR_NUMERO (integer): número do endereço<br>DS_BAIRRO (varchar): nome do bairro<br>DS_COMPLEMENTO (varchar): descrição do complemento<br>NR_CEP (varchar): número do CEP<br>CD_EMPRESA (integer): código da empresa ligada ao registro<br>CD_CIDADE (integer): código da cidade ligada ao registro<br>UF(varchar): código do estado |
| **Métodos:** | Void add (dados: Endereco);<br>Void remover (cod: integer);<br>Void atualizar (dados: Endereco); |

**Quadro 15 – Descrição da classe Endereço**

No Quadro 16 está a apresentação da classe Usuário.

| | |
|---|---|
| **Identificação:** | Usuário |
| **Descrição:** | Usuários do sistema. |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_USUARIO (varchar): código do registro<br>NM_USUARIO (varchar): nome do usuário<br>IMG_USUARIO (varchar): imagem para perfil |
| **Métodos:** | Void add (dados: Usuario);<br>Void remover (cod: integer);<br>Void atualizar (dados: Usuario); |

**Quadro 16 – Descrição da classe Usuário**

No Quadro 17 está a apresentação da classe Peça.

| | |
|---|---|
| **Identificação:** | Peça |
| **Descrição:** | Peças que cadastradas no sistema |
| **Requisitos:** | RF 05, RF 06, RF 07, RF 08, RF 10, RF 11, RF 13 |
| **Atributos:** | CD_PECA (integer): código do registro<br>NM_PECA (varchar): nome da peça<br>DS_PECA (varchar): descrição detalhada da peça |

| | |
|---|---|
| | VL_VENDA (float): valor de venda<br>DS_REFERENCIA (varchar): descrição de referência<br>IMG_PECA (varchar): imagem para peça<br>CD_USUARIO (varchar): código do usuário ligado a peça<br>CD_MARCA (varchar): código da marca ligada a peça<br>ST_VENDIDA (varchar): situação da peça<br>CD_USUARIOCOMPRA (varchar): código do usuário que comprou a peça<br>CD_ESTADO (varchar): código do estado ligado a peça<br>DS_TAG(varchar):descrição das palavras-chave<br>DT_CADASTRO(date): data de cadastro<br>VL_VENDAFINAL(varchar): valor da venda final |
| **Métodos:** | Void add (dados: Peca);<br>Void remover (cod: integer);<br>Void atualizar (dados: Peca);<br>Peca buscar (cod: integer);<br>Peca[] buscarPecas();<br>Void geraRel (info : Info); |

**Quadro 17 – Descrição da classe Peça**

No Quadro 18 está a apresentação da classe Marca.

| | |
|---|---|
| **Identificação:** | Marca |
| **Descrição:** | Marcas previamente cadastradas para serem utilizadas no cadastro de peças. |
| **Requisitos:** | RF 05, RF 06 |
| **Atributos:** | CD_MARCA (integer): código do registro<br>NM_MARCA (varchar): nome da marca |

**Quadro 18 – Descrição da classe Marca**

No Quadro 19 está a apresentação da classe Estado.

| | |
|---|---|
| **Identificação:** | Estado |
| **Descrição:** | Estados previamente cadastrados para utilizados no cadastro de cidades. |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_ESTADO (integer): código do registro<br>SG_ESTADO (varchar): sigla do estado |

**Quadro 19 – Descrição da classe Estado**

No Quadro 20 está a apresentação da classe Mensagem.

| | |
|---|---|
| **Identificação:** | Mensagem |
| **Descrição:** | Mensagens enviadas entre usuários. |
| **Requisitos:** | RF 09 |
| **Atributos:** | CD_MENSAGEM (varchar): código do registro<br>DS_MENSAGEM (varchar): corpo da mensagem<br>TIMESTAMP (time): valor de dia/hora do envio<br>CD_USUARIO (varchar): código do usuário ligado a mensagem<br>CD_CONVERSA(varchar): código da conversa<br>NOME(varchar): nome do usuário |
| **Métodos:** | Void add(msg : Mensagem); |

| | |
|---|---|
| | Mensagem[] buscaConversa(cod : varchar); |

**Quadro 20 – Descrição da classe Mensagem**

No Quadro 21 está a apresentação da classe Sugestão.

| | |
|---|---|
| **Identificação:** | Sugestão |
| **Descrição:** | Sugestões recebidas pelos usuários para melhorias e problemas encontrados no sistema. |
| **Requisitos:** | RF 12 |
| **Atributos:** | CD_SUGESTAO (integer): código do registro<br>CD_USUARIO (integer): código do usuário ligado a sugestão<br>DS_SUGESTAO (varchar): descrição da sugestão |
| **Métodos:** | Void add(dados : Sugestao); |

**Quadro 21 – Descrição da classe Sugestão**

No Quadro 22 está a apresentação da classe Código Validador.

| | |
|---|---|
| **Identificação:** | CodigoValidador |
| **Descrição:** | Código informado pelo usuário no momento do cadastro. |
| **Requisitos:** | RF 01 |
| **Atributos:** | CD_VALIDADOR (varchar): código para validar o cadastro do usuário<br>ST_CADASTRADO (varchar): situação do código |
| **Métodos:** | Boolean buscaCodigo ( codigo : varchar); |

**Quadro 22 – Descrição da classe Código Validador**

A Figura 3 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados do sistema.

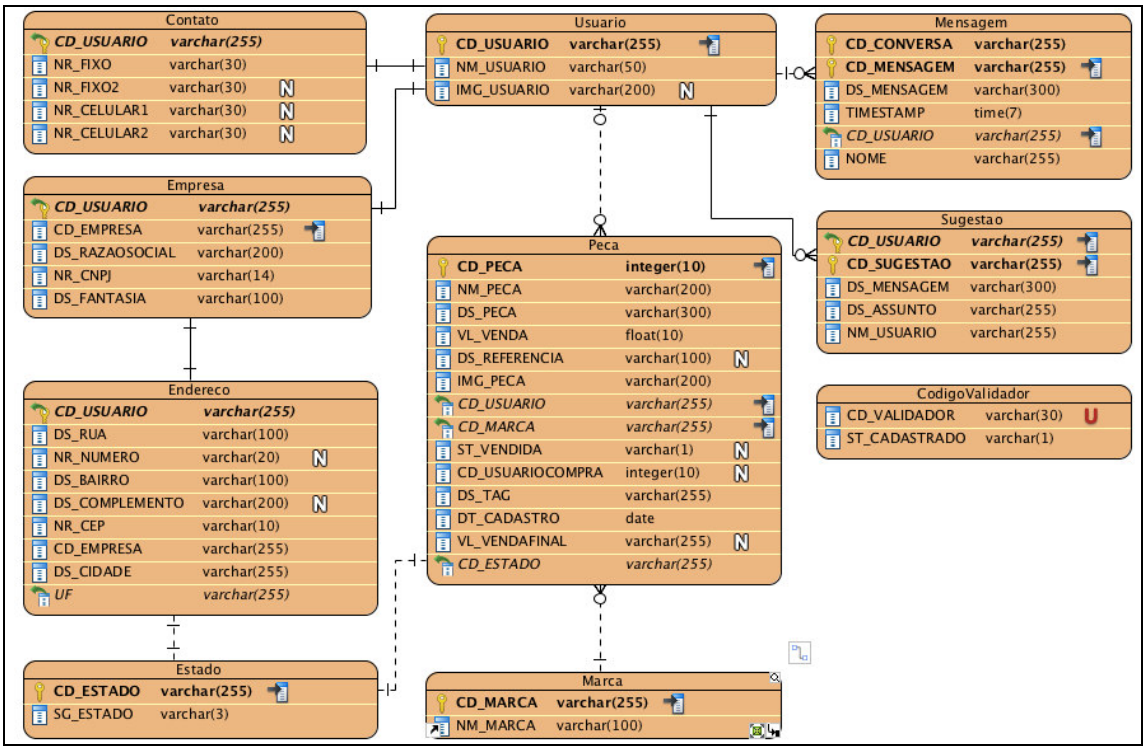


**Figura 3 – Diagrama de entidades e relacionamentos do banco de dados**

No Quadro 23 estão os campos da tabela Contato. Um ou mais números de telefone poderão estar relacionados a um usuário no momento do seu cadastro.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| NR_FIXO | Texto | Não | Não | Não | |
| NR_FIXO2 | Texto | Sim | Não | Não | |
| NR_CELULAR1 | Texto | Sim | Não | Não | |
| NR_CELULAR2 | Texto | Sim | Não | Não | |
| CD_USUARIO | TEXTO | Não | Sim | Sim | Da tabela usuário |

**Quadro 23 – Campos da tabela Contato**

No Quadro 24 estão os campos da tabela empresa. Uma empresa será cadastrada após a criação de um usuário, tendo relacionado a ela um registro de endereço.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_EMPRESA | Numérico | Não | Não | Não | |
| DS_RAZAOSOCIAL | Texto | Não | Não | Não | |
| NR_CNPJ | Texto | Não | Não | Não | |
| DS_FANTASIA | Texto | Não | Não | Não | |
| CD_USUARIO | Texto | Não | Sim | Sim | Da tabela Usuário |

**Quadro 24 – Campos da tabela Empresa**

No Quadro 25 estão os campos da tabela endereço. Um endereço estará relacionado a uma empresa no momento do seu cadastro no sistema.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_USUARIO | Texto | Não | Sim | Sim | Da tabela Usuário |
| DS_RUA | Texto | Não | Não | Não | |
| NR_NUMERO | Texto | Sim | Não | Não | |
| DS_BAIRRO | Texto | Não | Não | Não | |
| DS_COMPLEMENTO | Texto | Sim | Não | Não | |
| NR_CEP | Texto | Não | Não | Não | |
| CD_EMPRESA | Numérico | Não | Não | Sim | Da tabela Empresa |
| DS_CIDADE | Texto | Não | Não | Não | |
| UF | Texto | Não | Não | Sim | Da tabela Estado |

**Quadro 25 – Campos da tabela Endereço**

No Quadro 26 estão os campos da tabela usuário. Um usuário ao finalizar o seu cadastro, cadastrará a sua empresa, para que, assim, ele possa cadastrar, alterar e excluir peças no sistema. O usuário poderá entrar em contato com outros usuários por meio do envio de mensagens pelo próprio sistema e também enviar sugestões de melhorias ou problemas no sistema.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_USUARIO | Texto | Não | Sim | Não | |
| NM_USUARIO | Texto | Não | Não | Não | |
| IMG_USUARIO | Texto | Sim | Não | Não | |

**Quadro 26 – Campos da tabela Usuário**

No Quadro 27 são apresentados os campos da tabela peça. Uma peça ao ser cadastrada é relacionada a uma marca. Uma peça só pode ser excluída ou alterada pelo usuário que a cadastrou. Cada usuário terá acesso aos relatórios de conferência de peças vendidas e em estoque. Cada usuário possui acesso para edição somente das peças que ele cadastrou.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_PECA | Numérico | Não | Sim | Não | |
| NM_PECA | Texto | Não | Não | Não | |
| DS_PECA | Texto | Não | Não | Não | |
| VL_VENDA | Numérico | Não | Não | Não | |
| DS_REFERENCIA | Texto | Sim | Não | Não | |
| IMG_PECA | Texto | Não | Não | Não | |
| ST_VENDIDA | Texto | Sim | Não | Não | |
| CD_USUARIOCOMPRA | Texto | Sim | Não | Não | |
| UF_PECA | Texto | Não | Não | Sim | Da tabela Estado |
| CD_USUARIO | Texto | Não | Não | Sim | Da tabela usuário |
| CD_MARCA | Texto | Não | Não | Sim | Da tabela marca |
| DS_TAG | Texto | Não | Não | Não | |
| DT_CADASTRO | Data | Não | Não | Não | |
| VL_VENDAFINAL | Texto | Sim | Não | Não | |

**Quadro 27 – Campos da tabela Peça**

No Quadro 28 estão os campos da tabela marca. As marcas serão previamente cadastradas para serem utilizadas no cadastro de peças, auxiliando assim em buscas no sistema por meio de filtros.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_MARCA | Texto | Não | Sim | Não | |
| NM_MARCA | Texto | Não | Não | Não | |

**Quadro 28 – Campos da tabela Marca**

No Quadro 29 estão os campos da tabela estado. Os Estados serão previamente cadastrados e serão relacionados a um endereço ao ser realizado o cadastro da empresa e a uma peça quando cadastrada.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_ESTADO | Texto | Não | Sim | Não | |
| SG_ESTADO | Texto | Não | Não | Não | |

**Quadro 29 – Campos da tabela Estado**

No Quadro 30 estão os campos da tabela mensagem. Mensagens podem ser enviadas de usuários para usuários dentro do sistema, com a finalidade de facilitar o contato entre eles.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_MENSAGEM | Texto | Não | Sim | Não | |
| DS_MENSAGEM | Texto | Não | Não | Não | |
| TIMESTAMP | Data/Hora | Não | Não | Não | |
| CD_USUARIO | Texto | Não | Não | Sim | Da tabela usuário |
| NOME | Texto | Não | Não | Não | |
| CD_CONVERSA | Texto | Não | Sim | Não | |

**Quadro 30 – Campos da tabela Mensagem**

No Quadro 31 estão os campos da tabela sugestão. Um usuário envia uma sugestão ao sistema, seja para a correção de problemas encontrados ou para melhorias sugeridas.

| Campo | Tipo | Nulo | Chave primária | Chave estrangeira | Observações |
|---|---|---|---|---|---|
| CD_SUGESTAO | Texto | Não | Sim | Não | |

| | | | | | |
|---|---|---|---|---|---|
| DS_MENSAGEM | Texto | Não | Não | Não | |
| CD_USUARIO | Texto | Não | Sim | Sim | Da tabela usuário |
| NM_USUARIO | Texto | Não | Não | Não | |
| DS_ASSUNTO | Texto | Não | Não | Não | |

**Quadro 31 – Campos da tabela Sugestão**

No Quadro 32 estão os campos da tabela Código Validador. Cada código é único e será utilizado apenas uma vez para validar o cadastro do usuário, sendo que uma pessoa só poderá realizar o cadastro no sistema se possuir um código válido e não utilizado.

| **Campo** | **Tipo** | **Nulo** | **Chave primária** | **Chave estrangeira** | **Observações** |
|---|---|---|---|---|---|
| CD_VALIDADOR | Texto | Não | Não | Não | |
| ST_CADASTRADO | Texto | Não | Não | Não | |

**Quadro 32 – Campos da tabela Código Validador**

## 4.3 APRESENTAÇÃO DO SISTEMA

Todas as telas do sistema compartilham duas partes importantes para a padronização do leiaute que são o cabeçalho contendo o nome do sistema e os ícones utilizados para a navegação do usuário no sistema e o rodapé.

A Figura 4 exibe a tela que apresenta ao usuário a opção de realizar o cadastro para ter acesso às funcionalidades do sistema ou no caso de já possuir um cadastro, ele poderá simplesmente clicar no link “Já possui cadastro?” e ser redirecionado para a página de *login*. Novos usuários só poderão realizar cadastro perante o uso de um código validador, que pode ser obtido informando o seu *e-m*ail no campo do cadastro e clicando no *link* “Não possui código validador? Informe o e-mail e receba o seu!” para receber o referido código no *e-mail* informado.

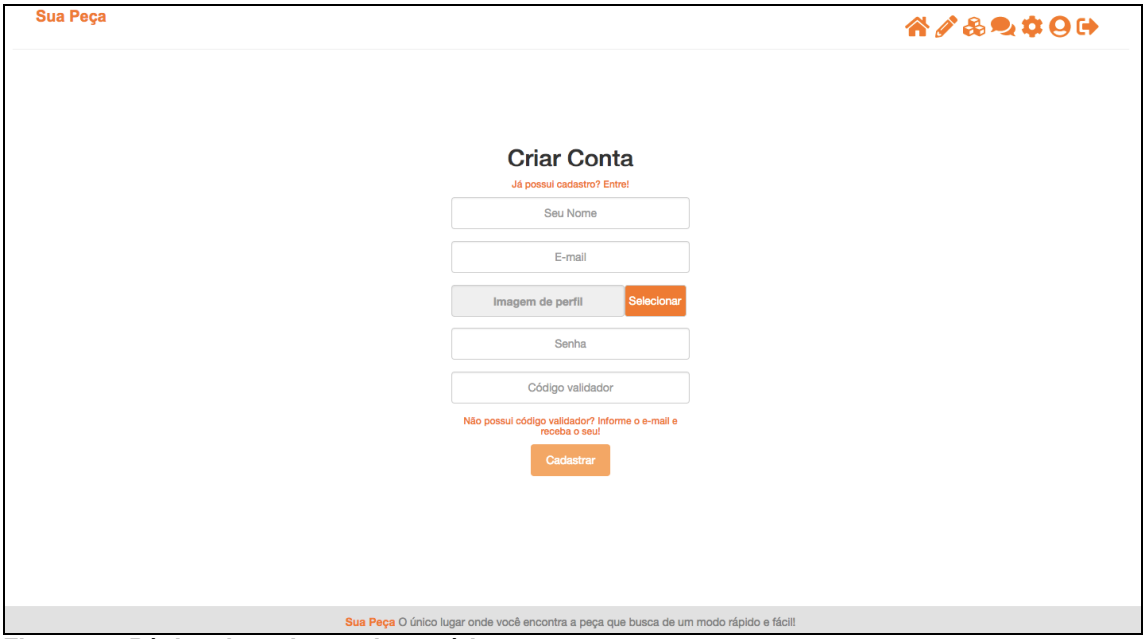


**Figura 4 – Página de cadastro de usuário**

Na Figura 5 é apresentada a tela de cadastro de empresa. Essa tela é apresentada ao usuário após o seu cadastro ser finalizado. Nessa tela, o usuário informará os seus dados principais para contato e localização de empresa. Além disso, deverá informar o número do CNPJ da empresa e utilizar o botão “Buscar Dados” para que seja feita a busca dos dados da empresa para consulta é utilizado o *webservice* da Receita Federal Brasileira para carregar as informações relacionadas ao CNPJ informado. Os campos buscados pelo CNPJ diretamente da Receita estarão sempre desabilitados e não poderão ser editados.

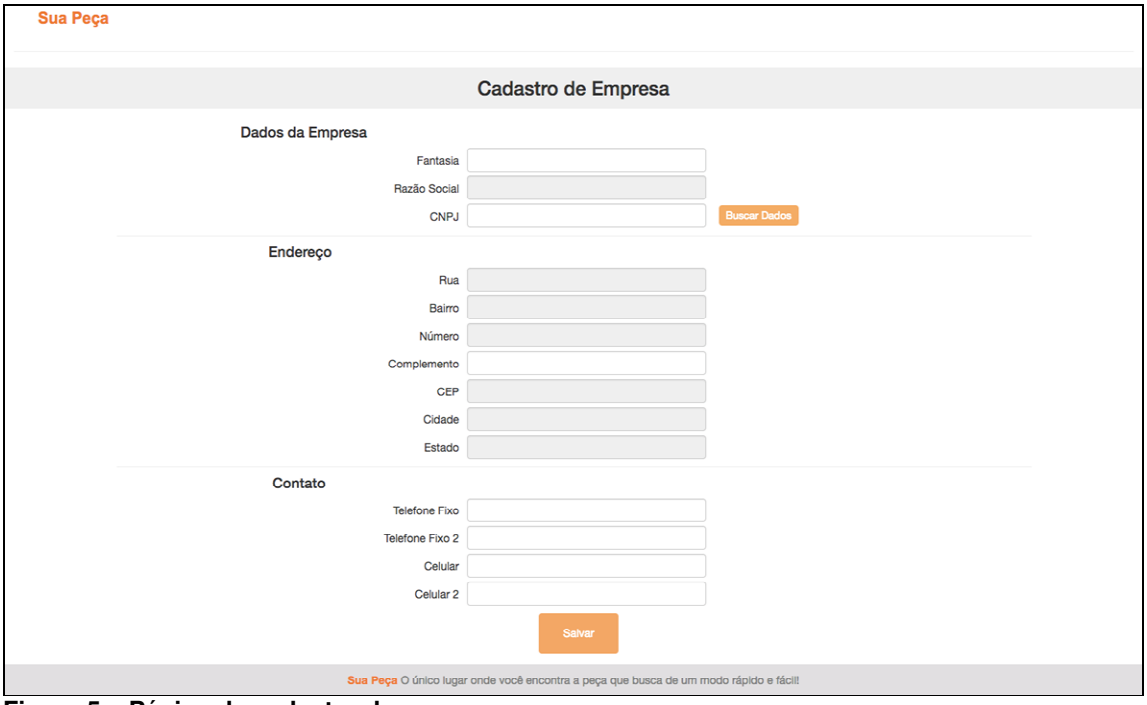


**Figura 5 – Página de cadastro da empresa**

Na Figura 6 é apresentada a tela de informações do usuário. Nessa tela, o usuário pode alterar o nome e a foto informados no momento do cadastro, bem como requisitar a mudança da sua senha por meio do *link* "Clique aqui para redefinir sua senha!". O sistema enviará um *e-mail* para o usuário com um *link* para que a alteração da senha possa ser realizada.


**Figura 6 – Página de informações do usuário**

Na Figura 7 é apresentada a tela de *login* do sistema que possui somente os campos “E-mail” e “Senha”. Caso o usuário não possua cadastro ou necessite realizar um novo, ele tem a opção de ser direcionado para a tela de cadastro mostrada na Figura 4, por meio do *link* “Não possui cadastro? Cadastre-se!”. Caso o usuário não lembrar a sua senha ele pode informar o seu *e-mail* e clicar no *link* “Esqueceu a senha?” que um *e-mail* será enviado para ele com um *link* para que ele possa redefinir a sua senha.

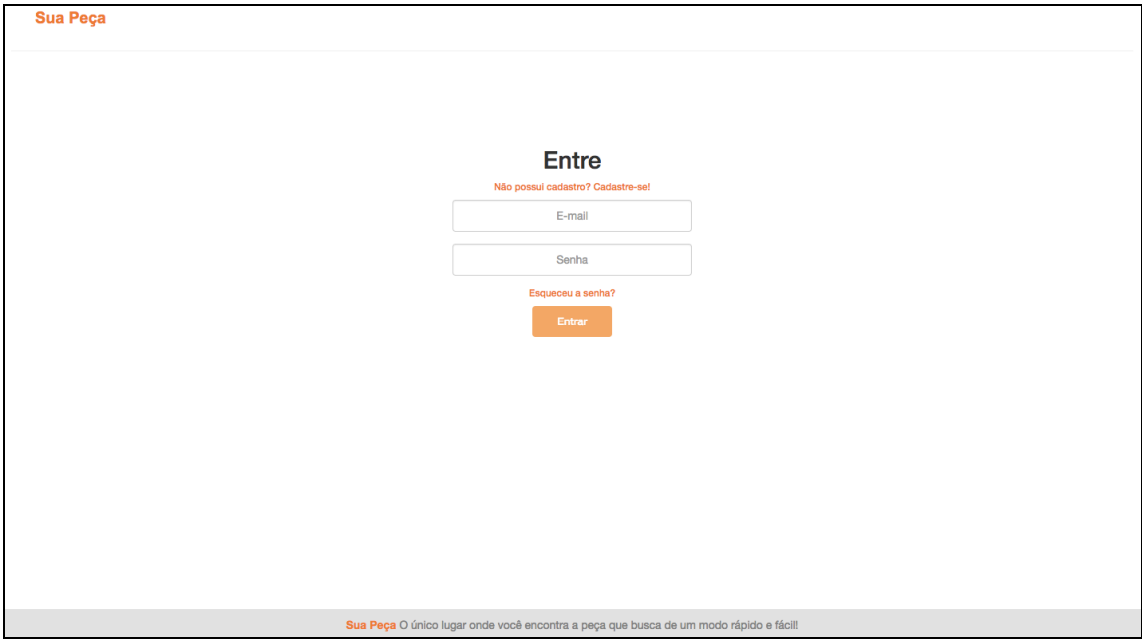


**Figura 7 – Página de login**

Na Figura 8 é apresentada a tela que o usuário tem acesso após realizar autenticação no sistema, sendo possível realizar buscas por peças. Essa tela apresenta o cabeçalho com os ícones habilitados. Nesta tela também está o botão “Fale Conosco” por meio do qual o usuário tem acesso à tela de envio de sugestões.

Os ícones para navegação, na parte superior à direita da imagem da Figura 8, somente são apresentados ao usuário após ele ter realizado o *login* com sucesso no sistema, sendo eles (da esquerda para a direita):

a) o ícone para redirecionamento para a página principal do sistema sendo possível realizar a busca por peças e enviar sugestões para o sistema por meio do botão “Fale Conosco”.

b) ícone para acesso à tela de cadastro de peças.

c) ícone para acesso ao estoque de peças do referido usuário.

d) ícone para acesso a página do *chat* do sistema, para entrar em contato com outros usuários.

e) ícone para a tela das informações cadastradas para a empresa, caso ele necessite realizar alguma alteração nas mesmas.

f) ícone para acesso às informações cadastrais do usuário, permitindo editar o cadastro.

g) ícone para *logout* do sistema, o usuário pode desconectar do sistema.

O cabeçalho contém os ícones para a navegação em todas as telas. Exceto nas telas de *login* e de cadastro do usuário por não existir, ainda, um usuário autenticado no sistema e na tela de cadastro de empresa, para que o usuário realize o cadastro de sua empresa para seguir com o uso do sistema.

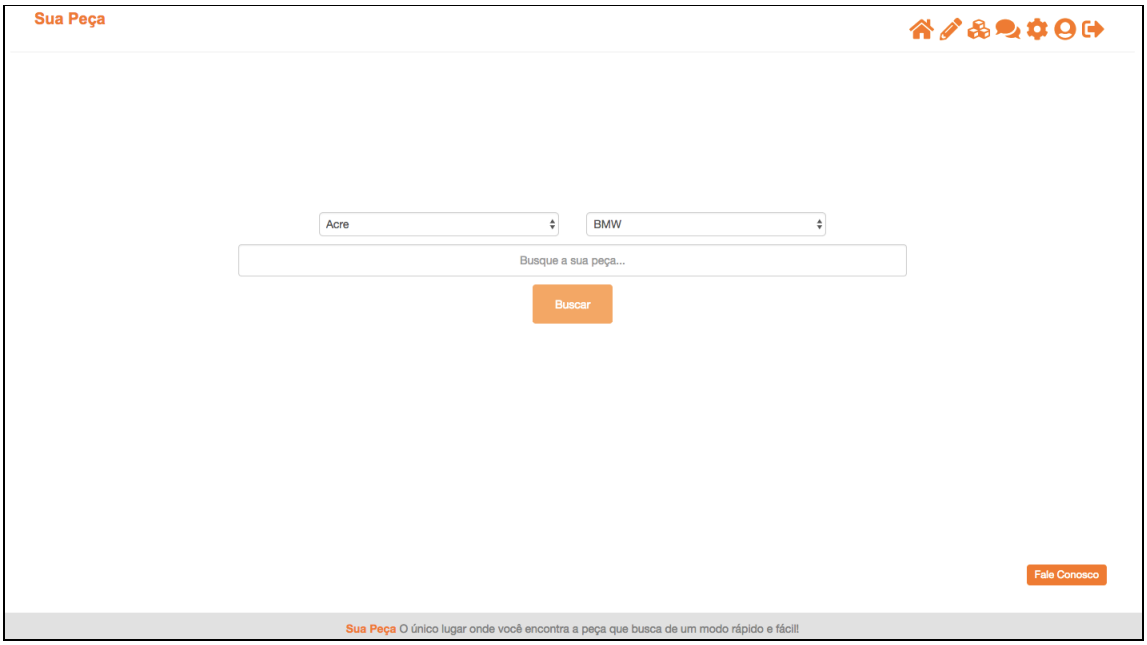


**Figura 8 – Página de busca**

Na Figura 9 é apresentada a tela de resultado de uma busca por meio dos filtros e das palavras-chaves informadas no campo de busca.. Na parte superior da tela, abaixo do cabeçalho, é apresentado o texto buscado e logo abaixo os resultados encontrados, baseados nas palavras-chave no Estado e Marca que foram cadastradas nas peças.

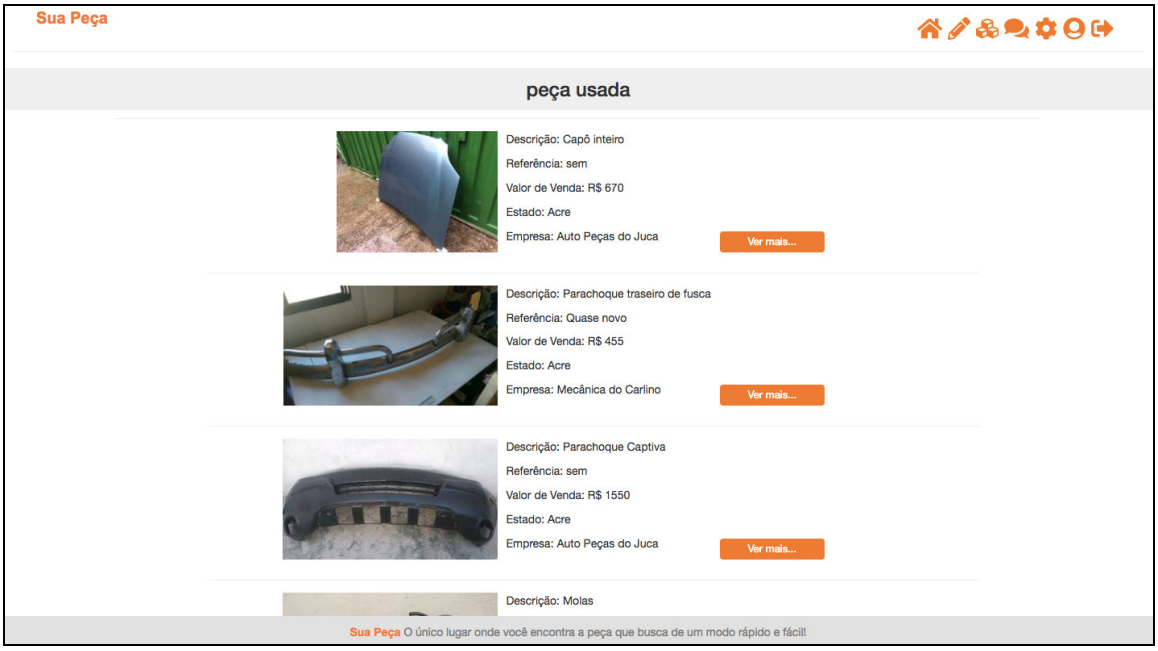


**Figura 9 – Página de resultado de busca**

Na Figura 10 está a tela de visualização de uma peça apresentada como resultado de uma busca. Após a busca, o usuário tem acesso às informações cadastradas da peça pelo botão “Ver mais...” sendo direcionado para a tela com as informações da peça que contém um botão “Enviar Mensagem” para que o usuário possa entrar em contato com a pessoa responsável pelo cadastro da peça.

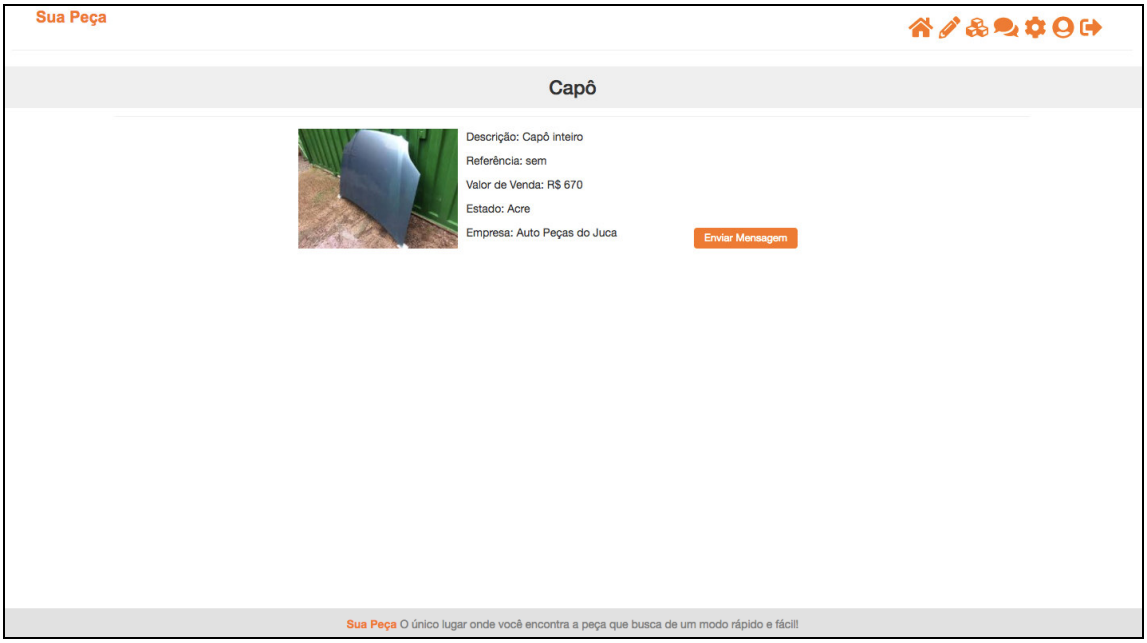


**Figura 10 – Página de visualização de peça da busca**

Na Figura 11 é apresentada a tela de estoque do usuário. Nela, o usuário tem a opção de clicar no botão “Ver mais...” para ter acesso a mais informações da peça, podendo alterar dados do cadastro da peça ou excluir a peça por meio do botão “Excluir”.

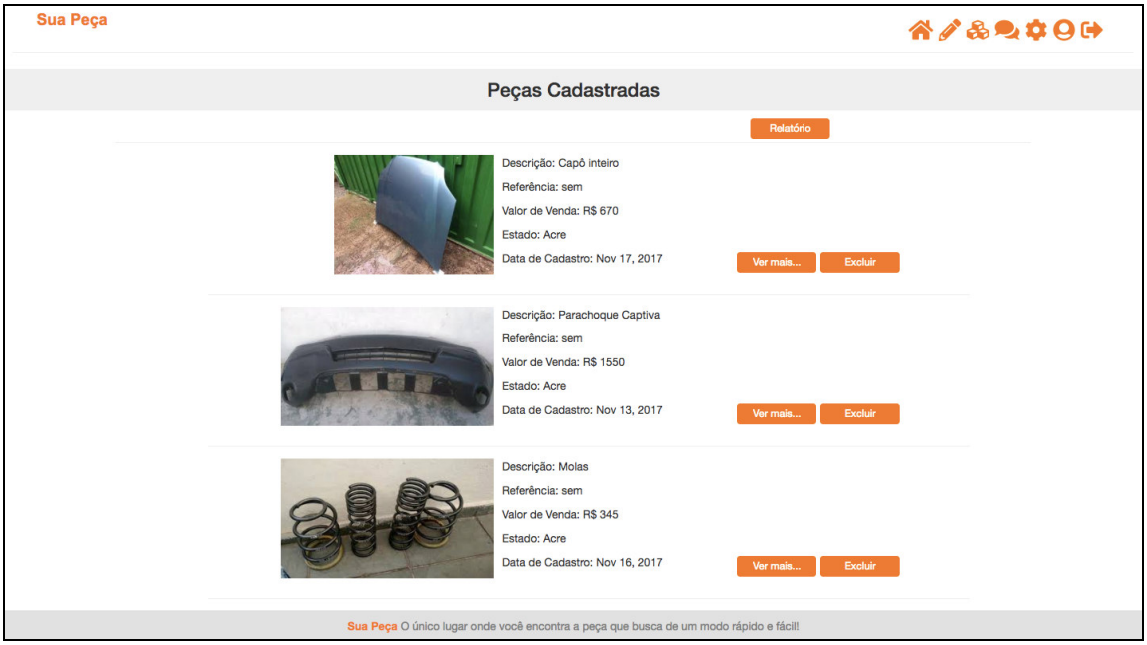


**Figura 11 – Página de estoque**

Na Figura 12 é apresentado o relatório que pode ser gerado por meio do botão “Relatório” na tela de peças cadastradas apresentado na Figura 11. Ao clicar nesse botão, o relatório será gerado e salvo em um arquivo formato PDF com o nome “Relatório_de_Pecas_dia-mes-ano_hora-minuto”. Nesse relatório é possível verificar as peças que estão cadastradas pelo usuário no estoque bem como a situação, se está vendida ou não.

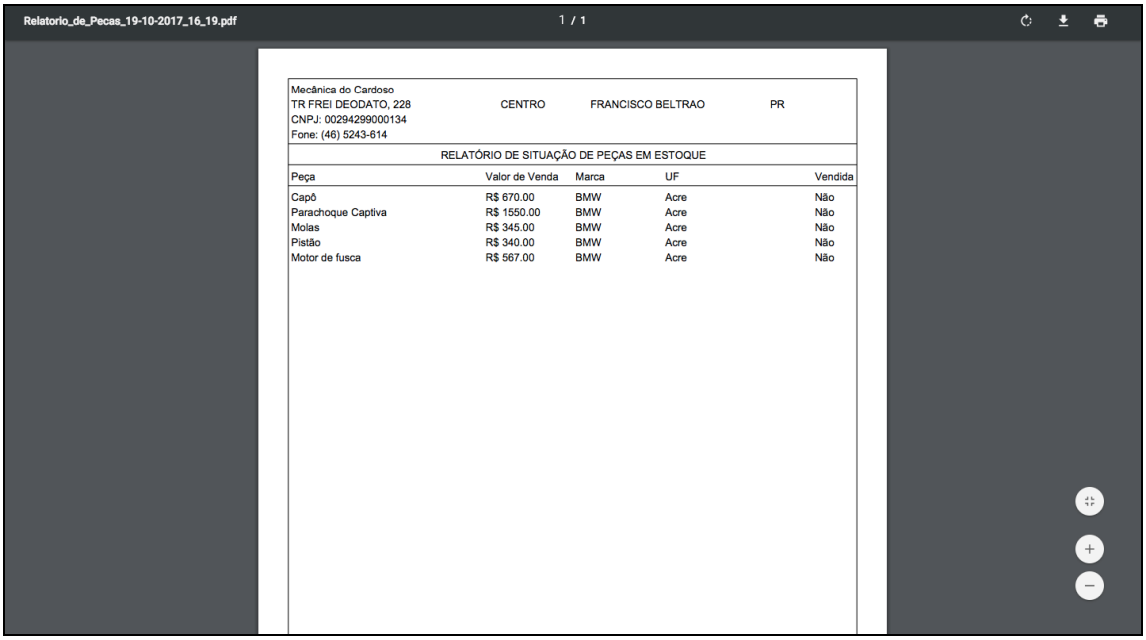


**Figura 12 – Página de relatório de peças**

Na Figura 13 é apresentada a tela de visualização da peça do estoque do usuário. Nessa tela o usuário tem a opção de alterar as informações de cadastro da peça por meio do botão “Editar”.

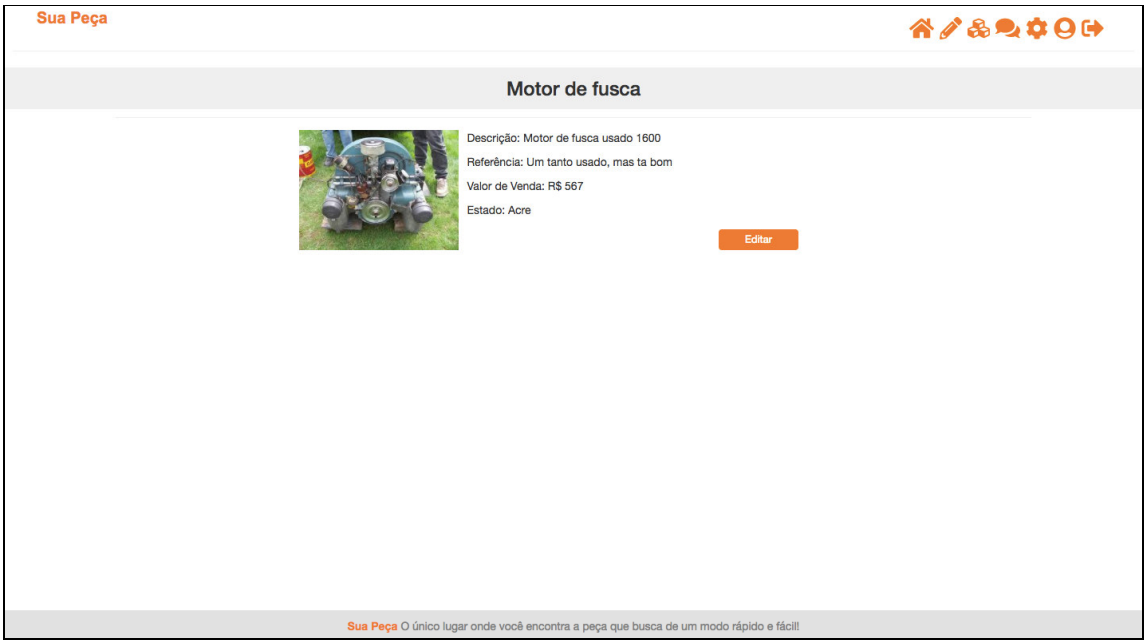


**Figura 13 – Página de visualização de peça do estoque**

Na Figura 14 é apresentada a tela de edição de peça. Nessa tela o usuário pode modificar os dados informados no momento do cadastro da peça e também

alterar a situação da peça, informando se ela foi vendida, para quem foi vendida, se o valor de venda final é diferente do ofertado e em que data foi realizada a venda.

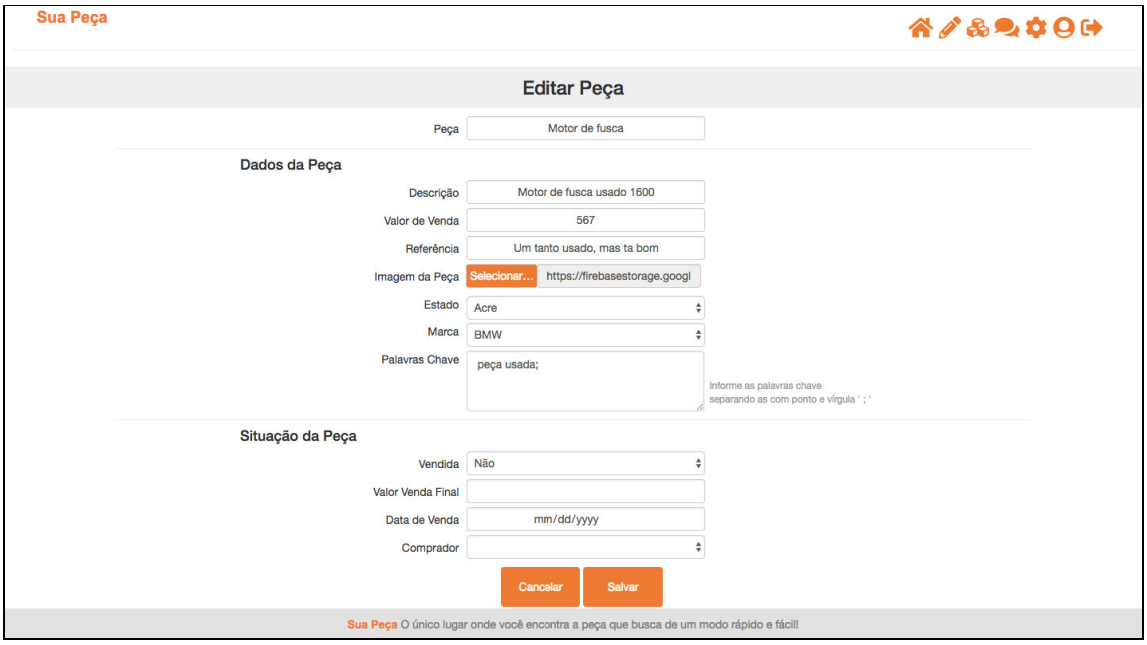


**Figura 14 – Página de edição de peça**

Na Figura 15 é apresentada a tela de cadastro de peça. Nela, dentre os campos para serem preenchidos três deles serão utilizados no mecanismo de busca do sistema, sendo eles: o campo “Marca”, o campo “Estado” e o campo “Palavras Chave”. O usuário, ao informar as palavras que deseja utilizar na peça para filtro no campo “Palavras Chave”, deve separá-las com ponto-vírgula, determinando assim cada chave individualmente.

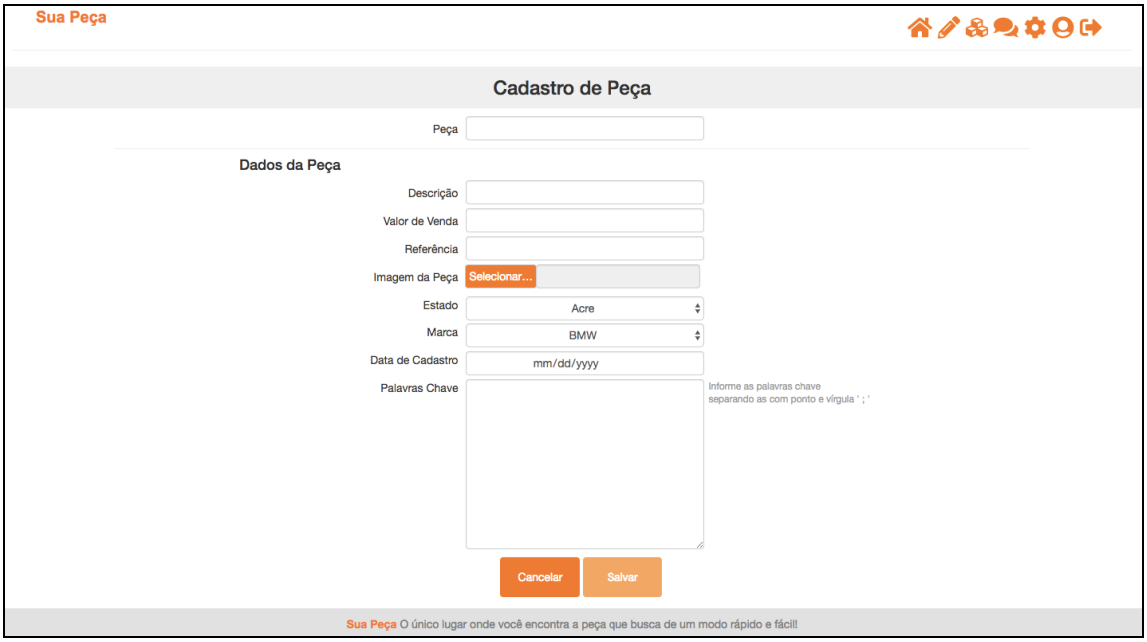


**Figura 15 – Página de cadastro de peça**

Na Figura 16 é apresentada a tela do *chat* para a qual o usuário será redirecionado ao clicar no botão “Enviar Mensagem” exibida na Figura 10. A tela será carregada já com o usuário responsável pela peça selecionado e o *chat* pronto para o uso, ou pode ser acessado também pelo do ícone do *chat* no cabeçalho.

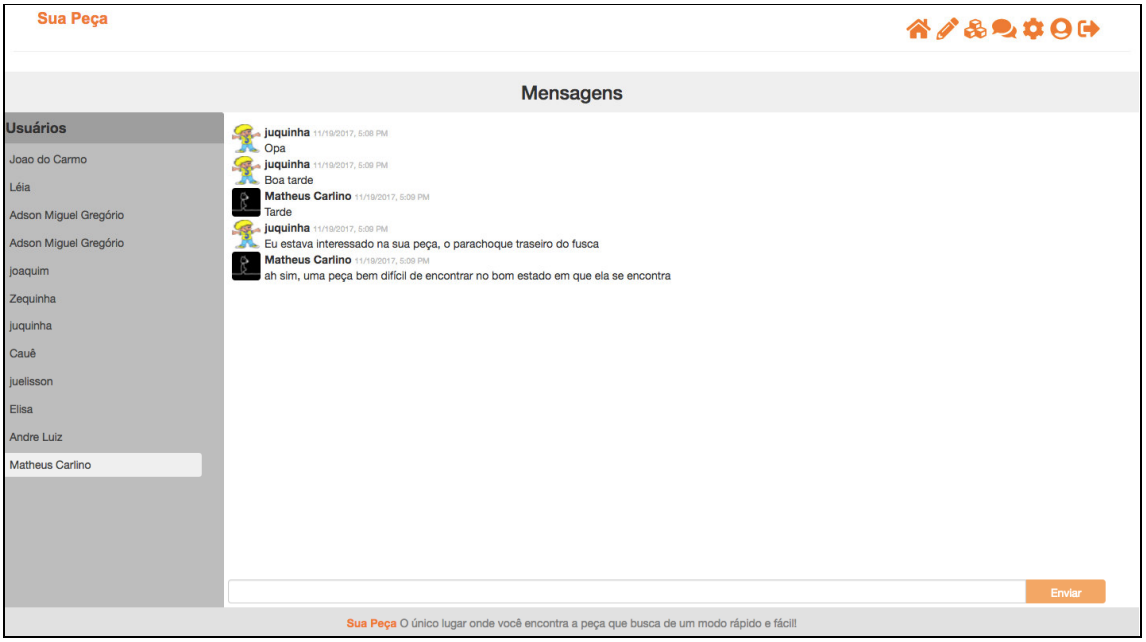


**Figura 16 – Página do chat**

Na Figura 17 é apresentada a tela de informações da empresa, que pode ser acessada pelo ícone que representa uma engrenagem no cabeçalho do lado direito. Nesta tela o usuário terá a possibilidade de fazer alterações nos dados cadastrais, sendo que os campos buscados pelo CNPJ diretamente da Receita Federal estarão sempre desabilitados.

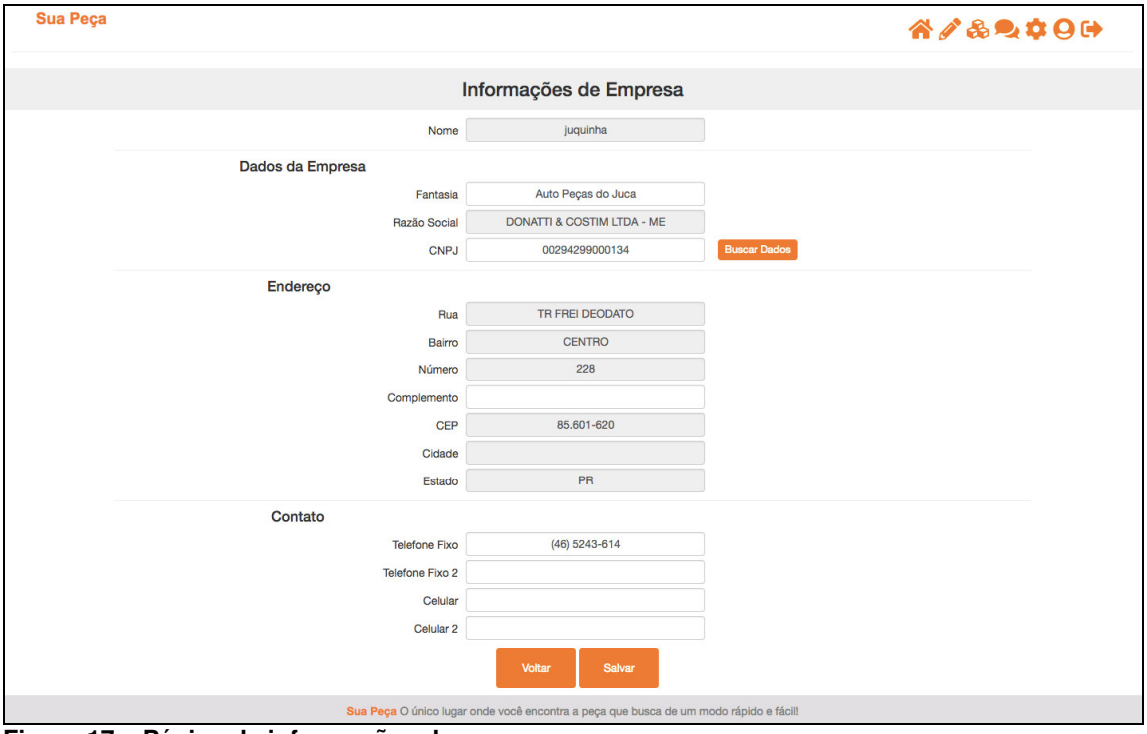


**Figura 17 – Página de informações da empresa**

## 4.4 IMPLEMENTAÇÃO DO SISTEMA

O sistema possui duas partes que são comuns para todas as telas, que são o cabeçalho e o seu rodapé. A seguir são apresentados exemplos de códigos de ambos e explicações de suas funções.

Na Figura 18 é possível observar o código HTML5 que pertence à parte visual do cabeçalho que é composto por uma *tag* “h1” que apresenta o nome do sistema na tela e um conjunto de *tags* “a” que são os *links* para navegação no sistema. Cada um contém uma *tag* “i” adicionada por uma “class” que é um ícone proveniente do *framework* de ícones FontAwesome. Todas as *tags* “a” utilizam um “routerLink” para designar qual é a rota da página a ser acessada, salvo a última *tag* pois ela utiliza

um evento de “(click)” para que chame o método “logout()” para que o usuário seja desconectado do sistema ao solicitar a execução dessa funcionalidade.

```
<div class="div-nav">
  <h1>Sua Peça</h1>
  <div class="div-links" [hidden]="hideButtons">
    <a routerLink="/home" routerLinkActive="active" title="Home Page">
      <i class="fa fa-home fa-2x"> </i>
    </a>
    <a routerLink="/cadpeca" routerLinkActive="active" title="Cadastrar Peça">
      <i class="fa fa-pencil-alt fa-2x"> </i>
    </a>

    <a routerLink="/pecasestoque" routerLinkActive="active" title="Peças Cadastradas">
      <i class="fa fa-cubes fa-2x"> </i>
    </a>

    <a routerLink="/chat" routerLinkActive="active" title="Chat">
      <i class="fa fa-comments fa-2x"> </i>
    </a>

    <a routerLink="/editempresa" routerLinkActive="active" title="Informações da Empresa">
      <i class="fa fa-cog fa-2x"> </i>
    </a>

    <a routerLink="/userinfo" routerLinkActive="active" title="Informações do Usuário">
      <i class="fa fa-user-circle fa-2x"> </i>
    </a>

    <a (click)="logout()" href="" title="Sair do sistema">
      <i class="fa fa-sign-out-alt fa-2x"> </i>
    </a>
  </div>
  <hr>
</div>
```

**Figura 18 – Código da estrutura HTML do cabeçalho**

Na Figura 19 está o início do arquivo que contém a lógica para apresentação visual do cabeçalho. E como em todos os componentes Angular2 haverá os *imports* para as classes nativas. Nesse caso, são as “Component, OnInit” e *imports* de classes criadas para o sistema sendo as “AuthService, EmpresaService” representando, dois serviços do Angular2.

```
import { Component,
         OnInit      } from '@angular/core';

import { AuthService } from '../services/authentication.service';
import { EmpresaService } from '../services/empresa.service';

```

**Figura 19 – Código da lógica do componente do cabeçalho**

A Figura 20 contém a declaração padrão de um componente Angular2 utilizando a diretiva “@Component”. Ele possui três propriedades que são o “selector” que é o nome da *tag* que o componente terá para ser utilizada em algum arquivo HTML para inserir o conteúdo do componente; o “styleUrls” que contém o caminho do arquivo de estilos para fazer o *design* visual do componente e o “templateUrl” que contém o caminho do arquivo que possui a estrutura visual do componente. Também nesta figura é apresentado como um componente é exportado pelo “export class..NomeComponente” para poder ser utilizado externamente. Neste componente é realizada a implementação de um método nativo do Angular2 o “... implements OnInit” descrito a seguir. Pode-se observar, ainda, que é apresentado o método de construção do componente, utilizado para realizar a declaração de atributos para os serviços importados AuthService e EmpresaService.

```
7   @Component({
8     selector: 'my-nav',
9     templateUrl: './nav.component.html',
10    styleUrls: [ './nav.component.css']
11  })
12  export class NavComponent implements OnInit{
13    userName;
14    hideButtons;
15
16    constructor(
17      private auth:       AuthService,
18      private empService: EmpresaService) {
19
20      }
```

**Figura 20 – Código da lógica do componente do cabeçalho**

Na Figura 21 estão listadas dois métodos implementados: ngOnInit e logout. O Método nativo “ngOnInit” será executado automaticamente no momento que o componente é carregado na tela do sistema. Neste caso, o método verificará, por meio do método “this.auth.isUserLogado()”, se o usuário está autenticado para ter acesso a página que ele está tentando acessar. Essa validação não será executada, caso o usuário esteja acessando as páginas de *login* ou de cadastro (/signup) do sistema.

No código da Figura 21, da linha 27 até a linha 37 é realizada uma validação para ocultar os botões dos ícones de navegação do cabeçalho. O método “logout()” realiza a chamada por meio do objeto “auth” anteriormente declarado para realizar o

acesso ao método “doLogout()” do serviço de autenticação para desconectar o usuário do sistema.

```
22    ngOnInit(){
23      if( (this.auth.route.url != '/login') && (this.auth.route.url != '/signup')){
24        this.auth.isUserLogado();
25      }
26
27      this.auth.afAuth.authState.subscribe(auth => {
28          if((auth == null) || (auth.email == 'admin@hotmail.com')){
29            this.hideButtons = true;
30          }else{
31            this.hideButtons = false;
32          }
33          if(this.auth.route.url == '/cadempresa'){
34            this.hideButtons = true;
35          }
36      });
37    }
38
39    logout(){
40      this.auth.doLogout();
41    }
42  }
```

**Figura 21 – Código da lógica do componente do cabeçalho**

Na Figura 22 está o código da estrutura HTML5 visual do rodapé, contendo algumas *tags* “div” utilizadas para agrupar e colocar as propriedades “class” da estilização do visual. Essa imagem também possui uma *tag* “p” que contém a mensagem de destaque do rodapé.

```
1  <div class="footer">
2    <div class="rodape">
3      <p class="pColorPadrao" id="fPrincipal">Sua Peça</p> <p>O único lugar onde você encontra a peça que
4        busca de um modo rápido e fácil!</p>
5    </div>
6  </div>
```

**Figura 22 – Código da estrutura HTML do componente do rodapé**

Na Figura 23 é possível observar a parte lógica do componente do rodapé, que é, basicamente, o esqueleto padrão do componente com o *import* da diretiva “Component” para ser declarado o componente. Esse componente possui as três propriedades que representam a *tag* do componente em HTML, o caminho para o estilo do componente e o caminho para o arquivo de estrutura em HTML do componente.

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'my-footer',
5    templateUrl: './footer.component.html',
6    styleUrls: ['./footer.component.css']
7  })
8  export class FooterComponent {
9
10 }
```

**Figura 23 – Código da lógica do componente do rodapé**

Na Figura 24 está a estrutura do arquivo com a estrutura visual da tela de *login* do sistema. Na linha 1 está a *tag* “my-nav” que representa o cabeçalho sendo inserido na página de *login*. Na linha 6 está uma *tag* “a” que possui um *link* para que o usuário possa ser direcionado à página de cadastro do sistema, caso necessário. Da linha 10 até a linha 33 está a declaração de um formulário e dos campos que o compõe, sendo que o seu envio é acionado pelo botão na linha 32, como ele é do tipo “submit” o envio do formulário é acionado automaticamente.

Na linha 13 está a declaração de um campo de edição para ser informado o *e-mail* do usuário para realizar *login* no sistema. Esse campo é obrigatório pelo uso da propriedade “required” e está vinculado à propriedade “model.email” da classe “model” na parte lógica do componente pelo uso da propriedade “[(ngModel)]”. Na linha 15 está um campo que apresenta uma mensagem de erro caso o *e-mail* informado seja inválido.

Na linha 28 está uma *tag* “a” que define um *link* para que o usuário possa redefinir a sua senha chamando a função “redefineSenha(model.email)”. Observa-se que na última linha dessa listagem de código está a declaração da *tag* “my-footer” que inserirá o rodapé na página de *login* do sistema.

```
<my-nav></my-nav>
<div class="container">
  <div>
    <h2>Entre</h2>

    <a routerLink="/signup" routerLinkActive="active">
      <h4>Não possui cadastro? Cadastre-se!</h4>
    </a>

    <form (ngSubmit)="onSubmit()" #loginForm="ngForm">

      <div class="form-group">
        <input type="text" class="form-control" id="email" required [(ngModel)]="model.email"
          name="email" #email="ngModel" placeholder="E-mail">
        <div [hidden]="email.valid || email.pristine" class="alert alert-danger alerta-erro">
          {{ userErrorMessage }}
        </div>
      </div>

      <div class="form-group">
        <input type="password" class="form-control" id="password" required [(ngModel)]="model.password"
          name="password" #password="ngModel" placeholder="Senha">
        <div [hidden]="password.valid || password.pristine" class="alert alert-danger alerta-erro">
          {{ passErrorMessage }}
        </div>
      </div>

      <a (click)="redefineSenha(model.email)">
        <h4>Esqueceu a senha?</h4>
      </a>

      <button type="submit" class="btn" [disabled]="!loginForm.form.valid">Entrar</button>
    </form>
  </div>
</div>
<my-footer></my-footer>
```

**Figura 24 – Código da estrutura HTML da tela de login**

Na Figura 25 está parte da lógica do componente de *login* do sistema, entre as linhas 6 e 10 está a declaração das propriedades padrões desse componente, sendo a sua *tag* para o HTML no “selector” e os dois caminhos para os arquivos de estrutura e *design*. A partir da linha 11 está o *export* do componente e três atributos são criados, o “model” para receber os dados de “e-mail” e “senha” do usuário para o *login* e os atributos “userErrorMessage” e “passErrorMessage” que mostrarão as mensagens de erro na tela caso o usuário ou a senha estejam incorretos, sendo que eles são inicializados no construtor do componente.

```
6   @Component({
7     selector: 'login-page',
8     templateUrl: './login.component.html',
9     styleUrls: [ './access.css']
10  })
11  export class LoginComponent {
12    model = new Login();
13    userErrorMessage : string;
14    passErrorMessage : string;
15
16    constructor(
17      private auth       : AuthService
18    ) {
19      this.userErrorMessage = 'Usuário obrigatório';
20      this.passErrorMessage = 'Senha obrigatória';
21    }
```

**Figura 25 – Código da lógica do componente de login**

Na Figura 26 entre as linhas 23 e 25 está o método “onSubmit()” que será chamado ao ser enviado o formulário contendo as informações para o *login* do usuário, que por meio do objeto “auth” é realizado o acesso ao método “doLogin()” passando o *e-mail* e senha do usuário para o método. A partir da linha 27 é definida a função “redefineSenha()” que faz a chamada por meio do objeto “auth” do método “...sendPasswordResetEmail()” do serviço de autenticação para que seja enviado um *e-mail* com o *link* para o usuário redefinir a sua senha.

```
23    onSubmit() {
24      this.auth.doLogin(this.model.email, this.model.password);
25    }
26
27    redefineSenha(email : string){
28      if((email != null) && (email != '')){
29        this.auth.afAuth.auth.sendPasswordResetEmail(email);
30        alert('Um email foi enviado para a sua caixa de entrada com um link para redefir a senha');
31      }else{
32        alert('Informe um email para redefinir a senha!');
33      }
34    }
```

**Figura 26 – Código da lógica do componente de login**

Na Figura 27 está o início da estrutura visual da tela de cadastro do usuário. Na linha 1 dessa listagem está, novamente, a *tag* declarando o cabeçalho na tela. Entre as linhas 6 e 8 há um *link* para caso o usuário já possua uma conta, ele possa ser redirecionado para a tela de *login* do sistema.

```
<my-nav></my-nav>

<div class="container">
  <div>
    <h2>Criar Conta</h2>
    <a routerLink="/login" routerLinkActive="active">
      <h4>Já possui cadastro? Entre!</h4>
    </a>

```

**Figura 27 – Código da estrutura HTML da tela de cadastro de usuário**

Na Figura 28, a partir da linha 10, está a declaração do início do formulário que conterá os campos para serem informados os dados do usuário, que serão enviados para a chamada do método “onSubmit()”. Entre as linhas 12 e 17 está um exemplo de campo de edição e de validação se o conteúdo do campo for inválido. Essas mensagens serão repetidas para cada campo do formulário, sempre utilizando a propriedade “[(ngModel)]” para popular o objeto usuário na parte lógica do componente.

```
    <form (ngSubmit)="onSubmit()" #loginForm="ngForm">
      <div class="form-group">
        <input type="text" class="form-control" id="name" required [(ngModel)]="model.nome"
          name="nome" #nome="ngModel" placeholder="Seu Nome">
        <div [hidden]="nome.valid || nome.pristine" class="alert alert-danger alerta-erro">
          Nome Obrigatório
        </div>
      </div>
```

**Figura 28 – Código da estrutura HTML da tela de cadastro de usuário**

É interessante observar na Figura 29 que ela contém um botão e um campo de edição para que seja informada a imagem do usuário, que ao clicar no botão disparará o evento “(change)” chamando o método “detectaArquivo($event)” da parte lógica do componente. Se uma imagem foi selecionada, será realizado o seu *upload* no momento de salvar as informações.

```
      <div class="form-group">
        <label class="input-group-btn">
          <span class="btn btn-primary sel">
            Selecionar&hellip; <input type="file" style="display: none;"
              (change)="detectaArquivo($event)" multiple>
          </span>
          <input type="text" class="form-control" readonly style="width: 207px; float: left;"
            value="{{filename}}" placeholder="Imagem de perfil">
        </label>
      </div>
```

**Figura 29 – Código da estrutura HTML da tela de cadastro de usuário**

Na Figura 30 há um *link* declarado que realiza a chamada o método “enviaCodigoValidador()” para caso o usuário não possua um código para criar a sua conta, ele possa requisitar um que será enviado para o seu *email*.

```
51        <a (click)="enviaCodigoValidador(model.email, model.nome)">
52          <h4>Não possui código validador? Informe o e-mail e receba o seu!</h4>
53        </a>
```

**Figura 30 – Código da estrutura HTML da tela de cadastro de usuário**

Na Figura 31 está a declaração da parte lógica do componente da tela de cadastro de usurário, havendo a declaração do objeto “model” para ser populado com os dados do usuário, atributo “imgPeca” que receberá a imagem selecionada para o *upload* e os atributos “filename” e “newImage” utilizados para fins de validação.

```
7   @Component({
8     templateUrl: './signup.component.html',
9     styleUrls: ['./access.css']
10  })
11  export class SignUpComponent {
12    model = new User();
13    imgPeca: FileList;
14    filename = '';
15    newImage;
```

**Figura 31 – Código da lógica do componente de cadastro de usuário**

Na Figura 32 está a implementação do método “detectaArquivo()” que é chamado na parte visual pelo botão de adicionar imagem no cadastro de usuário. Esse botão por meio do clique dispara um evento (“event”) que atribuirá para o atributo “imgPeca” o arquivo da imagem e para os atributos “filename” e “newImage” o nome do arquivo.

```
21    detectaArquivo(event){
22      this.imgPeca = event.target.files;
23      this.filename = this.imgPeca.item(0).name;
24      this.newImage = this.imgPeca.item(0).name;
25    }
```

**Figura 32 – Código da lógica do componente de cadastro de usuário**

Na Figura 33 está a implementação do método “onSubmit()” que realizará o envio das informações inseridas no formulário do cadastro do usuário. Como todas as operações envolvendo o banco de dados precisam ter um usuário autenticado, na

linha 28 está uma função chamada “doLoginAdmin()” por meio da qual é realizado o *login* de um usuário temporário para que sejam feitas as operações necessárias no banco e após isso é realizado o *logout*. Há dois modos de inserção do usuário apresentados sendo que ambos chamam o mesmo método “doSignUp()”, mas com uma diferença: a chamada da linha 31 está criando o usuário sem imagem de perfil e a da linha 40 está criando o mesmo usuário com a inserção da imagem.

```
27  onSubmit() {
28    this.auth.doLoginAdmin();
29    setTimeout(() =>{
30      if(this.filename == ''){
31        this.auth.doSignUp(
32          this.model.email,
33          this.model.password,
34          this.model.codValidador,
35          this.model.nome,
36          undefined);
37      }else{
38        let file = this.imgPeca.item(0);
39        let img = new Upload(file);
40        this.auth.doSignUp(
41          this.model.email,
42          this.model.password,
43          this.model.codValidador,
44          this.model.nome,
45          img);
46      }
47    },1500);
48  }
```

**Figura 33 – Código da lógica do componente de cadastro de usuário**

Na Figura 34 está o método “enviaCodigoValidador()” que realiza o envio do código para o usuário fazer seu cadastro. Utilizando o objeto “auth” é chamada o método do serviço “enviarEmailCodValidador()” e enviado um *email* para o usuário contendo o código de acesso.

```
50  enviaCodigoValidador(email, nome : string){
51    if ( (email != null) && (email != '')){
52      this.auth.enviarEmailCodValidador(email, nome);
53      alert('Um email contendo o código validador foi enviado, verifique sua caixa de entrada!');
54    }else{
55      alert('Informe um endereço de email para receber o código validador!');
56    }
57  }
```

**Figura 34 – Código da lógica do componente de cadastro de usuário**

A Figura 35 apresenta parte de estrutura da tela de *chat* do sistema. Na linha 2 está a declaração do cabeçalho, seguido pela declaração de uma *tag* “ul” que representa uma lista contendo itens da *tag* “li” que serão criados a partir de um *loop*

feito na linha 8 utilizando a diretiva do Angular2 “ngFor”. Isso é utilizado para listar todos os usuários do sistema em um menu lateral, sendo que a propriedade “[class.selected]” verifica se o usuário sendo listado no momento é o que está selecionado. Caso seja, ele terá uma cor de fundo diferente para ser destacado. Na linha 10 está o *link* que é criado para cada item de usuário listado que possui a declaração do “(click)” que realiza a chamada do método “carregaMensagens()” para carregar a conversa entre o usuário logado e o usuário selecionado.

```
2   <my-nav></my-nav>
3     <div class="main">
4       <h3 class="no-margin-bottom">Mensagens</h3>
5     <div class="sidebar">
6       <h6 class="usuarios-chat">Usuários</h6>
7       <ul>
8         <li *ngFor="let usuario of listaUsuarios" class="usuario-item"
9           [class.selected]="usuario.id === selectedUser">
10          <a (click)="carregaMensagens(usuario.id)">{{usuario.nome}}</a>
11        </li>
12      </ul>
13    </div>
```

**Figura 35 – Código da estrutura HTML da tela de Chat**

Na Figura 36 está parte do código que apresenta visualmente as mensagens aos usuários. Na linha 18 está a diretiva “ngFor” para realizar a listagem de cada “msg” da lista “conversa” como um item cada. Entre as linhas 19 e 27 é mostrado como será listada a mensagem, sendo a imagem do usuário primeiro e após o seu nome, data-hora e em seguida será listada a mensagem.

```
15    <div class="mensagens">
16      <div class="espaco-bottom">
17        <ul [hidden]="!showContent" class="lateral">
18          <li *ngFor="let msg of conversa | async" class="msg-user-one">
19            <div class="div-item">
20              <div>
21                <img class="user-pic" src="{{msg.imgPic}}" [hidden]="msg.imgPic == undefined"/>
22                <b>{{msg.nome}}</b>  <span class="timestamp">{{msg.timestamp | date:'short' }}</span>
23              </div>
24              <div>
25                {{msg.mensagem}}
26              </div>
27            </div>
28          </li>
29        </ul>
30      </div>
```

**Figura 36 – Código da estrutura HTML da tela de Chat**

Na Figura 37, entre as linhas 32 e 40, é declarado o início e o fim de um formulário que conterá o campo de edição da mensagem e o botão para o seu envio. O botão dispara o evento padrão do formulário “onSubmit()”.

```
32        <form (ngSubmit)="onSubmit()" #msgForm="ngForm">
33          <div class="form-group no-margin">
34            <input type="text" class="form-control chaves fim msg-box" id="mensagem" required
35              [(ngModel)]="msg.mensagem" name="mensagem" #mensagem="ngModel" >
36            <button type="submit" class="btn fim btn-envia-msg" [disabled]="!msgForm.form.valid">
37              Enviar
38            </button>
39          </div>
40        </form>
41      </div>
42      </div>
43    <my-footer></my-footer>
```

**Figura 37 – Código da estrutura HTML da tela de Chat**

A Figura 38 apresenta o início da parte lógica do componente do *chat*, com os *imports* padrão entre as linhas 1 e 3, e nas linhas 6 a 7 *imports* dos serviços de autenticação e de mensagem. Na linha 8 está o *import* da classe “Mensagem” que será utilizado para passar os dados da mensagem para o serviço tratar o envio.

```
1  import { Component,
2           OnInit          } from '@angular/core';
3  import { ActivatedRoute,
4           Params          } from '@angular/router';
5
6  import { AuthService     } from '../services/authentication.service';
7  import { MensagemService } from '../services/mensagem.service';
8  import { Mensagem        } from '../common/classes';
```

**Figura 38 – Código da lógica do componente de Chat**

A Figura 39 contém a declaração padrão do componente, que está entre as linhas 10 e 12. Ressalta-se que na declaração das variáveis está a “msg” que é criada a partir da classe que foi importada anteriormente. Há, ainda, atributos para identificar os usuários da conversa declaradas nas linhas 17 e 18. Na linha 20 está o atributo que receberá a lista contendo as mensagens trocadas entre os usuários. No construtor há um objeto declarado de uma classe ainda não abordada que é a “ActivatedRoute”, por meio desse objeto será possível identificar se na url passada possui parâmetros.

```
10  @Component({
11    templateUrl: './chat.component.html',
12    styleUrls: ['../crud/style-geral.css']
13  })
14  export class ChatComponent implements OnInit{
15    listaUsuarios;
16    msg = new Mensagem();
17    uidSender;
18    uidReceiver;
19    selectedUser;
20    conversa: any;
21    showContent = false;
22    imgUser;
23
24    constructor(
25      private auth:        AuthService,
26      private msgService:  MensagemService,
27      private router:      ActivatedRoute
28    ){
29
30    }
```

**Figura 39 – Código da lógica do componente de Chat**

A implementação do método nativo “ngOnInit” é apresentado na listagem da Figura 40. Na linha 33 está o uso do objeto “auth” para chamar o método “listaUsuarios()” para retornar os usuários cadastrados no sistema e listá-los no painel lateral. A partir da linha 34 a 38 é feita uma busca pelo id do usuário na base de dados para recuperar a imagem correspondente, para ser mostrada na listagem das mensagens.

```
32    ngOnInit(){
33      this.listaUsuarios = this.auth.listaUsuarios();
34      this.auth.afAuth.authState.subscribe(auth => {
35        this.uidSender = auth.uid;
36        this.auth.buscaDadosUser(auth.uid).then((data) => {
37          this.imgUser = data.urlPerfilPic;
38        });
```

**Figura 40 – Código da lógica do componente de Chat**

Na Figura 41, ainda no método “ngOnInit()”, entre as linhas 40 e 54 está o tratamento utilizando a variável “router” declarada anteriormente para fazer um *loop* percorrendo os parâmetros que possam ter sido passados nesta rota. O parâmetro buscado é o “user”, sendo que entre as linhas 41 e 45 é realizada uma verificação se

caso o “params[“user”]” tenha um valor definido ele será atribuído para o atributo “uiReceiver” utilizando o método “atob” no valor, pois ele é enviado codificado.

Na linha 46 é passado o usuário selecionado, caso haja algum, para o atributo “selectedUser” que fará o tratamento na parte visual para destacar o usuário na listagem mostrando que ele é o selecionado com a conversa carregada. Entre as linhas 47 e 51 é realizada a busca das mensagens utilizando o atributo “msgService” para acessar o método do serviço “listaMensagens()” recuperando a conversa entre dois usuários, caso exista alguma mensagem o conteúdo será apresentado ao usuário.

```
40        this.router.params.forEach((params: Params) =>{
41          if (params["user"] != undefined) {
42            this.uidReceiver = atob(params["user"]);
43          }else{
44            this.uidReceiver = '';
45          }
46          this.selectedUser = this.uidReceiver;
47          this.conversa = this.msgService.listaMensagens(this.uidSender, this.uidReceiver);
48          if(this.conversa.length > 0){
49            this.showContent = false;
50          }else{
51            this.showContent= true;
52          }
53        });
54      });
```

**Figura 41 – Código da lógica do componente de Chat**

A Figura 42 apresenta a implementação do método de envio do formulário. Nesse método é montada a mensagem com as informações da tela. Na linha 58 é passado o id do usuário que está enviando a mensagem. Na linha 59 é passada a data-hora do envio da mensagem. E entre as linhas 60 e 65 é realizada uma busca nos dados do usuário utilizando o id para recuperar o nome e a imagem de perfil, para serem enviadas junto com a mensagem. Na linha 63 é realizada a chamada por meio do objeto do serviço de mensagem o método “enviaMensagem()” para realizar o envio da mensagem. E na linha 64 é reinicializada o objeto sendo informados os dados de cada mensagem nova.

```
57   onSubmit(){
58     this.msg.uid = this.uidSender;
59     this.msg.timestamp = new Date().getTime();
60     this.auth.buscaDadosUser(this.msg.uid).then(data =>{
61       this.msg.nome   = data.nome;
62       this.msg.imgPic = data.urlPerfilPic;
63       this.msgService.enviaMensagem(this.msg, this.msg.uid, this.uidReceiver);
64       this.msg = new Mensagem();
65     });
66   }
```

**Figura 42 – Código da lógica do componente de Chat**

Na Figura 43 está o método que é chamado para carregar as mensagens de uma conversa ao ser selecionado o nome de um usuário no painel lateral, chamando novamente o método “listaMensagens()” do serviço de mensagens.

```
68   carregaMensagens(id: string){
69     this.conversa = this.msgService.listaMensagens(this.uidSender, id);
70     this.uidReceiver = id;
71     this.selectedUser = id;
72     if(this.conversa.length > 0){
73       this.showContent = false;
74     }else{
75       this.showContent= true;
76     }
77   }
```

**Figura 43 – Código da lógica do componente de Chat**

Na Figura 44 está a parte estrutural da tela principal (Home) do sistema. Entre as linhas 5 e 21 está a declaração dos dois combos que estão na tela acima da caixa de busca, na qual são listados os estados e as marcas para serem utilizadas no filtro da busca.

```
<my-nav></my-nav>
<div [hidden]="!showContent">
  <div class="home-content">
    <form (ngSubmit)="onSubmit()" #homeForm="ngForm">
      <div class="centraliza-filtros">
        <div class="form-group no-break-line home-center-align alinha-centro">
          <select class="btn input selector form-control" id="uf" required
            [(ngModel)]="pecaBuscaDet.uf" name="marca" #uf="ngModel" value="{{uf.$value}}" >
            <option *ngFor="let uf of estados | async">
              {{ uf.$value }}
            </option>
          </select>
        </div>
        <div class="form-group no-break-line home-center-align">
          <select class="btn input selector form-control" id="marca" required
            [(ngModel)]="pecaBuscaDet.marca" name="marca" #marca="ngModel" value="{{marca.$value}}">
            <option *ngFor="let marca of marcas | async">
              {{ marca.$value }}
            </option>
          </select>
        </div>
```

**Figura 44 – Código da estrutura HTML da tela Home**

A Figura 45, declarado entre as linhas 23 e 27, está o campo de edição para ser informado o texto a ser utilizado na busca. E nas linhas 28 a 30 está a declaração do botão que será utilizado para acionar o envio do formulário com as informações para a busca da peça.

```
        <div class="form-group home-center-align">
          <input type="text" class="form-control search-box-peca" id="pecaBuscaDet.textoBusca" required
            [(ngModel)]="pecaBuscaDet.textoBusca" name="textoBusca" #textoBusca="ngModel"
            placeholder="Busque a sua peça...">
        </div>
        <div class="home-center-align">
          <button type="submit" class="btn" [disabled]="!homeForm.form.valid">Buscar</button>
        </div>
```

**Figura 45 – Código da estrutura HTML da tela Home**

Na Figura 46 está a implementação da parte lógica do componente da página principal. No método nativo “ngOnInit()” entre linhas 28 a 31 é realizado o carregamento das marcas e dos estados para os atributos que serão atribuídos para os campos de *combo* na tela visual, sendo escolhido nas linhas 30 e 31 um estado e uma marca padrão para estarem inicialmente selecionados.

```
27   ngOnInit(){
28     this.marcas  = this.pecaService.getMarcas();
29     this.estados = this.pecaService.getEstados();
30     this.pecaBuscaDet.uf    = 'Acre';
31     this.pecaBuscaDet.marca = 'BMW';
32     setTimeout(() =>{
33       this.showContent = true;
34     },1400);
35   }
```

**Figura 46 – Código da lógica do componente da Home**

Na Figura 47 está a implementação do método chamado ao ser realizada a busca. É utilizado o método “navigate” do objeto “route” para acessar a rota “/resultadobusca” e passados os parâmetros que serão utilizados na busca, listados na linha 39.

```
37   onSubmit(){
38     this.auth.route.navigate([`/resultadobusca`,`
39       ${this.pecaBuscaDet.textoBusca}|${this.pecaBuscaDet.uf}|${this.pecaBuscaDet.marca}`]);
40   }
```

**Figura 47 – Código da lógica do componente da Home**

A seguir está a apresentação dos serviços utilizados para realizar o processamento dos dados e as chamadas de métodos do *framework backend* do AngularFire2.

Na Figura 48 estão os *imports* e as declarações iniciais de classes nativas e criadas para o sistema utilizadas ao longo do serviço de autenticação. Algumas delas já foram discutidas e explicadas, mas há alguns novos como o “AngularFireAuth” na linha 7. Essa é uma classe do *framework* AngularFire2 utilizada para declarar um objeto de acesso ao banco de dados do Firebase. O *import* da classe “firebase” na linha 8 é utilizado para acessar o “storage” do Firebase no qual serão armazenadas as imagens de peças e os dados dos usuários. Entre as linhas 13 e 15 são realizados o *import* de classes criadas para o sistema utilizadas para o manuseio de dados nos métodos.

```
import { Injectable,
         Input             } from '@angular/core';
import { Router,
         ActivatedRoute,
         Params            } from '@angular/router';

import { AngularFireAuth   } from 'angularfire2/auth';
import * as firebase from 'firebase/app';
import 'rxjs/add/operator/map';

import { AngularFireDatabase    } from 'angularfire2/database';

import { User,
         Upload,
         Sugestao              } from '../common/classes';

import { EmpresaService         } from '../services/empresa.service';
```

**Figura 48 – Código da lógica do Serviço de Autenticação**

Na Figura 49, na linha 19, está a declaração do objeto “emailjs” utilizado para realizar o envio de *e-mails* no sistema.

```
19   declare var emailjs: any;
```

**Figura 49 – Código da lógica do Serviço de Autenticação**

Na Figura 50 pode ser observado o objeto declarada a partir do uso do script “emailjs” na página *index* do sistema. Nela é injetado o serviço para que seja possível ser realizado o envio de *e-mails* do sistema sem precisar utilizar recursos de servidor.

```
<script type="text/javascript" src="https://cdn.emailjs.com/dist/email.min.js"></script>
<script type="text/javascript">
   (function(){
      emailjs.init("user_SpT2hIQ14D0Ofl6SfADqU");
   })();
</script>
```

**Figura 50 – Código da lógica do Serviço de Autenticação**

Na Figura 51 está o construtor do serviço de autenticação com a listagem dos objetos utilizados nos métodos. Destacando-se a linha 27 que contém o objeto que realizará o acesso aos métodos de autenticação do Firebase e a linha 29 que fará a comunicação com os dados no banco.

```
21  @Injectable()
22  export class AuthService {
23    sub;
24    private basePath:string = '/usuarioimagens';
25
26    constructor(
27      public  afAuth:     AngularFireAuth,
28      public  route:      Router,
29      private db:         AngularFireDatabase,
30      public  acRoute:    ActivatedRoute,
31      private empService: EmpresaService){
32
33    }
```

**Figura 51 – Código da lógica do Serviço de Autenticação**

Na Figura 52 está o método “isUserLogado()” que é chamado no cabeçalho do sistema e que realiza a verificação se um usuário está autenticado no sistema ou não. Sendo, assim, possível determinar se ele possui acesso à página que está tentando acessar. Na linha 37 é realizada uma subscrição ao objeto de autenticação “afAuth” anteriormente declarado e caso o objeto “auth” esteja *null* o usuário não está autenticado e será direcionado para a página de *login* do sistema.

```
35  isUserLogado(){
36    let sub;
37    sub = this.afAuth.authState.subscribe(auth => {
38      if(auth == null){
39        alert(`Você deve estar logado para ter acesso ao sistema!
40          Redirecionando para a página de login...`);
41        this.route.navigate(['/login']);
42      }
43      setTimeout(() =>{
44          sub.unsubscribe();
45        },300);
46      });
47  }
```

**Figura 52 – Código da lógica do Serviço de Autenticação**

Na Figura 53 está a implementação do método que realiza o *login* do usuário no sistema. Da linha 50 à linha 69 é realizada a chamada do método “signInWithEmailAndPassword()” utilizando o objeto “afAuth” para acessar os métodos da classe de autenticação do *framework* AngularFire2. Caso haja sucesso (na linha 51 “.then...”) nas linhas 53 a 58 é realizada uma validação para verificar se

o usuário realizando o *login* já possui o cadastro da empresa completo. Se sim, ele é redirecionado à página principal (linha 54). Se não, ele será redirecionado para a página de cadastro da empresa (linha 56). Caso ocorra algum erro, o tratamento é realizado a partir da linha 60 a 68, validando se o *e-mail* e/ou senha estão corretos.

```
49  doLogin(email, password){
50    this.afAuth.auth.signInWithEmailAndPassword(email, password)
51      .then((value) =>{
52        this.empService.buscaDadosEmpresa(this.afAuth.auth.currentUser.uid).then((data) => {
53          if(data.razaoSocial != null){
54            this.route.navigate(['/home']);
55          }else{
56            this.route.navigate(['/cadempresa']);
57          }
58        });
59      })
60      .catch( (error) => {
61        if ((error.message === `The email address is badly formatted.`) ||
62            (error.message === `There is no user record corresponding to this identifier.
63             The user may have been deleted.`)){
64          alert('Email Invalido');
65        }else if(error.message === `The password is invalid or the user does not have a password.`){
66          alert('Senha incorreta');
67        }
68      });
69  }
```

**Figura 53 – Código da lógica do Serviço de Autenticação**

Na Figura 54 está o método que realiza o *logout* do usuário do sistema, sendo bem simples. Por meio do objeto “afAuth” é acessado o método “signOut()”,caso obtenha sucesso (linha 73) o usuário será direcionado à página de *login* e caso haja erro (linha 76) ele será direcionado à página de erro padrão 404.

```
71    doLogout(){
72      this.afAuth.auth.signOut()
73      .then( (sucess) => {
74        this.route.navigate(['/login']);
75      })
76      .catch( (err) =>{
77        this.route.navigate(['**']);
78      })
79    }
```

**Figura 54 – Código da lógica do Serviço de Autenticação**

Na Figura 55 está o método que realiza a criação do usuário no sistema sendo que na linha 82 é recuperada uma lista de objetos contendo os códigos de validação para a criação de usuários no sistema. Na linha 83 é feita uma subscrição

a esta lista para verificar se o valor da variável “codValidador” passado por parâmetro existe nesta lista e se na sua propriedade “st_cadastrado” há o valor de “N”. Isso significa que é um código válido e pode ser utilizado para a inserção do novo usuário. Caso inválido será abortada a operação e apresentada uma mensagem ao usuário (linha 106).

Após essa verificação na linha 86 é utilizada o objeto “afAuth” para acessar o método “createUserWithEmailAndPassword()” do *framework backend* AngularFire2. Caso haja sucesso na operação (linha 87) será apresentada uma mensagem ao usuário (linha 88) e ele será redirecionado para a tela de cadastro da empresa (linha 89). Além de ser executada a chamada do método “invalidaCod()” para mudar o valor da propriedade “st_cadastrado” do código utilizado para “S”, invalidando-o.

Caso aconteça algum erro será tratado entre as linhas 92 e 100, sendo apresentada uma mensagem condizente com o erro encontrado. Na linha 101 é removida a subscrição do evento, ou seja, qualquer mudança que ocorrer não será mais observada. E entre as linhas 102 e 104 é executado um *sleep* de três segundos para que a requisição no Firebase seja tratada e, em seguida, sejam salvos os dados do usuário na base de dados.

```
81  doSignUp(email, password, codValidador, nome : string, img: Upload){
82    let codigos = this.db.object('/codigos', { preserveSnapshot: true});
83    this.sub = codigos.subscribe(snapshot => {
84      if( (snapshot.child(codValidador).exists()) &&
85          (snapshot.child(codValidador).val().st_cadastrado === 'N')){
86        this.afAuth.auth.createUserWithEmailAndPassword(email, password)
87          .then( (value) =>{
88            alert('Conta criada com sucesso!');
89            this.route.navigate(['/cadempresa']);
90            this.invalidaCod(codValidador);
91          })
92          .catch( (error) => {
93            if(error.message === `The email address is badly formatted.`){
94            alert('O e-mail informado é inválido!');
95            } else if (error.message === `The email address is already in use by another account.`) {
96              alert('O e-mail informado já possui uma conta cadastrada!');
97            } else if (error.message === `Password should be at least 6 characters`){
98              alert('A senha deve possuir pelo menos 6 caracteres!');
99            }
100         });
101       this.sub.unsubscribe();
102       setTimeout(() => {
103         this.salvaInfoUser(nome, img);
104       },3000);
105     }else{
106       alert('Código validador inválido!');
107     }
108   });
109 }
```

**Figura 55 – Código da lógica do Serviço de Autenticação**

A Figura 56 apresenta a declaração do método “invalidaCod()” que invalida o código validador utilizado pelo usuário para criar a sua conta. É simples, apenas buscando uma referência do banco de dados na linha 112, referenciando o código informado via parâmetro e realizando a alteração do valor para “S” na linha 113.

```
111    invalidaCod(codValidador : string){
112      let ref = this.db.database.ref(`/codigos/${codValidador}`);
113      ref.set({st_cadastrado: "S"});
114    }
```

**Figura 56 – Código da lógica do Serviço de Autenticação**

Na Figura 57 está o método “buscaDadosUser()”. Na linha 152 é declarado um objeto do tipo “User” e na linha 153 é buscada a referência aos dados do usuário utilizando o id que foi passado por parâmetro no método. Após isto, entre as linhas 155 e 160 é realizada a busca utilizando o objeto “ref” que contém a referência e é executado o método “once()” que busca os dados do registro apenas uma vez e após recebidos eles são atribuídos para as propriedades do objeto “dadosUsuario” que foi declarado anteriormente.

```
151    buscaDadosUser(id : string){
152      let dadosUsuario = new User();
153      let ref = this.db.database.ref(`/usuarios/${id}`);
154
155      return ref.once("value")
156        .then(function(snapshot){
157          dadosUsuario.nome         = snapshot.child("nm_usuario").val();
158          dadosUsuario.urlPerfilPic = snapshot.child("img_usuario").val();
159          return dadosUsuario;
160        });
161    }
```

**Figura 57 – Código da lógica do Serviço de Autenticação**

Na Figura 58 está o método “enviarEmailCodValidador()” que realiza o envio de um *email* com um código validador gerado para o usuário utilizar no seu cadastro. Na linha 172 é declarado o atributo “codigoValidador” e gerado um número aleatório de 5 ou 6 dígitos. Na linha 173 é realizado a chamada do método “send()” do objeto “emailjs” anteriormente declarado passando alguns parâmetros como o “outlook” que é o servidor de *email* utilizado, o “template_xUlsCe52” que é a referência do *template* a ser utilizado. Na linha 174 estão os objetos utilizados no *template* para preencher com valores.

Como todas as operações na base de dados exigem um usuário, na linha 175 está um método padrão que realiza o *login* de um usuário admin. Entre as linhas 177 e 180 é referenciada a "tabela" dos códigos e é criado o registro do "codigoValidador" gerado anteriormente com a sua propriedade "st_cadastrado" com o valor "N" deixando-o válido.

```
171   enviarEmailCodValidador(email, nome : string){
172     let codigoValidador = Math.round(Math.random() * 1000000);
173     emailjs.send("outlook","template_xUlsCe52",
174       {name: nome, codigoValidador: codigoValidador, toemail: email});
175     this.doLoginAdmin();
176     setTimeout(() =>{
177       let ref = this.db.database.ref("/codigos/");
178       ref.child(codigoValidador.toString()).set({
179         'st_cadastrado': 'N'
180       });
181       setTimeout(this.doLogoutAdmin(), 4000);
182     },2000);
183   }
```

**Figura 58 – Código da lógica do Serviço de Autenticação**

Na Figura 59 está o *template* que é configurável no serviço utilizado em (http://dashboard.emailjs.com/) e, como explicado anteriormente, os valores passados nos objetos na linha 174 do código apresentado na Figura 58 substituem as variáveis que são adicionadas entre "{{ }}" nos campos do *template*.

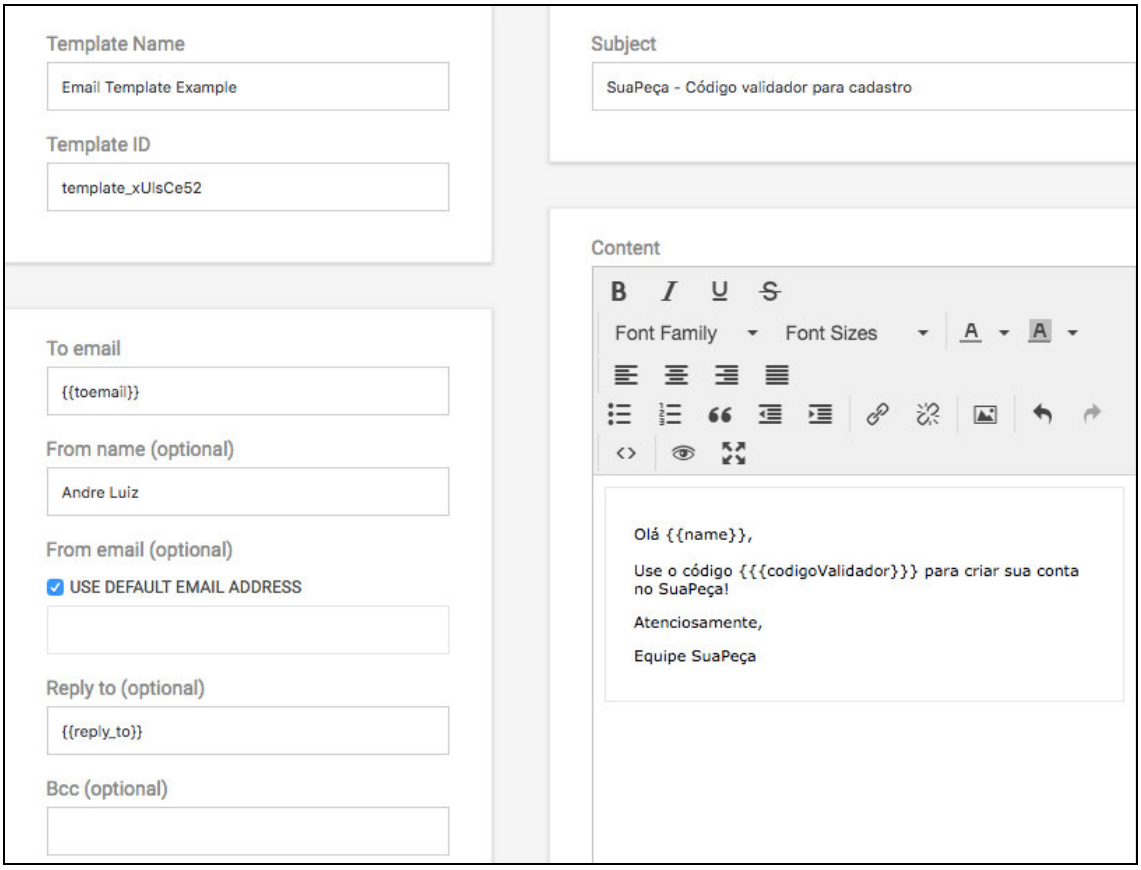


**Figura 59 – Template de envio do Código Validador**

Na Figura 60 estão os *imports* iniciais do serviço de mensagens. Na linha 1 está o *import* padrão para realizar a declaração de um serviço Angular, já na linha 2 está o *import* da classe "AngularFireDatabase" do *framework* AngularFire2 utilizado para realizar a comunicação com a base de dados. Na linha 4 está o *import* do serviço de autenticação criado para o sistema e na linha 5, o *import* da classe "Mensagem" que será utilizada para declarar a variável responsável por receber os dados de cada mensagem.

```
1  import { Injectable           } from '@angular/core';
2  import { AngularFireDatabase  } from 'angularfire2/database';
3
4  import { AuthService          } from '../services/authentication.service';
5  import { Mensagem             } from '../common/classes';
```

**Figura 60 – Código da lógica do Serviço de Mensagens**

A Figura 61 apresenta a declaração padrão do serviço Angular criando o objeto "mensagem" utilizando a classe "Mensagem", anteriormente importada, e

criando no construtor os objetos para acesso à base de dados (“db”) e para o uso do serviço de autenticação do sistema (“auth”).

```
7   @Injectable()
8   export class MensagemService {
9     mensagem = new Mensagem();
10
11    constructor(
12      private db: AngularFireDatabase,
13      private auth: AuthService){
14
15    }
```

**Figura 61 – Código da lógica do serviço de mensagens**

Na Figura 62 é apresentado o método “enviaMensagem()”. Na linha 18 é buscada a referência da tabela mensagens na base de dados e na linha 19 é criado um id salvo na variável “path” que é, simplesmente, a união dos ids dos dois usuários da conversa. Após isso, entre as linhas 20 e 25, é utilizada o objeto “ref” para acessar o método “child()” para realizar a inserção da nova mensagem na base de dados.

```
17    enviaMensagem(mensagem: Mensagem, uid1, uid2: string){
18      let ref = this.db.database.ref("/mensagens");
19      let path = uid1 < uid2 ? uid1+'/'+uid2 : uid2+'/'+uid1;
20      ref.child(path).push({
21        "mensagem":  mensagem.mensagem,
22        "timestamp": mensagem.timestamp,
23        "nome":      mensagem.nome,
24        "imgPic":    mensagem.imgPic
25      });
26    }
```

**Figura 62 – Código da lógica do serviço de mensagens**

Na Figura 63 está o método “listaMensagens()” que é chamado para carregar as mensagens de uma conversa utilizando os ids dos usuários passados por parâmetro. Na linha 29 é montado o id do caminho na base para a conversa dos usuários e na linha 30 é utilizada o objeto “db” para acessar o método “list()” que listará as mensagens da conversa tendo como base o id de caminho (“path”) informado e, assim, retornar na linha 31 o objeto “mensagens” contendo uma lista.

```
28    listaMensagens(uid1, uid2: string){
29      let path = uid1 < uid2 ? uid1+'/'+uid2 : uid2+'/'+uid1;
30      let mensagens = this.db.list(`/mensagens/${path}`);
31      return mensagens;
32    }
```

**Figura 63 – Código da lógica do serviço de mensagens**

Na Figura 64 está a parte inicial dos *imports* do serviço, que, contém, também, *import* da classe “FirebaseListObsersvable” do *framework* AngularFire2 para recuperar listas de objetos da base. O *import* da linha 11 é utilizado para gerar o arquivo pdf do relatório. E entre as linhas 13 a 15 estão alguns *imports* de classes que foram criadas para o sistema e serão utilizadas para o manuseio de dados durante o processamento.

```
1   import { Injectable,
2            Input                  } from '@angular/core';
3   import { Router                 } from '@angular/router';
4
5   import { AngularFireAuth        } from 'angularfire2/auth';
6   import * as firebase from 'firebase/app';
7   import { AngularFireDatabase,
8            FirebaseListObservable } from 'angularfire2/database';
9
10  import 'firebase/storage';
11  import * as jsPDF from 'jspdf';
12
13  import { Peca,
14           Upload,
15           EmpModel        } from '../common/classes';
16  import { EmpresaService  } from '../services/empresa.service';
```

**Figura 64 – Código da lógica do serviço de peças**

Na Figura 65 está a declaração padrão do serviço Angular2 com alguns objetos declarados como “marcas” e “estados” que receberão duas listas contendo os seus respectivos valores para serem utilizados na parte visual dos componentes. É declarado o objeto “basePath” determinando o caminho padrão do local no “storage” do Firebase no qual serão salvas as imagens de cada peça cadastrada. No construtor do serviço está a declaração das variáveis dos *imports* realizados para que os seus métodos possam ser acessados.

```
18  @Injectable()
19  export class PecaService {
20    marcas:  FirebaseListObservable<any>;
21    estados: FirebaseListObservable<any>;
22    private basePath:string = '/pecasimagens';
23
24    constructor(
25      public afAuth:          AngularFireAuth,
26      public route:           Router,
27      private db:             AngularFireDatabase,
28      private empresaService: EmpresaService){
29
30    }
```

**Figura 65 – Código da lógica do serviço de peças**

Na Figura 66 está o método “salvarPeca()” para realizar a inserção dos dados da peça informada pelo usuário. Entre as linhas 33 e 34 é gerado um código único para ser o id da peça. Na linha 35 é buscada a referência da tabela de peças no Firebase e atribuído ao objeto “ref” para, entre as linhas 36 e 49, ser invocado o método “child()” inserindo um novo registro com o id único gerado anteriormente e passando todos os valores das propriedades da peça para serem salvos na base. Alguns valores como “cd_usuariocompra”, “img_peca”, “st_vendida” são informados com valores estáticos, pois no momento da inserção da peça o usuário não os informa e poderá alterá-los na tela de edição da peça.

```
32    salvarPeca(peca: Peca, palavrasChave: string, img: Upload){
33      let cod = Math.round(Math.random() * 1000).toString();
34      cod = cod + Math.round(Math.random() * 1000).toString();
35      let ref = this.db.database.ref("/peca/");
36      ref.child(`/${cod}`).set({
37          "cd_marca":          peca.marca,
38          "cd_usuariocompra": "",
39          "ds_peca":           peca.descricao,
40          "img_peca":          "",
41          "nm_peca":           peca.nome,
42          "uf_peca":           peca.uf,
43          "vl_venda":          peca.valor,
44          "ds_referencia":     peca.referencia,
45          "st_vendida":        "Não",
46          "cd_usuario":        this.afAuth.auth.currentUser.uid,
47          "ds_tag":            peca.palavrasChave,
48          "dt_cadastro":       peca.dt_cadastro
49      });
50      this.salvarImgPeca(img, cod);
51      alert('Peça cadastrada com sucesso!');
52    }
```

**Figura 66 – Código da lógica do serviço de peças**

Na Figura 67 está o método “salvarImgPeca()” para realização do *upload* da imagem da peça no “storage” do Firebase e salvar a url de acesso nos detalhes da peça. Na linha 55 é passada para o objeto “storageRef” a referência do “storage” para que seja possível utilizá-la para acessar os métodos posteriormente. Nas linhas 56 e 57 é realizado a declaração do objeto “uploadTask” e é atribuída a referência de uma tarefa de *upload* utilizando o método “child()” e os parâmetros “this.basePath” para passar o caminho para salva a imagem e o “upload.file.name” para passar o nome do arquivo. Por fim, é chamado o método “put()” passando o parâmetro “upload.file” que apontará o arquivo a ser salvo.

Na linha 58 é acionada a tarefa por meio do método “on()”, caso ocorra algum erro no momento do *upload* da imagem o processo é abortado e uma mensagem é apresentada ao usuário (linha 63). Havendo sucesso no *upload*, a função executará, entre as linhas 68 e 73, a atualização do registro da peça com o *link* da imagem. Nas linhas 68 e 69 são recuperados da “uploadTask” o caminho para *download* da imagem e o nome do arquivo para serem salvos nos dados da peça.

```
54  salvarImgPeca(upload: Upload, idPeca: string){
55    let storageRef = firebase.storage().ref();
56    let uploadTask;
57    uploadTask = storageRef.child(`${this.basePath}/${upload.file.name}`).put(upload.file);
58    uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGED,
59      (snapshot) => {
60        //upload in progress
61      },
62    (error) => {
63      alert('Upload de imagem falhou!');
64      //error handling, upload failed
65    },
66    () => {
67      //upload sucess
68      upload.url = uploadTask.snapshot.downloadURL;
69      upload.name = upload.file.name;
70      let ref = this.db.database.ref(`/peca/${idPeca}`);
71      ref.update({
72        "img_peca": upload.url
73      });
74    });
75  }
```

**Figura 67 – Código da lógica do serviço de peças**

Na Figura 68 está o método “buscaDadosPeca()” na qual é realizada a consulta na base de dados por meio do id da peça que foi passada por parâmetro

para retornar os seus dados. Na linha 105 está a declaração do objeto “dadosPeca” que será populado com os dados, já na linha 106 é atribuído para o objeto “ref” a referência da peça pelo seu id. Entre as linhas 108 e 126 é utilizada o objeto “ref” para acessar o método “once()” trazendo as informações da peça da base de dados e atribuindo os dados para o objeto “dadosPeca” que será retornado após ser populado.

```
104  buscaDadosPeca(idUser, codPeca : string){
105    let dadosPeca = new Peca();
106    let ref = this.db.database.ref(`/peca/${codPeca}`);
107
108    return ref.once("value")
109      .then(function(snapshot){
110        dadosPeca.descricao     = snapshot.child("ds_peca").val();
111        dadosPeca.foto          = snapshot.child("img_peca").val();
112        dadosPeca.marca         = snapshot.child("cd_marca").val();
113        dadosPeca.nome          = snapshot.child("nm_peca").val();
114        dadosPeca.referencia    = snapshot.child("ds_referencia").val();
115        dadosPeca.situacao      = snapshot.child("st_vendida").val();
116        dadosPeca.uf            = snapshot.child("uf_peca").val();
117        dadosPeca.valor         = snapshot.child("vl_venda").val();
118        dadosPeca.usuarioCompra = snapshot.child("cd_usuariocompra").val();
119        dadosPeca.palavrasChave = snapshot.child("ds_tag").val();
120        dadosPeca.id            = snapshot.key;
121        dadosPeca.vl_vendafinal = snapshot.child("vl_vendafinal").val();
122        dadosPeca.dt_venda      = snapshot.child("dt_venda").val();
123        dadosPeca.cd_usuario    = snapshot.child("cd_usuario").val();
124
125        return dadosPeca;
126      });
127  }
```

**Figura 68 – Código da lógica do serviço de peças**

Na Figura 69 está a implementação do método “pesquisaPeca()” que utiliza os parâmetros “textoBusca”, “estado” e “marca” para realizar a busca. Na linha 154 é declarada o atributo “pecasEncontradas” que será a lista na qual serão adicionadas as peças que atendem aos critérios definidos pelos parâmetros passados para a busca. Na linha 156 é atribuída para o atributo “pecas” a listagem com todas as peças da base para que seja realizada a busca. Entre as linhas 157 e 190 é realizado o processo de busca na lista de peças, sendo que na linha 157 é realizada uma subscrição no atributo “pecas” que contém a lista das peças para que seja realizado um *loop* nesta lista (linha 158).

Nas linhas 159 e 160 é realizada a primeira parte da verificação, validando se o estado e a marca da peça são iguais aos foram passados por parâmetro e caso sejam será realizada a segunda parte da validação para verificar se a peça contém

alguma das palavras-chave indicadas na busca. Nas linhas 161 e 162 é atribuído para o atributo “pecaTags” as *tags* que a peça da base contém para serem comparadas ao texto utilizado na busca. Na linha 164 é feito um *loop* pelas *tags* da peça. Na linha 168 é aplicada no objeto “busca” o método “search()” passando o objeto “pecaTag” como parâmetro para ser verificada se ela existe no texto. Caso o resultado seja positivo, os dados dessa peça serão passados (linhas 169 a 182) para o objeto “peca” para que ele seja adicionada à lista de peças encontradas (linha 183). Após todas as peças serem percorridas será realizado o retorno da lista com as peças encontradas (linha 191).

```
153   pesquisaPeca(textoBusca, estado, marca: string){
154     let pecasEncontradas : Peca[] = [];
155     let busca = textoBusca.toUpperCase();
156     let pecas = this.db.list("/peca", { preserveSnapshot: true });
157     let sub = pecas.subscribe(snapshots => {
158       snapshots.forEach(snapshot => {
159         if( (snapshot.child("uf_peca").val() == estado) &&
160             (snapshot.child("cd_marca").val() == marca) ){
161           let pecaTags = snapshot.child("ds_tag").val();
162           pecaTags = pecaTags.split(";");
163
164           for(let i = 0; i < pecaTags.length; i++){
165             let sair = 0;
166             let pecaTag = pecaTags[i].toUpperCase();
167             if(pecaTag != ''){
168               if (busca.search(pecaTag) != -1 ){
169                 let peca = new Peca();
170                 peca.descricao  = snapshot.child("ds_peca").val();
171                 peca.foto       = snapshot.child("img_peca").val();
172                 peca.id         = snapshot.key;
173                 peca.marca      = snapshot.child("cd_marca").val();
174                 peca.nome       = snapshot.child("nm_peca").val();
175                 peca.referencia = snapshot.child("ds_referencia").val();
176                 peca.uf         = snapshot.child("uf_peca").val();
177                 peca.valor      = snapshot.child("vl_venda").val();
178
179                 this.empresaService.buscaDadosEmpresa(snapshot.child("cd_usuario").val())
180                 .then(dados =>{
181                   peca.nomeEmpresa = dados.fantasia;
182                 });
183                 pecasEncontradas.push(peca);
184                 break;
185               }
186             }
187           }
188         }
189       });
190     });
191     return pecasEncontradas;
192   }
```

**Figura 69 – Código da lógica do Serviço de Peças**

Na Figura 70 é listado o método “geraRelatorio()” na qual é passado por parâmetro uma lista de peças para serem apresentadas no relatório. Na linha 243 é

declarado o objeto “doc” que será populado com os dados do relatório. Entre as linhas 245 e 255 é declarada a variável “empresa” e populada com os dados da empresa cadastrada para o usuário logado, que será passado por parâmetro para gerar o cabeçalho do relatório (linha 258). Entre as linhas 259 e 275 é realizado um *loop* na lista de peças que foi passada e cada peça é adicionada ao relatório. Ao finalizar o *loop* entre as peças é utilizada a variável “doc” para chamar o método “save()” passando o nome padrão “Relatorio_de_Pecas_data_hora.pdf” que terá seu *download* realizado automaticamente para a máquina do usuário.

```
242   geraRelatorio(listaPecasEstoque: any){
243     let doc = new jsPDF();
244     let empresa = new EmpModel();
245      this.empresaService.buscaDadosEmpresa(this.afAuth.auth.currentUser.uid)
246         .then(dados =>{
247           empresa.fantasia     = dados.fantasia;
248           empresa.rua          = dados.rua;
249           empresa.bairro       = dados.bairro;
250           empresa.numero       = dados.numero;
251           empresa.cidade       = dados.cidade;
252           empresa.estado       = dados.estado;
253           empresa.CNPJ         = dados.CNPJ;
254           empresa.telefoneFixo = dados.telefoneFixo;
255           empresa.celular      = dados.celular;
256
257           let altura = 51;
258           this.geraCabecalho(empresa, doc, altura);
259           listaPecasEstoque.forEach(peca => {
260             doc.text(11,  altura, `${peca.nome}`);
261             doc.text(76,  altura, `R$ ${Number(peca.valor).toFixed(2)}`);
262             doc.text(106, altura, `${peca.marca}`);
263             doc.text(136, altura, `${peca.uf}`);
264             doc.text(186, altura, `${peca.situacao || 'Não'}`);
265             if((altura+5) < 280){
266               altura = altura + 5;
267             }else{
268               doc.addPage();
269               altura = 51;
270               this.geraCabecalho(empresa, doc, altura);
271             }
272           });
273           let dataHora = this.getDataHora();
274           doc.save(`Relatorio_de_Pecas_${dataHora}.pdf`);
275       });
276   }
```

**Figura 70 – Código da lógica do Serviço de Peças**

Na Figura 71 está o serviço da empresa que possui uma função que utiliza uma requisição JSON para fazer a busca de dados de um CNPJ utilizando o *web*

*service* da Receita Federal. No serviço é importada a classe “Jsonp” do pacote nativo angular/http para que seja possível realizar a declaração do objeto para fazer a requisição.


**Figura 71 – Código da lógica do Serviço de Peças**

Na Figura 72 está a declaração do objeto “jsonp” da classe Jsonp para ter acesso aos métodos. Entre as linhas 22 e 30 é declarado o método “retornaDadosCNPJ()” sendo passado um número de CNPJ para serem buscados os dados. Na linha 23 é utilizado o objeto “jsonp” para ter acesso ao método “get()” fazendo, assim, uma requisição ao *web service* da Receita passando o número de CNPJ como parâmetro na requisição. Sendo feito um mapeamento do resultado a partir da linha 24 e aguardado o resultado. Caso o resultado seja diferente de “null” (linha 25) é retornada o objeto “res” que foi a resposta obtida por meio da requisição e caso ocorra algum erro é retornado “null” (linha 28).


**Figura 72 – Código da lógica do Serviço de Peças**

Na Figura 73 estão listadas as imagens do arquivo que é o módulo no qual é realizado o gerenciamento das rotas do sistema. Na linha 1 está a declaração da classe “NgModule” que é necessária a todos os arquivos declarados como módulos Angular. Já nas linhas 2 e 3 são declarados o “RouterModule” responsável por atribuir as rotas ao sistema e a classe “Routes” utilizada para a declaração do objeto que possuirá as rotas. Entre as linhas 5 e 19 são realizados os *imports* de todos os componentes que possuirão uma rota no sistema.

```
1  import { NgModule      } from '@angular/core';
2  import { RouterModule,
3           Routes        } from '@angular/router';
4
5  import { LoginComponent                  } from '../access/login.component';
6  import { HomeComponent                   } from '../home/home.component';
7  import { PageNotFoundComponent           } from '../common/not-found.component';
8  import { SignUpComponent                 } from '../access/signup.component';
9  import { CadastroEmpresaComponent        } from '../crud/empresa/cadastro-empresa.component';
10 import { EditarEmpresaComponent          } from '../crud/empresa/editar-empresa.component';
11 import { CadastroPecaComponent           } from '../crud/peca/cadastro-peca.component';
12 import { UserInfoComponent               } from '../crud/usuario/user-info.component';
13 import { EditarPecaComponent             } from '../crud/peca/editar-peca.component';
14 import { VisualizaPecaEstoqueComponent   } from '../crud/peca/visualiza-peca-estoque.component';
15 import { VisualizaPecaBuscaComponent     } from '../crud/peca/visualiza-peca-busca.component';
16 import { VisualizaBuscaResultadoComponent } from '../crud/peca/visualiza-busca-resultado.component';
17 import { VisualizaEstoqueComponent       } from '../crud/peca/visualiza-estoque.component';
18 import { FaleConoscoComponent            } from '../contato/fale-conosco.component';
19 import { ChatComponent                   } from '../chat/chat.component';
```

**Figura 73 – Código da lógica do Módulo de Rotas**

Na Figura 74 é criada a constante das rotas “appRoutes” atribuindo a cada rota um caminho e o seu componente correspondente. Algumas rotas como as das linhas 30 a 33 possuem um parâmetro (“:id” ou “:search”) que é enviado junto de sua rota e será utilizado pelo componente para algum processamento. É interessante pontuar que a rota declarada na linha 37 é a rota padrão para caso não seja informado nenhum caminho na url do sistema. E a rota declarada na linha 38 é a que redirecionará qualquer rota que o usuário tente utilizar e que não exista para a página padrão de erro 404 (não encontrada).

```
22 const appRoutes: Routes = [
23   { path: 'home',                  component: HomeComponent                  },
24   { path: 'signup',                component: SignUpComponent                },
25   { path: 'login',                 component: LoginComponent                 },
26   { path: 'cadempresa',            component: CadastroEmpresaComponent       },
27   { path: 'editempresa',           component: EditarEmpresaComponent         },
28   { path: 'userinfo',              component: UserInfoComponent              },
29   { path: 'cadpeca',               component: CadastroPecaComponent          },
30   { path: 'editpeca/:id',          component: EditarPecaComponent            },
31   { path: 'pecaestdet/:id',        component: VisualizaPecaEstoqueComponent  },
32   { path: 'pecabuscadet/:id',      component: VisualizaPecaBuscaComponent    },
33   { path: 'resultadobusca/:search', component: VisualizaBuscaResultadoComponent },
34   { path: 'pecasestoque',          component: VisualizaEstoqueComponent      },
35   { path: 'faleconosco',           component: FaleConoscoComponent           },
36   { path: 'chat',                  component: ChatComponent                  },
37   { path: '', redirectTo: '/login', pathMatch: 'full'                        },
38   { path: '**',                    component: PageNotFoundComponent          }
39 ];
```

**Figura 74 – Código da lógica do Módulo de Rotas**

Na Figura 75 é apresentada a declaração do módulo “AppRoutingModule” utilizando a diretiva “@NgModule” na linha 41, sendo que no seu *array* de *imports*

(linha 42 a 46) é utilizado o “RouterModule” nativo para instanciar as rotas declaradas no objeto “appRoutes” e realizada a exportação (linhas 47 a 50) do “RouterModule”, agora com suas rotas declaradas e prontas para uso no sistema.

```
41  @NgModule({
42    imports: [
43      RouterModule.forRoot(
44        appRoutes,
45      )
46    ],
47    exports: [
48      RouterModule
49    ]
50  })
51  export class AppRoutingModule {}
```

**Figura 75 – Código da lógica do Módulo de Rotas**

Na Figura 76 está o início da apresentação do módulo principal do sistema, sendo importadas todas as classes, componentes e serviços que o sistema possui. A seguir são listados os *imports* de classes nativas do Angular2 e também algumas do *framework* AngularFire2.

```
1   import { BrowserModule              } from '@angular/platform-browser';
2   import { NgModule                   } from '@angular/core';
3   import { FormsModule                } from '@angular/forms';
4   import { AngularFireModule          } from 'angularfire2';
5   import { AngularFireDatabaseModule } from 'angularfire2/database';
6   import { AngularFireAuthModule      } from 'angularfire2/auth';
7   import { environment                } from '../environments/environment';
8   import { RouterModule,
9            Routes                     } from '@angular/router';
10  import { HttpModule                 } from '@angular/http';
11  import { JsonpModule                } from '@angular/http';
```

**Figura 76 – Código da lógica do Módulo de Principal**

Na Figura 77 estão listados todos os *imports* de todos os componentes criados e utilizados pelo para a criação do sistema.

```
import { AppComponent                  } from './app.component';
import { AppRoutingModule              } from './routes/app-routing.module';
import { HomeComponent                 } from './home/home.component';
import { LoginComponent                } from './access/login.component';
import { NavComponent                  } from './common/nav.component';
import { FooterComponent               } from './common/footer.component';
import { PageNotFoundComponent         } from './common/not-found.component';
import { SignUpComponent               } from './access/signup.component';
import { CadastroEmpresaComponent      } from './crud/empresa/cadastro-empresa.component';
import { EditarEmpresaComponent        } from './crud/empresa/editar-empresa.component';
import { UserInfoComponent             } from './crud/usuario/user-info.component';
import { CadastroPecaComponent         } from './crud/peca/cadastro-peca.component';
import { EditarPecaComponent           } from './crud/peca/editar-peca.component';
import { VisualizaPecaEstoqueComponent } from './crud/peca/visualiza-peca-estoque.component';
import { VisualizaPecaBuscaComponent   } from './crud/peca/visualiza-peca-busca.component';
import { VisualizaBuscaResultadoComponent } from './crud/peca/visualiza-busca-resultado.component';
import { VisualizaEstoqueComponent     } from './crud/peca/visualiza-estoque.component';
import { FaleConoscoComponent          } from './contato/fale-conosco.component';
import { ChatComponent                 } from './chat/chat.component';
```

**Figura 77 – Código da lógica do Módulo de Principal**

Na Figura 78 são listados os serviços que foram criados e utilizados para realizar todo o processo de busca e processamento de dados no sistema.

```
import { AuthService      } from './services/authentication.service';
import { EmpresaService   } from './services/empresa.service';
import { PecaService      } from './services/pecas.service';
import { MensagemService  } from './services/mensagem.service';
```

**Figura 78 – Código da lógica do Módulo de Principal**

Na Figura 79 está a declaração do módulo utilizando a diretiva “@NgModule” com todos os componentes do sistema listados dentro do array “declarations” do módulo para que seja possível o seu uso.

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent,
    NavComponent,
    FooterComponent,
    PageNotFoundComponent,
    SignUpComponent,
    CadastroEmpresaComponent,
    UserInfoComponent,
    CadastroPecaComponent,
    EditarPecaComponent,
    EditarEmpresaComponent,
    VisualizaPecaEstoqueComponent,
    VisualizaPecaBuscaComponent,
    VisualizaBuscaResultadoComponent,
    VisualizaEstoqueComponent,
    FaleConoscoComponent,
    ChatComponent
  ],
```

**Figura 79 – Código da lógica do Módulo de Principal**

Na Figura 80 está o *array* de *imports* das classes utilizadas no sistema, sendo que as classes “BrowserModule” e “FormsModule” se fazem necessárias para que seja possível ser exportado o aplicativo para o *browser* e os formulários no sistema possam ser utilizados. Nas linhas 61 a 63 está a inicialização do módulo do Firebase e os *imports* das classes necessárias para ser realizado o acesso à base de dados.

```
59   imports: [
60     BrowserModule,
61     AngularFireModule.initializeApp(environment.firebase),
62     AngularFireAuthModule,
63     AngularFireDatabaseModule,
64     FormsModule,
65     AppRoutingModule,
66     HttpModule,
67     JsonpModule,
68   ],
```

**Figura 80 – Código da lógica do Módulo de Principal**

Na Figura 81 está a declaração do *array* de *providers*, ou seja, as declarações dos serviços utilizados e compartilhados no sistema. Ao serem declarados no “providers” do módulo principal todos os componentes que declararem um objeto de um desses serviços estarão utilizando a mesma referência. Na linha 75 está o *array* de Bootstrap no qual é informado qual será o componente carregado no *browser* inicialmente, sendo informados os AppComponent apresentados.

```
69   providers: [
70     AuthService,
71     EmpresaService,
72     PecaService,
73     MensagemService
74   ],
75   bootstrap: [AppComponent]
76 })
77 export class AppModule { }
```

**Figura 81 – Código da lógica do Módulo de Principal**

Na Figura 82 está o AppComponent que é o componente principal do sistema. É um componente simples, contendo apenas os *imports* dos arquivos de estrutura, estilização e a declaração do seu “selector” para ser utilizado junto do HMTL do sistema.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
}
```

**Figura 82 – Código da estrutura lógica do componente principal**

Na Figura 83 está a parte visual do AppComponent que é bem simples. Ela apenas contém a *tag* “<router-outlet>” que exibirá o conteúdo dos componentes carregados por meio das rotas do “RouterModule”.

```
<router-outlet></router-outlet>
```

**Figura 83 – Código da estrutura HTML do componente principal**

Na Figura 84 observa-se que a *tag* do AppComponent “app-root” será declarada na página index.html do sistema para que sejam carregadas as páginas e apresentadas ao usuário.

```
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

**Figura 84 – Código da estrutura HTML da página index**

# 5 CONCLUSÃO

Em um processo de compra e venda de peças usadas, as peças são, geralmente, adquiridas de sobras de consertos de automóveis originados de acidentes de trânsito. Assim, a proposta de desenvolvimento do sistema apresentado neste trabalho, visa auxiliar na oferta e na localização desse tipo de peça.

Foi realizado um levantamento de informações para verificar os requisitos para o sistema. Tendo como base esses requisitos, foram desenvolvidas as regras de negócio e as funções com as quais o sistema seria composto. Essas regras e funções definidas serviram como roteiro durante do desenvolvimento do projeto e a implementação do sistema.

Além disso, foi realizada a análise durante o levantamento de requisitos sobre o modo em que o uso de um sistema poderia auxiliar na busca de peças usadas e influenciar no valor final de um orçamento de reparo de um automóvel. Aumentando, assim, a possibilidade de o cliente aprovar o orçamento, como também gerar uma renda extra para a mecânica em que esta peça se encontrasse pela possibilidade de anúncio por meio do sistema.

Nesse sentido, acredita-se que o sistema desenvolvido pode contribuir para facilitar o comércio de peças usadas. Reusar peças de automóveis é uma forma de reduzir a quantidade de peças em ferros velhos e lixões. Contribuindo, de certa forma, para a conservação do meio ambiente e na redução do uso de matérias primas para a fabricação de peças.

Será retomado o desenvolvimento do sistema buscando ser feita uma análise no sistema para serem aplicadas as melhorias sugeridas pela banca. E, ainda, desenvolvidas implementações com o intuito de prepará-lo para ser implantado visando capacidade de manipular grandes quantias de dados e usuários no sistema simultaneamente.

## REFERÊNCIAS

ALHIR, Sinan Si. **Understanding the Unified Modeling Language (UML)**. Disponível em: http://www.methodsandtools.com/archive/archive.php?id=76/>. Acesso em: 06. jun. 2017.

AXURE. **Design the right solution**. Disponível em: <https://www.axure.com/> Acesso em: 20 nov. 2017.

BI-FENG, Chen. Technology and application of rich client based on AJAX. **Computer Science**, v.38, n.10A, outubro 2011.

BOOTH, Joseph D. **Angular 2 Succinctly**. 2017, p17.

BOOTSTRAP. **Bootstrap 3 tutorial**. Disponível em: <https://www.w3schools.com/bootstrap/>. Acesso em: 20 nov. 2017.

CASTELEYN, Sven; GARRIGOS, Irene; MAZÓN, Jose-Norberto**. Ten years of Rich Internet Applications: a systematic mapping study, and beyond.** ACM Transactions on the Web, v. 8, n. 3, art. 18, June 2014, p. 18:1-18:46.

CROWTHER, Rob. **Hello! HTML5 & CSS3 – A user-friendly reference guide**. 2013, p4.

DEVMEDIA. **Introdução ao Visual Studio Code**. Disponível em: <http://www.devmedia.com.br/introducao-ao-visual-studio-code/34418/>. Acesso em: 12 jun. 2017.

FERNANDES, Nádia; COSTA, Daniel; NEVES, Sergio; DUARTE, Carlos; CARRIÇO, Luís. **Evaluating the accessibility of Rich Internet Applications.** W4A2012 - Communication, 212, p. 1-4.

FIREBASE. **Comece gratuitamente e pague em escala somente pelo que usar**. Disponível em: < https://firebase.google.com/pricing/?hl=pt-br/>. Acesso em: 06 jun. 2017.

FONT AWESONE. **Font Awesome**. Disponível: <http://fontawesome.io/>. Disponível em: 16 jun. 2017.

GITHUB. **AngularFire2 – The official Angular library for Firebase**. Disponível em: https://github.com/angular/angularfire2/>. Acesso em: 06 jun. 2017.

HOOSHMAND, Salman; MAHMUD, Akib; BOCHMANN, Gregor V.; FAHEEM, Muhammad; JOURDAN, Guy-Vincent. **D-ForenRIA: distributed reconstruction of**

**User-Interactions for Rich Internet Applications. In:** WWW'16 Companion, 2016, p. 211-2014.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Infográficos: frota municipal de veículos**. Disponível em: <http://cidades.ibge.gov.br/painel/frota.php>. Acesso em: 30 jan. 2017.

JEREMY, Allaire. **Macromedia flash MX - A next-generation rich client**. Tech. rep., Macromedia. March. 2002. Disponível em: <http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf>. Acesso em: 04 fev. 2017.

LABRIOLA, Michael; TAPPER, Jeff; BOLES, Matthew. Flex 4 authoritative guide. **People's Posts and Telecommunications**, October, 2011.

LEITE, Joel Silveira. **Brasil tem 41,5 milhões de veículos**. Disponível em: <http://omundoemmovimento.blogosfera.uol.com.br/2015/05/08/brasil-tem-415-milhoes-de-veiculos/>. Acesso em: 30 jan. 2017.

LI-LI, Chen; ZHENG-LONG, Liu. **Design of Rich Client Web Architecture based on HTML5.** 2012 Fourth International Conference on Computational and Information Sciences, 2012, p. 1009-1012.

MARTÍNEZ-RUIZ, Francisco J.; ARTEAGA, Jaime Muñoz; VANDERDONCKT, Jean; GONZÁLEZ-CALLEROS, Juan M. **A first draft of a Modeldriven method for designing Graphical User Interfaces of Rich Internet Applications**. In: 4th Latin American Web Congress LA-Web'2006, IEEE Computer Society Press, 2006, p. 1-5.

MELIÁ, Santiago; GÓMEZ, Jaime; PÉREZ, Sandy; DÍAZ, Oscar. Facing Architectural and technological variability of Rich Internet Applications. **IEEE Internet Computing**, 2010, p. 1-7.

ORGANIZAÇÃO DAS NAÇÕES UNIDAS BRASIL. **OMS: Brasil é o país com maior número de mortes de trânsito por habitante da América do Sul**. Disponível em: <https://nacoesunidas.org/oms-brasil-e-o-pais-com-maior-numero-de-mortes-de-transito-por-habitante-da-america-do-sul/>. Acesso em: 30 jan. 2017.

SINDICATO NACIONAL DA INDÚSTRIA DE COMPONENTES PARA VEÍCULOS AUTOMOTORES. **Relatório do Mercado de Reposição**. 5 ed. Disponível em: <http://www.sindipecas.org.br/sindinews/Economia/2017/RMR_JAN17.pdf/>. Acesso em: 30 jan. 2017.

THINKSTER. **What is RxJS?**. Disponível em: < https://thinkster.io/tutorials/learn-rxjs-observables/what-is-rxjs/>. Acesso em: 06 jun. 2017.