UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARLON EDUARDO BERTAN
THIAGO CAGNIN RODRIGUES

SISTEMA DESKTOP EM DELPHI PARA GERENCIAMENTO DE LOCADORA DE VEÍCULOS VINCULADO A UMA PÁGINA WEB EM PHP

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2012

MARLON EDUARDO BERTAN
THIAGO CAGNIN RODRIGUES

SISTEMA DESKTOP EM DELPHI PARA GERENCIAMENTO DE LOCADORA DE VEÍCULOS VINCULADO A UMA PÁGINA WEB EM PHP

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

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2012

ATA Nº: **201**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DOS ALUNOS **MARLON EDUARDO BERTAN e THIAGO CAGNIN RODRIGUES.**

Às 15:30 hrs do dia 19 de outubro de 2012, Bloco S da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Rúbia E. O. Schultz Ascari (Convidada) e Alisson Andrey Puska (Convidado), para avaliar o Trabalho de Diplomação do aluno Marlon Eduardo Bertan, matrícula 609170 e do aluno Thiago Cagnin Rodrigues, matrícula 610283, sob o título **Sistema Desktop em Delphi para Gerenciamento de Locadora de Veículos Vinculado a uma Página Web em PHP**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação os candidatos foram entrevistados pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:50 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Prof. Alisson Andrey Puska, Esp.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

BERTAN, Marlon Eduardo; RODRIGUES, Thiago Cagnin. Sistema desktop em Delphi para gerenciamento de locadora de veículos vinculado a uma página web em PHP. 91 f. Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2012.

O uso de sistemas computacionais para auxílio na realização de processos de negócio das organizações tem sido cada vez mais evidenciado. Isso ocorre por motivos diversos, incluindo o volume de informações geradas e manipuladas pelas empresas, a agilidade necessária na realização de atividades e mesmo pelo auxílio no gerenciamento e tomada de decisão. Em organizações de grande porte os sistemas de informação podem automatizar as mais diversas atividades da cadeia de suprimentos, como, por exemplo, fazer automaticamente pedido de matéria-prima ou produtos quando a quantidade em estoque atinge determinado valor. Mesmo para empresas de pequeno porte os benefícios do uso de sistemas de informações, geralmente denominados sistemas de informações gerenciais, é uma ferramenta importante ao suporte gerencial. Dados atuais e de histórico podem ser de muita valia no momento de tomar uma decisão. Assim, tendo em vista a relevância que sistemas de informação podem ter, mesmo em empresas de pequeno porte, neste trabalho é apresentado o desenvolvimento de um sistema gerencial para uma locadora de veículos. O sistema de gerenciamento é um sistema *desktop* implementado na linguagem Delphi e uma aplicação *web*, implementada em PHP, possibilita a visualização dos veículos a serem locados e realizar a locação dos mesmos. Tanto a aplicação *desktop* quanto a aplicação *web* utilizam a mesma base de dados.

Palavras-chave: Linguagem Delphi. Sistemas de informações gerenciais. Sistema para locadora de veículos. PHP.

ABSTRACT

BERTAN, Marlon; RODRIGUES, Thiago Cagnin. Delphi desktop system for managing a rent a car linked to a web page developed in PHP. 91 f. Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2012.

The use of computational systems to aid in the implementation of business processes has been increasingly evident. This occurs for several reasons, including the amount of information generated and manipulated by companies, the agility necessary in conducting activities and even aid in the management and decision making. In large organizations information systems can automate various activities of the supply chain, for example, to make request automatically for materials or products where the quantity in stock reaches a certain value. Even for small businesses the benefits of using information systems, commonly called management information systems, is an important tool to support management. Current and historical data can be of great value when making a decision. Thus, considering the importance that information systems can have, even in small businesses, this work presents the development of a management system for a car rental agency. The management system is a desktop system implemented in Delphi language and a web application, implemented in PHP, allows the visualization of vehicles to be leased. The desktop application and the web application use the same database.

Palavras-chave: Delphi language. Information management system. Car rental system. PHP.

LISTA DE FIGURAS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

API	*Application Programming Interface*
CEP	Código de Endereçamento Postal
CNPJ	Cadastro Nacional de Pessoas Jurídicas
CPF	Cadastro de Pessoas Físicas
CSS	*Cascading Style Sheet*
DENATRAN	Departamento Nacional de Trânsito
DETRAN	Departamento de Trânsito
FIPE	Fundação Instituto de Pesquisas Econômicas
GPL	*General Public License*
HTML	*Hypertext Markup Language*
IDE	*Integrated Development Environment*
IMAP	*Internet Message Application Protocol*
IPVA	Imposto sobre a Propriedade de Veículos Automotores
PHP	*PHP HypertextPreprocessor*
POP3	*Post Office Protocol*
RG	Registro Geral
SERASA	Serviços de Assessoria Sociedade Anônima
SIG	Sistemas de Informações Gerenciais
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
UF	Unidade de Federação
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto no qual se insere a proposta do trabalho, os seus objetivos e a justificativa. Por fim está a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Atualmente o volume de informações manipulado pelas organizações é bastante grande. Mesmo empresas de pequeno porte precisam tomar decisões operacionais, gerenciais e estratégicas baseadas em dados da própria empresa. Há que se ressaltar que essas decisões geralmente não são baseadas apenas nos dados da empresa. O mercado e outros fatores também são considerados em um processo de tomada de decisão e mesmo de gerenciamento.

Assim, as informações geradas pelos dados armazenados por sistemas computacionais que possam fornecer suporte ao processo de decisão das empresas são vistas como ferramenta de apoio. Sistemas computacionais no apoio à tomada de decisões, não são mais vistos como um diferencial competitivo é quase que uma questão de sobrevivência para as organizações. Para Bazzotti e Garcia (2012), a era da informação exige das organizações uma estratégia de gestão eficiente que pode ser facilitada pelo uso de recursos oferecidos pela tecnologia de informação e pelos sistemas de informação.

A tecnologia de informação oferece recursos tecnológicos e computacionais para a geração de informações e os sistemas de informação estão cada vez mais sofisticados, propondo mudanças nos processos, estrutura e estratégia de negócios (BAZZOTTI; GARCIA, 2012).

Para auxiliar os gestores nas suas atividades existem os SIGs (Sistemas de Informações Gerenciais). Oliveira (2002) define esse tipo de sistema como processos utilizados para transformar dados em informações que auxiliem no processo decisório da empresa. Um sistema de informação consiste nos componentes que fornecem suporte à tomada de decisão e ao controle dos processos organizacionais. Esses componentes são baseados em dados da organização (OLIVEIRA, 2002).

Considerando o contexto de importância de sistemas de informação para as organizações, mesmo as de pequeno porte, neste trabalho é apresentado o desenvolvimento de

um sistema de informação para uma empresa de locação de veículos. A parte gerencial do sistema, ou seja, cadastros, controle de locação (*check-in* e *check-out*) e de movimentações foi implementada utilizando a linguagem Delphi. E uma página *web* implementada em PHP (*PHP HipertextProcessor)* foi desenvolvida como forma de o cliente visualizar os veículos da empresa e fazer locações.

1.2 OBJETIVOS

O objetivo geral se refere ao resultado principal obtido com a realização do trabalho. E os objetivos específicos o complementam.

1.2.1 Objetivo Geral

Implementar um sistema para gerenciamento de uma locadora de veículos.

1.2.2 Objetivos Específicos

- Oferecer uma forma de controle de *check-in* e *check-out* bem como de movimentações de empresas de locação de veículos. As movimentações se referem às saídas e retorno dos veículos, seja para locação a clientes ou para manutenção e ao controle financeiro da empresa.
- Implementar uma página *web* para que os clientes possam visualizar os veículos a serem locados e fazer locações.

1.3 JUSTIFICATIVA

A proposta de um sistema de informação para empresa de locação de veículos é uma oportunidade de disponibilizar uma ferramenta de apoio aos processos operacionais e gerenciais da empresa.

Desta forma, o trabalho dos funcionários é minimizado, há maior agilidade na realização de atividades e maior segurança e controle dos dados. Isso se deve ao fato de as informações serem cadastradas e poderem ser consultadas. As organizações precisam reagir de modo rápido e eficiente aos problemas e às oportunidades que surgem. Os sistemas de informação podem auxiliar as organizações a expandir os seus limites de atuação, a oferecer novos produtos e serviços e a reorganizar fluxos de tarefas e trabalho (LAUDON; LAUDON, 2004).

No trabalho de estágio do aluno Marlon Eduardo Bertan, um dos autores deste trabalho, foi apresentada a modelagem e a implementação dos cadastros relacionados a veículos e localização. Localização se refere ao endereço de clientes, fornecedores, usuários do sistema e outros. Neste trabalho de conclusão de curso, que tem a participação do aluno Thiago Cagnin Rodrigues, os demais requisitos serão modelados e implementados, incluindo a página *web* que exibirá os veículos e permitirá a locação. Essa página *web* utilizará o mesmo banco de dados, mas será implementada em outra linguagem de programação. O sistema gerencial é implementado utilizando a linguagem Delphi e a página *web* utilizando PHP.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta as considerações iniciais, o objetivo e a justificativa. O Capítulo 2 apresenta o referencial teórico centrado em sistemas de informação com foco gerencial. O Capitulo 3 apresenta os materiais utilizados na modelagem e no desenvolvimento do sistema e o método. Os materiais se referem às tecnologias e ferramentas utilizadas. O método apresenta as principais atividades realizadas para desenvolver o trabalho. No Capítulo 4 está o resultado obtido da realização deste trabalho que é a modelagem e a implementação do sistema. A modelagem é apresentada por meio de diagramas e explicações textuais e a implementação por telas e partes de código do sistema. Por fim, está o Capítulo 5 com a conclusão.

2 SISTEMAS DE INFORMAÇÕES GERENCIAIS

Este capítulo apresenta o referencial teórico do trabalho. Esse referencial está organizado em seções relacionadas que apresentam o contexto dos Sistemas de Informações Gerenciais, conceitos, finalidades e importância desses sistemas.

2.1 CONTEXTO

Atualmente, face à competitividade do mercado e à facilidade de pesquisa por preços de produtos dentre outros fatores, as empresas precisam do suporte de sistemas computacionais. A Internet possibilita acesso facilitado e rápido à busca por parte do consumidor por preços e condições mais vantajosas por parte do consumidor. E o volume de dados que as empresas geram e precisam manipular está cada vez maior. Além de conseguir novos clientes e manter os atuais, as empresas têm se preocupado com a satisfação dos seus clientes. E para isso elas se vêem quase que obrigadas a lançar mão da utilização de ferramentas para gerenciamento das informações disponíveis em suas bases de dados, possibilitando aos executivos e aos gestores tomar decisões com base em informações atuais e fidedignas (EICHSTAEDT; DEGENHARDT, 2012). Uma das ferramentas computacionais que podem ser usadas para essa finalidade são os Sistemas de Informações Gerencias (SIG).

Essas ferramentas, de acordo com Oliveira (2002), permitem aos gestores obter de forma dinâmica e prática as informações necessárias para fundamentar as decisões empresariais. Essas decisões são de escopo operacional, gerencial e estratégico. Elas estão relacionadas às operações realizadas no dia-a-dia que definem o negócio da empresa e mesmo às decisões mais importantes que podem redefinir os rumos do negócio.

2.2 CONCEITOS

Sistemas de Informações Gerencias são definidos como sendo um processo de transformação de dados em informações utilizadas na estrutura decisória da empresa, proporcionando o suporte à administração (OLIVEIRA, 2004). Essas informações visam otimizar os resultados obtidos pela tomada de decisões que se tornam mais adequadas, por

serem fundamentadas em dados reais e dados que efetivamente representam as condições do negócio em relação ao mercado.

Um conceito de SIG mais diretamente relacionado à área de computação é proposto por Stair (1998), para quem sistema de informação é referido como uma série de elementos inter-relacionados que coletam (entrada), manipulam e armazenam (processamento) e fornecem dados e informações (saída). Entrada, processamento e saída compõem a base de qualquer sistema computacional que processa dados. Os SIGs se caracterizam por coletarem dados que forneçam resultados que efetivamente possam ser utilizados como ferramenta de apoio nas decisões da empresa. O processamento é realizado de forma a agregar valor aos dados transformando-os em informações úteis para o momento de negócio da empresa.

No contexto deste trabalho, os sistemas de informações gerenciais tratam conjuntos de dados que são transformados em informações, de forma que essas informações possam ser utilizadas no auxílio ao processo de decisão e de gerenciamento dos negócios.

É importante ressaltar que os dados não podem ser simplesmente armazenados em um repositório. Eles precisam ser disponibilizados de maneira a fornecer suporte efetivo para quem toma decisões. Assim, é importante que a interface do sistema seja de uso fácil e que as informações sejam organizadas visando agregar valor aos dados armazenados, considerando o contexto do negócio.

Rezende e Abreu (2000) explicitam algumas atuações dos sistemas de informação:

a) Ferramentas para auxiliar na realização das tarefas do cotidiano de funcionamento das empresas;

b) Instrumentos que possibilitam avaliação analítica e sintética das empresas;

c) Facilitadores dos processos internos e externos. Externos estão relacionados a fornecedores e parcerias de negócio, podendo abranger a cadeia produtiva;

d) Meios para suportar a qualidade de produtos, processos e serviços, melhoria da produtividade e inovação tecnológica do ponto de vista organizacional;

e) Fornecimento de informações para auxiliar nos processos decisórios empresariais;

f) Fornecimento de informações estratégicas visando lucratividade e competitividade empresarial.

As diversas formas de atuação dos sistemas permitem que as empresas conheçam a si mesmas, no sentido de efetivamente definir as suas capacidades e limitações, e que estejam preparadas para atuar no mercado, que geralmente é bastante competitivo (BAZZOTTI; GARCIA, 2012).

2.3 FINALIDADES DOS SISTEMAS DE INFORMAÇÕES GERENCIAIS

Os sistemas de informação têm por objetivo gerar informações para a tomada de decisões. Isso é realizado por meio de dados que são coletados, processados e transformados em informação (BAZZOTTI; GARCIA, 2012).

Um Sistema de Informações Gerenciais não se restringe ao *software*. Ele inclui pessoas, procedimentos, dados armazenados e fontes de dados aos gerentes e aos tomadores de decisão. As pessoas e seus procedimentos são essenciais para o entendimento de quais informações devem ser fornecidas e assim a origem dos dados seja selecionada. Áreas como *marketing*, produção e finanças recebem suporte dos sistemas de informações gerenciais e podem compartilhar dados armazenados (STAIR; REYNOLDS, 2002). Um sistema de informações gerenciais obtém dados dos diversos segmentos da empresa e mesmo externos. É necessário que a empresa seja entendida e tratada como um meio interligado em seus diversos setores e departamentos.

Os sistemas de informação têm por finalidade capturar e/ou recuperar dados e realizar sua análise em função de um processo de decisão. Pereira e Fonseca (1997) endossam a necessidade de envolvimento da organização como um todo, seja para alimentar o sistema de informação, seja para definir o processamento dos dados, seja para interpretar as informações fornecidas como resultado do processamento. Para esses autores, esse tipo de sistema envolve, de modo geral, o decisor, o contexto, o objetivo da decisão e a estrutura de apresentação das informações. A estrutura de representação se refere à forma de armazenamento dos dados, de processamento e de apresentação ao usuário.

As exigências do mercado - que é competitivo, dinâmico e globalizado - motivam as empresas a possuírem sistemas de informação que sejam eficientes, garantindo níveis mais elevados de produtividade e eficácia (BAZZOTTI; GARCIA, 2012).

Batista (2004) destaca que o objetivo de usar os sistemas de informação é criar um ambiente empresarial em que as informações sejam confiáveis e possam fluir na estrutura organizacional por meio dos seus diversos processos e procedimentos de negócio. Considerando a quantidade de recursos tecnológicos disponíveis e o volume de dados manipulado pelas empresas, o diferencial das empresas tem sido diretamente relacionado à valorização da informação e do conhecimento, proporcionando soluções mais efetivas.

Para serem efetivos, os sistemas de informação precisam corresponder às seguintes expectativas (PEREIRA; FONSECA, 1997):

a) Atender as reais necessidades dos usuários. As informações fornecidas devem atender as expectativas e as necessidades do negócio e dos seus usuários;

b) Estar centrados no usuário e não no profissional que o criou. Assim, esses sistemas precisam fornecer informações resultantes do processamento que sejam facilmente compreendidas pelos usuários;

c) Adaptar-se constantemente às novas tecnologias de informação. O sistema deve ser implementado de forma que possa ser ampliado ou adaptado com facilidade e sem necessidade de extensivos treinamentos aos usuários. A forma que os processos de negócio são realizados, seja manualmente ou por meio de sistema já existente na empresa, deve ser mantida no SIG. Assim, a dificuldade e possível rejeição pelos usuários são minimizadas;

d) Estar alinhado com as estratégias e os procedimentos de negócio da empresa. O sistema de informação deve levar em conta o processo de negócio e a cultura organizacional. O conhecimento existente na organização deve ser considerado.

Se um sistema de informação atender a esses requisitos é provável que as pessoas se sintam mais confiante em utilizá-lo seja para as atividades rotineiras ou como suporte ao processo de tomada de decisão.

2.4 IMPORTÂNCIA DOS SISTEMAS DE INFORMAÇÕES GERENCIAIS

Geralmente há dificuldades para avaliar de forma quantitativa qual o efetivo benefício de um sistema de informações gerenciais, ou seja, a melhoria no processo decisório da organização (MIRANDA, 2012).

Oliveira (2002) destaca os seguintes como os possíveis benefícios que os sistemas de informações gerenciais podem, sob determinadas condições, trazer para as empresas:

a) Redução de custos das operações. Muitas tarefas podem ser realizadas automaticamente pelo sistema de informação. Para outras tarefas o sistema fornece informações que auxiliam na sua realização. Um volume menor de tarefas sendo realizadas por pessoas e tarefas sendo realizadas em menos tempo pode impactar na redução de custos da empresa;

b) Melhoria no acesso às informações que pode ocorrer por meio de relatórios mais completos por terem fontes distintas de dados, mais dados e combinados de maneiras mais efetivas e de acordo com necessidades dos usuários. Os sistemas de informação também

podem propiciar que o usuário elabore os seus próprios relatórios. Esses sistemas podem contar com gerenciadores de relatórios, ainda que eles possam requerer algum conhecimento do usuário em termos de consultas SQL (*Structured Query Language)* ou de tabelas em banco de dados para serem utilizados.

c) Melhoria nos serviços realizados e oferecidos por ter base em dados reais e considerados confiáveis;

d) Melhoria na tomada de decisões, por meio do fornecimento de informações mais rápidas e precisas;

e) Melhoria na estrutura organizacional, por facilitar o fluxo de informações. Tendo os seus processos mapeados no sistema de informação, o fluxo de informações se torna conhecido por todos os usuários do sistema;

f) Melhoria na adaptação da empresa para enfrentar acontecimentos não previstos. Isso pode ser obtido por meio de dados utilizados pelo sistema de informação da empresa e que são provenientes do meio externo ou mesmo por constantes de mudanças identificadas nos fatores ambientais;

g) Melhor interação com os elos da cadeia de suprimentos. Um sistema de informação pode fornecer integração direta com fornecedores, como, por exemplo, a emissão de pedidos quando itens de estoque alcançam determinados níveis;

Essas premissas permitem que as empresas definam possíveis fortalecimentos do processo de gestão, garantindo o diferencial de atuação e por consequência, vantagem competitiva.

Um sistema de informações gerenciais resulta em vantagem competitiva para a empresa, pois o mesmo deve ser desenvolvido de forma a dar apoio às metas da organização (STAIR, 1998). Por exemplo, os executivos de nível estratégico usam relatórios dos SIGs no desenvolvimento de estratégias para o sucesso dos negócios; os gestores de nível gerencial usam os relatórios para comparar as metas estabelecidas para a empresa com os resultados reais obtidos; e os gerentes de nível tático (executivo) utilizam relatórios para acompanhar se as equipes estão realizando suas atividades de forma a atender os objetivos propostos. Dessa forma, a empresa justifica o cumprimento de suas metas com a ajuda dos sistemas de informações gerenciais (BAZZOTTI; GARCIA, 2012).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados na realização deste trabalho. Materiais são as ferramentas, as tecnologias, os ambientes de desenvolvimento e outros que são utilizados para realizar as atividades desde a definição dos requisitos à implantação do sistema. O método define as principais atividades realizadas.

3.1 MATERIAIS

As ferramentas e as tecnologias utilizadas para as atividades de modelagem, implementação e execução do sistema são:

a) Astah Community - como ferramenta para elaboração dos diagramas de casos de uso do sistema.

b) Microsoft SQL Server Management 2012 - como ferramenta para gerenciamento do banco de dados e definição do diagrama de entidades e relacionamentos para o banco de dados.

c) Delphi Rad XE2 - como IDE (*Integrated Development Environment*) de desenvolvimento do sistema, parte *desktop*, com Borland Delphi versão 7 como linguagem de programação.

d) Ferramenta NetBeans - a IDE para implementação da página *web*.

e) Ferramenta Notepad++ - a IDE para implementação da página *web*. Esse editor foi utilizado para ajustes mais simples nas páginas *web* sendo implementadas. Isso porque, nesse editor a página *web* é aberta como um arquivo de texto e o não há carregamento de componentes e outros elementos como ocorre com o NetBens. Assim, os arquivos são abertos de forma mais rápida que no NetBeans.

f) Linguagem PHP versão 5.3.1 - para a programação *back-end* da página *web*, para programação do servidor do site.

g) CSS2 - para definir os estilos e alguns efeitos visuais das páginas *web*.

h) Linguagem de marcação HTML (*Hypertext Markup Language*) versão 4 - para a composição da página *web*.

i) JavaScript e bibliotecas JavaScript - para a programação de alguns efeitos visuais da página *web* e comunicação entre cliente e servidor via Ajax. As bibliotecas JavaScript utilizadas foram: jQuery JavaScript Library v. 1.6.1, jQuery Nivo Slider v2.3, jQuery validation plug-in pre-1.5.2, jQuery Easing v1.3, Masked Input plugin for jQuery v. 1.2.2, Javascript Mouse-Wheel v. 3.0.4, FancyBox - jQuery Plugin v. 1.3.4.

j) Adobe Photoshop CS5 Extended - para criação do leiaute da página *web*, a edição das imagens utilizadas no site e a criação da logomarca.

k) XAMPP v1.7.3 - como servidor *web* Apache e intérprete de *scripts* para a linguagem PHP.

3.1.1 Astah Community

Astah Community (ASTAH, 2012) é uma ferramenta de modelagem gratuita para projeto de sistemas orientados a objetos. Essa ferramenta é baseada nos diagramas e na notação da UML 2 (*Unified Modeling Language*) e permite gerar código em Java. A Figura 1 apresenta a interface principal dessa ferramenta de modelagem.

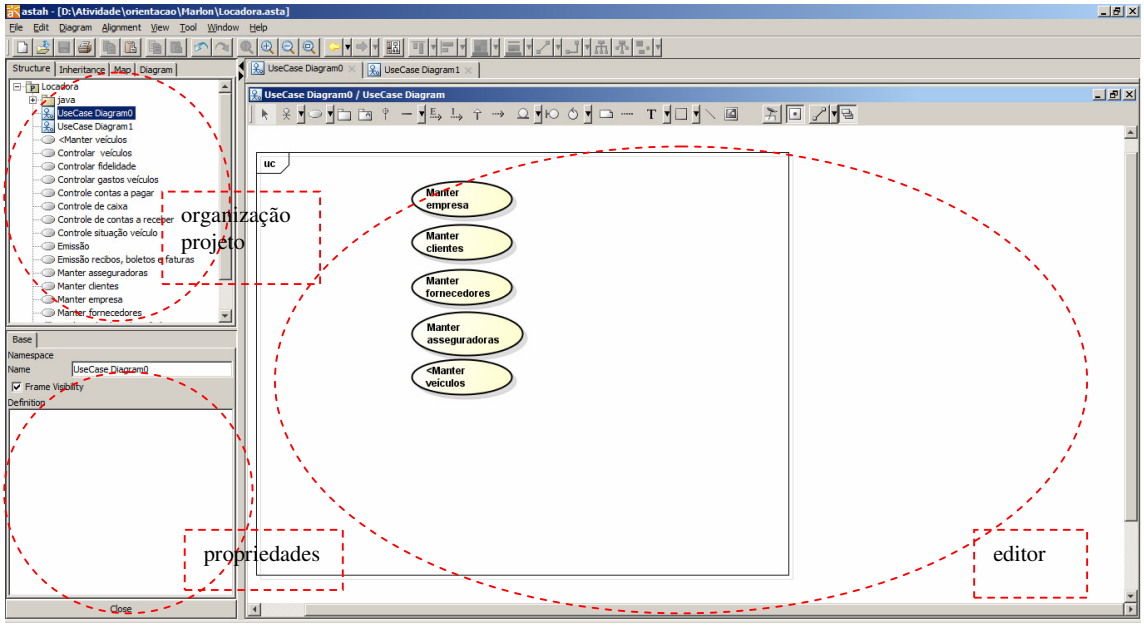

Figura 1 – Ferramenta de modelagem Astah Community

A ferramenta Astah Community está dividida em três partes principais, como pode ser visualizado pelas regiões circuladas na Figura 1:

a) Organização do projeto (área superior à esquerda) - é a área na qual estão as diferentes visões do projeto.

b) Visão das propriedades (área inferior à esquerda) - é a área na qual podem ser alteradas as propriedades dos elementos dos diagramas.

c) Editor do diagrama (área à direita) - é a área na qual são exibidos e compostos os diagramas.

3.1.2 Microsoft SQL Server Management

SQL Server Management Studio é um ambiente integrado para acessar, configurar, gerenciar, administrar e desenvolver todos os componentes do SQL Server (MICROSOFT, 2012). Esse ambiente combina um conjunto de ferramentas gráficas com editores de *script* para prover acesso em diversos níveis aos desenvolvedores e administradores de SQL Server.

A Figura 2 apresenta a tela da ferramenta Microsoft SQL Server Management com uma visão do ambiente gráfico de editoração.

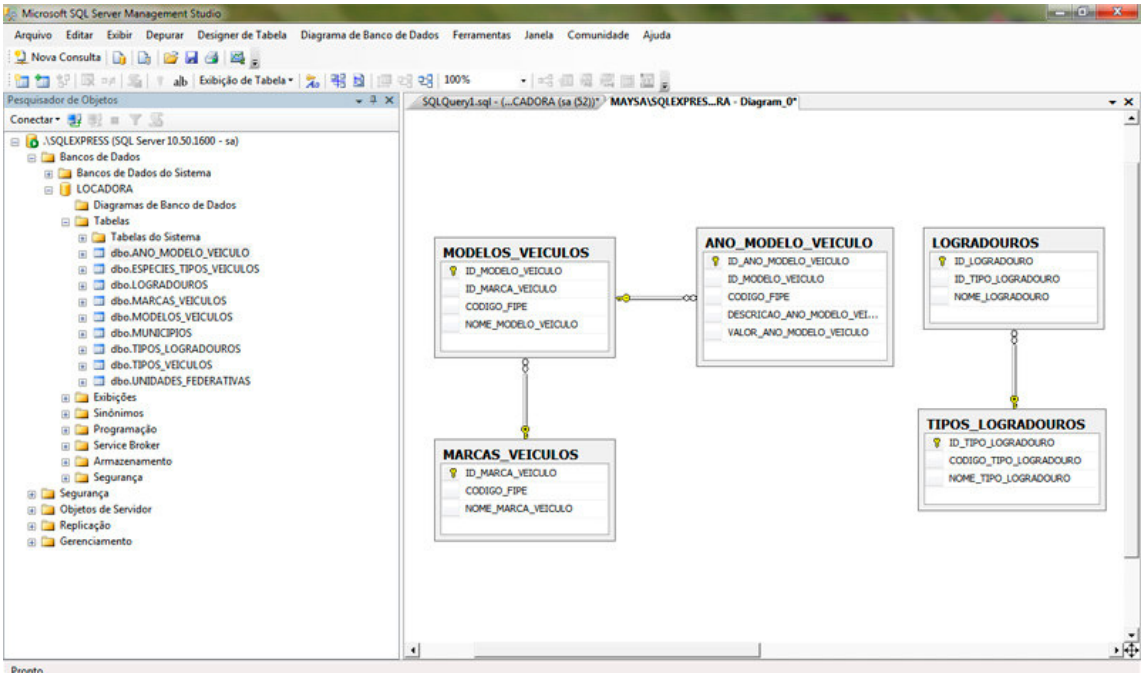

Figura 2 – Ferramenta SQL Server – composição gráfica de banco de dados

As tabelas e respectivos campos que são definidos de forma gráfica podem ser visualizados e editados em termos de atributos e propriedades. A área superior à direita da Figura 3 apresenta os campos da tabela selecionada e na parte inferior da área a direita dessa figura são apresentadas as propriedades da coluna selecionada. Uma coluna é um “campo” da tabela.

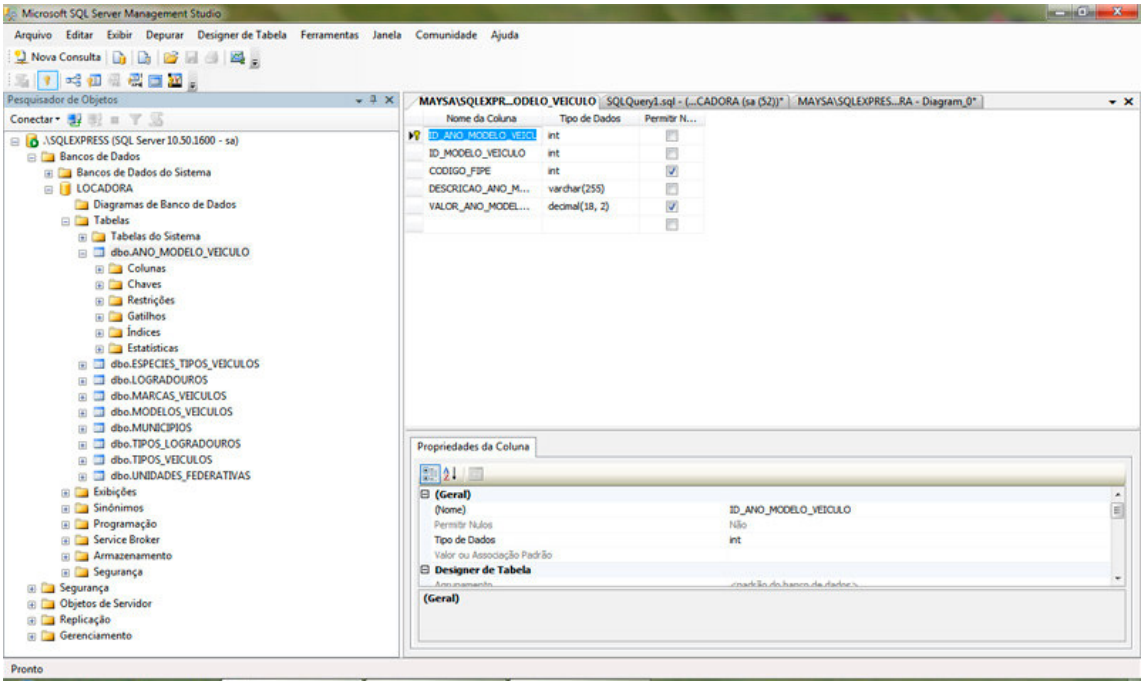

Figura 3 – Ferramenta SQL Server – propriedades de campos de tabelas

A Figura 4 apresenta o *script* que a ferramenta gera da tabela composta. Esse script é utilizado pela linguagem de programação, no caso deste trabalho é a linguagem Delphi.

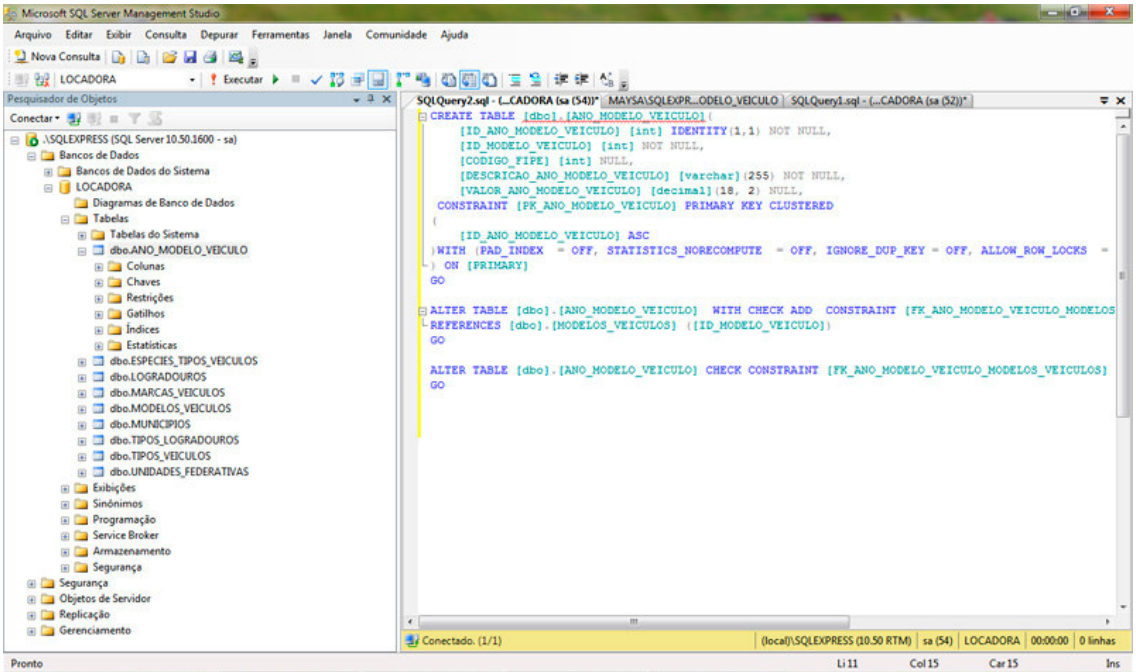

Figura 4 – Ferramenta SQL Server – geração de script das tabelas

3.1.3 Delphi

A linguagem Delphi, que está vinculada a um ambiente de desenvolvimento, se baseia em uma extensão orientada a objetos da linguagem de programação Pascal, também conhecida como Object Pascal (CANTU, 2003). Esse ambiente de desenvolvimento é composto de várias ferramentas agregadas.

Como ambiente de desenvolvimento, Delphi é um compilador e também uma IDE para o desenvolvimento de *software*. A linguagem utilizada é *Object Pascal* (Pascal com extensões orientadas a objetos) que a partir da versão 7.0 passou a se chamar Delphi Language. A Figura 5 apresenta a tela de interface do ambiente de desenvolvimento da linguagem Delphi que é na versão Rad XE2.

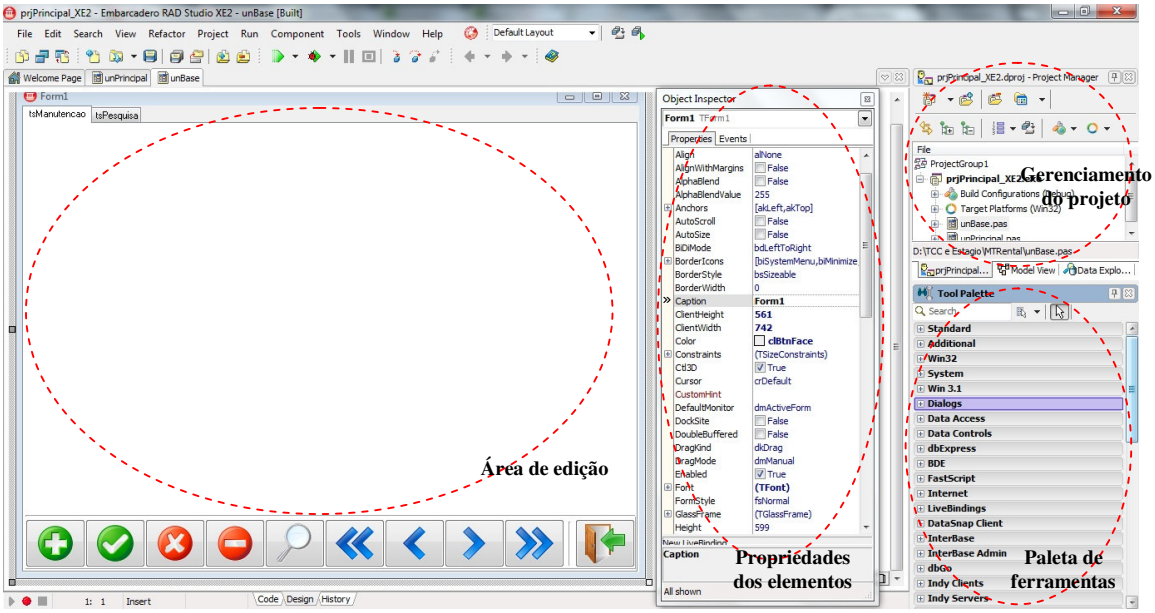

Figura 5 – Ambiente de desenvolvimento da Linguagem Delphi

As regiões marcadas na Figura 5 indicam as principais partes da IDE de desenvolvimento da Linguagem Delphi, em sua versão Embarcadero RAD Studio XE2. Essa IDE oferece uma área de edição na qual são compostos os formulários a partir dos componentes da paleta de ferramentas (área inferior à direita) e também para escrita do código. Em “Propriedades dos elementos” são apresentadas as propriedades do elemento selecionado na área de edição. E em “Gerenciamento do projeto” os arquivos que compõem o projeto são organizados sob a forma de uma árvore de diretórios. Isso facilita a localização para edição dos componentes, por exemplo.

3.1.4 NetBeans

A IDE NetBeans (NETBEANS, 2012) é um ambiente de desenvolvimento que reúne diversas funcionalidades necessárias para implementar um sistema. Essa IDE possui um grande conjunto de bibliotecas, módulos e APIs (*Application Programming Interface*) que compõem um conjunto de rotinas, protocolos e ferramentas para a construção de *software*). Com a IDE NetBeans, é possível desenvolver aplicativos para as plataformas *desktop*, *web* e *mobile*. A Figura 6 apresenta a tela principal da IDE NetBeans.

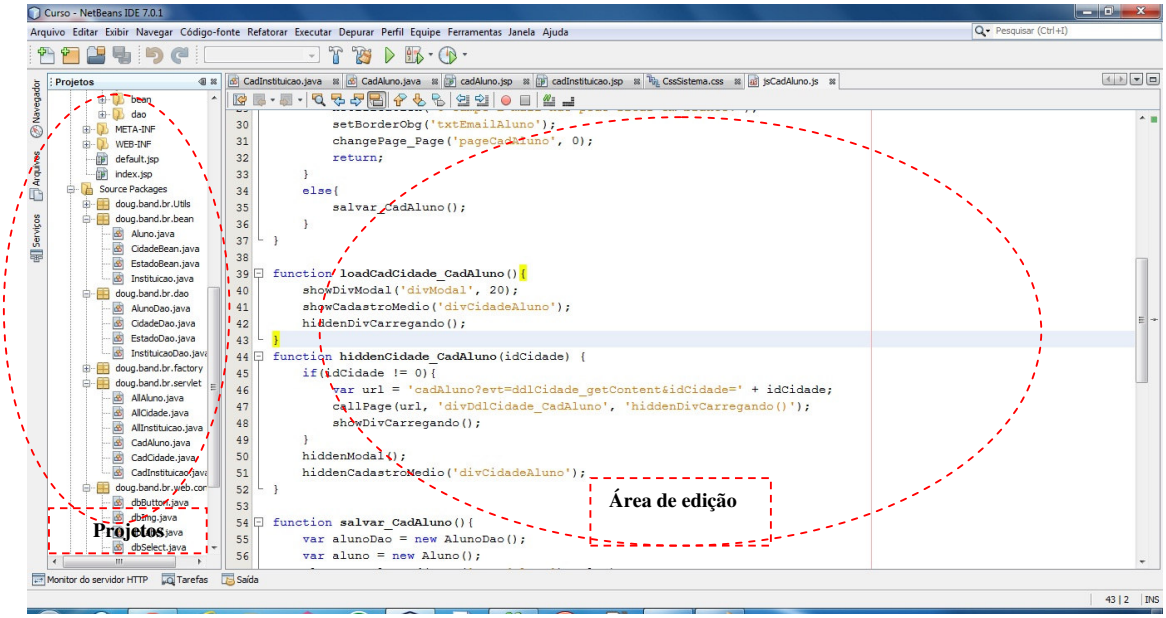

Figura 6 – Tela principal da IDE NetBeans

As partes destacadas na Figura 6 indicam:

a) Projetos – é a área na qual são organizados os arquivos de código definidos durante a implementação do projeto.

b) Área de edição – nesta área o código é escrito e editado. Além da escrita de código, nessa área a interface do sistema é composta por meio de componentes visuais vinculados à ferramenta e disponibilizados em suas paletas de componentes.

3.1.5 Notepad++

A ferramenta Notepad++ é um editor de código fonte livre que suporta várias linguagens (NOTEPAD, 2012). Essa ferramenta executa no ambiente Microsoft Windows e é

baseada na licença GPL (*General Public License*). A Figura 7 apresenta a tela principal do Notepad++.

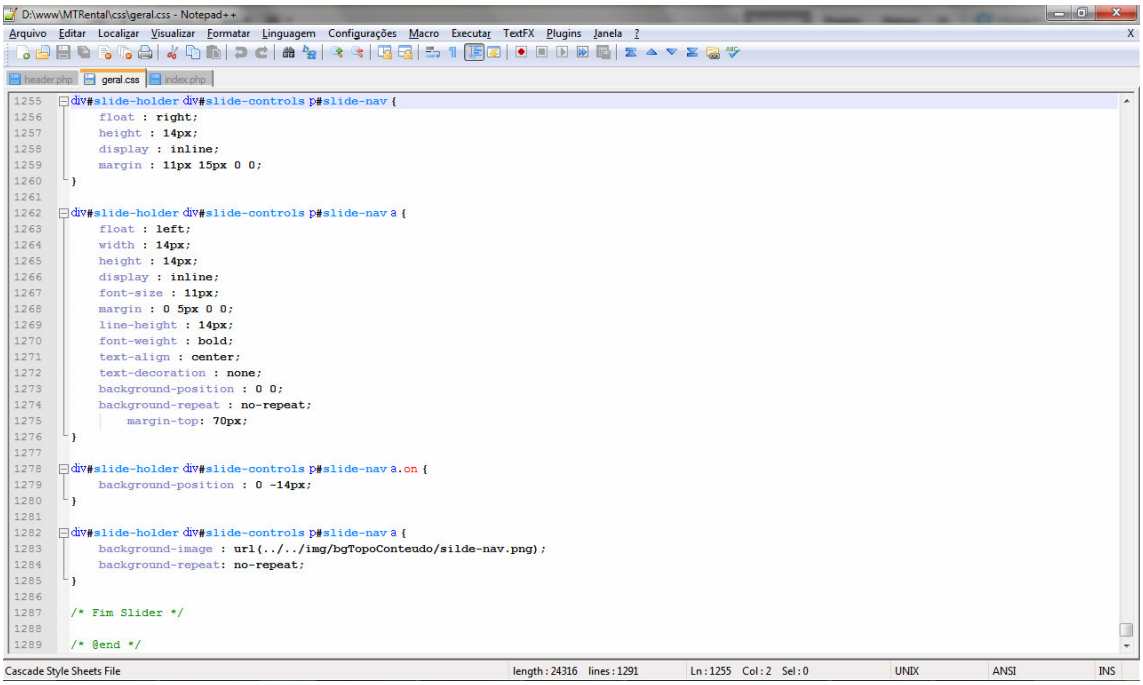

Figura 7 – Tela principal da ferramenta NotePad++

Como pode ser observado pela Figura 7 essa ferramenta de edição é bastante simples. Com menus, barras de ferramentas e uma área de edição.

3.1.6 Linguagem PHP

PHP é uma linguagem de programação para *web* (PHP, 2012) que é processada no servidor, retornando para o cliente somente HTML. A linguagem PHP tem suporte para diversos bancos de dados existentes. Isso facilita a integração com aplicações que necessitem de banco de dados. Essa linguagem também suporta protocolos como SMTP (*Simple Mail Transfer Protocol)*, POP3 (*Post Office Protocol)* e IMAP (*Internet Message Application Protocol*).

3.1.7 HTML

HTML é uma linguagem de marcação para a criação de páginas *web*. Essa linguagem provê um meio para descrever a estrutura de informação baseada em texto em um documento.

Essa estrutura define a forma como o texto é apresentado. Isso é feito pela definição de tipos de formatação de texto para cabeçalho, parágrafos, listas e outros; e pela inclusão de imagens e outros objetos. Mikkonen e Taivalsaari (2008) consideram HTML como um dos componentes fundamentais das páginas *web*.

3.1.8 Cascading Style Sheet

Cascading Style Sheet (CSS) é uma linguagem de estilo que é usada para descrever aspectos de apresentação de um documento escrito em uma linguagem de marcação, como a HTML. Um estilo permite estabelecer regras de formatação de uma página *web*, como, por exemplo, cores e fontes, que são definidos independentemente do conteúdo da página *web* e podem ser aplicados em páginas distintas. A alteração de formatação das diversas páginas de uma mesma aplicação ou site é facilitada porque é necessário alterar apenas o arquivo CSS (que pode ser mais de um) responsável pela formatação. Para Mikkonen e Taivalsaari (2008) CSS também é um dos componentes fundamentais das páginas *web*.

3.1.9 JavaScript

JavaScript possibilita a execução de código em um documento *web*. JavaScript é uma linguagem de *script*, caracterizada por suas instruções serem interpretadas e com execução vinculada a outras linguagens. As linguagens de *script* como JavaScript são um dos componentes fundamentais das páginas *web* (MIKKONEN; TAIVALSAARI, 2008).

3.1.10 Adobe Photoshop

A ferramenta Adobe Photoshop é um programa de editoração gráfica desenvolvido e distribuído pela Adobe Systems (ADOBE, 2012).

O Photoshop foi criado para edição de imagens para impressão em papel, mas está sendo cada vez mais usado também para produzir imagens destinadas à *web* (ADOBE, 2012). O Adobe ImageReady é utilizado em conjunto com o Photoshop para a edição e criação de imagens e animações para a *web*.

3.1.11 XAMPP

XAMPP é uma distribuição do Apache contendo MySQL, PHP e Perl (XAMPP, 2012). O nome provem da abreviação de X (para diferentes sistemas operacionais, como Linux Windows, Solaris e Mac), Apache, MySQL, PHP, Perl (APACHE, 2012). A distribuição para Windows 2000, 2003, XP, Vista e 7 contém: Apache, MySQL, PHP + PEAR, Perl, mod_php, mod_perl, mod_ssl, OpenSSL, phpMyAdmin, Webalizer, Mercury Mail Transport System for Win32 and NetWare Systems v3.32, Ming, FileZilla FTP Server, mcrypt, eAccelerator, SQLite e WEB-DAV + mod_auth_mysql.

A Figura 8 apresenta a tela do painel de controle do XAMPP.

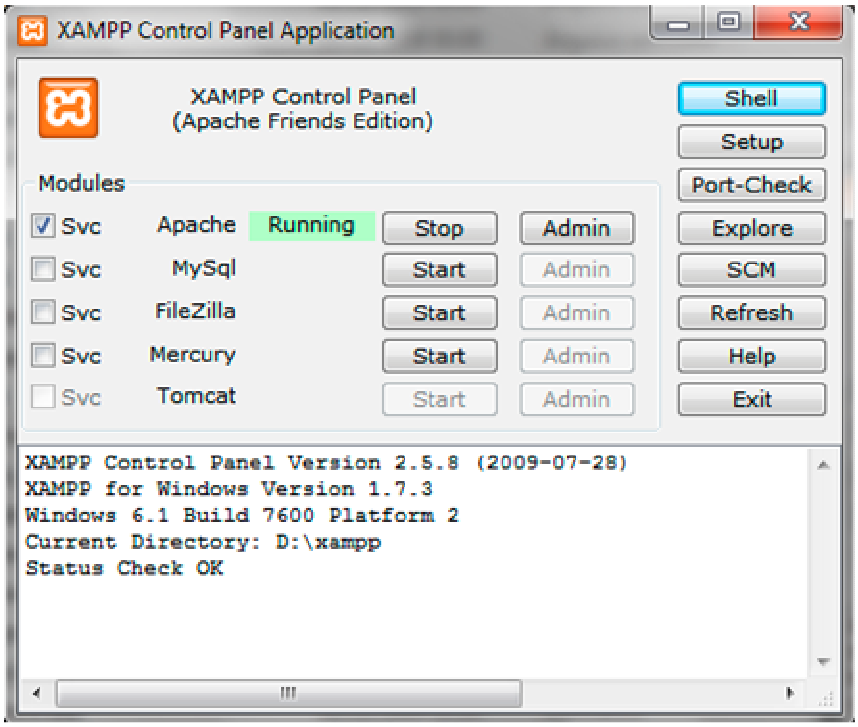

Figura 8 – Tela do painel de controle do XAMPP

3.2 MÉTODO

Os procedimentos para definir o sistema estão baseados nas fases propostas por Pressman (2002) para o modelo sequencial linear que são análise, projeto, codificação e testes. A seguir as fases definidas para este trabalho:

a) Planejamento

Inicialmente foram analisados os requisitos definidos para o sistema como trabalho de estágio do aluno Marlon Eduardo Bertan. Em seguida os requisitos foram complementados e foi estabelecido o que cada um dos alunos implementaria. Estabeleceu-se que o desenvolvimento seria feito como um trabalho coordenado, mas cada aluno com atribuições específicas. Ficou definido que aluno Marlon estaria centrado na aplicação *desktop* embora a definição dos requisitos, a revisão da modelagem e a definição do banco de dados tenham sido definidos em conjunto pelos dois alunos. O aluno Thiago estaria centrado na implementação da página *web* e na definição de procedimentos que agilizariam as operações com o banco de dados.

b) Revisão dos requisitos

O levantamento de requisitos foi realizado com base no conhecimento de um dos autores deste trabalho, o aluno Marlon, que já trabalhou em uma locadora de veículos. Com base no conhecimento dos processos, procedimentos e necessidades percebidos e identificados na empresa foram definidos os requisitos.

A revisão dos requisitos foi realizada em reuniões com a orientadora em que alguns aspectos de negócio foram definidos, especialmente os relacionados ao *check-in* e *check-out* de veículos.

c) Análise e projeto do sistema

A análise e o projeto foram realizados por meio da definição dos casos de uso utilizando a ferramenta Astah Community e a modelagem do banco de dados foi realizada com a ferramenta SQL Server Management Studio.

Como trabalho de estágio do aluno Marlon apenas os requisitos relacionados ao veículo e localização (endereço) foram modelados. Assim, a modelagem do sistema foi complementada e requisitos foram reavaliados.

Em relação aos requisitos da página *web* foi definido que a página exibiria imagens dos veículos disponíveis para locação e permitiria aos clientes realizar reserva de veículos.

d) Implementação (codificação) do sistema

A implementação do sistema para a "parte" *desktop* foi realizada por meio da linguagem Delphi utilizando o ambiente de desenvolvimento da própria linguagem.

A página *web* que apresenta os veículos da locadora e permite o cliente realizar locação foi implementada utilizando a linguagem PHP e como IDE foram utilizados NetBeans e Notepad++.

e) Realização dos testes

Os testes realizados foram basicamente de codificação e para verificar o atendimento das funcionalidades. Foram testes informais e realizados pelo próprio autor deste trabalho.

4 RESULTADO

Este capítulo apresenta o que foi obtido como resultado do trabalho: o sistema desenvolvido.

4.1 APRESENTAÇÃO DO SISTEMA

O sistema de gestão de locadoras de veículos permitirá o controle de usuários através de níveis de acesso, o cadastro de clientes, de fornecedores, asseguradoras com sistema de bloqueio ou inativação do mesmo, cadastro de veículos com seus tipos e modelos, controle dos gastos e custos de cada veículo. O sistema permitirá que proprietário tenha acesso a informações gerenciais para, por exemplo, verificar se os veículos de sua frota estão trazendo lucro ou não, por exemplo.

Dentre outras funcionalidades do sistema estão: o controle de *check-in* e *check-out*, no momento da locação e devolução do veículo e da situação dos veículos em relação a veículos locados, não locados ou reservados. Além de um sistema de reservas *online*, por meio de site próprio e opção de envio de e-mail para clientes avisando em quanto tempo sua locação terminará, ou caso haja alguma promoção. O sistema também proverá integração a sites como FIPE (Fundação Instituto de Pesquisas Econômicas), DETRAN (Departamento de Trânsito) e SERASA (Serviços de Assessoria Sociedade Anônima, também conhecida como Centralizadora dos Serviços dos Bancos SA) controlando multas, sinistros, licenciamento, seguro e IPVA (Imposto sobre a Propriedade de Veículos Automotores).

O sistema de reservas *online* permitirá ao cliente visualizar os veículos que a empresa possui e realizar a locação dos mesmos por meio de uma página *web*.

4.2 MODELAGEM DO SISTEMA

A Figura 9 apresenta o diagrama de casos de uso contendo as funcionalidades do sistema.

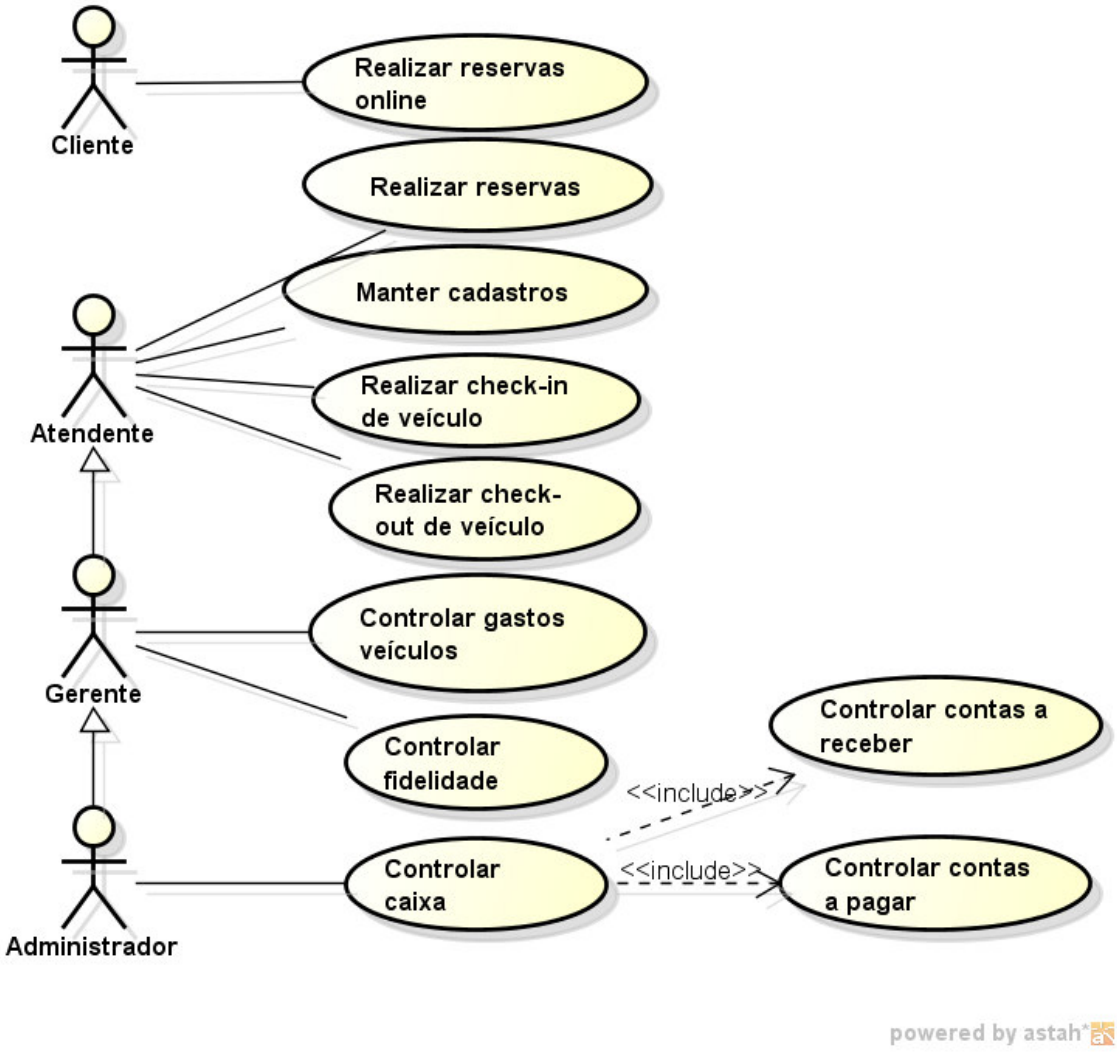

Figura 9 – Diagrama de casos de uso

As Figuras 10 a 13 apresentam os diagramas de banco de dados com as entidades e os respectivos relacionamentos das tabelas do sistema. A modelagem do banco foi dividida em módulos para facilitar a visualização. Esses módulos indicam apenas agrupamentos de funcionalidades de forma a facilitar a visualização das tabelas.

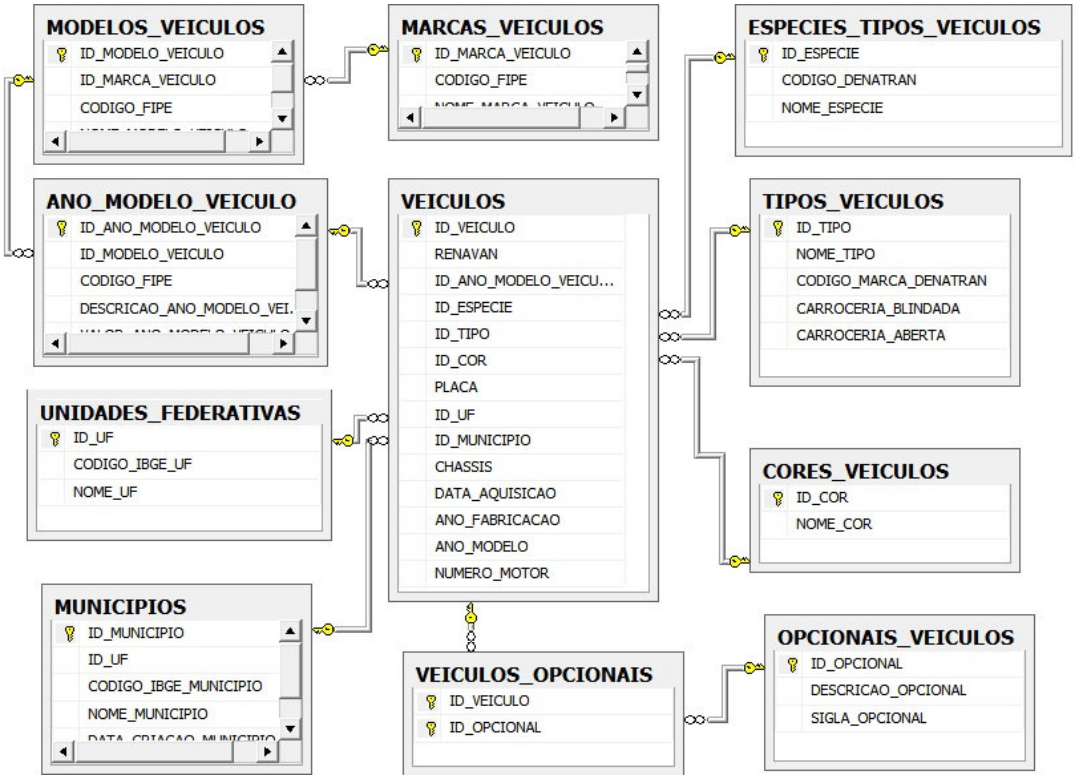

Figura 10 – Diagrama de entidades e relacionamentos "veículos"

De acordo com o representado na Figura 10, a tabela veículos está relacionada com espécies, tipos, cores, ano do modelo e opcionais do veículo. A tabela ano de modelos de veículos está relacionada com a tabela que contém os modelos de veículos e esta, por sua vez, com a tabela de marcas.

Na Figura 11 estão as tabelas relacionadas ao endereço utilizado nos cadastros.

Figura 11 – Diagrama de entidades e relacionamentos "Localização"

As tabelas relacionadas ao cadastro de clientes estão apresentadas na Figura 12.

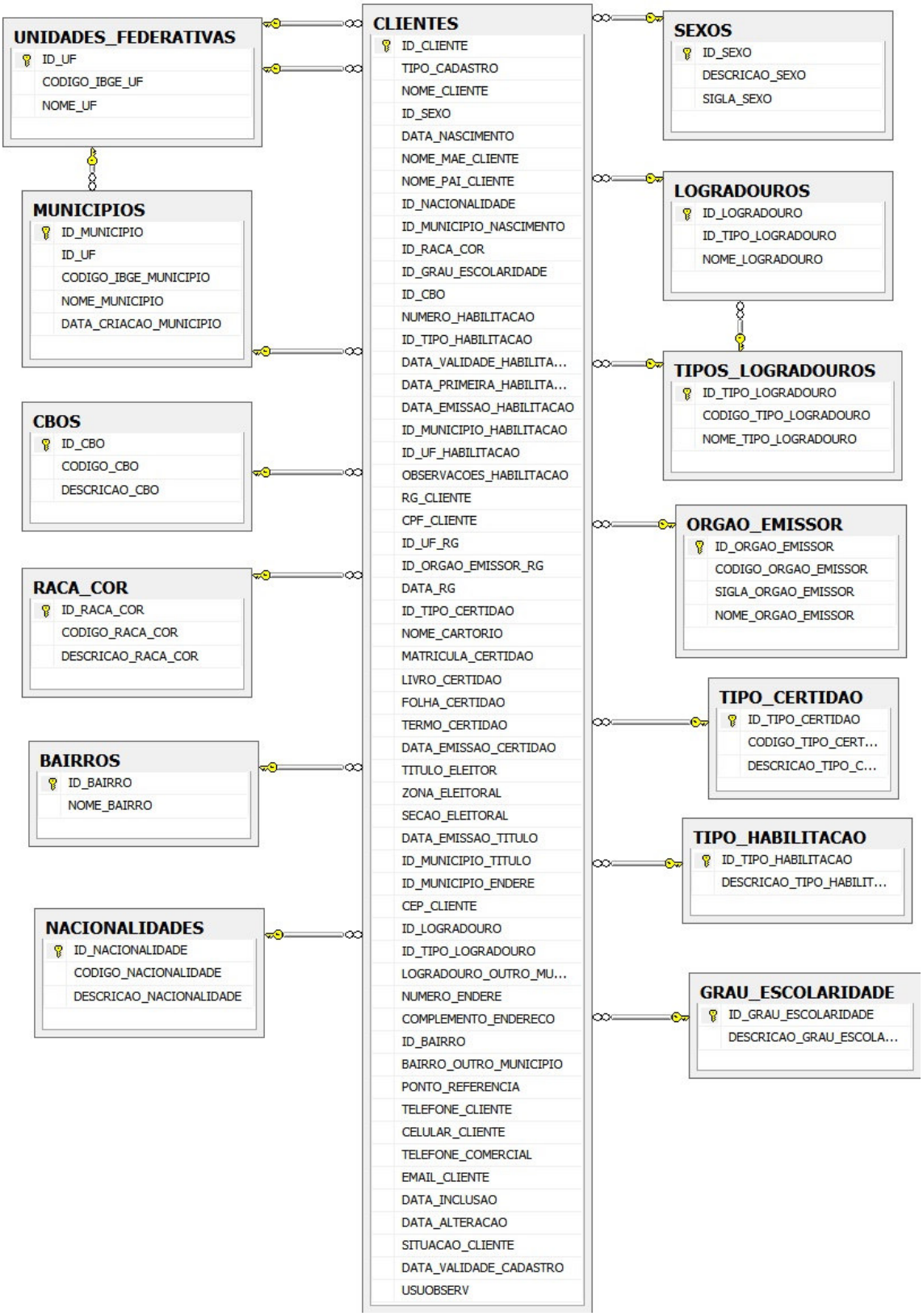

Figura 12 – Diagrama de entidades e relacionamentos "Clientes"

As tabelas relacionadas às movimentações estão apresentadas na Figura 13.

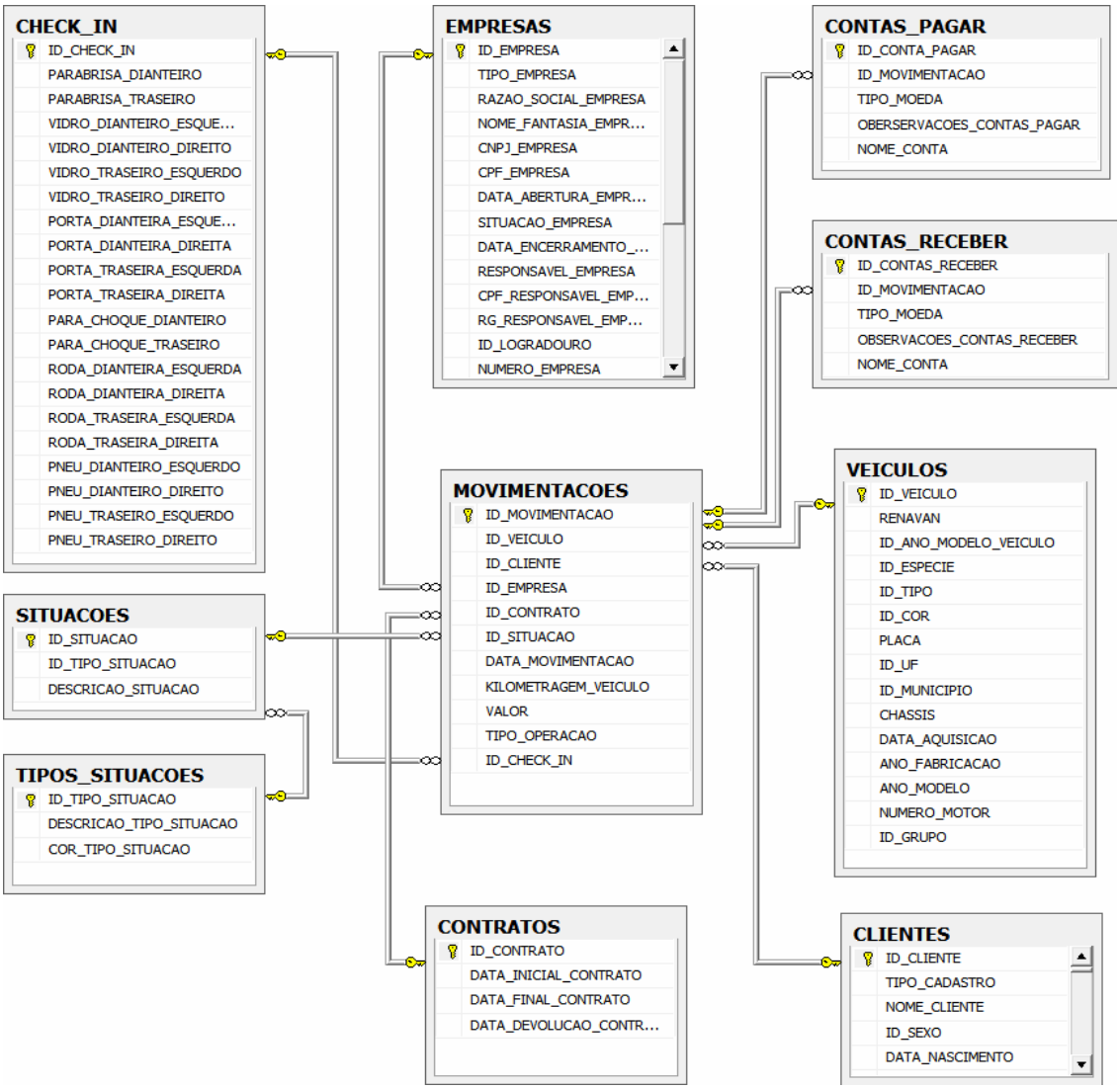

Figura 13 – Diagrama de entidades e relacionamentos "Movimentações"

Na Figura 13, percebe-se que na tabela "Movimentações" relaciona-se uma Conta a Pagar ou uma Conta a Receber. Todos os outros vínculos são gerados pelas tabelas filhas da movimentação. O que define o tipo de movimentação (entrada ou saída) é o campo "Tipo_Operacao". Já o que define em que situação o veículo ficará é o campo "Id_Situacao" que vem da tabela filha "Situacoes". Nessa tabela, o usuário do sistema poderá cadastrar qualquer situação que o veículo poderá se encontrar, mas o campo "Id_Tipo_Situacao" terá seus registros fixos que serão: Disponível, Reservado, Locado, Manutenção ou Baixado. De acordo com o tipo de movimentação, poderá haver vinculo com a tabela "Check_In" que se refere à verificação do estado do veículo no momento em que o mesmo está sendo locado ou devolvido.

4.3 DESCRIÇÃO DO SISTEMA

A tela apresentada na Figura 14 apresenta a tela inicial do sistema. Como trabalho de estágio foram implementados apenas as funcionalidades relacionadas a veículos e localização. Essas funcionalidades, de certa forma, definem quatro módulos do sistema: Geral que inclui Localização (definido como Geral no menu apresentado na Figura 14), Veículos, Movimentações e Financeiro. A Figura 14 apresenta a tela inicial do sistema, sendo que o menu geral permite acesso aos cadastros relacionado à localização.

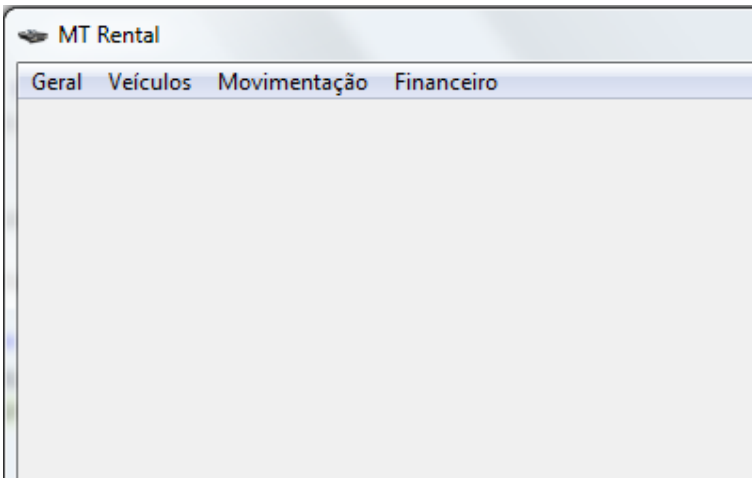

Figura 14 – Tela inicial do sistema

Para facilitar a apresentação, a descrição do sistema foi dividida em aplicação *deskotp* e aplicação *web*. As funcionalidades da aplicação *desktop* foram organizadas em módulos. Esses módulos representam os menus da tela do sistema. A denominação "módulo" foi utilizada apenas como forma de agrupamento das telas e funcionalidades.

4.3.1 Aplicação Desktop - Módulo Localização

O módulo Localização se refere aos cadastros relacionados a endereço. A Figura 15 apresenta os itens do menu Geral correspondente aos cadastros.

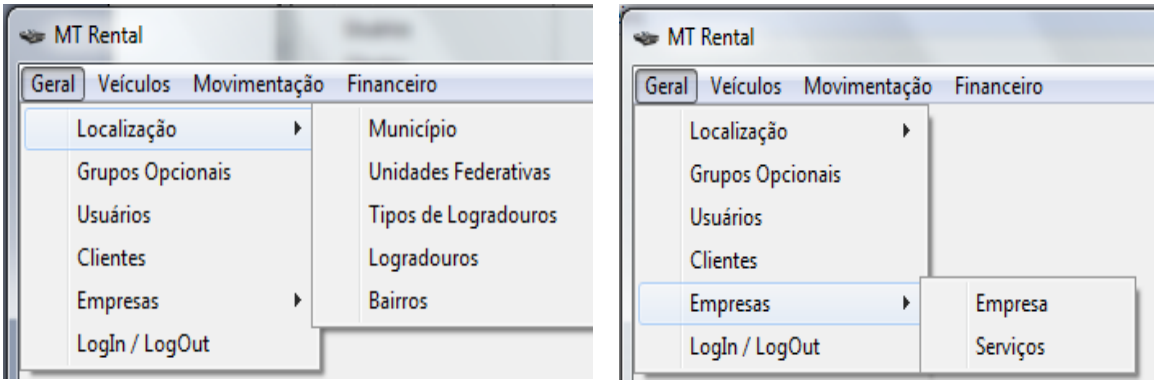

Figura 15 – Menu geral e seus itens de menu

A tela de cadastro de Unidades Federativas (representada na Figura 16) compreende os campos do código interno do sistema, código do IBGE da Unidade Federativa e o nome da respectiva Unidade Federativa.

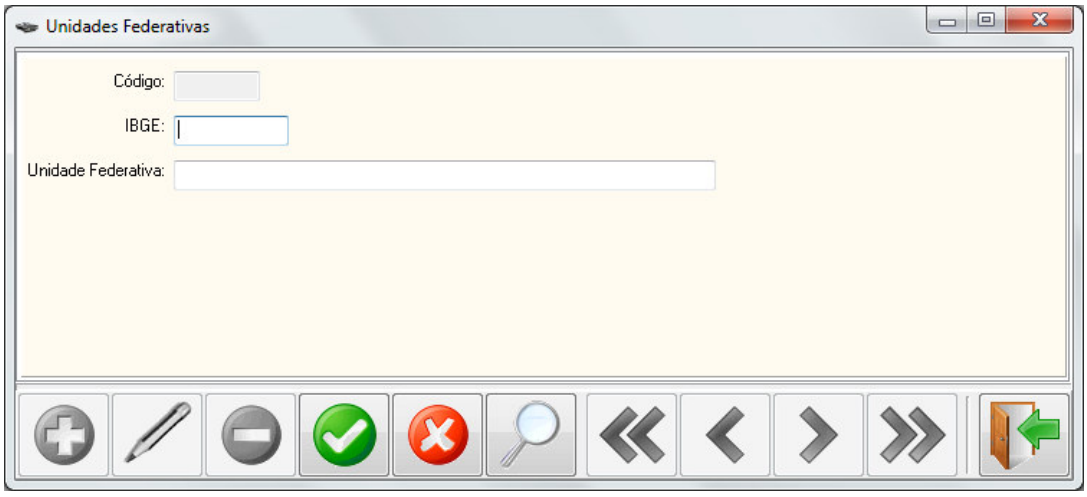

Figura 16 – Tela de cadastro de Unidades Federativas

A tela de cadastro de Tipos de Logradouros é apresentada na Figura 17. Nessa tela são cadastrados os tipos de logradouro como, por exemplo, Rua, Avenida e Travessa. Essa tela contém os campos código, tipo do logradouro que é um código próprio e a descrição do tipo do logradouro.

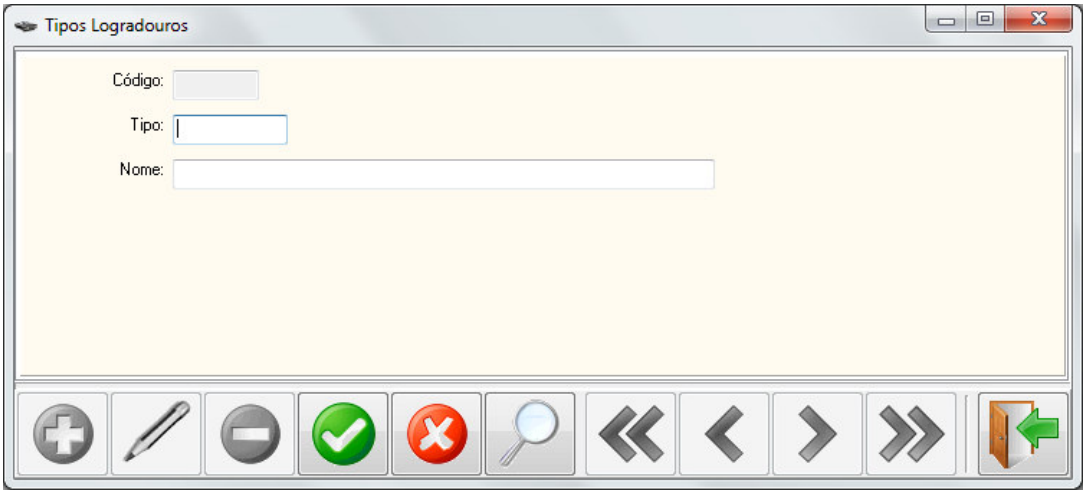

Figura 17 – Tela de tipos de logradouros

O cadastro de tipos de logradouros é utilizado no cadastro de logradouros que é apresentado na Figura 18. A tela de cadastro de logradouros contém o código, o tipo do logradouro e a descrição do logradouro. O tipo possui um botão para selecionar os tipos de logradouros cadastrados pelo formulário apresentado na Figura 17.

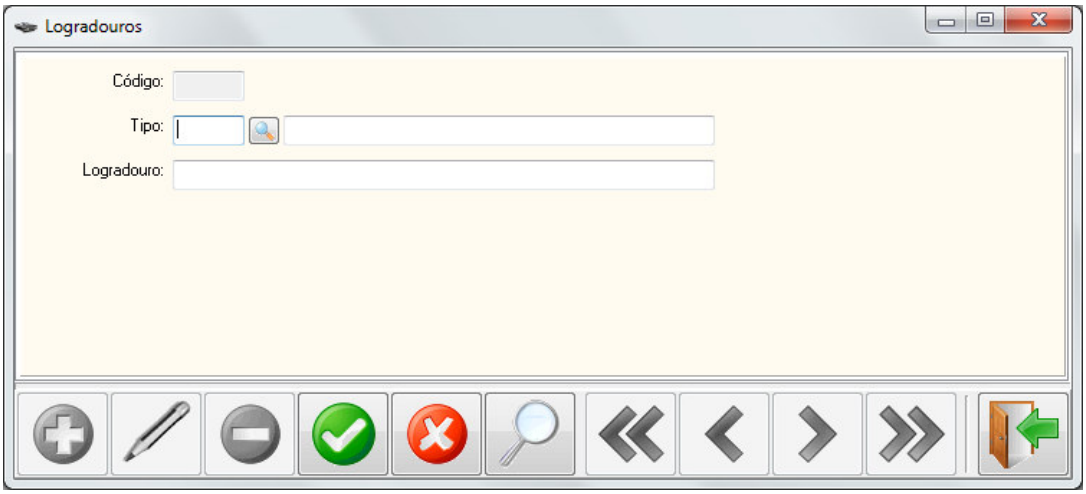

Figura 18 – Tela de logradouros

Na Figura 19 está a tela responsável pelo cadastro de clientes. A aba “Dados Pessoais” contém as informações específicas do cliente tais como nome, sexo, data de nascimento, etc. Nessa tela há um *checkbox* “Cadastro Simplificado” que faz com que alguns campos fiquem desabilitados, criando assim, uma forma de cadastro mais rápida, porém sem tantas informações. Já por sua vez o *combobox* “Situação Cadastro” apresenta qual é a atual situação cadastral do cliente em questão. Quando ativo o cliente está apto a qualquer movimentação de locação no sistema, e os dados no cadastro são apresentados como habilitados.

Quando selecionada a opção bloqueado, nesse cadastro, significa que o cliente tem alguma pendência para com a empresa e não poderá fazer nenhuma movimentação de locação. Quando Inativo, o cliente já não utiliza os serviços da empresa em questão há um bom tempo, ou por alguma razão específica o mesmo foi categorizado nessa situação. Quando em situação inativo todos os dados do cliente ficam inativos, exceto o campo "Situação Cadastro".

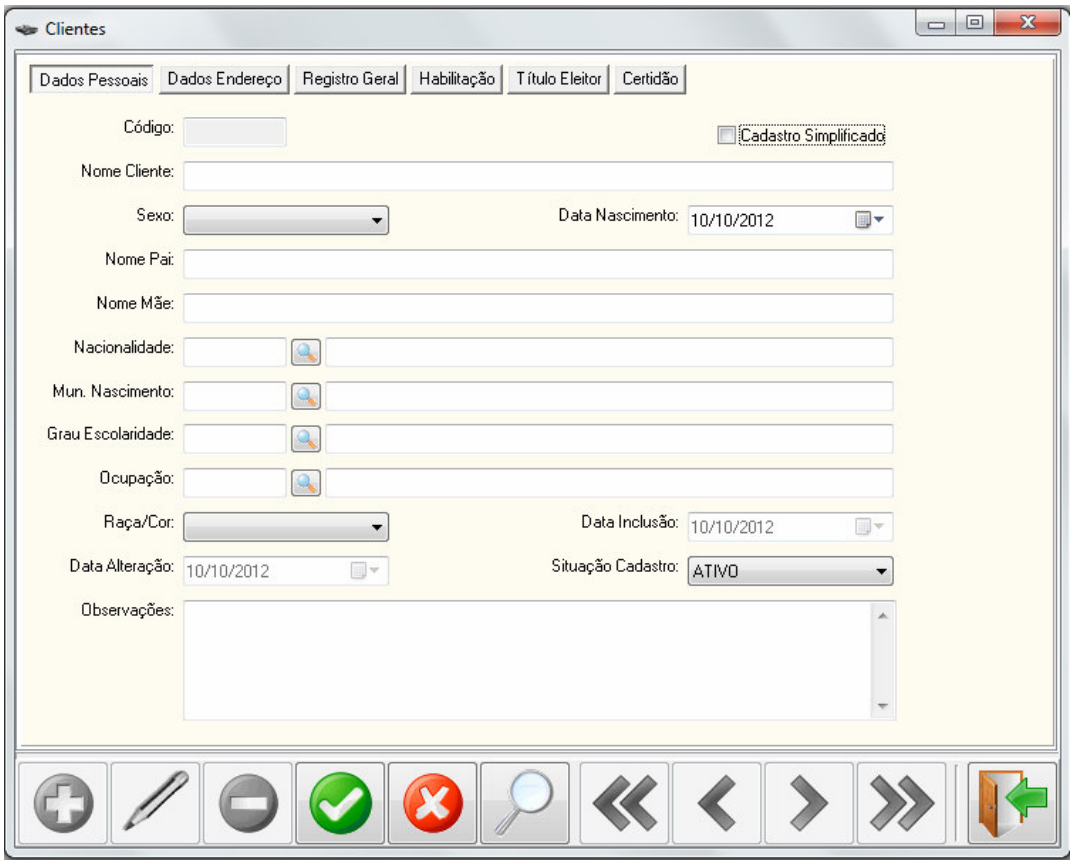

Figura 19 – Tela de cadastro veículos – aba dados pessoais

A aba "Dados Endereço" contém os dados do endereço do cliente, tais como CEP (Código de Endereçamento Postal), logradouro, bairro, complemento e telefones. Nesse cadastro é necessário que o cliente informe preferencialmente o seu endereço residencial, uma vez que abrange apenas o cadastro permite apenas um endereço por cliente.

A aba "Registro Geral" compreende os dados do RG (Registro Geral) e CPF (Cadastro de Pessoa Física) do cliente, que são indispensáveis no cadastramento do mesmo. Assim como dados mais específicos do RG, tais como Órgão Emissor e UF (Unidade de Federação) do mesmo.

A aba “Habilitação” contém alguns dos dados mais importantes e indispensáveis para o cadastro, tais como o número da habilitação, data de validade e o tipo de habilitação. Esses dados indicam se o cliente está apto a locar alguma espécie de veículo ou não.

A aba “Título Eleitor” contém um conteúdo os dados cívicos do cliente em questão, tais como o número do título de eleitor, data de emissão, zona eleitoral, seção eleitoral, etc.

A aba “Certidão”, assim como a aba “Título Eleitor”, contém conteúdo adicional, relacionado à certidão em questão, seja ela de nascimento, casamento, ou outra.

A Figura 20 apresenta um exemplo de uma tela genérica criada para tabelas internas do sistema em que os usuários Cliente e Atendente não possuem acesso, somente os usuários Gerente e Administrador. Tabelas como Nacionalidades e Órgãos emissores, são algumas das tabelas que utilizam essa pesquisa. Contendo somente o código e a descrição da tabela que está relacionada a ela no momento. Nesta tela de pesquisa apenas os campos de navegação ficam habilitados para seleção e os demais todos ficam desabilitados.

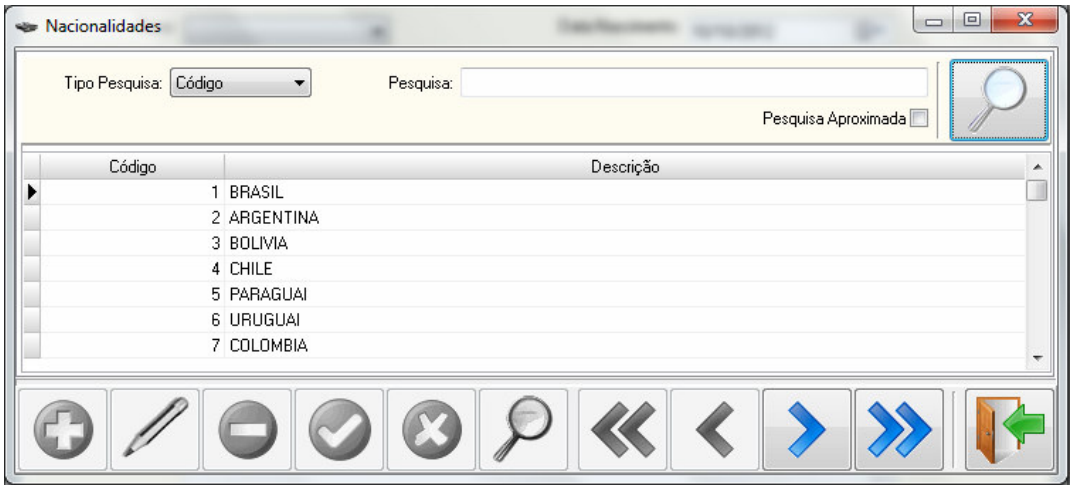

Figura 20 – Tela de cadastro de nacionalidades

A tela responsável pelo cadastro de Empresas, prestadoras ou tomadoras de serviços é apresentada na Figura 21. Essa tela é composta pelas seguintes abas:

a) Aba “Dados Empresa” - essa aba contém os principais dados da empresa, tais como nome fantasia, razão social, CNPJ (Cadastro Nacional de Pessoas Jurídicas) ou CPF dependendo do campo tipo da empresa ser física ou jurídica. O campo “Situação Empresa” possui o mesmo comportamento do campo “Situação Cliente” do cadastro de Clientes. Quando ativo os campos são deixados habilitados. Quando inativo todos os restantes desabilitados, impedindo a empresa selecionada de realizar movimentações quando bloqueado.

b) Aba “Endereço Empresa” - essa aba é responsável pelo cadastramento do endereço da empresa em questão, contendo os campos CEP, número, bairro e logradouro como sendo os principais.

c) Aba “Dados Responsável” - essa aba tem como finalidade cadastrar o proprietário ou o responsável pela empresa que está prestando ou tomando um serviço da locadora em questão.

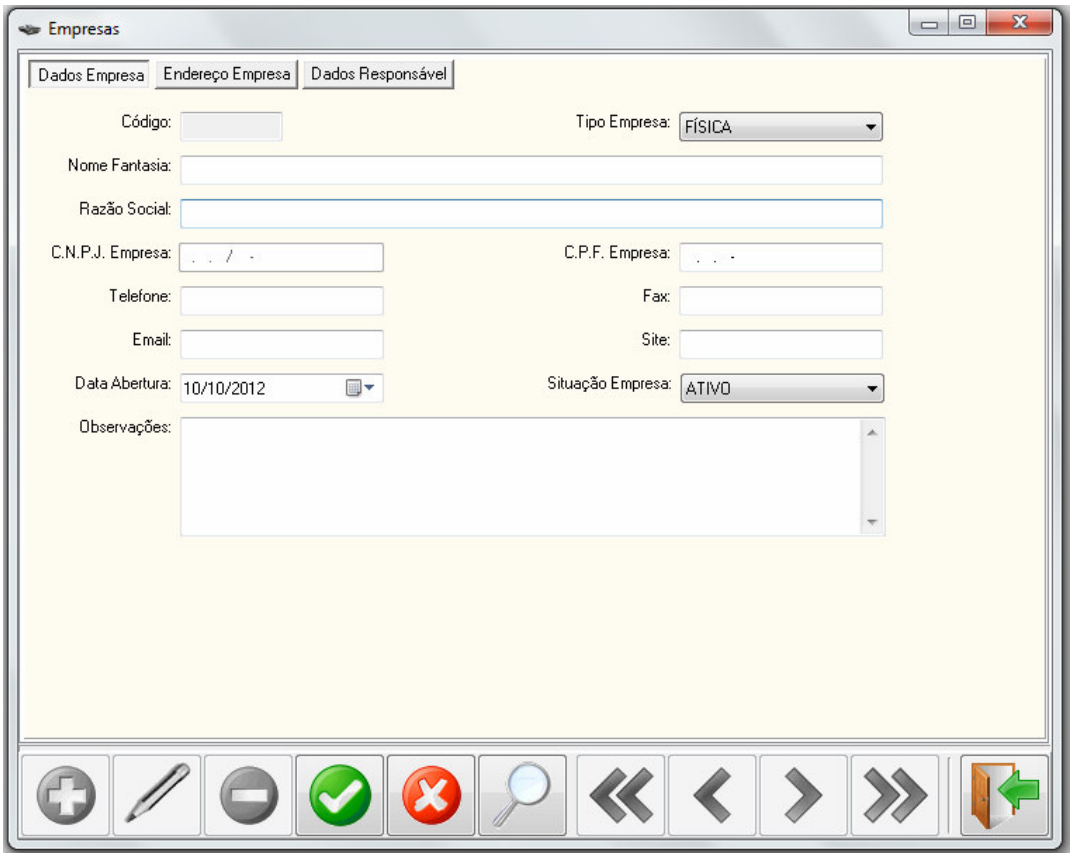

Figura 21 – Tela de cadastro de empresas

A Figura 22 apresenta o cadastro de usuário.

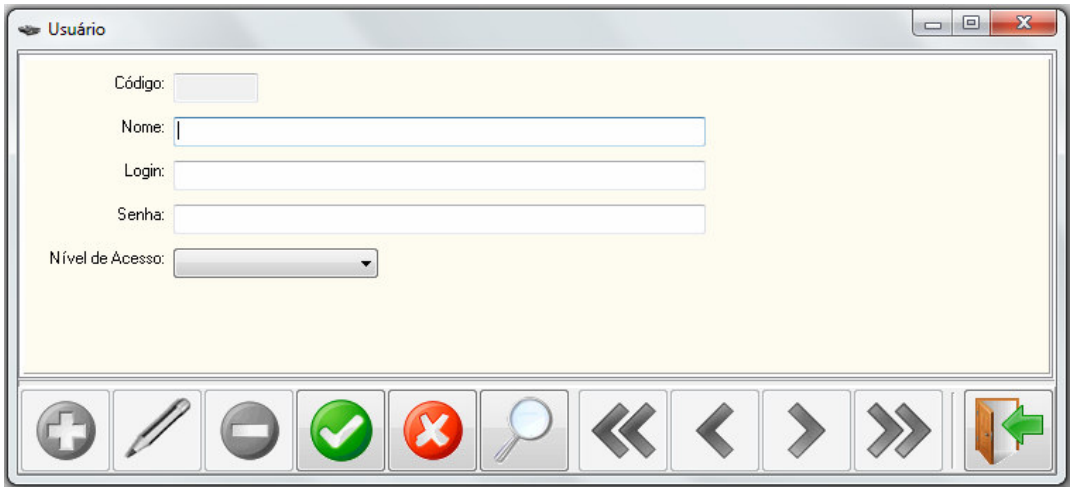

Figura 22 – Tela de cadastro de usuários

O cadastro de usuários contém os campos nome do usuário, *login* e senha do mesmo e também seu nível de acesso, que pode ser atendente, gerente ou administrador. Apenas o administrador tem acesso a este cadastro, portanto somente ele poderá designar os níveis de acesso dos demais envolvidos.

4.3.2 Aplicação Desktop - Módulo Veículos

Na Figura 23 estão os itens de menu que se referem aos cadastros relacionados ao veículo. O menu veículos permite acesso aos formulários relacionados aos cadastros do módulo denominado veículos.

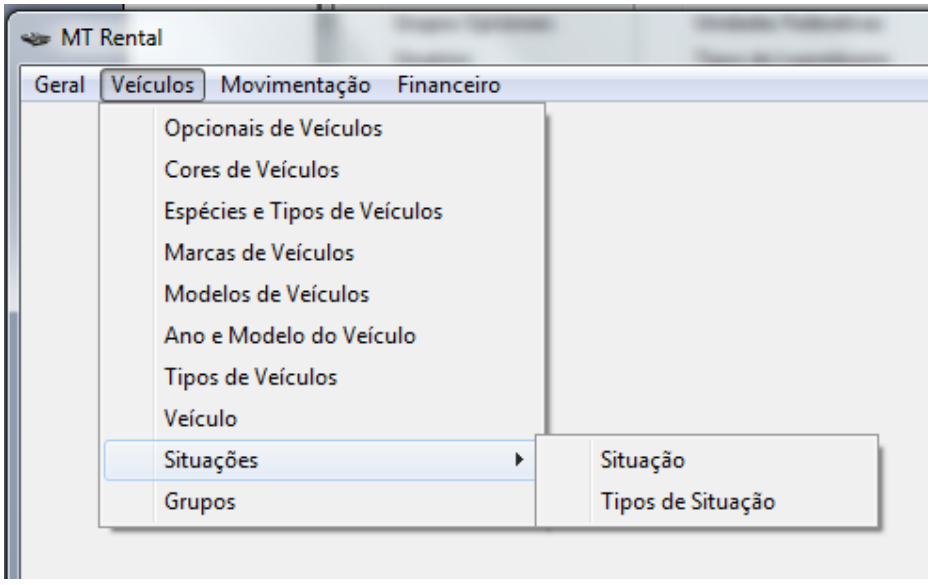

Figura 23 – Menu Veículos e seus itens de menu

A seguir serão apresentadas as telas de cadastro relacionadas a veículos. As telas desses cadastros são acessadas através dos itens de menu exibidos na Figura 23.

Será utilizado o cadastro de Opcionais de Veículos para mostrar a tela de pesquisa dos cadastros (Figura 24). Esta tela é padrão para todos os cadastros. No seu topo há um campo de pesquisa. O campo "Tipo de Pesquisa" serve para saber se a pesquisa é por código ou por descrição. Ao seu lado está o campo "Pesquisa" no qual o usuário informará o código ou a descrição do item que está procurando. É feita a validação no sentido de esse campo aceitar apenas números, se o campo "Tipo de Pesquisa" está selecionado como código. "Pesquisa Aproximada" faz com que sejam pesquisadas as descrições ou códigos por aproximação e bem ao lado direito da tela o botão de "Pesquisa", representado pelo ícone de uma lupa.

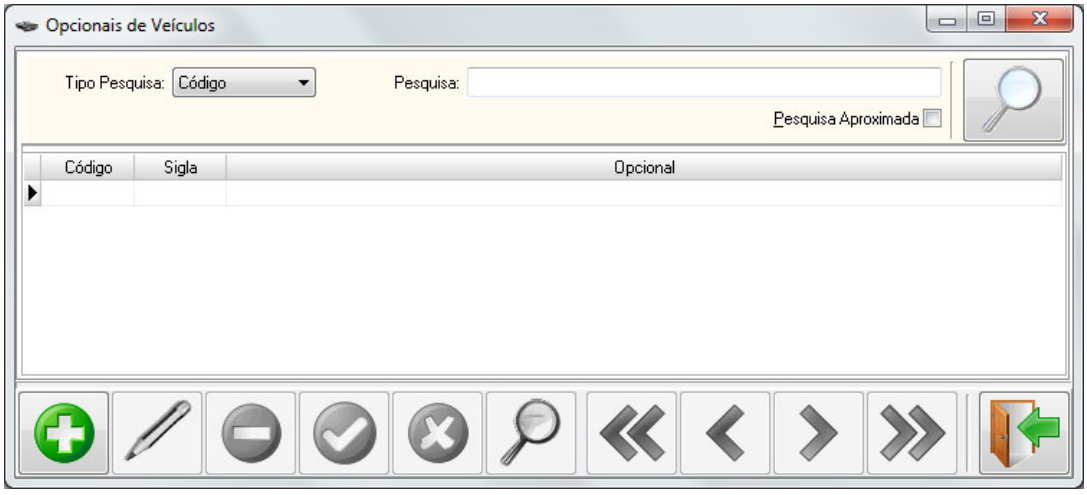

Figura 24 – Cadastro de opcionais de veículos

Abaixo das pesquisas, como pode ser visualizado na Figura 24, está um *grid* com os campos disponíveis de cada tela. Na parte inferior estão os seguintes botões, apresentados pela ordem sequencial da esquerda para a direita:

a) Incluir (ícone que representa um sinal "+") - Inclui um novo registro. Se estiver na tela de Pesquisa, abrirá a tela de Manutenção.

b) Editar (ícone representado um lápis) - Abre o registro selecionado no *grid* para edição. Se estiver na tela de Pesquisa, abrirá a tela de Manutenção. Um duplo clique no registro do *grid* tem o mesmo efeito deste botão.

c) Excluir (ícone que representa um sinal "-) - Exclui o registro selecionado.

d) Salvar (ícone que representa uma marca de verificação) - Quando em edição ou em inserção gravará o registro selecionado no banco de dados.

e) Cancelar (ícone que representa “X”) - Cancela a edição ou a inserção que ainda não foi gravada em banco.

f) Pesquisar (ícone de uma lupa) - Utilizada para pesquisar em registros. Quando na tela de manutenção, a opção de pesquisa fica desabilitada.

g) Primeiro (ícone “<<”) - Posiciona no primeiro registro.

h) Anterior (ícone “<”) - Posiciona no registro anterior ao atual.

i) Próximo (ícone “>”) - Posiciona no próximo registro.

j) Último (ícone “>>”) - Posiciona no último registro.

h) Sair (ícone que representa uma porta aberta com uma flecha indicativa) - Fecha a tela.

Esses botões são utilizados em todas as telas de cadastro e possuem as mesmas funcionalidades. Os botões se comportam de forma “inteligente”, por exemplo, quando está sendo inserido um item, o botão de inserção fica desabilitado até que o registro seja efetivado no banco de dados, o mesmo ocorre para os demais botões.

Os botões de navegação trabalham de acordo com o número do registro em que está posicionado (selecionado). Por exemplo, se estiver no primeiro registro os botões “Primeiro” e “Anterior” ficam desabilitados.

A tela de manutenção do cadastro de Opcionais de Veículos (Figura 25) contém o campo código que é responsável para identificar cada item, a sigla de cada opcional e o nome do opcional do veículo, exemplo: vidros elétricos, travas elétricas.

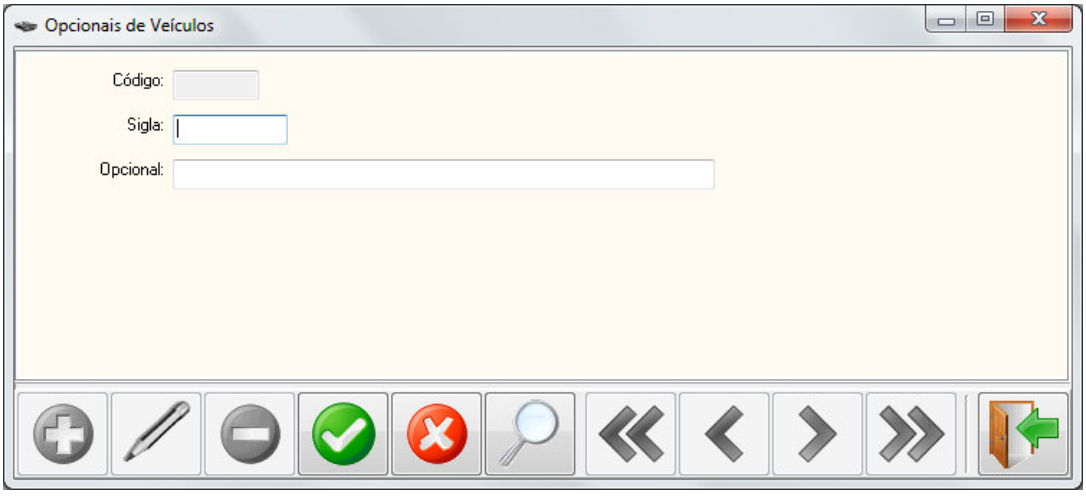

Figura 25 – Tela de manutenção de opcionais de veículos

A tela de Cores de Veículos, apresentada na Figura 26, contém os campos código da cor e a descrição da mesma.

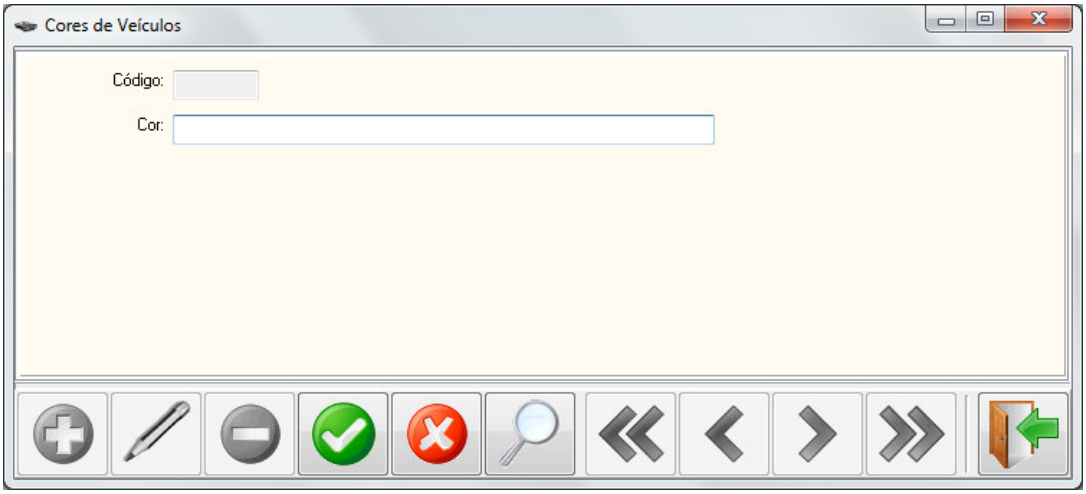

Figura 26 – Tela de manutenção de cores de veículos

A tela de cadastro de tipos de veículos que serve para identificar qual é a espécie do veículo, ou seja, se é um veículo de carga, de passageiros e etc. Compreendida pelos campos código, o código do Denatran (Departamento Nacional de Trânsito) e a descrição da espécie (campo Espécie). Essa tela é apresentada na Figura 27.

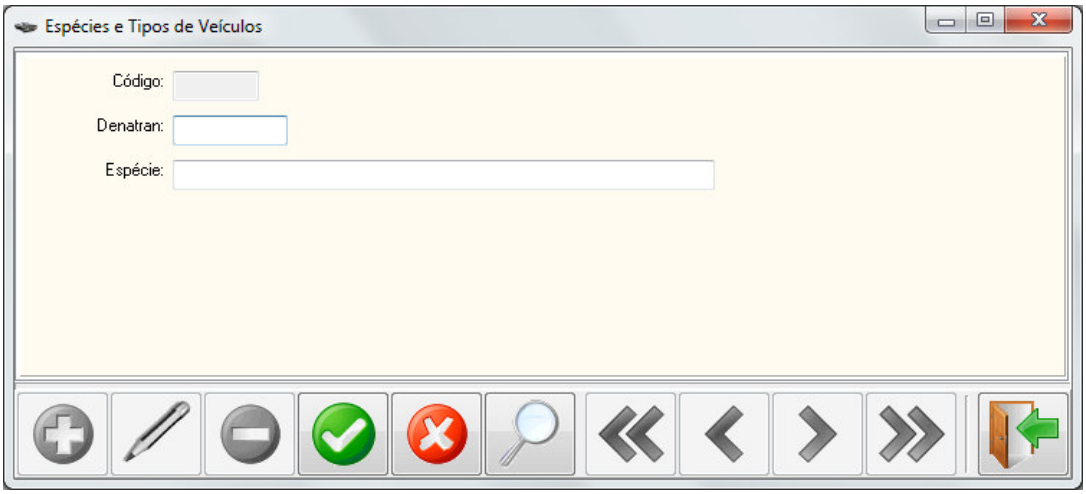

Figura 27 – Tela de cadastro de espécie e tipos de veículos

Na Figura 28 está a tela que é responsável pelo cadastro da marca do veículo, incluindo a tabela FIPE e a descrição da marca.

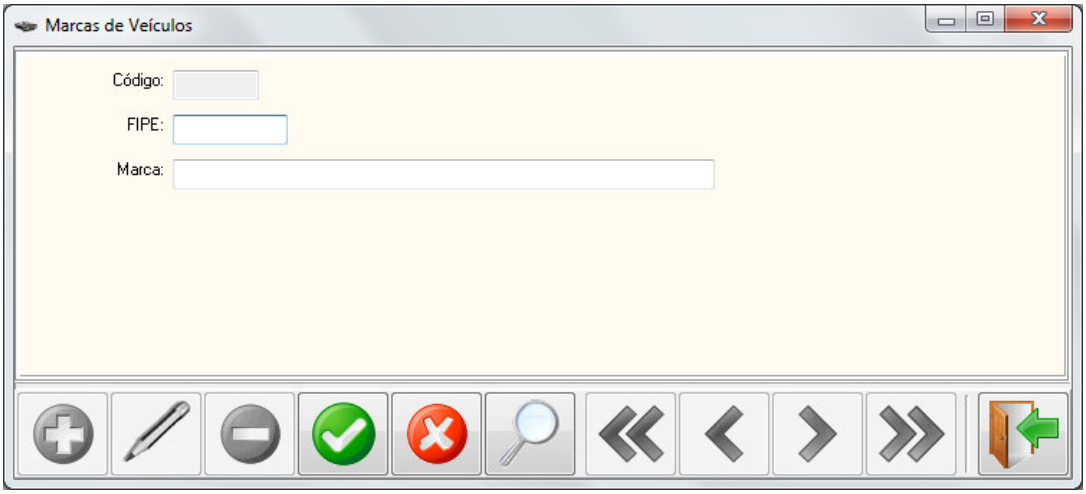

Figura 28 – Tela de cadastro de marcas de veículos

Na Figura 29 está apresentada a tela de cadastro dos modelos de veículos. Esse formulário contém os campos código, tabela FIPE, marca e modelo. O campo marca possui um botão de pesquisar, para que as marcas cadastradas na tela apresentada na Figura 28 possam ser selecionadas. Ao ser escolhido o código de uma marca é apresentada a sua descrição automaticamente ao lado do ícone de pesquisa. O usuário pode digitar o código correspondente à marca ao invés de pesquisá-lo.

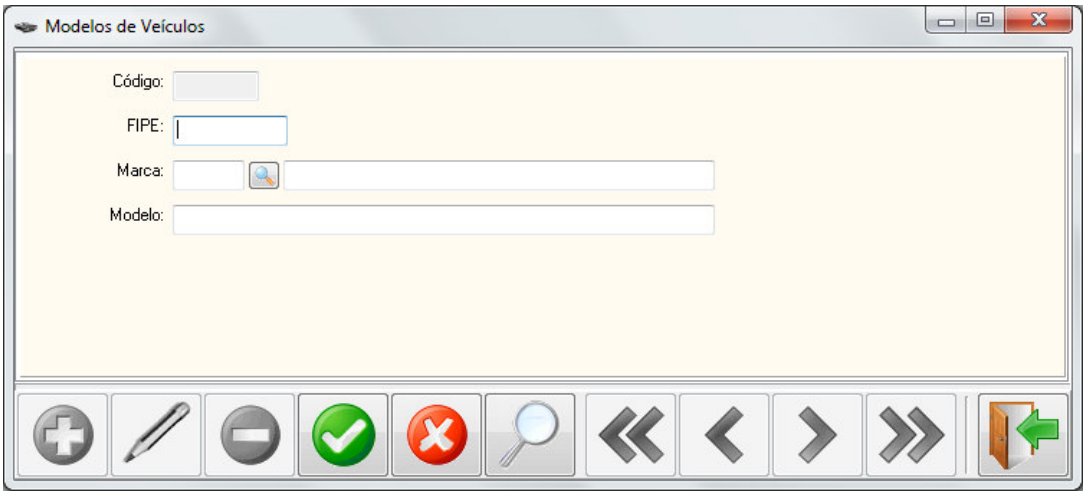

Figura 29 – Tela de cadastro de modelos de veículos

A tela de cadastro de ano e modelo do veículo (Figura 30) contém o código, a tabela FIPE, o campo modelo com sua pesquisa própria, o campo descrição que contém o ano e detalhes do modelo do veículo concatenados e também contém um campo para o valor do veículo.

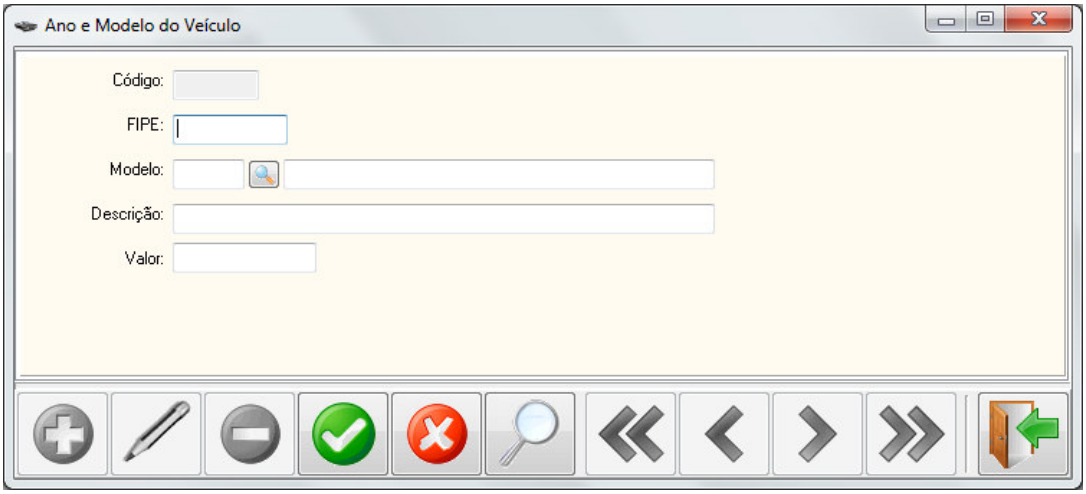

Figura 30 – Tela de cadastro de ano e modelo de veículos

A tela de tipos de veículo (Figura 31) contém o código interno do sistema, o código do Denatran, o tipo do veículo (exemplo: motocicleta, caminhão, etc.), a espécie do veículo que foi cadastrada anteriormente na tela de cadastros de espécie de veículos, com sua própria pesquisa e o grupo tipo de carroceria que informa se o veículo é blindado ou um utilitário.

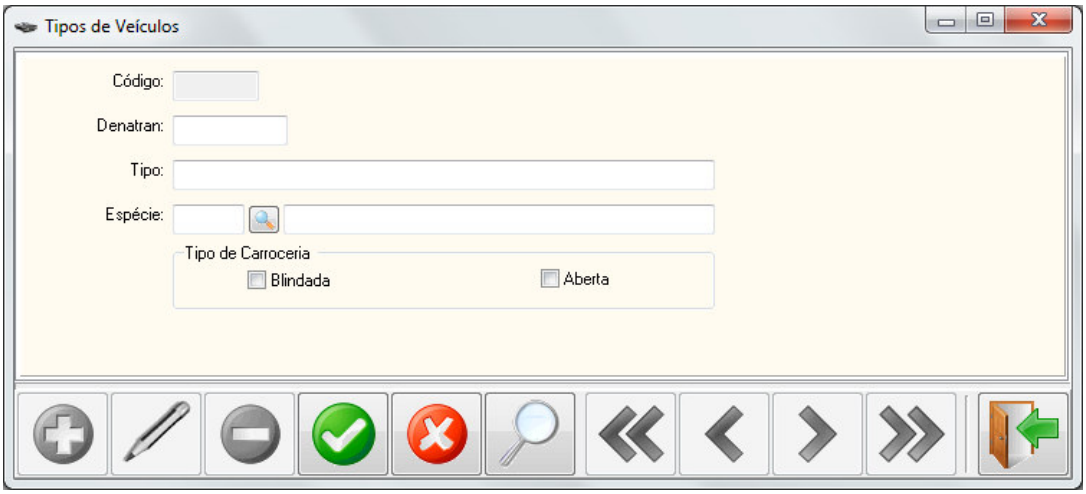

Figura 31 – Tela de cadastro de tipo de veículo

A tela de cadastro de veículos é apresentada na Figura 32 e que utiliza diversos outros cadastros. É o formulário mais completo desse módulo denominado Veículo.

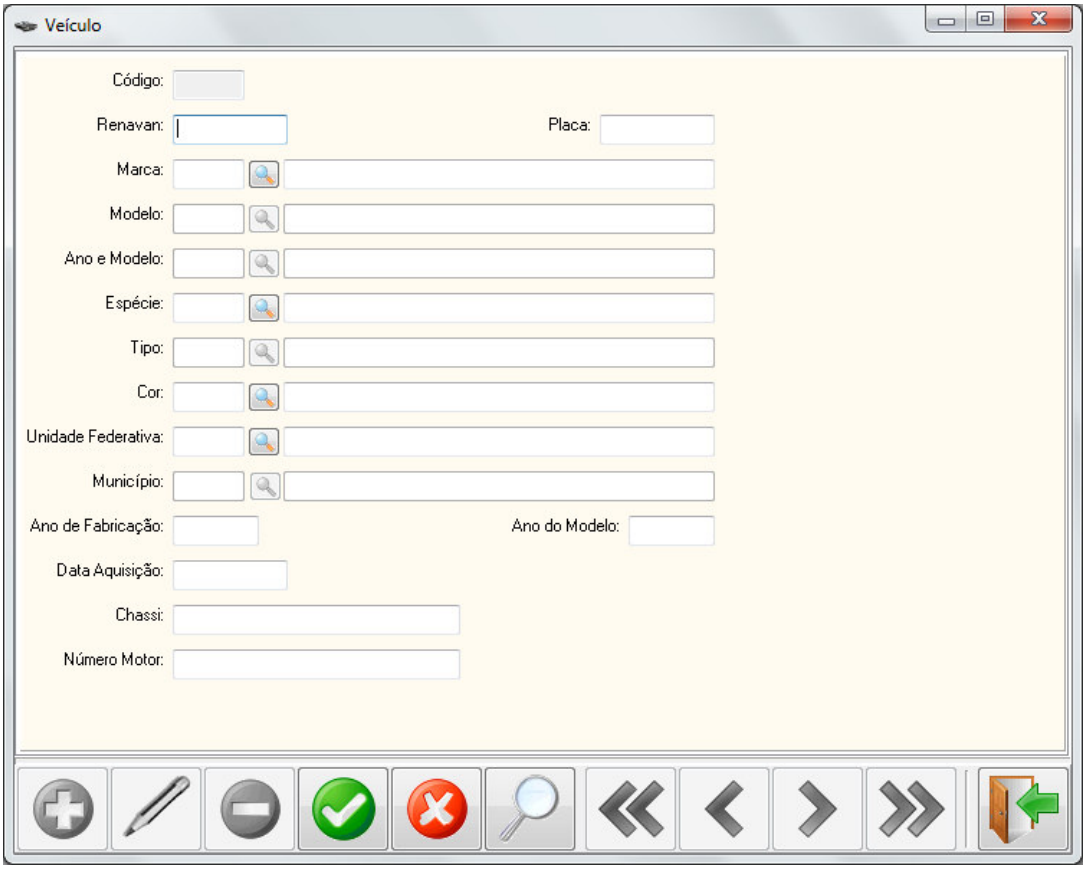

Figura 32 – Tela de cadastro veículos

Essa tela de cadastro de veículos contém o campo código interno do sistema, Renavan do veículo, placa, marca, modelo, ano e modelo, espécie, tipo, cor, Unidade Federativa e Município. O campo código é gerado automaticamente. Os campos Renavan e placa são indicados e os demais campos listados são obtidos por meio de outros cadastros. Isso é indicado pelo ícone que representa uma lupa e está posicionado ao lado do campo. Ao ser digitado o respectivo código ou o mesmo ser obtido por meio de pesquisa o campo a descrição desse código é automaticamente preenchida.

No cadastro de veículos (Figura 32), o campo modelo vem como padrão desabilitado e só será habilitado para a edição quando o campo marca estiver valorizado. Caso o campo marca volte a ficar sem valor, o campo modelo ficará desabilitado. O mesmo ocorre com o campo ano e modelo, em relação ao campo modelo, espécie e tipo e Unidades Federativas com Município.

Esse formulário contém também os campos ano de fabricação e ano do modelo, data da aquisição do veículo, número do chassi e número do motor. Em todas as telas que tenham relacionamentos com outras, exemplo marcas com modelos, o ícone de pesquisa chamará a

pesquisa. Ao ser realizado um duplo clique no registro escolhido, a tela atual é fechada e os dados são atribuídos ao campo de pesquisa. O código do item também poderá ser informado pelo usuário e ao passar o campo com a tecla "*enter*" ou "*tab*" o sistema carrega automaticamente a descrição da pesquisa. Se o usuário informar o valor zero e clicar em "*enter*" ou "*tab*" o sistema chamará a tela de pesquisa referente ao código informado.

O cadastro de grupos, apresentado na Figura 33, é responsável pelos grupos de categorias em que cada veículo se enquadra, fazendo assim com que o proprietário da empresa crie os grupos que contém em sua frota. Um grupo permite identificar veículos com conjuntos semelhantes de acessórios, por exemplo, veículos com trava e ar condicionado.

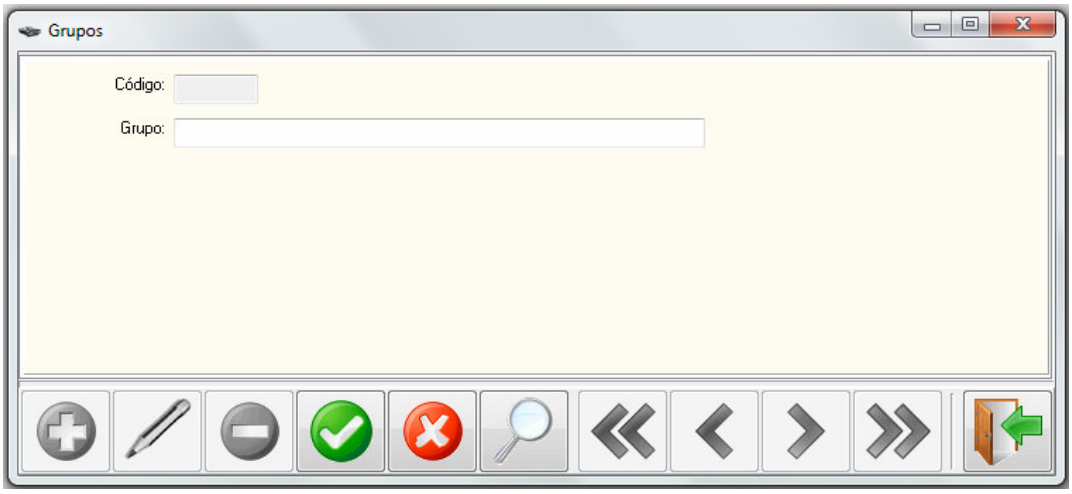

Figura 33 – Tela de cadastro de grupos

O cadastro de serviços é apresentado na Figura 34. Nessa tela são cadastrados os serviços que as outras empresas prestam para a locadora em questão.

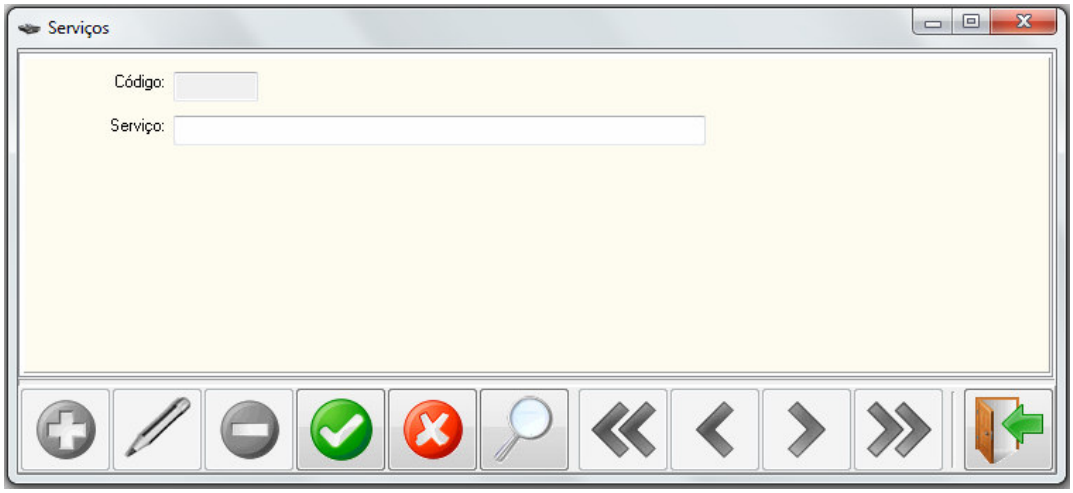

Figura 34 – Tela de cadastro de serviços

O cadastro de situações (Figura 35) tem como objetivo cadastrar as situações em que se encontram os veículos e suas respectivas cores, como por exemplo: cor – verde e tipo de situação - disponível.

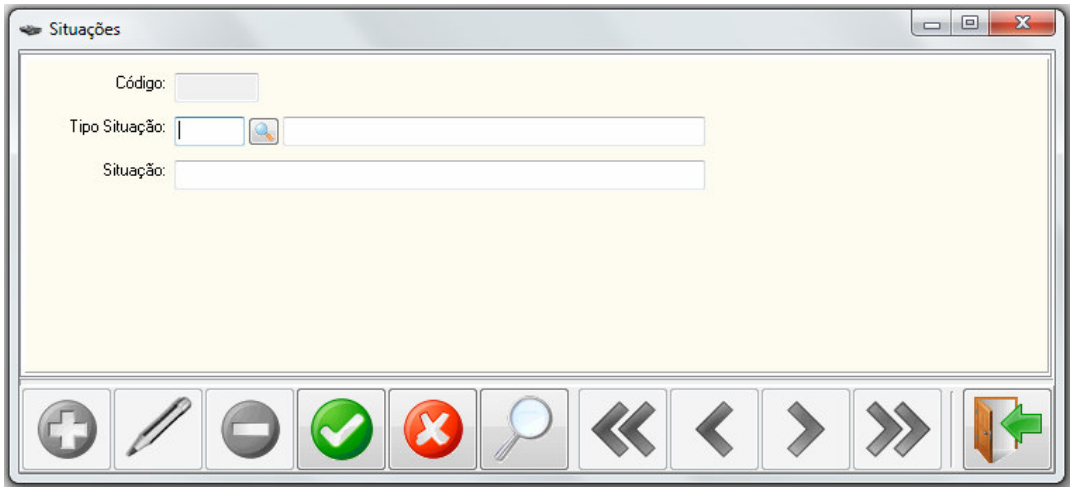

Figura 35 – Tela de cadastro de situações

No cadastro de situações é escolhido um tipo. Esse tipo é cadastrado por meio do formulário apresentado na Figura 36. Esse cadastro tem ligação com o cadastro de situações e tem como objetivo ajudar o usuário saber se o veículo com o tipo de situação "Manutenção" está em uma chapeação ou em uma oficina mecânica, por exemplo.

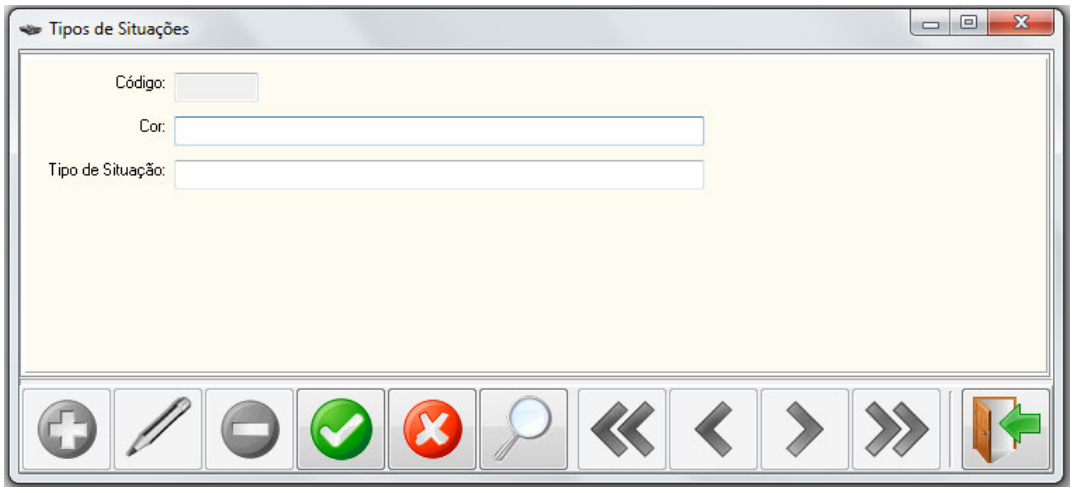

Figura 36 – Tela de cadastro de tipos de situações

O cadastro de grupos opcionais é apresentado na Figura 37. Esse cadastro une os grupos com os opcionais dos veículos, fazendo com que grupos com diferentes tipos de opcionais possam ser criados, possibilitando mais escolhas ao seu cliente.

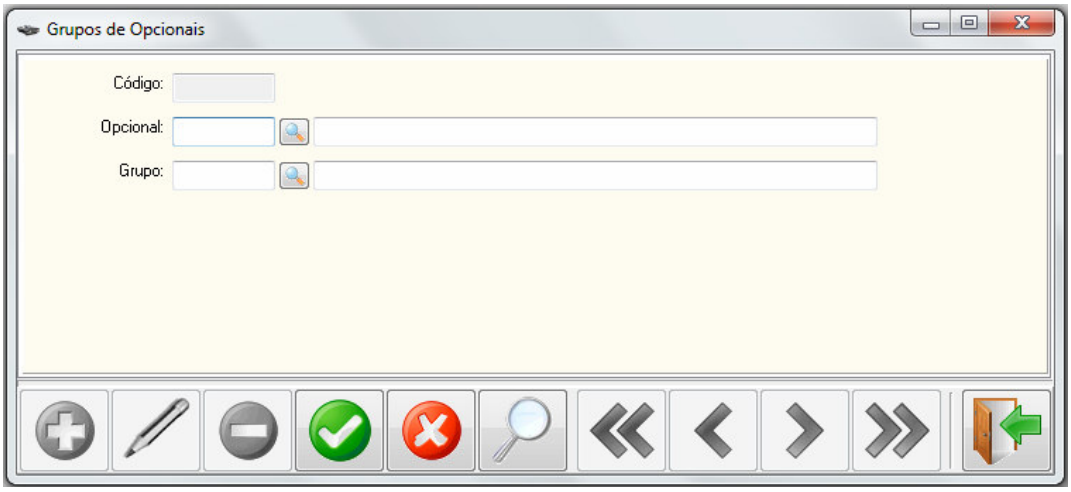

Figura 37 – Tela de cadastro de grupos de opcionais

4.3.3 Aplicação Desktop - Módulo Movimentação

O módulo movimentação se refere às opções de locação de veículos e de manutenção dos mesmos. A Figura 38 apresenta as opções do menu “Movimentação”.

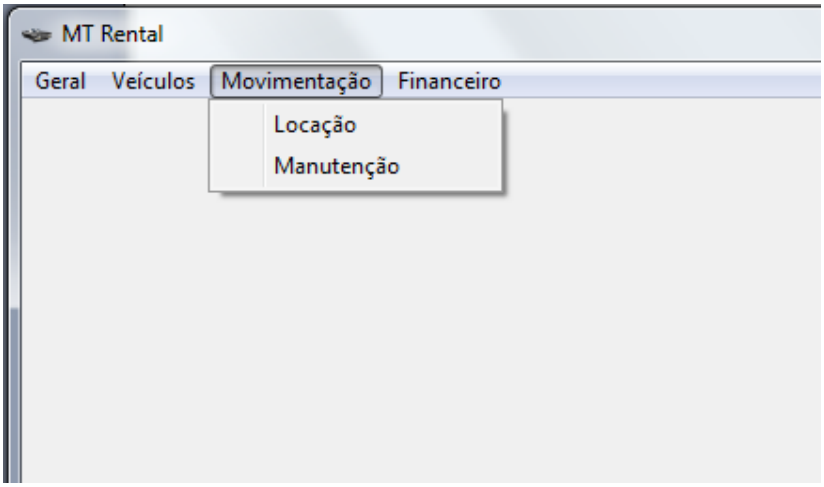

Figura 38 – Menu Movimentação e seus itens de menu

Na Figura 39 é apresentada a tela de manutenção. Essa tela é utilizada quando um veículo é enviado para manutenção.

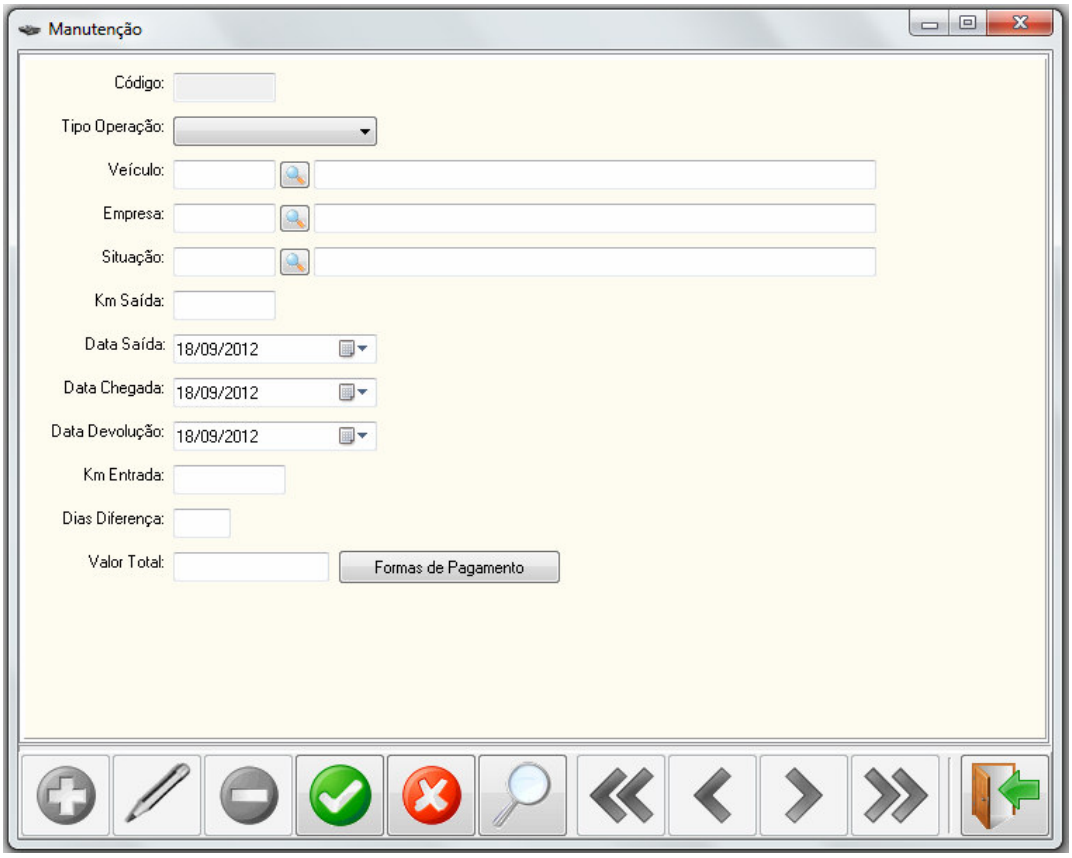

Figura 39 – Tela de manutenção de veículos

A tela de manutenção contém o campo tipo operação com os valores entrada e saída. Quando saída, o veículo parte da locadora para a empresa que prestará a manutenção. Quando entrada ele acaba de retornar desta manutenção. Nesse cadastro, os dados do veículo são acompanhados pelos dados da empresa que prestará o serviço de manutenção. A situação informará que o veículo foi enviado para uma mecânica ou chapeação, por exemplo. São informados também a quilometragem de saída do veículo, a data de saída e a data pré-estipulada para a chegada, a data de devolução que é quando o veículo de fato voltou da manutenção são informados. Se a data de devolução não é igual a data de chegada, então o campo "Dias Diferença" armazena diferença de dias entre a data chegada e a data devolução, para que assim o proprietário possa, caso queira, reajustar os valores quando houver atrasos no combinado. E por fim, estão o valor total e o botão Formas de Pagamento que abre a tela de "Contas a Pagar".

A tela de locação é composta por duas abas que são locação e *check-in*, conforme apresenta a Figura 40.

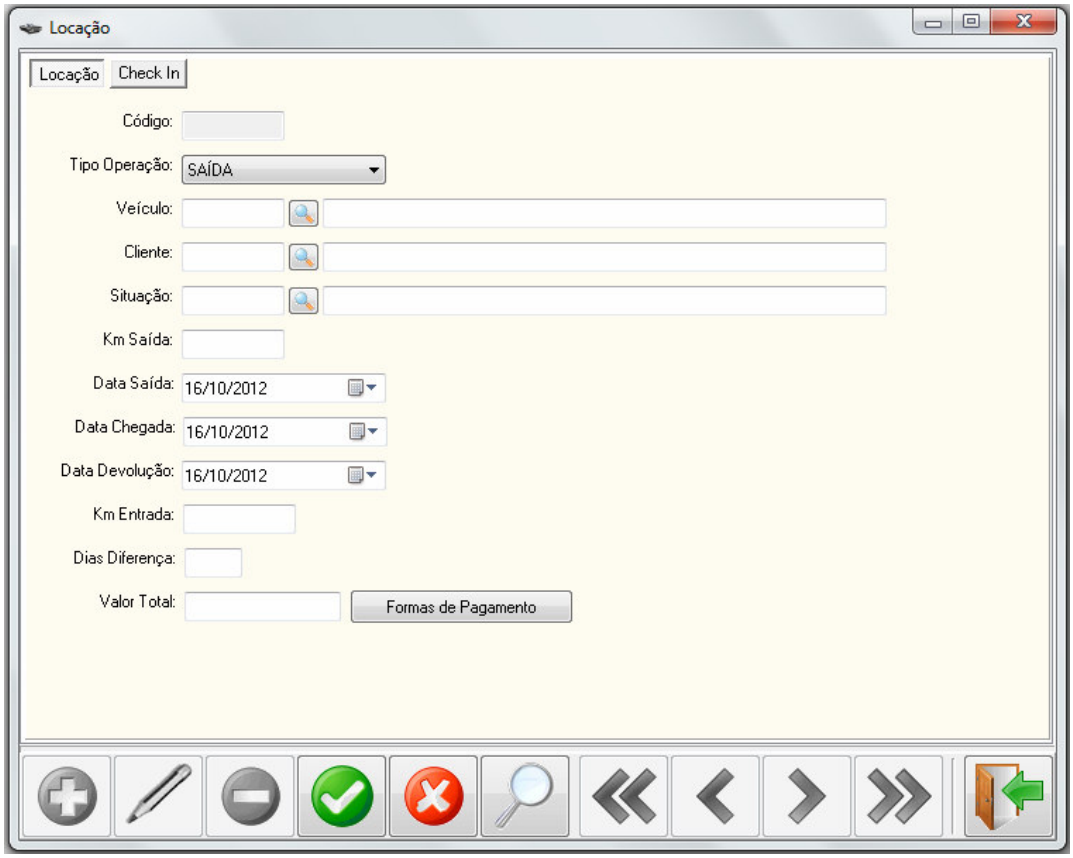

Figura 40 – Tela de locação de veículos

Na aba "Locação" (Figura 40) é realizada a locação dos veículos gerando uma saída através do campo "Tipo Operação". Quando o veículo é devolvido o usuário altera o campo para entrada que indica a devolução do veículo. Esse formulário conta com os campos veículo que é o carro a ser locado; cliente que é a pessoa (física ou jurídica) que está locando; a situação do veículo, que para a locação deve estar como disponível; a quilometragem de saída e a de entrada que é somada para ter em média quantos quilômetros o cliente andou e assim podendo gerar o preço final, caso seja assim que a locadora trabalha; data de saída do veículo; data de chegada, que é pré-estipulada na saída; e a data de devolução que é quando o veículo realmente volta à locadora. Os dias de diferença que são obtidos pela diferença entre a data de chegada e a data de devolução, podendo assim gerar novas diárias a serem pagas. incrementando o valor total. Ao final desta tela está a forma de pagamento que apresenta a tela de contas a receber.

A verificação de *check-in* é realizada através da indicação de possíveis danos existentes ou gerados no veículo durante a locação. As Figuras 41 e 42 apresentam as abas disponíveis para realizar o check-in, sendo que essa tela é revista após a devolução do veículo,

para assim saber se não houve danos ao veículo durante a etapa em que o mesmo foi locado. A aba “Frente e Lateral Esquerda” apresenta na figura de um veículo os possíveis lugares que se pode marcar quando houver algum dano antes ou depois da locação. Essa aba abrange a lateral esquerda e a parte dianteira do veículo (Figura 41).

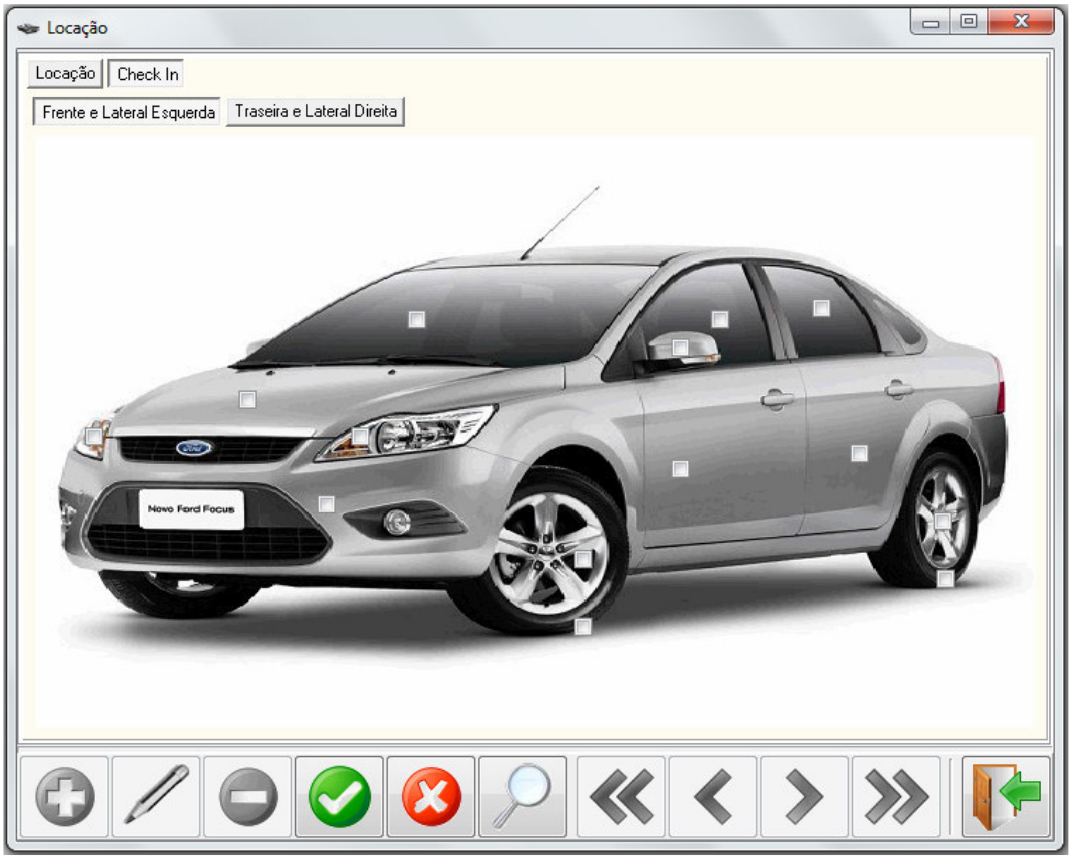

Figura 41 – Tela de locação de veículos: check-in 1

Na tela de locação, a aba “Check-in/aba Traseira e Lateral Direita”, apresenta também os vários pontos que podem ser marcados caso o veículo esteja danificado antes da locação ou volte danificado após uma locação. Essa aba apresenta a parte lateral direita e traseira do veículo (Figura 42).

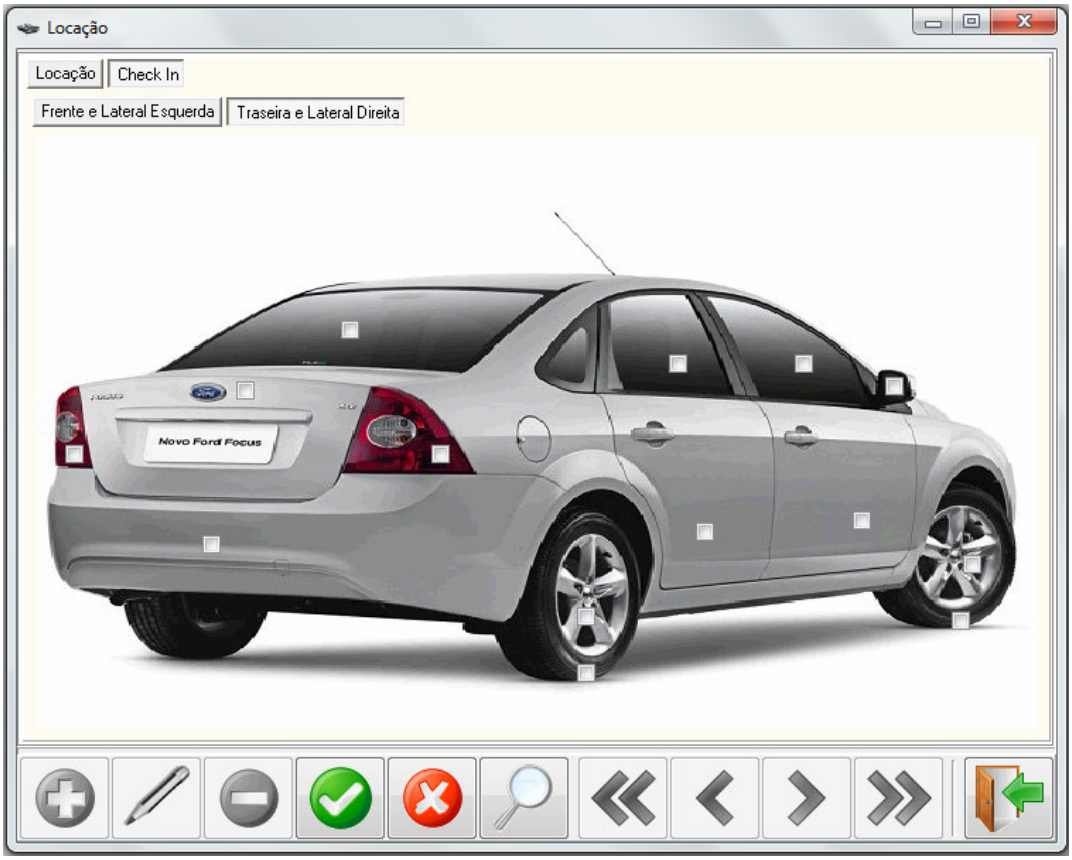

Figura 42 – Tela de locação de veículos: check-in 2

4.3.4 Aplicação Desktop - Módulo Financeiro

A Figura 43 apresenta as opções do menu Financeiro. As opções disponíveis são contas a pagar e contas a receber.

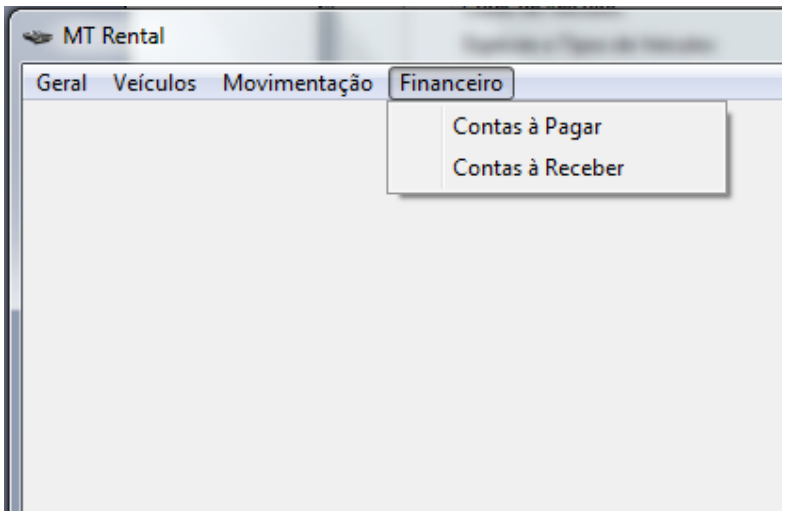

Figura 43 – Menu Financeiro e seus itens de menu

A tela de contas a pagar é apresentada na Figura 44.

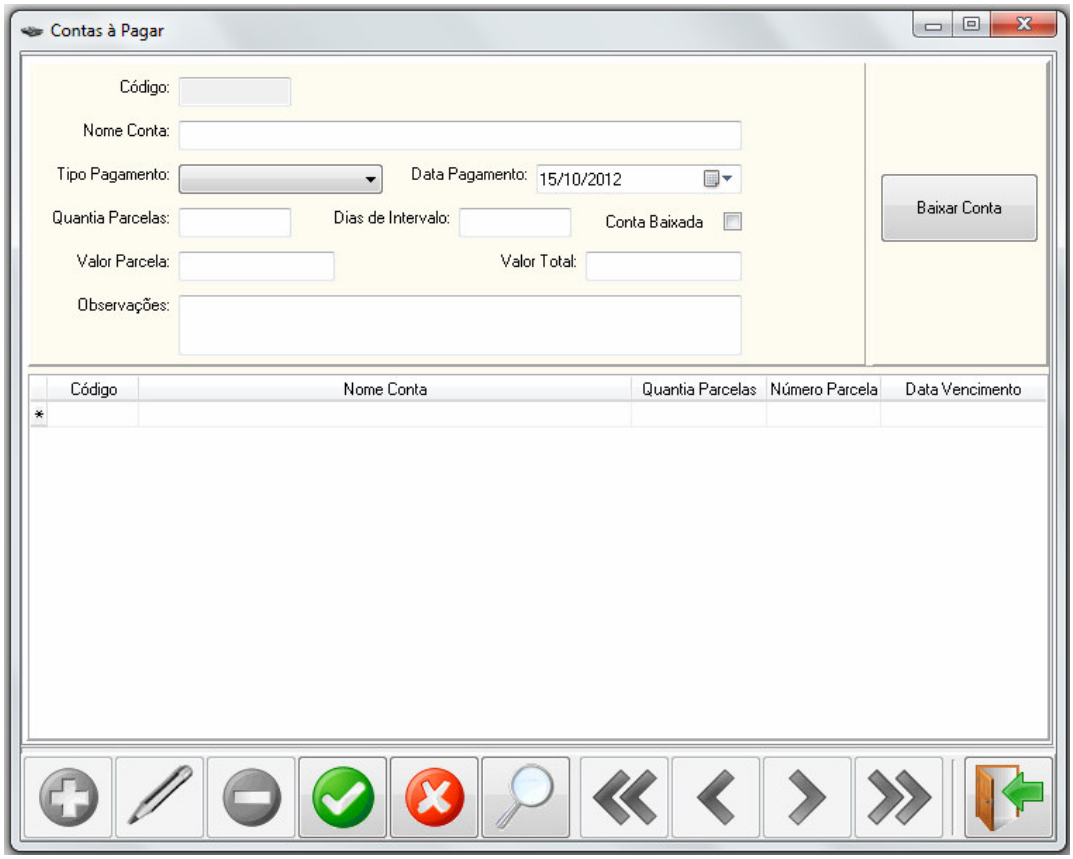

Figura 44 – Tela de cadastro de contas a pagar

Essa é a tela na qual o usuário lança as contas da empresa, informa o nome da conta e o tipo de pagamento, que compreende os valores à vista e a prazo. Quando à vista os campos "Dias de Intervalo" e "Valor Parcela" ficam desabilitados. Quando a opção a prazo é a marcada o usuário deve preencher os campos "Quantia Parcelas", "Dias de intervalo" e "Valor Parcela", para que após salvar todas a suas parcelas elas sejam apresentadas no grid da região inferior da tela. Após geradas as contas, o botão "Baixar Conta" serve para mudar o status da conta para baixado, depois de a mesma ter sido paga. Assim o *checkbox* "Conta Baixada" ficará marcado.

A tela de contas a receber possui os mesmos dados que a tela de contas a pagar e o seu funcionamento é semelhante. Assim como a tela de contas a pagar, a tela de contas a receber é utilizada para incluir contas, neste caso, contas a receber de clientes que locaram veículos, por exemplo.

4.3.5 Aplicação Web

A Figura 45 apresenta a tela principal da página *web* desenvolvida. É o módulo *web* do sistema.

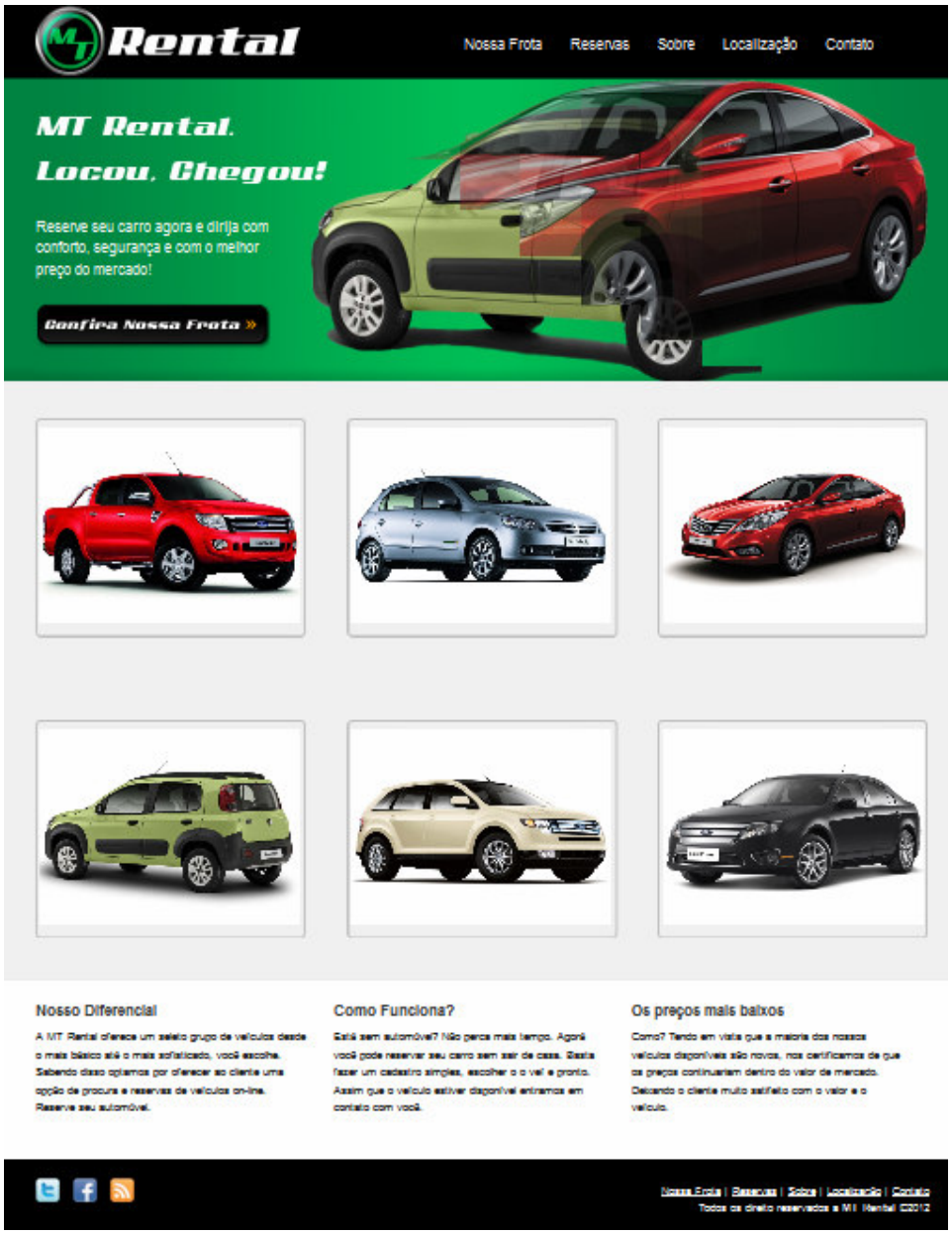

Figura 45 – Página inicial do site

Na região central da tela são apresentadas imagens aleatórias de alguns veículos ilustrativos com efeitos de transição entre eles. Ao lado da imagem é apresentado um botão

chamado "Confira Nossa Frota" que ao ser clicado mostra a mesma página disponível no menu "Nossa Frota".

Logo abaixo da região que apresenta o veículo está uma lista de veículos disponíveis para locação. Ao clicar em algum desses veículos é apresentado uma página com os detalhes do mesmo e com uma opção para reservar o veículo.

Após a lista de veículos são apresentados três textos referentes a empresa. Sendo quer um deles se refere ao diferencial da empresa, o segundo apresenta um texto explicativo sobre os veículos e o último sobre o motivo dos preços tão acessíveis da empresa.

No rodapé da página é apresentada uma mensagem de direitos reservados, alguns *links* para o Facebook, Twitter e Youtube da empresa. Também são apresentados novamente os mesmos menus disponíveis no cabeçalho da página.

O menu “Nossa Frota” lista todos os veículos da locadora com sua descrição e se cada um deles está disponível ou não. Quando disponível permite abrir os detalhes do veículo com um clique, onde são apresentadas todas as imagens do veículo e um botão para efetuar a Reserva do Veículo é disponibilizado. A Figura 46 apresenta a tela que é mostrada na opção “Nossa Frota”.

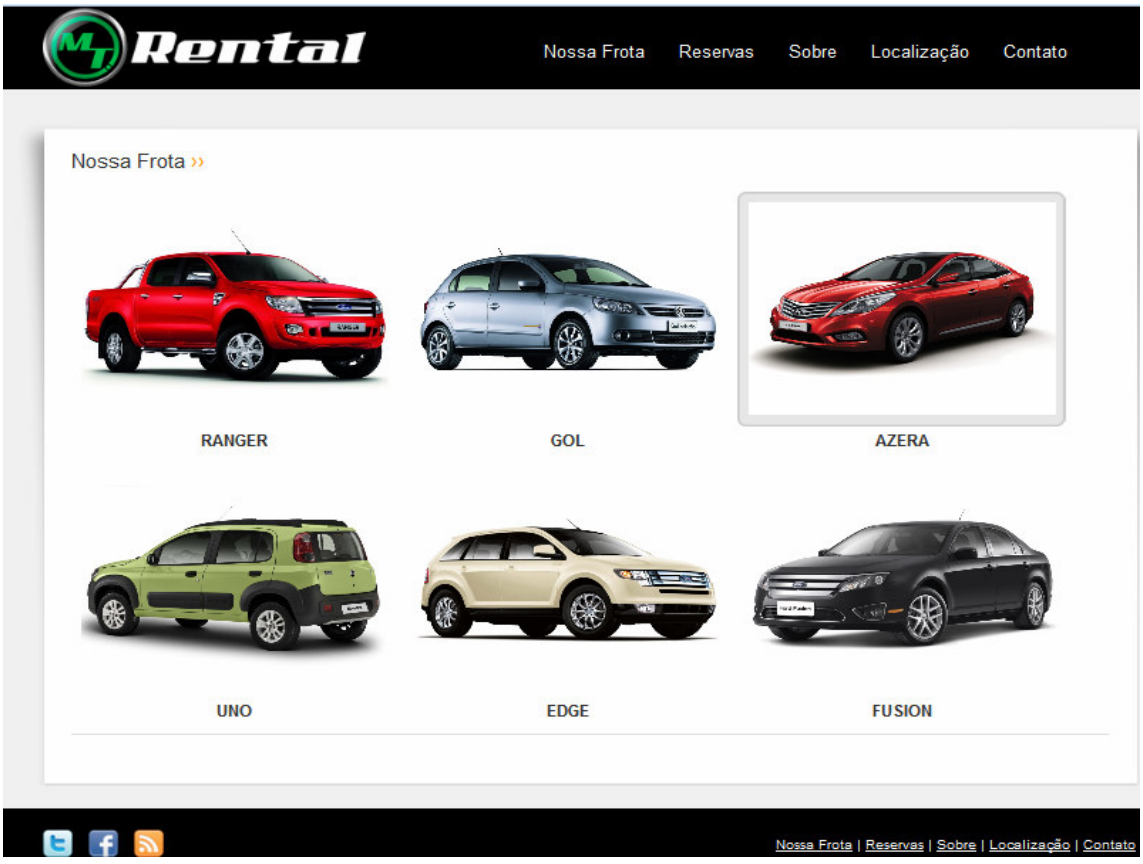

Figura 46 – Página da frota de veículos

As imagens dos veículos são carregadas diretamente do banco da aplicação *desktop*. Essas imagens são cadastradas por meio do cadastro de veículos.

A ser clicado sobre a imagem de um veículo o mesmo é apresentado na tela juntamente com algumas informações cadastrais. Essas informações são provenientes do cadastro de veículos realizado por meio da aplicação *desktop*. A Figura 47 apresenta a página com a apresentação do veículo e os seus dados cadastrais. Na parte superior da página é apresentado o nome do veículo selecionado e em uma listagem a direita, a sua marca, ano e modelo e a descrição do modelo. Essa página também apresenta um botão reservar, que abre o cadastro de reserva apresentado na Figura 48.

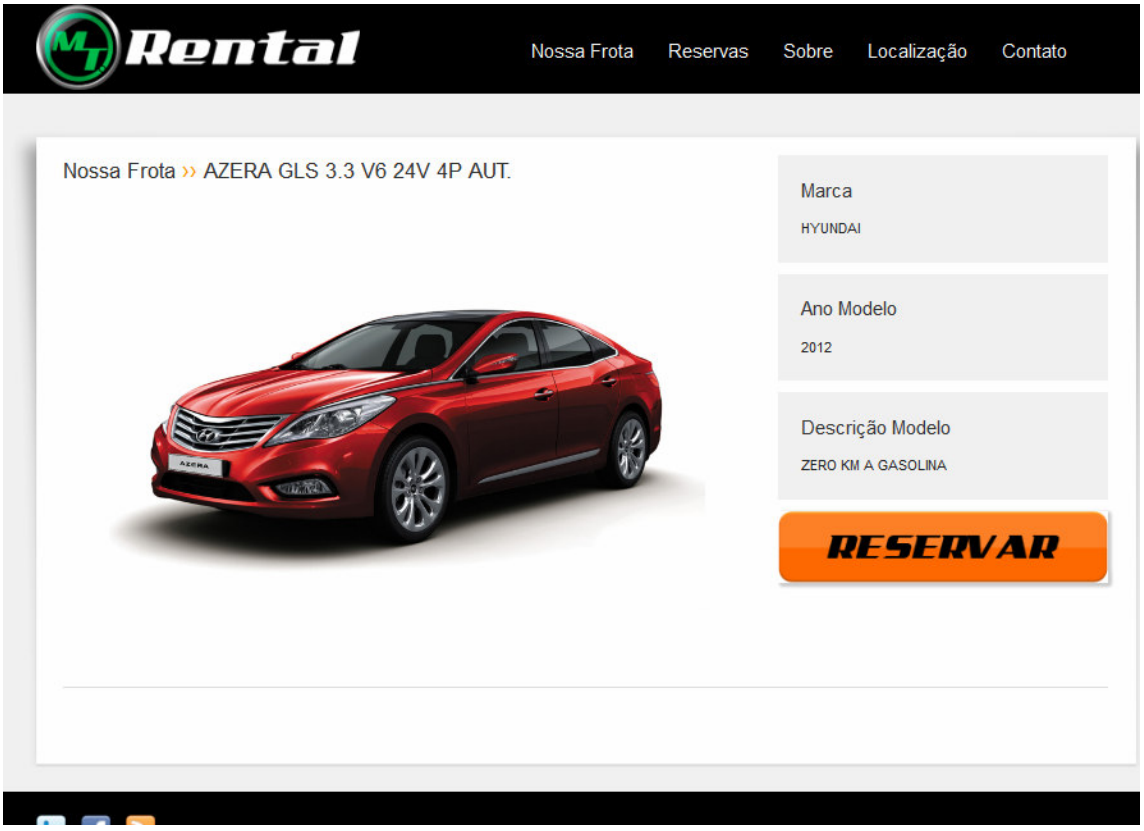

Figura 47 – Detalhes do veículo selecionado

O menu “Reservas” abre uma nova página para efetuar a reserva de um veículo. O usuário informa alguns dados pessoais, como nome e CPF, seleciona algum veículo disponível e informa a data para a qual deseja fazer a reserva. Ao salvar a reserva, caso o usuário não seja cliente, o cadastro é efetuado automaticamente junto com a reserva do veículo.

A página de cadastro para reserva do veículo é apresentada na Figura 48. Nessa tela o usuário informa o seu CPF, nome telefone e *email*. Se o usuário já está cadastrado no banco

de dados ao ser informado o CPF os demais campos são preenchidos. Caso o usuário não esteja cadastrado, verificação realizada pelo CPF, um cadastro de usuário simplificado (apenas com nome, CPF e *email*) é inserido no banco de dados no momento que o cliente clicar no botão "Reservar".

Figura 48 – Página para reserva de veículo

No menu "Sobre" é apresentado um breve histórico da empresa e também são apresentado dados referentes a empresa. Dentre esses dados estão os nomes dos proprietários, a data de fundação, o objetivo da empresa e o seu diferencial em termos de competitividade no mercado.

No menu "Localização" é apresentada uma página com um mapa indicando a localização da empresa. Também nessa mesma página há um campo para o usuário informar qual é a sua localização e um botão traçar rota, que ao ser clicado mostra o caminho que precisa ser feito para chegar até a empresa. A Figura 49 apresenta a página que permite indicar o endereço de origem e será indicada a rota para chegar até a empresa.

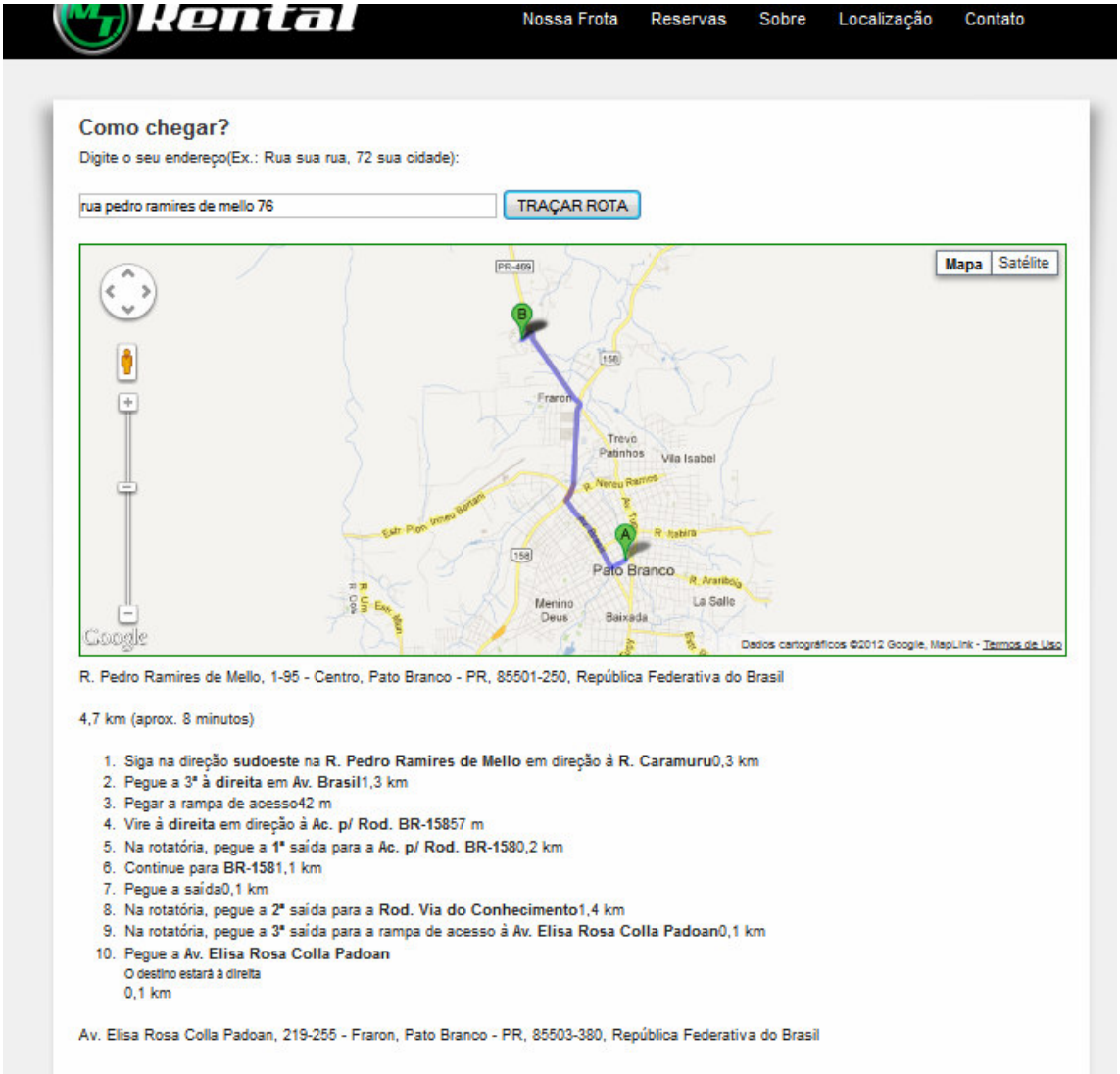

Figura 49 – Página de localização da empresa e indicação da rota

No menu “Contato”, em uma nova página é apresentado um formulário para o usuário entrar em contato com a empresa por *e-mail*. Também ficarão destacadas as informações de contato da empresa, como telefone, fax e *e-mail*.

A Figura 50 apresenta a página com o formulário de contato.

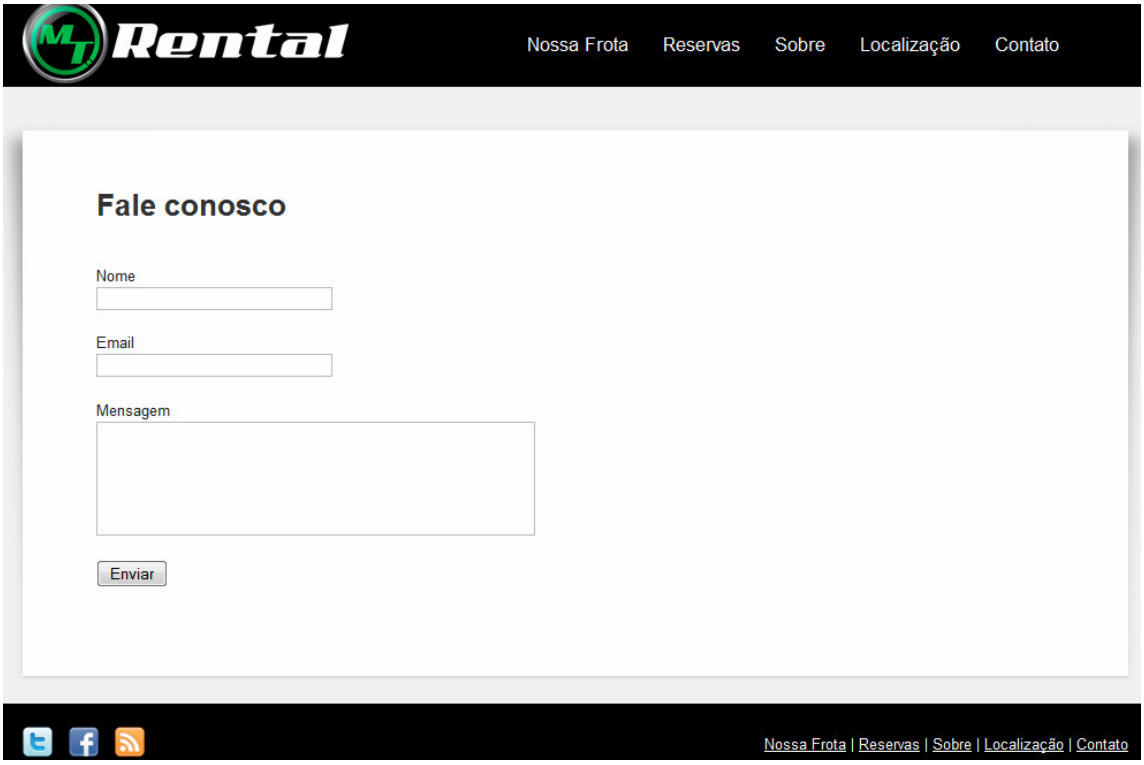

Figura 50 – Página com formulário de contato

4.4 IMPLEMENTAÇÃO DO SISTEMA

A seguir estão as listagens de código que representam exemplos das funcionalidades implementadas. Essas listagens se referem aos códigos usados para implementar funcionalidades padrão para o sistema. Elas exemplificam as operações realizadas com o sistema, incluindo cadastros, movimentação, controle de contas, dentre outras. Na Seção 4.4.1 estão os códigos relacionados à aplicação *desktop* e na Seção 4.4.2 os códigos relacionados à página *web*.

4.4.1 Aplicação Desktop

A Listagem 1 apresenta o método Cancelar, padrão em todas as telas. Esse método, inicialmente herda as funcionalidades da unfBase e verifica se o pgcPrincipal está na aba de registro. Se estiver, atribui o foco ao campo dbeNOME_COR senão o foco vai para o campo cbTipoPesq.

```
procedure TunfCoresVeiculos.bmCancelarClick(Sender: TObject);
begin
```

```
  inherited;
  if pgcPrincipal.ActivePage = tsRegistro then
    dbeNOME_COR.SetFocus
  else if pgcPrincipal.ActivePage = tsConsulta then
    cbTipoPesq.SetFocus;
end;
```

Listagem 1 –Código do evento clique do botão cancelar do Formulário de Cadastro de Cores.

Na Listagem 2 está o código para o evento onClique do botão Editar. Esse código se refere ao método Editar, que executa o que foi herdado da unfBase e atribui o foco ao campo dbeNOME_COR.

```
procedure TunfCoresVeiculos.bmEditarClick(Sender: TObject);
begin
  inherited;
  dbeNOME_COR.SetFocus;
end;
```

Listagem 2 – Código do evento onClique do botão editar do Formulário de Cadastro de Cores

A Listagem 3 apresenta o código do método de exclusão do formulário de cores de veículos. Esse método verifica se a qryPrincipal está no estado dsBrowser, ou seja, não está inserindo nem editando. E utiliza a spPrincipal, que conecta diretamente com a *procedure* LOCADORA_INS_UPD_DEL desenvolvida pelos autores deste trabalho e armazenado no banco de dados e atribui valor aos parâmetros. A ação, que é um inteiro, com valores iguais a 0, 1 e 2, que neste caso é representado pela constante C_DELETAR que tem o valor 2. Em seguida vem o nome da tabela e após o código SQL usado na cláusula *where* da exclusão. E por fim é chamado o método ExecProc, que executa a *procedure* do banco de dados. Essa *procedure* retorna o parâmetro @RETORNO. Se esse retorno for *true* é mostrada a mensagem de registro excluído com sucesso. Caso não seja *true* é apresentada uma mensagem de que ocorreu erro ao tentar excluir. Após isso é executado o que foi herdado da unfBase, e atribuído o foco ao campo dbeNOME_COR ou ao campo cbTipoPesq, de acordo com o ActivePage da pgcPrincipal. E por fim, é executado o método Requery para realizar um *refresh* na *query* e posicioná-la no último registro.

```
procedure TunfCoresVeiculos.bmExcluirClick(Sender: TObject);
begin
  if qryPrincipal.State = dsBrowse then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_DELETAR;
      Parameters.ParamByName('@TABELA').Value         := 'CORES_VEICULOS';
      Parameters.ParamByName('@WHERE_INS_DEL').Value  :=  '  ID_COR  =  '  +
qryPrincipalID_COR.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
```

```
        MessageBox(Application.Handle , 'Registro Excluído com Sucesso!' ,
'MT Rental' , MB_OK + MB_ICONINFORMATION )
      else
        MessageBox(Application.Handle  ,  'Erro  ao  Tentar  Excluír!',  'MT
Rental' , MB_OK + MB_ICONINFORMATION )
    end;
  end;
  inherited;
    if pgcPrincipal.ActivePage = tsRegistro then
    dbeNOME_COR.SetFocus
  else if pgcPrincipal.ActivePage = tsConsulta then
    cbTipoPesq.SetFocus;
  qryPrincipal.Requery;
  qryPrincipal.Last;
end;
```

Listagem 3 – Código do evento clique do botão excluir do Formulário de Cadastro de Cores

O código do método para criar um novo registro está apresentado na Listagem 4. Esse método herda as funcionalidades da unfBase e atribui o foco ao campo dbeNOME_COR.

```
procedure TunfCoresVeiculos.bmNovoClick(Sender: TObject);
begin
  inherited;
  dbeNOME_COR.SetFocus;
end;
```

Listagem 4 – Código do evento clique do botão Novo Registro do Formulário de Cadastro de Cores

Na Listagem 5 é apresentado o código do método salvar.

```
procedure TunfCoresVeiculos.bmSalvarClick(Sender: TObject);
begin
  if qryPrincipal.State = dsInsert then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_INSERIR;
      Parameters.ParamByName('@TABELA').Value         := 'CORES_VEICULOS';
      Parameters.ParamByName('@CAMPOS_INSERT').Value  := 'NOME_COR';
      Parameters.ParamByName('@VALORES_INSERT').Value                    :=
QuotedStr(qryPrincipalNOME_COR.AsString);
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        MessageBox(Application.Handle , 'Registro Inserido com Sucesso!' ,
'MT Rental' , MB_OK + MB_ICONINFORMATION )
      else
        MessageBox(Application.Handle  ,  'Erro  ao  Tentar  Inserir!',  'MT
Rental' , MB_OK + MB_ICONINFORMATION )
    end;
  end
  else if qryPrincipal.State = dsEdit then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_ALTERAR;
      Parameters.ParamByName('@TABELA').Value         := 'CORES_VEICULOS';
      Parameters.ParamByName('@VALORES_UPDATE').Value  :=  '  NOME_COR  =  '  +
QuotedStr(qryPrincipalNOME_COR.AsString);
```

```
      Parameters.ParamByName('@WHERE_INS_DEL').Value   :=  '  ID_COR  =  '  +
qryPrincipalID_COR.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        MessageBox(Application.Handle , 'Registro Alterado com Sucesso!' ,
'MT Rental' , MB_OK + MB_ICONINFORMATION )
      else
        MessageBox(Application.Handle  ,  'Erro  ao  Tentar  Alterar!',  'MT
Rental' , MB_OK + MB_ICONINFORMATION )
    end;
  end;
  dbeNOME_COR.SetFocus;
  inherited;
end;
```

Listagem 5 – Código do evento clique do botão Salvar do Formulário de Cadastro de Cores

O método salvar verifica se o *state* da qryPrincipal está como dsInsert. São atribuídos valores aos parâmetros da spPrincipal, a ação, a tabela, os campos_insert e os valores_insert. Em seguira a *procedure* é executada e o retorno é verificado.

Caso o *state* da qryPrincipal esteja como dsEdit, são atribuídos valores aos parâmetros da spPrincipal (que é o componente *stored* p*rocedure* do Delphi), a ação, a tabela, os valores_update e a where_ins_del. O retorno é verificado e as mensagens são apresentadas.

No botão Buscar (código apresentado na Listagem 6), são atribuídos valores às variáveis idCampo (nome do campo chave-primária da tabela), descriçãoCampo (nome do campo descrição) e sSQLWhereBusca que é responsável por buscar todos os registros da tabela.

```
procedure TunfCoresVeiculos.btnBuscarClick(Sender: TObject);
begin
  idCampo        := 'ID_COR';
  descricaoCampo := 'NOME_COR';
  sSQLWhereBusca := ' WHERE (1=1) ';
  inherited;
end;
```

Listagem 6 – Códig Código do evento clique do botão Buscar do Formulário de Cadastro de Cores

Na Listagem 7 está o evento onKeyDown que verifica se a tecla pressionada (*key*) é igual a vk_return, ou seja, *enter*, e então atribui o foco do sistema para o próximo campo. Esse código permite que a tecla *enter* seja considerada como a tecla de tabulação (*tab*).

```
procedure  TunfCoresVeiculos.dbeNOME_CORKeyDown(Sender:  TObject;  var  Key:
Word; Shift: TShiftState);
begin
  inherited;
  if Key = VK_RETURN then
    Perform(Wm_NextDlgCtl,0,0);
end;
```

Listagem 7 – Código do evento OnKeyDown do Campo `dbeNOME_COR` do Formulário de Cadastro de Cores

A Listagem 8 apresenta o evento onC*reate*. Esse evento atribui valor à sentença SQL e à cláusula *where* que serão usados ao abrir a tela e ao clicar no botão de pesquisar para que sejam mostrados os registros.

```
procedure TunfCoresVeiculos.FormCreate(Sender: TObject);
begin
  sSQLSelect :=
    'SELECT ID_COR,      '+
    'NOME_COR            '+
    'FROM CORES_VEICULOS ';
  sSQLWhere :=
    'WHERE ID_COR IS NULL';
  inherited;
end;
```

Listagem 8 – Código do evento onCreate do Formulário de Cadastro de Cores

No evento onDblClick do *grid*, é verificado se a origem da chamada do formulário foi o *menu*. Se sim, são atribuídos os campos código e descrição da tabela para o Record Parâmetros criado na unfBase, que é utilizado para as telas que chamam o cadastro de cor. O código desse evento é apresentado na Listagem 9.

```
procedure TunfCoresVeiculos.grdConsultaDblClick(Sender: TObject);
begin
  inherited;
  if Self.Owner.Name <> 'FPrincipal' then
  begin
    Parametros.Codigo    := qryPrincipalID_COR.AsInteger;
    Parametros.Descricao := qryPrincipalNOME_COR.AsString;
  end;
end;
```

Listagem 9 – Código do evento onDblClique do componente grdConsulta do Formulário de Cadastro de Cores

Na Listagem 10 está o código do evento que é utilizado para remover a barra de rolagem horizontal, pois em telas como esta que tem apenas dois campos mostrados no *grid*, não há necessidade de barra de rolagem horizontal.

```
procedure TunfCoresVeiculos.grdConsultaDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn; State:
TGridDrawState);
  begin
    inherited;
    ShowScrollBar(grdConsulta.Handle,SB_HORZ,False);
  end;
```

Listagem 10 – Código do Evento OnDrawColumnCell do compoente grdConsulta do Formulário de Cadastro de Cores

A Listagem 11 apresenta a *procedure* LOCADORA_INS_UPD_DEL utilizada para as operações com banco de dados descritas nas listagens de código apresentadas anteriormente.

```
Procedure no Banco de Dados, que realiza o CRUD
```

```
USE [LOCADORA]
GO
/****** Object:   StoredProcedure [dbo].[LOCADORA_INS_UPD_DEL]      Script
Date: 28/07/2012 18:22:08 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[LOCADORA_INS_UPD_DEL]
    @ACAO SMALLINT, -- 0-INSERE / 1-UPDATE / 3-DELETE
    @TABELA VARCHAR(100),
    @CAMPOS_INSERT NVARCHAR(4000) = NULL,
    @VALORES_INSERT NVARCHAR(4000) = NULL,
    @VALORES_UPDATE NVARCHAR(4000) = NULL,
    @WHERE_INS_DEL NVARCHAR(4000) = NULL,
    @ID_INSERIDO INT = 0 OUTPUT,
 @RETORNO BIT = 1 OUTPUT, -- 1-SUCESSO / 0-ERRO
 @MSG_ERRO NVARCHAR(4000) = '' OUTPUT
AS
 SET NOCOUNT ON;
 BEGIN TRY
  DECLARE @sSQL NVARCHAR(4000);
  BEGIN TRANSACTION
   SELECT @RETORNO = 1;
   IF (@ACAO = 0)
    SET  @sSQL  =  'INSERT  INTO  '+@TABELA+'  ('+  @CAMPOS_INSERT  +')
VALUES('+@VALORES_INSERT+');';
   ELSE IF (@ACAO = 1)
    SET  @sSQL  =  'UPDATE  '+@TABELA+'  SET  '+  @VALORES_UPDATE  +'  WHERE
'+@WHERE_INS_DEL+';';
   ELSE IF (@ACAO = 2)
    SET @sSQL = 'DELETE FROM '+@TABELA+' WHERE '+@WHERE_INS_DEL+';';
   EXECUTE(@sSQL)
   IF (@ACAO = 0)
    SET @ID_INSERIDO = @@IDENTITY;
   ELSE
    SET @ID_INSERIDO = 0;
  COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  -- Verifica se ocorreu erro na transação e não efetua ROLLBACK.
  IF @@TRANCOUNT > 0
   ROLLBACK TRANSACTION
  -- Retorna a informação do erro.
  SELECT @MSG_ERRO = ERROR_MESSAGE();
  SELECT @RETORNO = 0;
 END CATCH;
```

Listagem 11 – Código da procedure do banco de dados

A *procedure* LOCADORA_INS_UPD_DEL do banco de dados (Listagem 11), cria as variáveis utilizadas como parâmetros no código fonte. Essa *procedure* inicia a transação, verifica se a @Acao é Insert, Update ou Delete e monta as consultas SQL de acordo com os parâmetros passados pelas telas. Após isso executa o *commit*. Se ocorreu algum erro, executa um *rollback* e retorna a situação para a tela que fez uso dela.

A *procedure* da Listagem 12 é chamada com um clique no *grid* ou com um clique do botão Editar. Essa *procedure* atribui os dados que vem da *query* para os campos de data e o *comboBox* da tela.

```
procedure TunfLocacao.AtribuiDadosEdit;
begin
 cbTipoOperacao.ItemIndex := qryPrincipalTIPO_OPERACAO.AsInteger;
 dtDataSaida.DateTime  :=  qryPrincipalDATA_INICIAL_CONTRATO.AsDateTime;
  dtDataChegada.DateTime   := qryPrincipalDATA_FINAL_CONTRATO.AsDateTime;
  dtDataDevolucao.DateTime := qryPrincipalDATA_DEVOLUCAO_CONTRATO.AsDateTime;
end;
```

Listagem 12 – Código da procedure do evento click no grid ou botão editar da tela de Locação

Na Listagem 13 está o código da *procedure* que é chamada no clique do botão incluir. Essa *procedure* seta os dados padrão dos campos de data e *combo box*.

```
procedure TunfLocacao.AtribuiDadosInsert;
begin
  cbTipoOperacao.ItemIndex := 1;
  dtDataSaida.DateTime     := Now;
  dtDataChegada.DateTime   := Now;
  dtDataDevolucao.DateTime := Now;
end;
```

Listagem 13 – Código da procedure do evento click no botão incluir da tela de Locação

O evento onClick do botão Cancelar chama o que foi herdado da unfBase e verifica se o PageControl está posicionado na aba tsRegistro ou na tsConsulta e então atribui o foco ao campo ID_VEICULO ou cbTipoPesq (Listagem 14).

```
procedure TunfLocacao.bmCancelarClick(Sender: TObject);
begin
  inherited;
  if pgcPrincipal.ActivePage = tsRegistro then
    dbeID_VEICULO.SetFocus
  else if pgcPrincipal.ActivePage = tsConsulta then
    cbTipoPesq.SetFocus;
end;
```

Listagem 14 – Código do evento click no botão cancelar da tela de Locação

O clique no botão Editar, chama o que foi herdado da unfBase, após isso chama a *procedure* AtribuiDadosEdit e verifica se o PageControlMovimentacao está posicionado na aba tsMovimentacao para então atribuir o foco ao campo ID_VEICULO (Listagem 15).

```
procedure TunfLocacao.bmEditarClick(Sender: TObject);
begin
  inherited;
  AtribuiDadosEdit;
  if pgcMovimentacao.ActivePage <> tsMovimentacao then
    pgcMovimentacao.ActivePage := tsMovimentacao;
  dbeID_VEICULO.SetFocus;
end;
```

Listagem 15 – Código do evento click no botão editar da tela de Locação

O clique no botão Excluir verifica o resultado das funções ExcluirContrato e ExcluirCheckIn e o armazena na variável dadosExcluidos. Verifica o *state* da *query* e atribui valor aos parâmetros da *procedure* do banco de dados e a executa. Se o parâmetro retorno for verdadeiro ele verifica a variável dadosExcluidos, se ela estiver *true* será apresentada uma mensagem de sucesso ao usuário senão apresentará uma mensagem de erro. Após é chamado o que foi herdado da unfBase, verificado qual a página ativa do PageControl atribuindo assim o foco ao campo ID_VEICULO ou cbTipoPesquisa, e é usada a função *requery* para atualizar a *query* e posicionar o cursor no último registro.

```
procedure TunfLocacao.bmExcluirClick(Sender: TObject);
var
  dadosExcluidos : Boolean;
begin
  if ExcluiContrato and ExcluiCheckIn then
    dadosExcluidos := True
  else
    dadosExcluidos := False;
  if qryPrincipal.State = dsBrowse then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_DELETAR;
      Parameters.ParamByName('@TABELA').Value         := 'MOVIMENTACOES';
      Parameters.ParamByName('@WHERE_INS_DEL').Value   := '  ID_MOVIMENTACAO
= ' + qryPrincipalID_MOVIMENTACAO.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
      begin
        if dadosExcluidos then
          MessageBox(Application.Handle , 'Registro Excluido com Sucesso!'
, 'MT Rental' , MB_OK + MB_ICONINFORMATION )
      end
      else
          MessageBox(Application.Handle  ,  'Erro  ao  Tentar  Excluir!',  'MT
Rental' , MB_OK + MB_ICONINFORMATION );
      end;
    end;
  end;
  inherited;
  if pgcPrincipal.ActivePage = tsRegistro then
    dbeID_VEICULO.SetFocus
  else if pgcPrincipal.ActivePage = tsConsulta then
    cbTipoPesq.SetFocus;
  qryPrincipal.Requery;
  qryPrincipal.Last;
end;
```

Listagem 16 – Código do evento click no botão excluir da tela de Locação

O clique no botão Novo chama o que foi herdado da unfBase, chama a *procedure* AtribuiDadosInsert, verifica se o PageControlMovimentacao está posicionado na página tsMovimentacao e então atribui o foco ao campo ID_VEICULO (Listagem 17).

```
procedure TunfLocacao.bmNovoClick(Sender: TObject);
begin
  inherited;
  AtribuiDadosInsert;
  if pgcMovimentacao.ActivePage <> tsMovimentacao then
    pgcMovimentacao.ActivePage := tsMovimentacao;
  dbeID_VEICULO.SetFocus;
end;
```

Listagem 17 – Código do evento onClick no botão novo da tela de Locação

O clique no botão Salvar verifica se os dados informados são válidos, através do método ValidaDados (cujo código está na Listagem 18), atribui o retorno das funções GravaDadosCheckIn e GravaDadosContrato ao campo dadosGravados, e verifica o *state* da *query*. Se estiver como dsInsert atribui os parâmetros de *insert* da *procedure* do banco de dados e a executa. Se estiver como dsEdit, atribui os dados de edição, ao fim verifica se o parâmetro retorno e a variável dadosGravados estão como *true*. Se estiver apresenta mensagem de sucesso ao usuário, do contrario apresenta mensagem de erro.

```
procedure TunfLocacao.bmSalvarClick(Sender: TObject);
var
  dadosGravados: Boolean;
begin
  ValidaDados;
  if GravaDadosCheckIn and GravaDadosContrato then
    dadosGravados := True
  else
    dadosGravados := False;
  if qryPrincipal.State = dsInsert then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_INSERIR;
      Parameters.ParamByName('@TABELA').Value         := 'MOVIMENTACOES';
      Parameters.ParamByName('@CAMPOS_INSERT').Value  := 'ID_VEICULO,
'+ 'TIPO_OPERACAO'+'ID_CLIENTE,'+'ID_SITUACAO,'+'ID_CONTRATO,+
'DATA_MOVIMENTACAO, '+ 'KM_ENTRADA,'+'KM_SAIDA,'+'VALOR,
'+'ID_CONTA_RECEBER,  '+'LOCACAO_MANUTENCAO,';
      Parameters.ParamByName('@VALORES_INSERT').Value :=
QuotedStr(qryPrincipalID_VEICULO.AsString)            + ', ' +
QuotedStr(IntToStr(cbTipoOperacao.ItemIndex))         + ', ' +
QuotedStr(qryPrincipalID_CLIENTE.AsString)            + ', ' +
QuotedStr(qryPrincipalID_SITUACAO.AsString)           + ', ' +
QuotedStr(qryPrincipalID_CONTRATO.AsString)           + ', ' +
QuotedStr(qryPrincipalDATA_INICIAL_CONTRATO.AsString) + ', ' +
QuotedStr(qryPrincipalKM_ENTRADA.AsString)            + ', ' +
QuotedStr(qryPrincipalKM_SAIDA.AsString)              + ', ' +
QuotedStr(qryPrincipalVALOR.AsString)                 + ', ' +
QuotedStr(qryPrincipalID_CONTA_RECEBER.AsString)      + ', ' +
QuotedStr(qryPrincipalLOCACAO_MANUTENCAO.AsString);
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
      begin
        if dadosGravados then
          MessageBox(Application.Handle , 'Registro Inserido com Sucesso!'
, 'MT Rental' , MB_OK + MB_ICONINFORMATION )
```

```
        else
          MessageBox(Application.Handle , 'Erro ao Tentar Inserir!', 'MT
Rental' , MB_OK + MB_ICONINFORMATION );
      end
      else
          MessageBox(Application.Handle , 'Erro ao Tentar Inserir!', 'MT
Rental' , MB_OK + MB_ICONINFORMATION );
    end;
      if pgcMovimentacao.ActivePage <> tsMovimentacao then
        pgcMovimentacao.ActivePage := tsMovimentacao;
      dbeID_VEICULO.SetFocus;
  end
  else if qryPrincipal.State = dsEdit then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_ALTERAR;
      Parameters.ParamByName('@TABELA').Value         := 'MOVIMENTACOES';
      Parameters.ParamByName('@VALORES_UPDATE').Value :=
'ID_VEICULO =  '+ QuotedStr(qryPrincipalID_VEICULO.AsString)            +
', ' + 'TIPO_OPERACAO =      '+
QuotedStr(IntToStr(cbTipoOperacao.ItemIndex))         + ', ' +
'ID_CLIENTE =         '+ QuotedStr(qryPrincipalID_CLIENTE.AsString)
+ ', ' + 'ID_SITUACAO =        '+
QuotedStr(qryPrincipalID_SITUACAO.AsString)           + ', ' +
'ID_CONTRATO =        '+ QuotedStr(qryPrincipalID_CONTRATO.AsString)
+ ', ' + 'DATA_MOVIMENTACAO =  '+
QuotedStr(qryPrincipalDATA_INICIAL_CONTRATO.AsString) + ', ' +'KM_ENTRADA =
'+ QuotedStr(qryPrincipalKM_ENTRADA.AsString)            + ', ' +
 'KM_SAIDA =           '+ QuotedStr(qryPrincipalKM_SAIDA.AsString)
+ ', ' +  'VALOR =              '+ QuotedStr(qryPrincipalVALOR.AsString)
+ ', ' +  'ID_CONTA_RECEBER =   '+
QuotedStr(qryPrincipalID_CONTA_RECEBER.AsString)      + ', ' +
'LOCACAO_MANUTENCAO = '+
QuotedStr(qryPrincipalLOCACAO_MANUTENCAO.AsString);
     Parameters.ParamByName('@WHERE_INS_DEL').Value  := ' ID_MOVIMENTACAO =
'+ qryPrincipalID_MOVIMENTACAO.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
      begin
        if dadosGravados then
          MessageBox(Application.Handle , 'Registro Inserido com Sucesso!'
, 'MT Rental' , MB_OK + MB_ICONINFORMATION )
        else
          MessageBox(Application.Handle , 'Erro ao Tentar Inserir!', 'MT
Rental' , MB_OK + MB_ICONINFORMATION );
      end
      else
          MessageBox(Application.Handle , 'Erro ao Tentar Inserir!', 'MT
Rental' , MB_OK + MB_ICONINFORMATION );
      end;
    end;
  end;
  inherited;
end;
```

Listagem 18 – Código do evento onCllick no botão salvar da tela de Locação

O código do evento onClick do botão Buscar da aba tsConsulta do PageConrolPrincipal, que é apresentado na Listagem 19, atribui valor às variáveis idCampo

(variável responsável pelo campo chave da tabela), descriçãoCampo (responsável pelo nome ou dado mais relevante para buscas futuras) e sSQLWhereBusca. Em seguida chama o que foi herdado da unfBase.

```
procedure TunfLocacao.btnBuscarClick(Sender: TObject);
begin
  idCampo        := 'MOVIMENTACOES.ID_MOVIMENTACAO';
  descricaoCampo := 'CLIENTES.NOME_CLIENTE';
  sSQLWhereBusca := ' WHERE (1=1) ';
  inherited;
end;
```

Listagem 19 – Código do evento onClick no botão buscar da tela de Locação

O código do evento onClick do botão Formas de Pagamento, verifica se o objeto CadContasReceber está criado. Se ele não estiver o cria e então executa o método *showmodal* e ao fim libera o objeto criado da memória. O código desse botão é apresentado na Listagem 20.

```
procedure TunfLocacao.btnFormasPagtoClick(Sender: TObject);
begin
  inherited;
  if not Assigned(CadContasReceber) then
    CadContasReceber := TunfContasReceber.Create(Self);
  CadContasReceber.ShowModal;
  FreeAndNil(CadContasReceber);
end;
```

Listagem 20 – Código do evento onClick no botão forma de pagamento da tela de Locação

O evento on*Change* do campo cbTipoOperacao (Listagem 21), verifica se o itemIndex do campo cbTipoOperacao é igual a 1 (entrada) e então desabilita os campos dbeKM_ENTRADA, dtDataDevolucao e edtDiasDiferenca, do contrário habilita os mesmos.

```
procedure TunfLocacao.dbeID_CLIENTEKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  inherited;
  if Key = VK_RETURN then
    Perform(Wm_NextDlgCtl,0,0);
end;

Função executada evento onkeydown de todos os campos da tela, para que o
botão enter funcione como o tab.

procedure TunfLocacao.cbTipoOperacaoChange(Sender: TObject);
begin
  if cbTipoOperacao.ItemIndex = 1 then
  begin
    dbeKM_ENTRADA.Enabled    := False;
    dtDataDevolucao.Enabled  := False;
    edtDiasDiferenca.Enabled := False;
  end
  else
  begin
```

```
    dbeKM_ENTRADA.Enabled    := True;
    dtDataDevolucao.Enabled  := True;
    edtDiasDiferenca.Enabled := True;
  end;
end;
```

Listagem 21 - Código do evento onChange do campo cbTipoOperacao

A função ExcluirCheckIn (Listagem 22) verifica se o *state* da qryPrincipal está em dsBrowse e então atribui os parâmetros de exclusão para a *procedure* do banco de dados, e retorna o valor do parâmetro Retorno.

```
function TunfLocacao.ExcluiCheckIn: Boolean;
begin
  if qryPrincipal.State = dsBrowse then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_DELETAR;
      Parameters.ParamByName('@TABELA').Value         := 'CONTRATOS';
      Parameters.ParamByName('@WHERE_INS_DEL').Value  := ' ID_CONTRATO = '
+ qryPrincipalID_CONTRATO.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        Result := True
      else
        Result := False;
    end;
  end;
end;
```

Listagem 22 - Código da função ExcluirCheckIn da tela de Locação

O evento onCreate, atribui valor aos campos sSQLSelect e sSQLWhere, para que a qryPrincipal seja aberta (Listagem 23).

```
procedure TunfLocacao.FormCreate(Sender: TObject);
begin
  AtribuiDadosInsert;
  sSQLSelect :=
    'SELECT'+ 'MOVIMENTACOES.ID_MOVIMENTACAO,
'+
    'MOVIMENTACOES.TIPO_OPERACAO,
'+
    'MOVIMENTACOES.ID_VEICULO,
'+
    'VEICULOS.ID_MODELO_VEICULO,
'+
    'MODELOS_VEICULOS.NOME_MODELO_VEICULO,
'+
    'MOVIMENTACOES.ID_SITUACAO,
'+
    'MOVIMENTACOES.KM_ENTRADA,
'+
    'MOVIMENTACOES.KM_SAIDA,
'+
    'MOVIMENTACOES.ID_CHECK_IN,
'+
```

```
    'MOVIMENTACOES.VALOR,
'+
    'MOVIMENTACOES.ID_CONTRATO,
'+
    'CONTRATOS.DATA_INICIAL_CONTRATO,
'+
    'CONTRATOS.DATA_FINAL_CONTRATO,
'+
    'CONTRATOS.DATA_DEVOLUCAO_CONTRATO,
'+
    'MOVIMENTACOES.ID_CONTA_RECEBER,
'+
    'MOVIMENTACOES.ID_CLIENTE,
'+
    'CLIENTES.NOME_CLIENTE,
'+
    'SITUACOES.DESCRICAO_SITUACAO,
'+
    'MOVIMENTACOES.LOCACAO_MANUTENCAO
'+
    'FROM MOVIMENTACOES
'+
    'LEFT JOIN VEICULOS ON
'+
    'VEICULOS.ID_VEICULO = MOVIMENTACOES.ID_VEICULO
'+
    'LEFT JOIN MODELOS_VEICULOS ON
'+
    'MODELOS_VEICULOS.ID_MODELO_VEICULO = VEICULOS.ID_MODELO_VEICULO
'+
    'LEFT JOIN CONTRATOS ON
'+
    'CONTRATOS.ID_CONTRATO = MOVIMENTACOES.ID_CONTRATO
'+
    'LEFT JOIN CONTAS_RECEBER ON
'+
    'CONTAS_RECEBER.ID_CONTA_RECEBER = MOVIMENTACOES.ID_CONTA_RECEBER
'+
    'LEFT JOIN CLIENTES ON
'+
    'CLIENTES.ID_CLIENTE = MOVIMENTACOES.ID_CLIENTE
'+
    'LEFT JOIN SITUACOES ON
'+
    'SITUACOES.ID_SITUACAO = MOVIMENTACOES.ID_SITUACAO
';
  sSQLWhere :=
    'WHERE
'+
    'MOVIMENTACOES.ID_MOVIMENTACAO IS NULL AND
'+
    'MOVIMENTACOES.LOCACAO_MANUTENCAO = 0';
  inherited;
end;
```

Listagem 23 – Evento onCreate da tela de Locação

A função GravaDadosCheckIn (Listagem 24) verifica o *state* da *query* e se estiver como dsInsert atribui os parâmetros de *insert* para a *procedure* do banco de dados e a executa,

se estiver como dsEdit, atribui os dados de edição. Ao final verifica se o parâmetro retorno está verdadeiro e então apresenta a mensagem de sucesso ao usuário, caso contrário apresenta mensagem de erro.

```
function TunfLocacao.GravaDadosCheckIn: Boolean;
begin
  if qryPrincipal.State = dsInsert then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_INSERIR;
      Parameters.ParamByName('@TABELA').Value         := 'CHECK_IN';
      Parameters.ParamByName('@CAMPOS_INSERT').Value  :=
'PARABRISA_DIANTEIRO,      '+
'PARABRISA_TRASEIRO,       '+
'VIDRO_DIANTEIRO_ESQUERDO, '+
'VIDRO_DIANTEIRO_DIREITO,  '+
'VIDRO_TRASEIRO_ESQUERDO,  '+
'VIDRO_TRASEIRO_DIREITO,   '+
'PORTA_DIANTEIRA_ESQUERDA, '+
'PORTA_DIANTEIRA_DIREITA,  '+
'PORTA_TRASEIRA_ESQUERDA,  '+
'PORTA_TRASEIRA_DIREITA,   '+
'PARA_CHOQUE_DIANTEIRO,    '+
'PARA_CHOQUE_TRASEIRO,     '+
'RODA_DIANTEIRA_ESQUERDA,  '+
'RODA_DIANTEIRA_DIREITA,   '+
'RODA_TRASEIRA_ESQUERDA,   '+
'RODA_TRASEIRA_DIREITA,    '+
'PNEU_DIANTEIRO_ESQUERDO,  '+
'PNEU_DIANTEIRO_DIREITO,   '+
'PNEU_TRASEIRO_ESQUERDO,   '+
'PNEU_TRASEIRO_DIREITO,    '+
'RETROVISOR_ESQUEDO,       '+
'RETROVISOR_DIREITO,       '+
'FAROL_DIANTEIRO_DIREITO,  '+
'FAROL_DIANTEIRO_ESQUERDO, '+
'FAROL_TRASEIRO_DIREITO,   '+
'FAROL_TRASEIRO_ESQUERDO,  '+
'CAPO,                     '+
'BAGAGEIRO                 ';
      Parameters.ParamByName('@VALORES_INSERT').Value :=
QuotedStr(qryCheckInPARABRISA_DIANTEIRO.AsString)      + ', '+
QuotedStr(qryCheckInPARABRISA_TRASEIRO.AsString)       + ', '+
QuotedStr(qryCheckInVIDRO_DIANTEIRO_ESQUERDO.AsString) + ', '+
QuotedStr(qryCheckInVIDRO_DIANTEIRO_DIREITO.AsString)  + ', '+
QuotedStr(qryCheckInVIDRO_TRASEIRO_ESQUERDO.AsString)  + ', '+
QuotedStr(qryCheckInVIDRO_TRASEIRO_DIREITO.AsString)   + ', '+
QuotedStr(qryCheckInPORTA_DIANTEIRA_ESQUERDA.AsString) + ', '+
QuotedStr(qryCheckInPORTA_DIANTEIRA_DIREITA.AsString)  + ', '+
QuotedStr(qryCheckInPORTA_TRASEIRA_ESQUERDA.AsString)  + ', '+
QuotedStr(qryCheckInPORTA_TRASEIRA_DIREITA.AsString)   + ', '+
QuotedStr(qryCheckInPARA_CHOQUE_DIANTEIRO.AsString)    + ', '+
QuotedStr(qryCheckInPARA_CHOQUE_TRASEIRO.AsString)     + ', '+
QuotedStr(qryCheckInRODA_DIANTEIRA_ESQUERDA.AsString)  + ', '+
QuotedStr(qryCheckInRODA_DIANTEIRA_DIREITA.AsString)   + ', '+
QuotedStr(qryCheckInRODA_TRASEIRA_ESQUERDA.AsString)   + ', '+
QuotedStr(qryCheckInRODA_TRASEIRA_DIREITA.AsString)    + ', '+
QuotedStr(qryCheckInPNEU_DIANTEIRO_ESQUERDO.AsString)  + ', '+
QuotedStr(qryCheckInPNEU_DIANTEIRO_DIREITO.AsString)   + ', '+
```

```
QuotedStr(qryCheckInPNEU_TRASEIRO_ESQUERDO.AsString)   + ', '+
QuotedStr(qryCheckInPNEU_TRASEIRO_DIREITO.AsString)    + ', '+
QuotedStr(qryCheckInRETROVISOR_ESQUEDO.AsString)       + ', '+
QuotedStr(qryCheckInRETROVISOR_DIREITO.AsString)       + ', '+
QuotedStr(qryCheckInFAROL_DIANTEIRO_DIREITO.AsString)  + ', '+
QuotedStr(qryCheckInFAROL_DIANTEIRO_ESQUERDO.AsString) + ', '+
QuotedStr(qryCheckInFAROL_TRASEIRO_DIREITO.AsString)   + ', '+
QuotedStr(qryCheckInFAROL_TRASEIRO_ESQUERDO.AsString)  + ', '+
QuotedStr(qryCheckInCAPO.AsString)                     + ', '+
QuotedStr(qryCheckInBAGAGEIRO.AsString);
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        Result := True
      else
        Result := False;
    end;
  end
  else if qryPrincipal.State = dsEdit then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_ALTERAR;
      Parameters.ParamByName('@TABELA').Value         := 'CHECK_IN';
      Parameters.ParamByName('@VALORES_UPDATE').Value :=
'PARABRISA_DIANTEIRO =      '+
QuotedStr(qryCheckInPARABRISA_DIANTEIRO.AsString)      + ', '+
'PARABRISA_TRASEIRO =       '+
QuotedStr(qryCheckInPARABRISA_TRASEIRO.AsString)       + ', '+
'VIDRO_DIANTEIRO_ESQUERDO = '+
QuotedStr(qryCheckInVIDRO_DIANTEIRO_ESQUERDO.AsString) + ', '+
'VIDRO_DIANTEIRO_DIREITO =  '+
QuotedStr(qryCheckInVIDRO_DIANTEIRO_DIREITO.AsString)  + ', '+
'VIDRO_TRASEIRO_ESQUERDO =  '+
QuotedStr(qryCheckInVIDRO_TRASEIRO_ESQUERDO.AsString)  + ', '+
'VIDRO_TRASEIRO_DIREITO =   '+
QuotedStr(qryCheckInVIDRO_TRASEIRO_DIREITO.AsString)   + ', '+
'PORTA_DIANTEIRA_ESQUERDA = '+
QuotedStr(qryCheckInPORTA_DIANTEIRA_ESQUERDA.AsString) + ', '+
'PORTA_DIANTEIRA_DIREITA =  '+
QuotedStr(qryCheckInPORTA_DIANTEIRA_DIREITA.AsString)  + ', '+
'PORTA_TRASEIRA_ESQUERDA =  '+
QuotedStr(qryCheckInPORTA_TRASEIRA_ESQUERDA.AsString)  + ', '+
'PORTA_TRASEIRA_DIREITA =   '+
QuotedStr(qryCheckInPORTA_TRASEIRA_DIREITA.AsString)   + ', '+
'PARA_CHOQUE_DIANTEIRO =    '+
QuotedStr(qryCheckInPARA_CHOQUE_DIANTEIRO.AsString)    + ', '+
'PARA_CHOQUE_TRASEIRO =     '+
QuotedStr(qryCheckInPARA_CHOQUE_TRASEIRO.AsString)     + ', '+
'RODA_DIANTEIRA_ESQUERDA =  '+
QuotedStr(qryCheckInRODA_DIANTEIRA_ESQUERDA.AsString)  + ', '+
'RODA_DIANTEIRA_DIREITA =   '+
QuotedStr(qryCheckInRODA_DIANTEIRA_DIREITA.AsString)   + ', '+
'RODA_TRASEIRA_ESQUERDA =   '+
QuotedStr(qryCheckInRODA_TRASEIRA_ESQUERDA.AsString)   + ', '+
'RODA_TRASEIRA_DIREITA =    '+
QuotedStr(qryCheckInRODA_TRASEIRA_DIREITA.AsString)    + ', '+
'PNEU_DIANTEIRO_ESQUERDO =  '+
QuotedStr(qryCheckInPNEU_DIANTEIRO_ESQUERDO.AsString)  + ', '+
'PNEU_DIANTEIRO_DIREITO =   '+
QuotedStr(qryCheckInPNEU_DIANTEIRO_DIREITO.AsString)   + ', '+
```

```
'PNEU_TRASEIRO_ESQUERDO =   '+
QuotedStr(qryCheckInPNEU_TRASEIRO_ESQUERDO.AsString)   + ', '+
'PNEU_TRASEIRO_DIREITO =    '+
QuotedStr(qryCheckInPNEU_TRASEIRO_DIREITO.AsString)    + ', '+
'RETROVISOR_ESQUEDO =       '+
QuotedStr(qryCheckInRETROVISOR_ESQUEDO.AsString)       + ', '+
'RETROVISOR_DIREITO =       '+
QuotedStr(qryCheckInRETROVISOR_DIREITO.AsString)       + ', '+
'FAROL_DIANTEIRO_DIREITO =  '+
QuotedStr(qryCheckInFAROL_DIANTEIRO_DIREITO.AsString)  + ', '+
'FAROL_DIANTEIRO_ESQUERDO = '+
QuotedStr(qryCheckInFAROL_DIANTEIRO_ESQUERDO.AsString) + ', '+
'FAROL_TRASEIRO_DIREITO =   '+
QuotedStr(qryCheckInFAROL_TRASEIRO_DIREITO.AsString)   + ', '+
'FAROL_TRASEIRO_ESQUERDO =  '+
QuotedStr(qryCheckInFAROL_TRASEIRO_ESQUERDO.AsString)  + ', '+
'CAPO ='+ QuotedStr(qryCheckInCAPO.AsString) + ', '+
'BAGAGEIRO = '+ QuotedStr(qryCheckInBAGAGEIRO.AsString);
      Parameters.ParamByName('@WHERE_INS_DEL').Value  := ' ID_CHECK_IN =
'+ qryPrincipalID_CHECK_IN.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        Result := True
      else
        Result := False;
    end;
  end;
end;
```

Listagem 24 – Função GravaDadosCheckIn da tela de Locação

A função GravaDadosContrato (Listagem 25) verifica o *state* da *query* e se estiver como dsInsert atribui os parâmetros de *insert* da *procedure* para o banco de dados e a executa, se estiver como dsEdit, atribui os dados de edição. Ao fim verifica se o parâmetro retorno esta verdadeiro então apresenta mensagem de sucesso ao usuário, do contrário apresenta mensagem de erro.

```
function TunfLocacao.GravaDadosContrato: Boolean;
begin
  if qryPrincipal.State = dsInsert then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_INSERIR;
      Parameters.ParamByName('@TABELA').Value         := 'CONTRATOS';
      Parameters.ParamByName('@CAMPOS_INSERT').Value  :=
'DATA_INICIAL_CONTRATO, '+
'DATA_FINAL_CONTRATO,   '+
'DATA_DEVOLUCAO_CONTRATO';
      Parameters.ParamByName('@VALORES_INSERT').Value :=
QuotedStr(qryPrincipalDATA_INICIAL_CONTRATO.AsString)   + ', ' +
QuotedStr(qryPrincipalDATA_FINAL_CONTRATO.AsString)     + ', ' +
QuotedStr(qryPrincipalDATA_DEVOLUCAO_CONTRATO.AsString);
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        Result := True
      else
```

```
        Result := False;
    end;
  end
  else if qryPrincipal.State = dsEdit then
  begin
    with spPrincipal do
    begin
      Parameters.ParamByName('@ACAO').Value           := C_ALTERAR;
      Parameters.ParamByName('@TABELA').Value         := 'CONTRATOS';
      Parameters.ParamByName('@VALORES_UPDATE').Value :=
'DATA_INICIAL_CONTRATO =    '+
QuotedStr(qryPrincipalDATA_INICIAL_CONTRATO.AsString)   + ', ' +
                                                          '
DATA_FINAL_CONTRATO =     '+
QuotedStr(qryPrincipalDATA_FINAL_CONTRATO.AsString)     + ', ' +
                                                          '
DATA_DEVOLUCAO_CONTRATO = '+
QuotedStr(qryPrincipalDATA_DEVOLUCAO_CONTRATO.AsString);
      Parameters.ParamByName('@WHERE_INS_DEL').Value  := ' ID_CONTRATO =
'+ qryPrincipalID_CONTRATO.AsString;
      ExecProc;
      if Parameters.ParamByName('@RETORNO').Value then
        Result := True
      else
        Result := False;
    end;
  end;
end;
```

Listagem 25 – Função GravaDadosContrato da tela de Locação

O evento onDblClick do *grid* chama a *procedure* atribuiDadosEdit. O código desse evento está na Listagem 26.

```
procedure TunfLocacao.grdConsultaDblClick(Sender: TObject);
begin
  AtribuiDadosEdit;
  inherited;
end;
```

Listagem 26 – Evento onDblClick do grid da tela de Locação

O evento DrawColumnCell do *grid* faz com que seja apresentada a barra de *scroll* horizontal (Listagem 27).

```
procedure TunfLocacao.grdConsultaDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn; State:
TGridDrawState);
begin
  inherited;
  ShowScrollBar(grdConsulta.Handle,SB_HORZ,False);
end;
```

Listagem 27 – Evento onDrawColumnCell do grid da tela de Locação

O evento onChange do campo ID_VEICULO (Listagem 28) verifica se os campos ID_VEICULO e NOME_MODELO_VEICULO da qryPrincipal estão iguais a zero e vazio

respectivamente, então chama o clique do speedbutton sbVeiculos. Senão verifica se o campo ID_VEICULO está igual a zero e o campo NOME_MODELO_VEICULO está diferente de vazio, então é retirado o evento on*Change* do campo ID_VEICULO para que não entre em um *loop* infinito. São limpadas as informações dos campos ID_VEICULO e NOME_MODELO_VEICULO e por fim é devolvido o evento on*Change* do campo ID_VEICULO, do contrário é chamado o onClick do botão sbVeiculoClick(nil) passando o valor *nil* como *sender*.

```
procedure TunfLocacao.qryPrincipalID_VEICULOChange(Sender: TField);
begin
  inherited;
  if (qryPrincipalID_VEICULO.AsInteger = 0) and
     (qryPrincipalNOME_MODELO_VEICULO.AsString = '') then
    sbVeiculo.Click
  else if (qryPrincipalID_VEICULO.AsInteger = 0) and
          (qryPrincipalNOME_MODELO_VEICULO.AsString <> '') then
  begin
    try
      qryPrincipalID_VEICULO.OnChange  := nil;

      qryPrincipalID_VEICULO.Clear;
      qryPrincipalNOME_MODELO_VEICULO.Clear;

    finally
       qryPrincipalID_VEICULO.OnChange  := qryPrincipalID_VEICULOChange;
    end;
  end
  else
    sbVeiculoClick(nil);
end;
```

Listagem 28 – Evento onChange do campo Id_Veiculo da tela de Locação

A função que realiza a chamada do cadastro de veículos para que o usuário possa selecionar o veiculo que desejar é apresentada na Listagem 29. O usuário pode clicar no botão de busca para que a tela seja aberta ou informar o código do veículo, para que os campos de código e descrição sejam valorizados automaticamente. Caso seja informado zero ou algum número que não exista no cadastro a ação será a mesma como se tivesse clicado no botão de busca. O mesmo ocorre para o evento on*Change* dos campos ID_CLIENTE e ID_SITUACAO.

```
procedure TunfLocacao.sbVeiculoClick(Sender: TObject);
begin
  inherited;
  if not Assigned(CadVeiculo) then
    CadVeiculo := TunfVeiculo.Create(Self);
  if Sender <> nil then
  begin
    CadVeiculo.ShowModal;
    if CadVeiculo.ModalResult = mrOk then
    begin
```

```
      try
        qryPrincipalID_VEICULO.OnChange   := nil;
        if not (qryPrincipal.State in [dsInsert, dsEdit]) then
          qryPrincipal.Edit;
        qryPrincipalID_VEICULO.AsInteger         :=
CadVeiculo.Parametros.Codigo;
        qryPrincipalNOME_MODELO_VEICULO.AsString :=
CadVeiculo.Parametros.Descricao;
        dbeID_CLIENTE.SetFocus;
      finally
        qryPrincipalID_VEICULO.OnChange          :=
qryPrincipalID_VEICULOChange;
      end;
    end
    else
    begin
      try
        qryPrincipalID_VEICULO.OnChange   := nil;
        if not (qryPrincipal.State in [dsInsert, dsEdit]) then
          qryPrincipal.Edit;
        qryPrincipalID_VEICULO.Clear;
        qryPrincipalNOME_MODELO_VEICULO.Clear;
        dbeID_VEICULO.SetFocus;
      finally
        qryPrincipalID_VEICULO.OnChange   := qryPrincipalID_VEICULOChange;
      end;
    end;
  end
  else
  begin
    CadVeiculo.btnBuscarClick(nil);
    if
CadVeiculo.qryPrincipal.Locate('ID_VEICULO',qryPrincipalID_VEICULO.AsIntege
r,[]) then
    begin
      try
        qryPrincipalID_VEICULO.OnChange          := nil;
        qryPrincipalID_VEICULO.AsInteger         :=
CadVeiculo.qryPrincipalID_MARCA_VEICULO.AsInteger;
        qryPrincipalNOME_MODELO_VEICULO.AsString :=
CadVeiculo.qryPrincipalNOME_MARCA_VEICULO.AsString;
        dbeID_CLIENTE.SetFocus;
      finally
        qryPrincipalID_VEICULO.OnChange          :=
qryPrincipalID_VEICULOChange;
      end;
    end
    else
    begin
      try
        qryPrincipalID_VEICULO.OnChange   := nil;
        if not (qryPrincipal.State in [dsInsert, dsEdit]) then
        begin
          qryPrincipal.Edit;
          qryPrincipalID_VEICULO.Clear;
          qryPrincipalNOME_MODELO_VEICULO.Clear;
        end
        else
          sbVeiculo.Click;
        dbeID_CLIENTE.SetFocus;
      finally
```

```
        qryPrincipalID_VEICULO.OnChange   := qryPrincipalID_VEICULOChange;
      end;
    end;
  end;
  if Assigned(CadVeiculo) then
    FreeAndNil(CadVeiculo);
end;
```

Listagem 28 – Função para a chamada do cadastro de veículos do cadastro de Locação

A função que recebe por parâmetro uma *string* com o nome do campo que vem do banco de dados, em que Tabsheet do Pagecontrol o referido campo se encontra e a qual campo da tela ele pertence. Porém, esse campo deve ser herança da TWinControl, apresentada na Listagem 29. Essa função verifica se o campo informado é igual a vazio. Caso seja, é exibida uma mensagem e atribuído foco ao campo.

```
procedure TunfLocacao.ValidaCampo(vNomeCampo: String; vTabSheet: TTabSheet;
vTField: TWinControl);
begin
  if qryPrincipal.FieldByName(vNomeCampo).AsString = '' then
  begin
    ShowMessage('O campo "' +
qryPrincipal.FieldByName(vNomeCampo).DisplayLabel + '" precisa ser
preenchido!');
    if pgcClientes.ActivePage <> vTabSheet then
      pgcClientes.ActivePage := vTabSheet;
    vTField.SetFocus;
    Abort;
  end;
end;
```

Listagem 29 – Função para validação de campos obrigatórios da tela de Locação

4.4.2 Aplicação Web

A Figura 51 apresenta a organização do projeto em PHP. O código PHP foi desenvolvido utilizando a IDE NetBeans. Nessa organização do projeto é possível identificar a organização do projeto utilizando MVC por meio das pastas "*controller*", "*model*" e "*view*". Outras pastas também foram criadas para organizar o projeto, como, por exemplo, para armazenar os arquivos de configuração das páginas que estão em "*css*" e as imagens que são armazenada em "*images*".

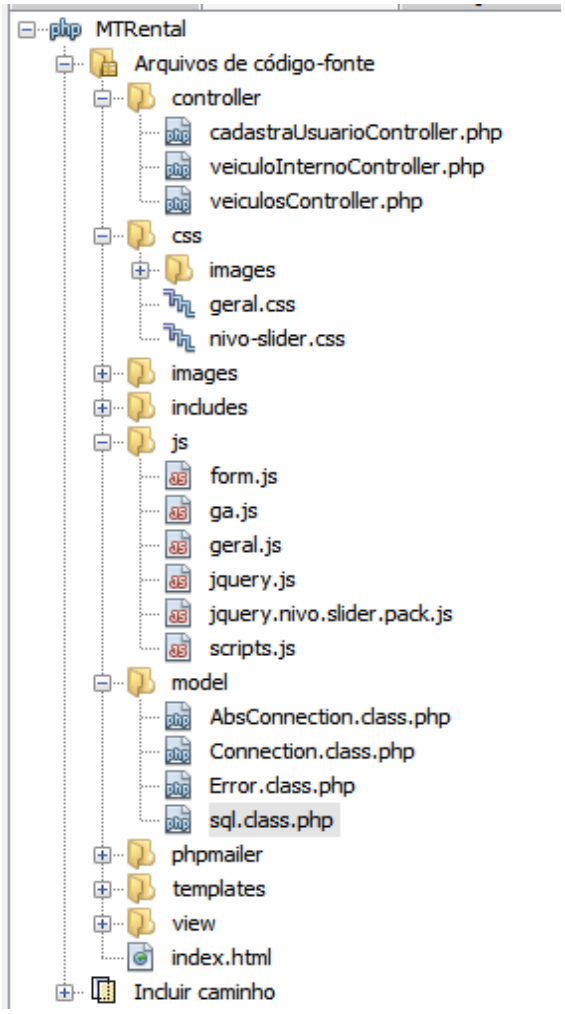

Figura 51 – Organização do projeto PHP na IDE NetBeans

O código da parte do *controller* da página de reserva, apresentado na Listagem 30, verifica se o método chamado pela página é *get* ou *post*. Caso seja *get* é selecionado o nome do modelo do veículo escolhido para ser mostrado na página de Reserva. Se o método utilizado para chamar a página é *post*, significa que foram enviados dados do formulário de reserva. E, assim, os dados enviados são armazenados em variáveis para, de acordo com o cadastro do Cliente, inserir um novo cliente ou utilizar o cliente já existente e inserir a reserva do veículo.

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {

    $id_veiculo = anti_injection($_GET['id_veiculo']);

    $conn = new db();

    $conn->connect();
    $resVeiculos = $conn->exec(selectVeiculo($id_veiculo));
```

```
    $modelo_veiculo = mssql_result($resVeiculos, 0, 'NOME_MODELO_VEICULO');
} else if ($_SERVER['REQUEST_METHOD'] == 'POST') {

    $id_veiculo = anti_injection($_POST['id_veiculo']);
    $id_cliente = anti_injection($_POST['id_cliente']);
    $nome       = strtoupper(anti_injection($_POST['nome']));
    $cpf        = preg_replace("/[^0-9]/", "",
anti_injection($_POST['cpf']));
    $email      = anti_injection($_POST['email']);
    $telefone   = preg_replace("/[^0-9]/", "",
anti_injection($_POST['telefone']));

    $conn = new db();

    $conn->connect();

    if (($id_cliente == '') && ($cpf != '')){
        $id_cliente = $conn->execLastID(cadastraClienteSimplificado($cpf,
$nome, 0, $telefone, $email));
    }
    if (($id_cliente != '') && ($id_veiculo != '')){
        $id_movimentacao = $conn->execLastID(cadastraReserva($id_veiculo,
$id_cliente));
    }

}
```

Listagem 30 – Código do arquivo de controller da página de Reserva de Veículos.

O código apresentado na Listagem 31 se refere ao *controller* da página de Contato que define a configuração necessária para que o *email* de contato seja enviado ao proprietário da empresa. O comando "$mail->Send()" envia o *email* de acordo com as configurações e retorna "True" se foi enviado com sucesso.

```
    // Setando o conteudo
        $mail->IsHTML(true);
        $mail->Subject = 'Contato pelo Site MT Rental';
        $mail->Body     = "<p>Nome: $nome</p><p>Mensagem: $msm</p>";
        $mail->AltBody = 'Conteudo sem HTML para editores que não suportam,
sim, existem alguns';

        // Validando a autenticação
        $mail->IsSMTP();
        $mail->SMTPAuth = true;
        $mail->Host     = "ssl://smtp.googlemail.com";
        $mail->Port     = 465;
        $mail->Username = 'thiagocr.utfpr@gmail.com';
        $mail->Password = 't20c3r18';
        //$mail->Username = 'bertan.marlon@gmail.com';
        //$mail->Password = 'akroma666';

        // Setando o endereço de recebimento
        $mail->AddAddress('thiagocr.utfpr@gmail.com', 'Nome');

        // Enviando o e-mail
        if( $mail->Send() )
            $enviado = 'Email enviado com sucesso!';
```

Listagem 31 – Código do arquivo de controller da página de Contato.

O arquivo *controller* da página que mostra os detalhes do veículo e esse está ou não disponível para reserva (Listagem 32) executa uma função que permite utilizar classes de outros arquivos. Essa função "require_once" está no início da listagem, logo em seguida é atribuído o valor à variável que armazena o "id" do veiculo. E caso o método utilizado para chamar a página seja *get* é executado um comando para selecionar os dados do veículo e armazenar em uma variável do tipo *array*.

```
<?php

require_once (ARQ_SEGURANCA);
require_once (DIR_MODEL . 'Connection.class.php');
require_once (ARQ_SQL);

 $id = anti_injection($_GET['id_veiculo']);

$conn = new db();

if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    try {
            $conn->connect();
            $resVeiculos = $conn->exec(selectVeiculo($id));
    } catch(Exception $e) {
            print_r($e);
    }

}
?>
```

Listagem 32 – Código do arquivo de controller da página de detalhes do veículo.

O trecho de código apresentado na Listagem 33 é executado sempre ao sair do campo CPF ao efetuar a reserva do veículo na página de reserva de veículos. Ao sair do campo CPF é executada, via Ajax, com ajuda da biblioteca JQuery, a função "load()" que é responsável por chamar o código apresentado nessa listagem sem alterar ou modificar a página de reserva. O código por sua vez recebe por parâmetro via método *get* o CPF informado na página de reserva, faz uma busca pelo CPF no banco de dados e caso encontre o CPF retorna o nome, telefone e e-mail do cliente preenchendo automaticamente os respectivos campos na página de reserva de veículos.

```
<?php
include '../includes/config.php';
require_once (ARQ_SEGURANCA);
require_once (DIR_MODEL . 'Connection.class.php');
require_once (ARQ_SQL);

if ($_SERVER['REQUEST_METHOD'] == 'GET') {

     $cpf = preg_replace("/[^0-9]/", "", anti_injection($_GET['cpf']));

    $conn = new db();
```

```
    $conn->connect();
    $resCliente = $conn->exec(selectCliente($cpf));
      die($cpf);
    if (mssql_num_rows($resCliente) > 0){

        $id_cliente = mssql_result($resCliente, 0, 'ID_CLIENTE');
        $nome       = mssql_result($resCliente, 0, 'NOME_CLIENTE');
        $telefone   = mssql_result($resCliente, 0, 'TELEFONE_CLIENTE');
        $email      = mssql_result($resCliente, 0, 'EMAIL_CLIENTE');

        echo "$id_cliente&&$nome&&$telefone&&$email";
    }else{
        echo "0";
    }
}
?>
```

Listagem 33 – Código do arquivo de controller para validar a existência do CPF do usuário no banco de dados.

Na Listagem 34 está a parte inicial do *plugin* para a biblioteca JavasSript JQuery que é responsável por aplicar máscaras nos campos como CPF, telefone e *email*. Percebe-se que no início do código existe um comentário sobre o desenvolvedor do *plugin*, versão e tipo de licença.

```
/*
      Masked Input plugin for jQuery
      Copyright (c) 2007-2009 Josh Bush (digitalbush.com)
      Licensed under the MIT license
(http://digitalbush.com/projects/masked-input-plugin/#license)
      Version: 1.2.2 (03/09/2009 22:39:06)
*/
(function($) {
      var pasteEventName = ($.browser.msie ? 'paste' : 'input') + ".mask";
      var iPhone = (window.orientation != undefined);

      $.mask = {
            //Predefined character definitions
            definitions: {
                  '9': "[0-9]",
                  'a': "[A-Za-z]",
                  '*': "[A-Za-z0-9]"
            }
      };
```

Listagem 34 – Parte do código do Plugin de Mascaras Javascript.

O código responsável por gerar o mapa da localização e traçar a rota de determinado ponto de partida até a locadora é apresentado na Listagem 35. Esse o código em JavaScript utiliza a biblioteca JQuery e foi desenvolvido manualmente utilizando outro *plugin* disponibilizado pelo Google Maps. Primeiramente são valoradas as variáveis, em seguida, via JQuery está o código que é responsável por sempre que ler o documento (sempre que for

aberta a página) executar a função que carrega o mapa e sempre no clique do botão “Traçar Rota” é chamada um função para traçar a rota entre Origem e Destino.

```
var endereco;
var map, destino;
var localAtual = 1;

$(document).ready(function () {

    carregaMapa('utfpr pato branco');

    // Mostra dados
    $('#tracarRota').click(function(){
      mostra_evento();
    });
});

function carregaMapa(cidade){

    endereco = cidade;

    var geocoder = new google.maps.Geocoder();
    var geo_options = {
        address: endereco
    };

    geocoder.geocode(geo_options, function (results, status) {
        if (status == google.maps.GeocoderStatus.OK) {
            destino = results[0].geometry.location;
            var myOptions = {
                zoom: 16,
                center: results[0].geometry.location,
                mapTypeId: google.maps.MapTypeId.ROADMAP
            };

            map = new
google.maps.Map(document.getElementById('mapa_evento'), myOptions);

            var marker = new google.maps.Marker({
                map: map,
                position: results[0].geometry.location,
                title: 'Endereço da Locadora'
            });
        } else {
            alert("Dados passados inválidos: " + status);
        }
    });

}
```

Listagem 35 – Parte do código javascript responsável por mostrar o Mapa e Traçar a Rota.

A principal classe *php* situada na estrutura pertencente ao *model*, a classe “db” apresentada na Listagem 36, é responsável por fazer a conexão com o banco de dados e executar as *querys* (SQLs) que fazem a seleção, alteração, exclusão e inclusão dos dados no banco de dados. Na classe construtora são atribuídos valores às variáveis de que são utilizadas como parâmetros da conexão ao banco de dados. Logo depois está uma função pública

responsável por fazer a conexão com o banco de dados utilizando os parâmetros que foram atribuídos valores na construção do método. Já as funções “exec” e “execLastID” são responsáveis por executar *querys* no banco de dados. A diferença entre elas é que a última executa um *insert* retornando o último ID inserido.

```
class db {

    //parâmetros de conexão
   var $db;
   var $host;
   var $usuario;
   var $senha;

   var $conn;

   var $result;
   var $numReg;

   function db() {
        $varDb = "LOCADORA";
        $varHost = ".\SQLEXPRESS";
        $varUsuario = "sa";
        $varSenha = "t20c3r18";

        $this->db = $varDb;
        $this->host = $varHost;
        $this->usuario = $varUsuario;
        $this->senha = $varSenha;
    }

    public function connect(){
        $this->conn = mssql_connect($this->host, $this->usuario, $this-
>senha) or die("Erro ao conectar a base"."<br>".mssql_get_last_message());
        if ($this->conn) {
            mssql_select_db($this->db, $this->conn);
        }
    }

    public static function exec($sql) {
        $res = mssql_query($sql) or die('Erro: '.$sql.' --
'.mssql_get_last_message());
        if (!$res) {
            return false;
        } else {
            if (substr($sql, 0, 6) == "INSERT") {
                return '1';
            } else {
                return $res;
            }
        }
    }

    public function execLastID($sql) {
        $res = mssql_query($sql) or die('Erro: ' . $sql . ' -- ' .
mssql_get_last_message());
        $res = mssql_query("SELECT @@IDENTITY") or die('Erro: ' . $sql . '
-- ' . mssql_get_last_message());
        if (!$res) {
            return false;
```

```
        } else {
            return mssql_result($res, 0, 0);
        }

    }
```

Listagem 36 – Parte do código php da Classe "DB"

Também localizada na parte *model* da estrutura da página web está o trecho de código apresentado na Listagem 37 que pertence ao arquivo "sql.class.php". Esse arquivo é responsável por armazenar todas as instruções SQLs que são executados na página. São armazenados separados por função que recebem os parâmetros para montar as sentenças SQL. Essas funções retornam as SQLs prontas para serem executados.

```
function selectCliente($cpf = null){
    $sql = "SELECT TOP 1 ID_CLIENTE, NOME_CLIENTE, TELEFONE_CLIENTE,
EMAIL_CLIENTE FROM CLIENTES ";

    if(($cpf != null)&&($cpf != '')){
        $sql .= " WHERE CPF_CLIENTE = '$cpf'";
    }else{
            $sql .= " WHERE ID_CLIENTE IS NULL";
      }

    return $sql;
}

function cadastraReserva($id_veiculo, $id_cliente){

    $sql  = " INSERT INTO MOVIMENTACOES (ID_VEICULO, ID_CLIENTE,
ID_SITUACAO, DATA_MOVIMENTACAO, VALOR, TIPO_OPERACAO,
LOCACAO_MANUTENCAO)";

    $sql .= "VALUES ($id_veiculo, $id_cliente, 2, current_timestamp, 0, 1,
0)";

    return $sql;

}

function selectVeiculo($id_veiculo = null){

    $sql  = "SELECT TOP 1 ";
    $sql .= "VEICULOS.ANO_MODELO, ";
    $sql .= "ANO_MODELO_VEICULO.DESCRICAO_ANO_MODELO_VEICULO, ";
    $sql .= "MODELOS_VEICULOS.NOME_MODELO_VEICULO, ";
    $sql .= "MARCAS_VEICULOS.NOME_MARCA_VEICULO, ";
    $sql .= "IMAGENS_VEICULOS.* ";
    $sql .= "FROM IMAGENS_VEICULOS ";
    $sql .= "INNER JOIN VEICULOS ON (VEICULOS.ID_VEICULO =
IMAGENS_VEICULOS.ID_VEICULO) ";
    $sql .= "INNER JOIN ANO_MODELO_VEICULO ON
(ANO_MODELO_VEICULO.ID_ANO_MODELO_VEICULO = VEICULOS.ID_ANO_MODELO_VEICULO)
";
    $sql .= "INNER JOIN MODELOS_VEICULOS ON
(MODELOS_VEICULOS.ID_MODELO_VEICULO = ANO_MODELO_VEICULO.ID_MODELO_VEICULO)
";
```

```
    $sql .= "INNER JOIN MARCAS_VEICULOS ON
(MARCAS_VEICULOS.ID_MARCA_VEICULO = MODELOS_VEICULOS.ID_MARCA_VEICULO) ";

    if(($id_veiculo != null) && ($id_veiculo != '')){
        $sql .= " WHERE IMAGENS_VEICULOS.ID_VEICULO = $id_veiculo";
    }

    $sql .= " ORDER BY IMAGENS_VEICULOS.ID_IMAGEM_VEICULO";

    return $sql;
}
```

Listagem 37 – Parte do código PHP da classe SQL

O código da Listagem 38 é responsável pela visualização do site o código HTML montado através do código PHP que é responsável por listar em tela os veículos retornados pela SQL executada. Caso não seja retornado nenhum registro é exibida através do <h1> a informação que não há veículos cadastrados. Caso contrário é criptografada a imagem em formato base64 para exibir as imagens do banco de dados.

```
                <h1>Nossa Frota <span>&rsaquo;&rsaquo;</span>
<?=$tipo?></h1>

                <?php if(mssql_num_rows($resVeiculos)> 0){

                    while ($row = mssql_fetch_array($resVeiculos)) {?>

                            <div class="product">
                                <a
href="veiculoInterno.php?id_veiculo=<?=$row['ID_VEICULO'];?>">
                                    <?php echo '<img width="270"
alt="'.$row['NOME_IMAGEM'].'"
src="'.base64_encode($row['IMAGEM']).'" /><br/>';?>
                                    <h5 style="margin-
top:8px"><?=$row['NOME_IMAGEM']?></h5>
                                </a>
                            </div>

                        <?php } ?>

                        <hr />

                    <?php } else{ ?>

                        <br /><br /><br /><br /><br />
                        <center>
                            <h1>N&atilde;o h&aacute; ve&iacute;
cadastrados!</h1>
                        </center>

                     <?php } ?>
```

Listagem 38 – Parte do código HTML da página Nossa Frota.

O código JavaScript localizado na parte *view* da estrutura do site dentro do arquivo de "reserva.php" está na Listagem 39. Esse código é responsável pela atribuição de máscaras aos

campos, verificar se o CPF é válido, validar o formulário com as informações preenchidas e através de Ajax pelo método *get* validar a existência do CPF no banco de dados da Locadora.

```
        <script type="text/javascript">

            $("#telefone").mask("(99)9999-9999");
            $("#cpf").mask("999.999.999-99");
            $("#fmReserva").validate({rules:{'cpf':{cpf : 'both'}}});

            $("#cpf").bind('blur', function() {

$.get('../controller/validaClienteController.php?cpf='+$("#cpf").val(),
function(result) {
                    if (result != '0'){
                        var arr = result.split("&&");
                        $('#id_cliente').val(arr[0]);
                        $('#nome').val(arr[1]);
                        $('#telefone').val(arr[2]);
                        $('#email').val(arr[3]);
                    }else{
                                    $('#id_cliente').val('');
                        $('#nome').val('');
                        $('#telefone').val('');
                        $('#email').val('');
                             }
                });
            });
        </script>
```

Listagem 39 – Parte do código Javascript da página reserva.php

5 CONCLUSÃO

O objetivo desse trabalho foi fazer uma modelagem inicial de um sistema para locadora de veículos e implementar alguns cadastros. A implementação teve a finalidade de familiarizar o autor deste trabalho com as ferramentas utilizadas. Assim, os cadastros de dois módulos (veículos e localização) foram implementados. Foram considerados módulos porque os formulários implementados estavam relacionados ao cadastro de veículos e de localização.

O sistema é considerado de informações gerenciais porque visa fornecer suporte operacional e gerencial para as atividades realizadas em uma locadora de veículos. Esse sistema poderá ser utilizado como um apoio às atividades do gestor. A implementação da página *web* vinculada ao sistema possibilita que os clientes possam realizar reserva por meio da Internet.

Em termos da linguagem utilizada para implementação, a Delphi, os autores deste trabalho consideram que trabalhar com o Embarcadero Rad XE2 foi importante, pois em alguns aspectos facilitou a implementação. Em termos gerais a versão XE2 pode ser comparada à versão Delphi 7 da Borland, com uma interface mais bem apresentada. E, além disso, há propriedades novas nos componentes, como a propriedade "Glyph" dos componentes "TBitBtn" e "TSpeedButton" que agora reconhece uma imagem com duas figuras diferentes. No caso dos cadastros implementados é uma imagem colorida e outra não. E isso fica automaticamente ligado à propriedade "Enabled" do mesmo componente, que faz com que ela mude qual imagem apresenta, conforme o componente está "Enabled True" ou "Enabled False".

Outro recurso importante da versão XE2 é relacionada à pesquisa pelo nome dos componentes que foi adicionada na paleta de Ferramentas. Isso facilita muito, uma vez que há muitos componentes.

Nas telas, os botões e os componentes ficaram com uma interface nova, eles são mais arredondados, poder-se-ia dizer que estão no estilo "Windows 7", fornecendo uma aparência nova aos sistemas.

Porém, o Rad XE2 trouxe muito mais do que esses detalhes descritos. O FireMonkey Framework, por exemplo, com seus objetos em três dimensões e diversas opções de leiaute de tela e componentes. Esses componentes não foram utilizados na implementação do sistema resultado deste trabalho.

REFERÊNCIAS

ADOBE. **Adobe Photoshop**. Disponível em: <http://www.adobe.com/br/products/photoshop.html>. Acesso em: 20 ago. 2012.

APACHE. **Apache friends.** Disponível em: <http://www.apachefriends.org/en/index.html>. Acesso em 21 out. 2012

ASTAH. **Astah community**. Disponível em <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: 08 ago. 2012.

BATISTA, Emerson de Oliveira. **Sistema de informação:** o uso consciente da tecnologia para o gerenciamento. São Paulo: Saraiva, 2004.

BAZZOTTI, Cristiane, GARCIA, Elias **A importância do sistema de informação gerencial para tomada de decisões**. Disponível em: <http://www.unioeste.br/campi/cascavel/ccsa/VISeminario/Artigos%20apresentados%20em%20Comunica%E7%F5es/ART%203%20-%20A%20import%E2ncia%20do%20sistema%20de%20informa%E7%E3o%20gerencial%20para%20tomada%20de%20decis%F5es.pdf>. Aceso em 19 de jun. 2012.

CANTU, Marco. **Dominando o Delphi 7 "A Bíblia".** São Paulo: Makron Books, 2003.

EICHSTAEDT, John F., DEGENHARDT, Toni Édio, **Sistemas de informações gerenciais.** Disponível em http://www.ceavi.udesc.br/arquivos/id_submenu/387/john_frank_eichstaedt_toni_edio_degenhardt.pdf. Aceso em 20 jun. 2012.

LAUDON, Kenneth C.; LAUDON, Jane Price. **Sistemas de informação gerenciais:** administrando a empresa digital. 5. ed. São Paulo, SP: Pearson Prentice Hall, 2004.

MICROSOFT. **Use SQL server management studio**, 2012. Disponível em: < http://msdn.microsoft.com/en-us/library/ms174173.aspx >. Acesso em: 09 jul. 2012.

MIKKONEN, Tommi; TAIVALSAARI, Antero. **Web applications – spaghetti code for the 21st century.** Sixth International Conference on Software Engineering Research, Management and Applications, 2008, p. 319-328.

MIRANDA, Ozineide Alves. **A importância do sistema de informação gerencial na empresa Sol Distribruidora de Combustíveis Ltda.** Disponível em http://www.profsergio.net/artigos/artigoozineidealves.pdf. Acesso em 20 jun.2012

NETBEANS. **NetBeans IDE**. Disponível em: <http://www.netbeans.org>. Acesso em: 20 jul. 2012

NOTEPAD. **Notepad++**. Disponível em: <http://notepad-plus-plus.org/>. Acesso em: 23 set. 2012.

OLIVEIRA, Djalma de Pinho Rebouças de. **Sistemas de informações gerenciais**. 9 ed. São Paulo: Atlas 2004.

OLIVEIRA, Figueiredo de Oliveira. **Sistemas de informação: um enfoque gerencial inserido no contexto empresarial e tecnológico**. 3ª ed. São Paulo: Érica 2002.

PEREIRA, Maria José Lara de Bretãs; FONSECA, João Gabriel Marques. **Faces da decisão: as mudanças de paradigmas e o poder da decisão**. São Paulo: Makron Books, 1997.

PHP. **Linguagem PHP**. Disponível em: <http://www.php.net>. Acesso em: 20 jul. 2012.
PRESSMAN, Roger. **Engenharia de software**. 5ª ed. Rio de Janeiro: McGraw-Hill, 2002

REZENDE, Denis Alcides; ABREU, Aline França de. **Tecnologia da informação aplicada a sistemas de informação empresariais**: o papel estratégico da informação e dos sistemas de informação nas empresas. São Paulo: Atlas, 2000.

STAIR, Ralph M. e REYNOLDS George W. **Princípios de sistemas de informações: uma abordagem gerencial**. 4a ed. São Paulo: LTC, 2002.

STAIR, Ralph M. **Princípios de sistemas de informação**. Rio de Janeiro: LTC, 1998.

XAMPP. **XAMPP**. Disponível em: <http://www.apachefriends.org/en/xampp.html>. Acesso em: 15 out. 2012.