FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimentos de Sistemas

João Paulo Frezzarin

Lucas Rafael da Silva

Pedro Guilherme A. Leme

LEGOCAR

Americana, SP

2018

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

João Paulo Frezzarin

Lucas Rafael da Silva

Pedro Guilherme A. Leme

LEGOCAR

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso SuperiordeTecnologia em Análise e Desenvolvimento de Sistemas do Prof.(a) Me.Clerivaldo José Roccia

Americana, SP.

2018

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

S581L SILVA, Lucas Rafael da

Legocar. / Lucas Rafael da Silva, João Paulo Frezzarin, Pedro Guilherme de Almeida Leme. – Americana, 2018.

53f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Clerivaldo José Roccia

1. Engenharia automotiva I. FREZZARIN, João Paulo II. LEME, Pedro Guilherme de Almeida III. ROCCIA, Clerivaldo José IV. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 621

João Paulo Frezzarin

Lucas Rafael da Silva

Pedro Guilherme A. Leme

LEGOCAR

Relatório técnico apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Americana,6 de dezembro de 2018.

Banca Examinadora:

Clerivaldo José Roccia (Presidente)
Mestre
Faculdade de Tecnologia de
Americana

Murilo Fujita (Membro)
Mestre
Faculdade de Tecnologia de
Americana

Antonio Alfredo Lacerda (Membro)
Mestre
Faculdade de Tecnologia de
Americana

RESUMO

Veículo é um bem que pode ser imprescindível para certas pessoas, devido aos ganhos que ele traz de locomoção. Entretanto, devido a sua engenharia complexa, ter conhecimento do mesmo e conseguir realizar a sua manutenção não é uma tarefa para qualquer um.

Com o problema apresentado acima, foi desenvolvido o sistema LegoCar, tendo como objetivo a gestão da manutenção preventiva de veículos. Para a elaboração desse projeto foram utilizadas técnicas de engenharia de software (na produção dos diagramas de classe, sequência, caso de uso e atividade), conhecimento de desenvolvimento (com o foco voltado para web), e, por último, mas não menos importante toda a elaboração do banco de dados. Através desses estudos o foco deste trabalho foi totalmente voltado para a produção de um sistema para gerenciamento de veículos.

Palavras-chave: Veículo, Manutenção, Gerenciamento.

ABSTRACT

Vehicle is a property that may be essential for certain people, due to the gains that it brings of locomotion. However, due to its complex engineering, being aware of it and being able to perform its maintenance is not a task for anyone.
With the problem presented above, the LegoCar system was developed, aiming the management of preventive maintenance of vehicles. For the elaboration of this project we used software engineering techniques (in the production of class diagrams, sequence, use case and activity), development knowledge (with the focus turned to the web), and last but not least all the elaboration of the database. Through these studies the focus of this work was totally focused on the production of a vehicle management system.

Keywords: *Vehicle, Maintenance, Management.*

SUMÁRIO

1. INTRODUÇÃO

Aproveitando que os sistemas vieram para nos ajudar, o grupo resolveu desenvolver um sistema para controle de veículos, sejam eles pessoais ou profissionais, pois por muitas das vezes as pessoas esquecem de realizar determinado serviço ou aquisição de determinada peça que é de suma importância para o bom funcionamento do automóvel.

Com o propósito de facilitar a manutenção do veículo, o presente trabalho propõe o desenvolvimento de um sistema web, capaz de gerenciar as informações e peças de determinado automóvel do usuário.

Diante desse cenário foi criado o LegoCar, onde o administrador pode criar uma conta, e após a criação da mesma é possível cadastrar o modelo do veículo, se é carro ou moto, a cor do mesmo, placa, ano e marca. Isso tudo vai ser utilizado quando o dono do automóvel decidir realizar alguma manutenção sendo ela uma troca de peça, óleo ou algo do tipo.

Para a utilização deste aplicativo, o usuário pode utilizar dispositivos como desktops, notebooks, tablets e celulares, com acesso a internet e alguma aplicação com recurso de navegação na internet. Pois o desenvolvimento deste trabalho está relacionado à tecnologia responsiva (que fornece uma fácil navegação e visualização de conteúdos web em diferentes tamanhos de dispositivos).

1.1 Objetivos

A seguir serão apresentados os objetivos gerais e específicos do trabalho.

1.1.1 Objetivos Gerais

Desenvolver um software web que permita gerenciar o veículo, seja ele pessoal ou profissional, permitindo que o usuário consiga fazer determinada manutenção ou aquisição quando for necessário.

1.1.2 Objetivos Específicos

Dentre os objetivos específicos do trabalho, destacam-se:

º Realizar a análise da documentação de requisitos, identificando as características para desenvolver o sistema de gerenciamento de veículos.

º Aplicar as técnicas e linguagens de programação web para que o desenvolvimento seja possível.

º Utilizar a UML como linguagem de modelagem para o projeto LegoCar.

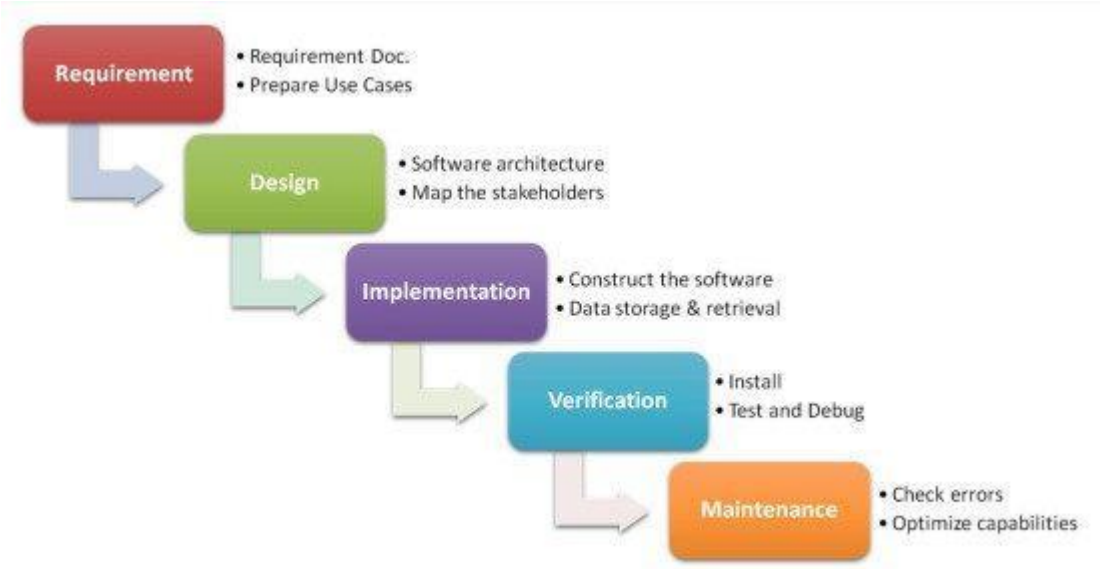
1.2 JUSTIFICATIVA

A ideia de criar um sistema voltado para o gerenciamento de veículos é para viabilizar o atendimento quando o mesmo estiver precisando de algum serviço ou alguma peça essencial. Por muitas das vezes acabamos esquecendo e deixando algo crucial passar batido, isso pode acarretar danos que talvez se tornem irreversíveis futuramente. Isso agilizaria bastante o trabalho de mecânicos também, que muitas vezes anotam o serviço no papel e acaba perdendo a anotação. Com um sistema na mão todo e qualquer reparo se torna mais fácil.

2. METODOLOGIA

A metodologia adotada para a criação do LegoCar foi o modelo de cascata, segue a foto do modelo abaixo:

Figura 1 – metodologia cascata



Fonte: https://casadaconsultoria.com.br/wp-content/uploads/2016/09/modelo-cascata.jpg

A metodologia de cascata ou top-down foi criada na década de 1970 por Royce, sendo o modelo mais aceito até 1980.

Esse modelo tem a finalidade de determinar a ordem em que o desenvolvimento será posto em prática. Se comparado com outros modelos, o de cascata é mais rígido e menos administrativo.

Seu objetivo principal é que as fases do desenvolvimento sigam uma sequência. A primeira etapa, quando finalizada se direciona para a segunda e esta se movimenta para a terceira, assim por diante.

As etapas do modelo são divididas em 5, sendo elas:

1 – Requerimento

2 – Projeto

3 – Implementação

4 – Verificação

5 – Manutenção

Na primeira etapa (**Requerimento**) são estabelecidos os requisitos do produto que está para ser desenvolvido.

Depois que isso é determinado, os requisitos precisam ser qualificados de forma adequada, para que também sejam úteis para a etapa seguinte. Nessa etapa está localizada a documentação e o estudo de viabilidade, é compreendida como o começo do projeto.

Já na segunda etapa (**Projeto**), estão localizadas a estrutura de dados, a arquitetura do software, caracterização das interfaces e detalhes procedimentais.

Podemos compreender essa etapa como a que possibilita a codificação do produto.

A **Implementação**, por sua vez, é quando os programas são criados. É recomendado realizar a codificação por módulos, tornando mais fácil o seu desenvolvimento. Quando for realizar testes, os mesmos devem acontecer de maneira individual para cada módulo.

No quarto patamar (**Verificação**) são onde ocorrem os testes do sistema como um todo. Nesse módulo são focadas as lógicas internas e as funcionalidades externas do software. Essa etapa serve para solucionar todos os possíveis erros que podem se encontrar no projeto.

A última fase do modelo de cascata (**Manutenção**) se baseia na correção de erros que são detectados durante os testes, melhorias e mais suporte. As melhorias fazem parte do processo de desenvolvimento.

O modelo de cascata em si funciona com a ideia de que a saída de determinado módulo seja a entrada da próxima etapa.

3.IMPLEMENTAÇÃO

3.1 REQUISITOS

Requisitos podem ser entendidos como propriedade, comportamento ou restrição de operação que um produto ou serviço deve atender (WTHREEX.COM, 2014).

Gerenciamento de requisitos é um modelo sistemático para compreender e controlar as mudanças nos requisitos de sistemas, nesse processo inclui-se identificar, organizar, documentar os requisitos de sistema e estabelecer e manter acordo entre o cliente e a equipe de desenvolvimento (WTHREEX.COM, 2014)

º Thayer e Dorfmann (2000) definem que requisito é uma característica do software necessária para que o usuário e/ou cliente possa encontrar a solução de um problema de forma a atingir um objetivo.

º Sommerville (2007) expressa que requisitos de um sistema são descrições dos serviços fornecidas pelo sistema e as suas restrições operacionais. Os requisitos demonstram as necessidades de um cliente de um sistema que ajuda a resolver um determinado problema.

º Requisitos são como uma declaração completa do que o software irá fazer sem referir-se a como fazê-lo (LOPES apud SIDDIQI, 1996).

3.1.2 REQUISITOS DE USUÁRIO

Os requisitos de usuário são feitos em linguagem natural, com diagramas os diagramas de quais serviços o sistema deve oferecer e as restrições que ele deve realizar. Deve ser compreensível pelos usuários que

não possuem conhecimento técnico, é recomendável não usar termos técnicos inclusive.

3.1.3 REQUISITOS DE SISTEMA

Requisitos de sistema são as funções, serviços e restrições. Deve existir um documento desses requisitos, o mesmo deve ser bem detalhado para que o sistema saia exatamente como o esperado.

De acordo com Sommerville (2007) os requisitos de sistema são classificados em requisitos funcionais e em requisitos não funcionais ou de domínio.

3.1.4 REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem explicitamente as funcionalidades e serviços do sistema, documentando como o sistema deve reagir a entradas específicas, como deve se comportar em determinadas situações e no que o sistema não deve fazer.

Os atributos dos requisitos funcionais são: A **Completude** e a **Consistência**.

A **Completude**, onde todos os serviços devem estar definidos.

A **Consistência**, onde os requisitos não devem ter definições contraditórias.

Os requisitos funcionais do sistema foram desenvolvidos separados por telas, segue abaixo:

1.1. Telas

1.1.1. O sistema deverá ter uma tela de login para autenticação, apelidada de “Login”.

1.1.2. O sistema deverá ter uma tela de início, apelidada de “Home”.

1.1.3. O sistema deverá ter uma tela de listagem de veículos, apelidada de “Veículos”.

1.1.4. O sistema deverá ter uma tela de cadastro de veículo, apelidada de “Cadastro de veículo”.

1.1.5. O sistema deverá ter uma tela de detalhes do veículo, apelidada de “Detalhes do veículo”

1.1.6. O sistema deverá ter uma tela de listagem de usuários, apelidada de “Usuários”.

1.1.7. O sistema deverá ter uma tela de cadastro de usuário, apelidada de “Cadastro de usuário”.

1.1.8. O sistema deverá ter uma tela de detalhes do usuário, apelidada de “Detalhes do usuário”.

1.1.9. O sistema deverá ter uma tela de listagem de itens, apelidada de “Itens”.

1.1.10. O sistema deverá ter uma tela de cadastro de item, apelidada de “Cadastro de item”.

1.1.11. O sistema deverá ter uma tela de detalhes do item, apelidada de “Detalhes do item”.

1.2. Menu

1.2.1. O sistema deverá exibir um menu de auxílio, o qual estará disponível em todas as telas e mudará conforme a permissão do usuário.

1.2.2. Para usuários comuns, o menu exibirá as opções de “Home” e “Sair”.

1.2.3. Para funcionários, o menu exibirá as opções de “Home”, “Usuários”, “Veículos” e “Sair”.

1.2.4. Para mecânicos, o menu exibirá as opções de “Home”, “Veículos”, “Itens” e “Sair”.

1.2.5. Para administradores, o menu exibirá as opções “Home, “Usuários”, “Veículos”, “Itens” e “Sair”.

1.2.6. Ao clicar na opção do menu “Home”, o sistema deverá direcionar para a tela “Home”.

1.2.7. Ao clicar na opção do menu “Usuários”, o sistema deverá direcionar para a tela “Usuários”.

1.2.8. Ao clicar na opção do menu “Veículos”, o sistema deverá direcionar para a tela “Veículos”.

1.2.9. Ao clicar na opção do menu “Itens”, o sistema deverá direcionar para a tela “Itens”.

1.2.10. Ao clicar na opção do menu “Logout”, o sistema deverá realizar o logout do usuário no sistema.

1.3. Tela Login

1.3.1. A tela de “Login” deverá ter um campo de “Email” e “Senha” e um botão “Logar”, para realizar a autenticação no sistema.

1.3.2. O usuário (usuário comum, funcionário, mecânico ou administrador), ao autenticar no sistema com sucesso, deverá ser direcionado para a tela “Home”.

1.3.3. O usuário (usuário comum, funcionário, mecânico ou administrador), ao autenticar no sistema com falha, deverá ser exibido a seguinte mensagem “Usuário ou senha incorretos”, permanecendo na tela de “Login”.

1.4. Tela Home

1.4.1. A tela “Home” deverá exibir uma lista de todos os veículos ativos do usuário autenticado com os seguintes dados: “Placa” e “Quilometragem” .

1.4.2. Ao clicar em um dos veículos da lista “Home”, o sistema deverá direcionar para a tela de “Detalhes do veículo”, exibindo os dados detalhados do veículo selecionado.

1.5. Tela Usuários

1.5.1. A tela de “Usuários” deverá ser somente visível e operada por funcionários e administradores.

1.5.2. A tela de “Usuários” deverá ter um botão “Cadastrar usuário”, que irá direcionar para a tela de “Cadastro de Usuário”.

1.5.3. A tela de “Usuários” deverá exibir uma lista contendo todos os usuários cadastrados no sistema.

1.5.4. O usuário, ao clicar em um dos usuários da lista, será direcionado para a tela de “Detalhes do usuário”, exibindo todos os dados detalhados do usuário selecionado.

1.6. Tela Cadastro de Usuário

1.6.1. A tela de “Cadastro de Usuários” deverá ser somente visível e operada por funcionários e administradores.

1.6.2. A tela de “Cadastro de Usuários” deverá ter os campos “Nome”, “Email”, “Senha”, “Permissão” e um botão “Cadastrar”.

1.6.3. Ao clicar no botão “Cadastrar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá cadastrar o usuário e retornar a mensagem de “Usuário cadastrado”.

1.6.4. Ao clicar no botão “Cadastrar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.6.5. Ao cadastrar um usuário, o sistema deverá validar que os campos “Nome”, “Email” e “Senha” estejam preenchidos e que não exista nenhum email igual à algum usuário já cadastrado no sistema.

1.6.6. Caso nenhum seja preenchido o campo “Permissão”, o sistema deverá cadastrar o usuário como sendo usuário comum.

1.7. Tela Detalhes do Usuário

1.7.1. A tela de “Detalhes do Usuário” deverá ser somente visível e operada por funcionários e administradores.

1.7.2. A tela de “Detalhes do Usuário” deverá ter os campos “Nome”, “Email”, “Senha” e “Permissão”, preenchidos conforme os dados do usuário selecionado, e um botão “Atualizar”.

1.7.3. Caso o usuário esteja ativo, deverá aparecer um botão “Inativar” e, caso esteja inativo, deverá aparecer um botão “Reativar”.

1.7.4. Ao clicar no botão “Atualizar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá atualizar o usuário e retornar a mensagem de “Usuário atualizado”.

1.7.5. Ao clicar no botão “Atualizar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.7.6. Ao atualizar um usuário, o sistema deverá validar que os campos “Nome”, “Email” e “Senha” estejam preenchidos e que não exista nenhum email igual à algum usuário já cadastrado no sistema.

1.7.7. Caso nenhum seja preenchido o campo “Permissão”, o sistema deverá atualizar o usuário como sendo usuário comum.

1.7.8. Ao clicar no botão “Inativar”, o sistema deverá inativar o usuário em questão.

1.7.9. Ao clicar no botão “Reativar”, o sistema deverá ativar o usuário em questão.

1.8. Tela Itens

1.8.1. A tela de “Itens” deverá ser somente visível e operada por mecânicos e administradores.

1.8.2. A tela de “Itens” deverá ter um botão “Cadastrar item”, que irá direcionar para a tela de “Cadastro de Item”.

1.8.3. A tela de “Itens” deverá exibir uma lista contendo todos os itens cadastrados no sistema.

1.8.4. O usuário, ao clicar em um dos itens da lista, será direcionado para a tela de “Detalhes do item”, exibindo todos os dados detalhados do item selecionado.

1.9. Tela Cadastro de Item

1.9.1. A tela de “Cadastro de Item” deverá ser somente visível e operada por mecânicos e administradores.

1.9.2. A tela de “Cadastro de Item” deverá ter os campos “Nome”, “Marca”, “Validade por Mês”, “Validade por Quilometragem”, “Modelo do item” e um botão “Cadastrar”.

1.9.3. Ao clicar no botão “Cadastrar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá cadastrar o item e retornar a mensagem de “Item cadastrado”.

1.9.4. Ao clicar no botão “Cadastrar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.9.5. Ao cadastrar um item, o sistema deverá validar que os campos “Nome” e “Modelo do Item” estejam preenchidos, que ou a “Validade por Mês” ou a “Validade por Quilometragem” esteja preenchida e seja acima de zero, e que não exista nenhum Nome igual à algum item já cadastrado no sistema.

1.10. Tela Detalhes do Item

1.10.1. A tela de “Detalhes do Item” deverá ser somente visível e operada por mecânicos e administradores.

1.10.2. A tela de “Detalhes do Item” deverá ter os campos “Nome”, “Marca”, “Validade por Mês”, “Validade por Quilometragem”, “Modelo do item” e um botão “Atualizar”.

1.10.3. Caso o item esteja ativo, deverá aparecer um botão “Inativar” e, caso esteja inativo, deverá aparecer um botão “Reativar”.

1.10.4. Ao clicar no botão “Atualizar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá atualizar o item e retornar a mensagem de “Item atualizado”.

1.10.5. Ao clicar no botão “Atualizar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.10.6. Ao atualizar um item, o sistema deverá validar que os campos “Nome” e “Modelo do Item” estejam preenchidos, que ou a “Validade por Mês” ou a “Validade por Quilometragem” esteja preenchida e seja acima de zero, e que não exista nenhum Nome igual à algum item já cadastrado no sistema.

1.10.7. Ao clicar no botão “Inativar”, o sistema deverá inativar o item em questão.

1.10.8. Ao clicar no botão “Reativar”, o sistema deverá ativar o item em questão.

1.11. Tela Veículos

1.11.1. A tela de “Veículos” deverá ser somente visível e operada por funcionários, mecânicos e administradores.

1.11.2. A tela de “Veículos” deverá ter um botão “Cadastrar veículo”, que irá direcionar para a tela de “Cadastro de Veículos”.

1.11.3. A tela de “Veículos” deverá exibir uma lista contendo todos os veículos cadastrados no sistema.

1.11.4. O usuário, ao clicar em um dos itens da lista, será direcionado para a tela de “Detalhes do veículo”, exibindo todos os dados detalhados do item selecionado.

1.12. Tela Cadastro de Veículos

1.12.1. A tela de “Cadastro de Veículo” deverá ser somente visível e operada por funcionários, mecânicos e administradores.

1.12.2. A tela de “Cadastro de Veículo” deverá ter os campos “Placa”, “Quilometragem” e “Ano de Fabricação”, “Modelo do Veículo” e um botão “Cadastrar”.

1.12.3. Ao clicar no botão “Cadastrar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá cadastrar o veículo e retornar a mensagem de “Veículo cadastrado”.

1.12.4. Ao clicar no botão “Cadastrar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.12.5. Ao cadastrar um veículo, o sistema deverá validar que os campos “Placa” e “Quilometragem”, “Ano de Fabricação” e “Modelo de Veículo” estejam preenchidos, que os campos “Quilometragem” e a “Ano de Fabricação” seja acima de zero, que o “Modelo de Veículo” esteja cadastrado e que não exista nenhuma Placa igual à algum veículo já cadastrado no sistema.

1.13. Tela Detalhes do Veículos

1.13.1. A tela de “Detalhes do Veículo” deverá ser somente visível e operada por funcionários, mecânicos, administradores e o usuário comum que for dono atual do veículo.

1.13.2. A tela de “Detalhes do Veículo” deverá ter os campos “Placa”, “Quilometragem” e “Ano de Fabricação”, “Modelo do Veículo” e um botão “Atualizar”.

1.13.3. Caso o veículo esteja ativo, deverá aparecer um botão “Inativar” e, caso esteja inativo, deverá aparecer um botão “Reativar”.

1.13.4. Ao clicar no botão “Atualizar”, e o sistema validando como satisfatórios os dados preenchidos no campo acima, o sistema deverá atualizar o veículo e retornar a mensagem de “Veículo atualizado”.

1.13.5. Ao clicar no botão “Atualizar”, e o sistema validando como insatisfatórios os dados preenchidos no campo acima, deverá retornar a mensagem de erro com os itens que não estão válidos.

1.13.6. Ao atualizar um veículo, o sistema deverá validar que os campos “Placa” e “Quilometragem”, “Ano de Fabricação” e “Modelo de Veículo” estejam preenchidos, que os campos “Quilometragem” e a “Ano de Fabricação” seja acima de zero, que o “Modelo de Veículo” esteja cadastrado e que não exista nenhuma Placa igual à algum veículo já cadastrado no sistema.

1.13.7. Ao clicar no botão “Inativar”, o sistema deverá inativar o veículo em questão.

1.13.8. Ao clicar no botão “Reativar”, o sistema deverá ativar o veículo em questão.

3.1.5 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais são utilizados para definir as propriedades e as restrições do sistema.

Exemplos: segurança e espaço em disco.

Lembrando que os requisitos não-funcionais podem ser mais críticos que os requisitos funcionais, pois se os mesmos não satisfazerem, tem a possibilidade do sistema ser visto como inútil.

Podem ser classificados como: Requisitos do produto, Requisitos Organizacionais e Requisitos Externos.

Requisitos do produto - Especificam o comportamento do sistema (ex: desempenho).

Requisitos Organizacionais - Política e procedimentos das empresas.

Requisitos Externos - Derivados do ambiente ou fatores externos.

3.3 FERRAMENTAS PARA DOCUMENTAÇÃO DO SISTEMA

3.3.1 UML

UML (UnifiedModelingLanguage), é uma linguagem unificada para modelagem orientada a objetos. Surgiu da unificação de três grandes métodos, o BOOCH, OMT (Rumbaugh) e OOSE (Jacobson). Auxilia bastante na visualização e comunicação entre objetos. Em suma, uma modelagem UML oferece um desenho do software que se pretende desenvolver.

Pode ser utilizada em várias fases de um sistema, desde os primeiros contatos até a parte de gerar o código em si. É aplicada em qualquer sistema com o foco voltado para a orientação de objetos, porém, pode também ajudar na organização de processos de determinada organização.

Os diagramas da UML estão divididos em duas categorias: Estruturais e Comportamentais.

Diagramas Estruturais

º Caso de Uso (Use Case): É um diagrama Geral e Informal para as fases de levantamento e análise de Requisitos do Sistema.

º Máquina de Estados: Acompanha as mudanças em um objeto dentro de um processo.

º Atividades: Descreve os caminhos a serem percorridos para concluir determinada atividade.

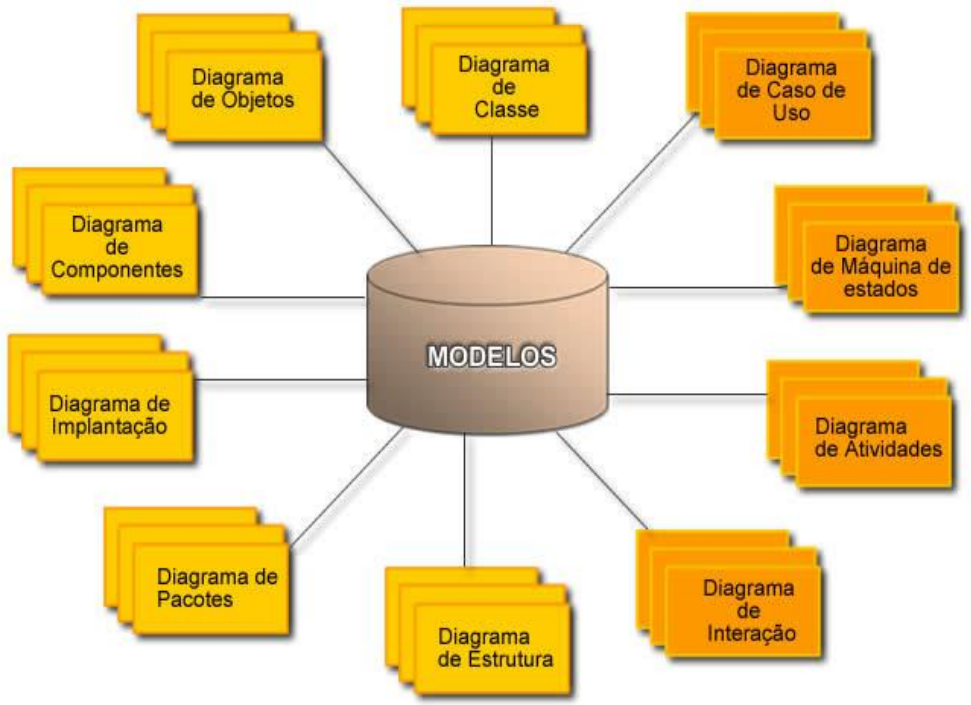
º Interação: Pode ser dividido em:

1. De Sequência: Descreve a ordem que as mensagens são trocadas entre os objetos.

2. Geral interação: Fornece visão geral dentro do sistema

3. De comunicação: Associado ao diagrama de sequência, complementando o mesmo.

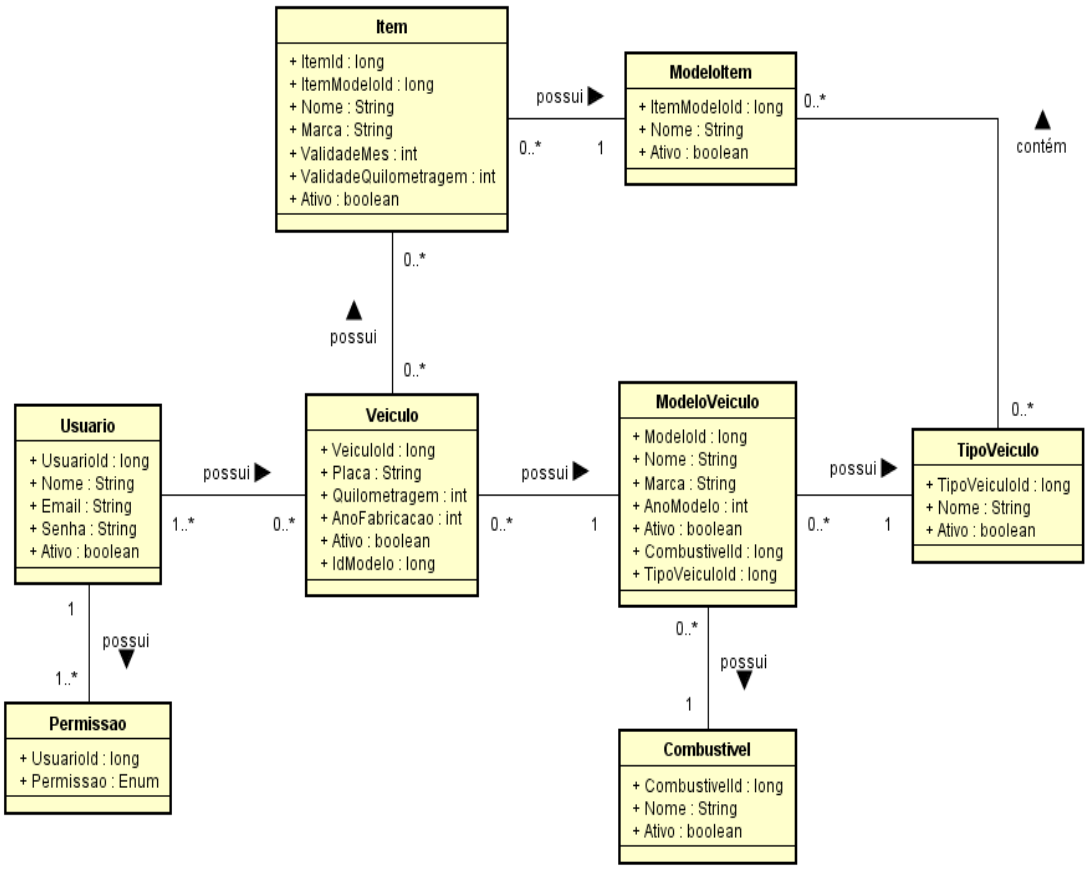
4. De tempo: Descreve a mudança ou condição de uma instância perante o tempo.



Fonte:https://www.infoescola.com/wp-content/uploads/2010/03/DiagramasdaUML.jpg

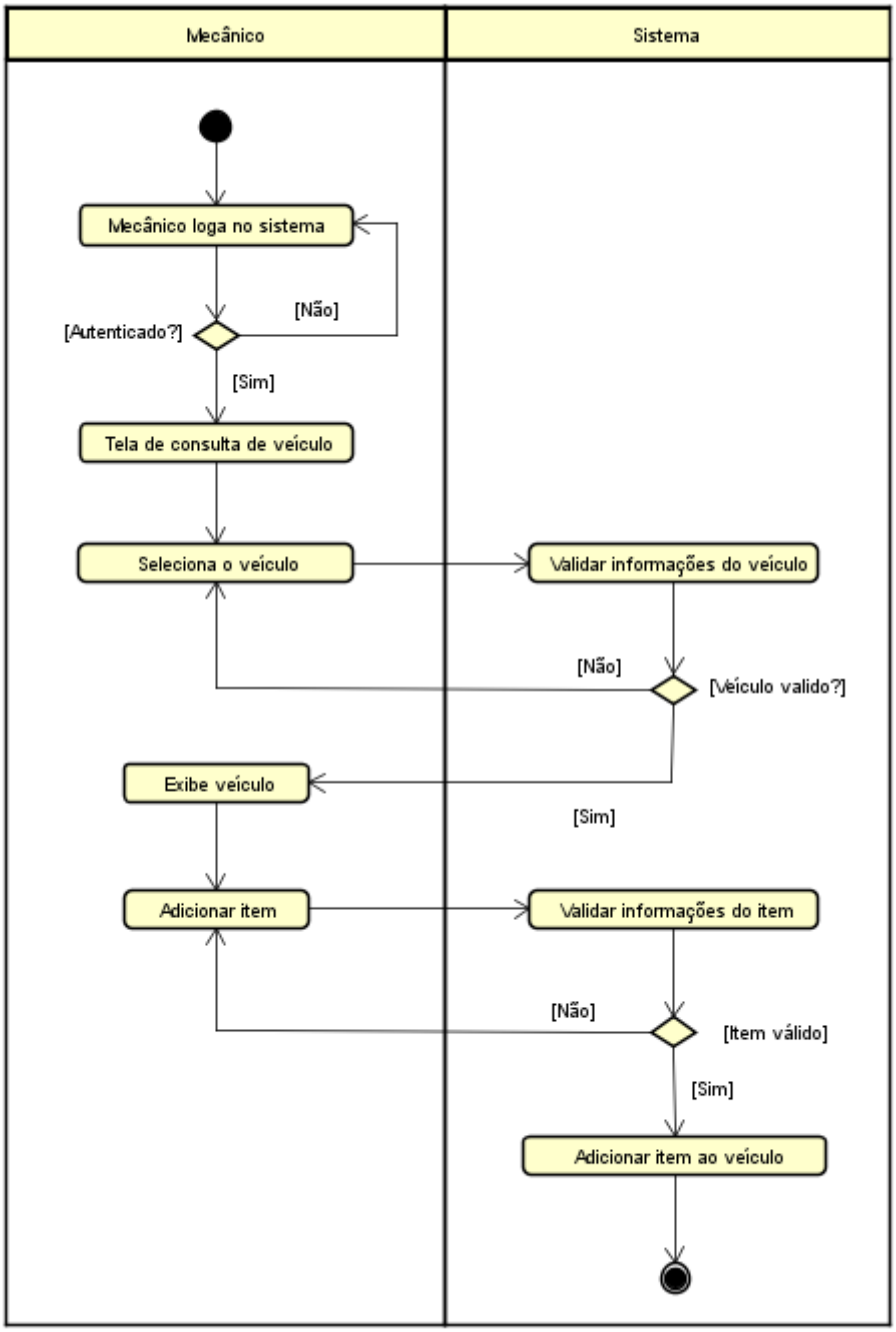
Seguem abaixo os diagramas desenvolvidos para a criação do sistema LegoCar:

Figura 2 - DIAGRAMA DE CLASSE



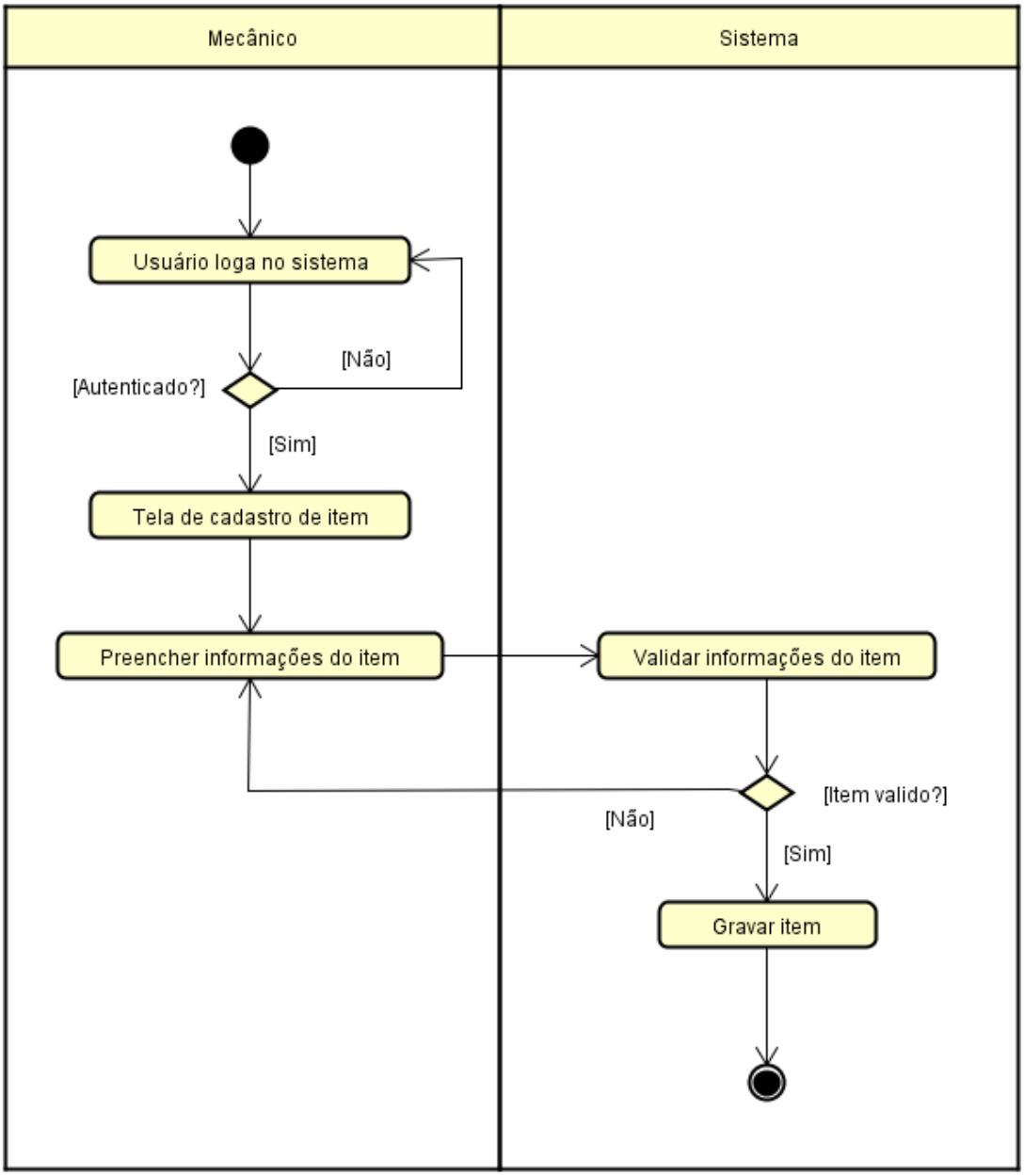
Fonte: Lucas Rafael da Silva

Figura 3 - DIAGRAMA DE ATIVIDADES (Adição de item ao veículo)



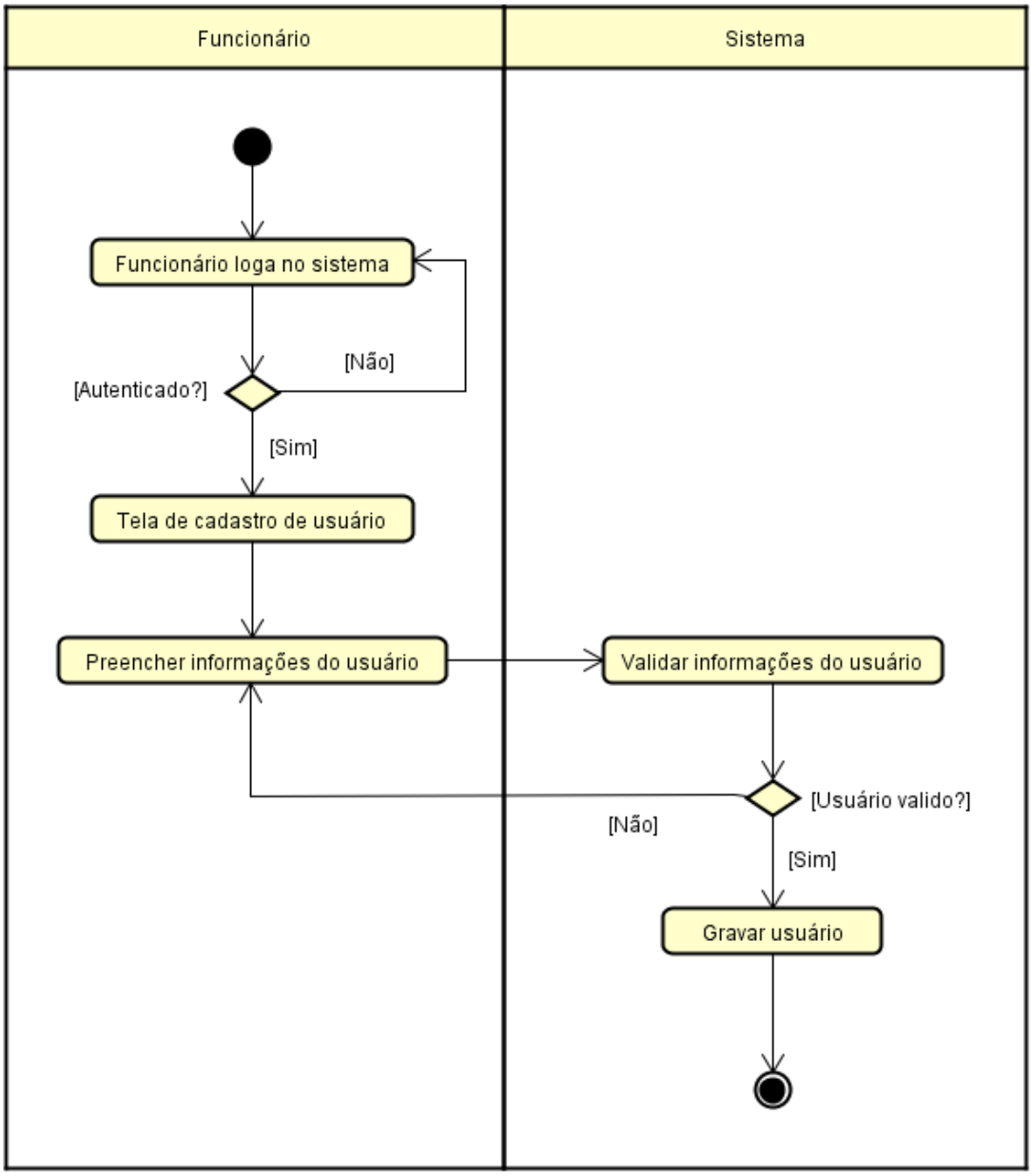
Fonte: João Paulo Frezzarin

Figura 4 - DIAGRAMA DE ATIVIDADES (Cadastro de itens)



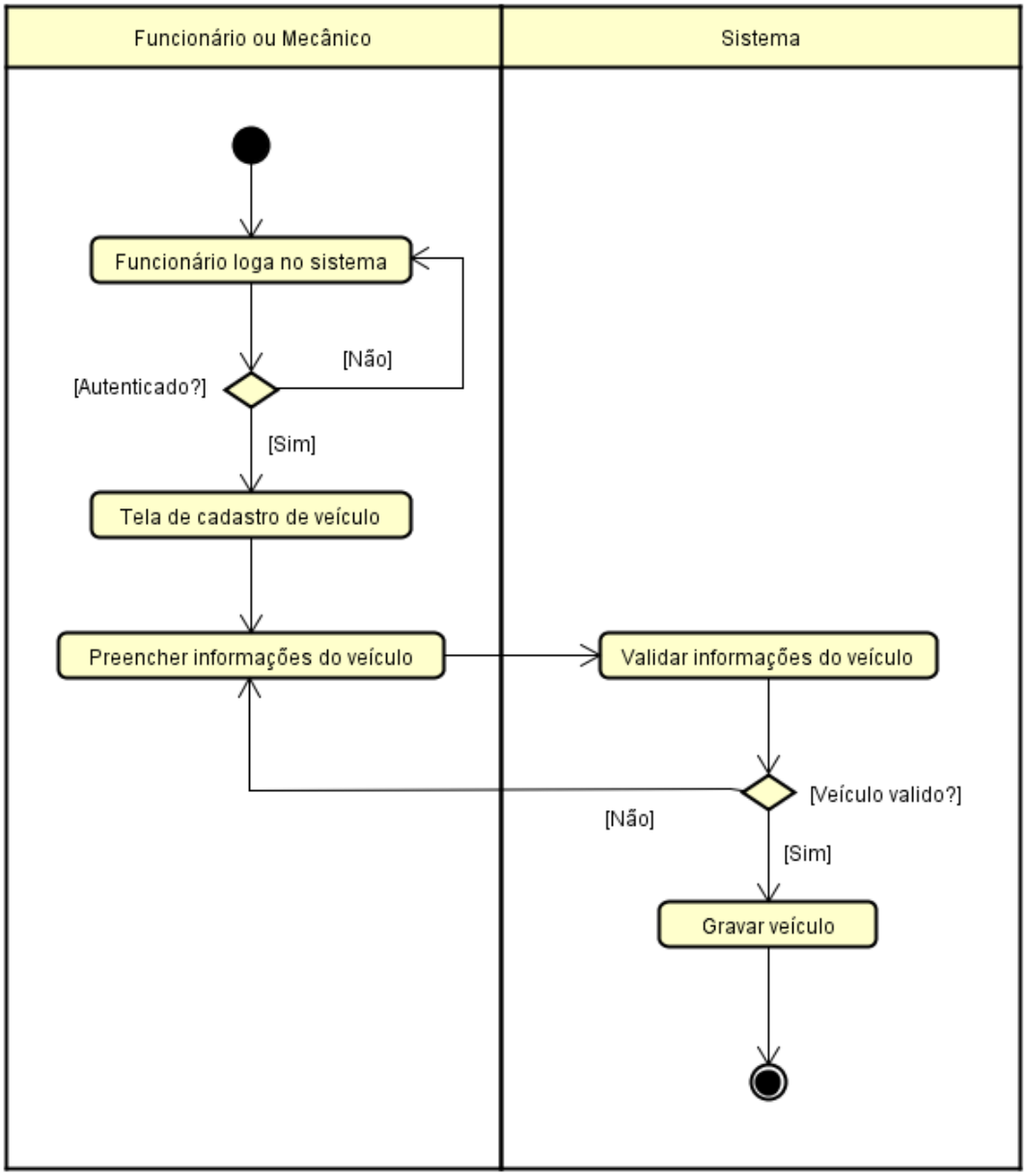
Fonte: João Paulo Frezzarin

Figura 5 - DIAGRAMA DE ATIVIDADES (Cadastro de usuários)



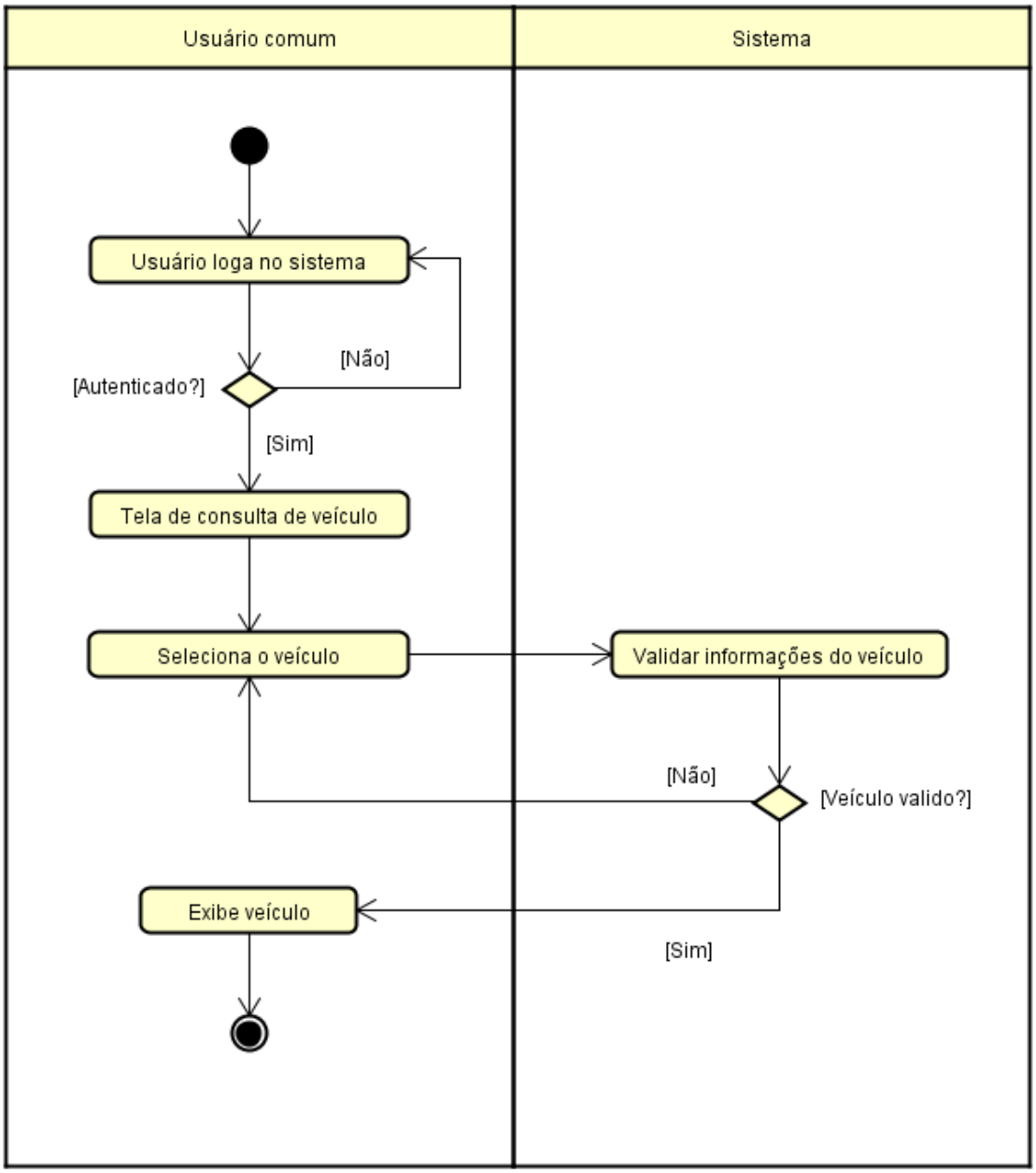
Fonte: João Paulo Frezzarin

Figura 6 - DIAGRAMA DE ATIVIDADES (CADASTRO DE VEÍCULOS)



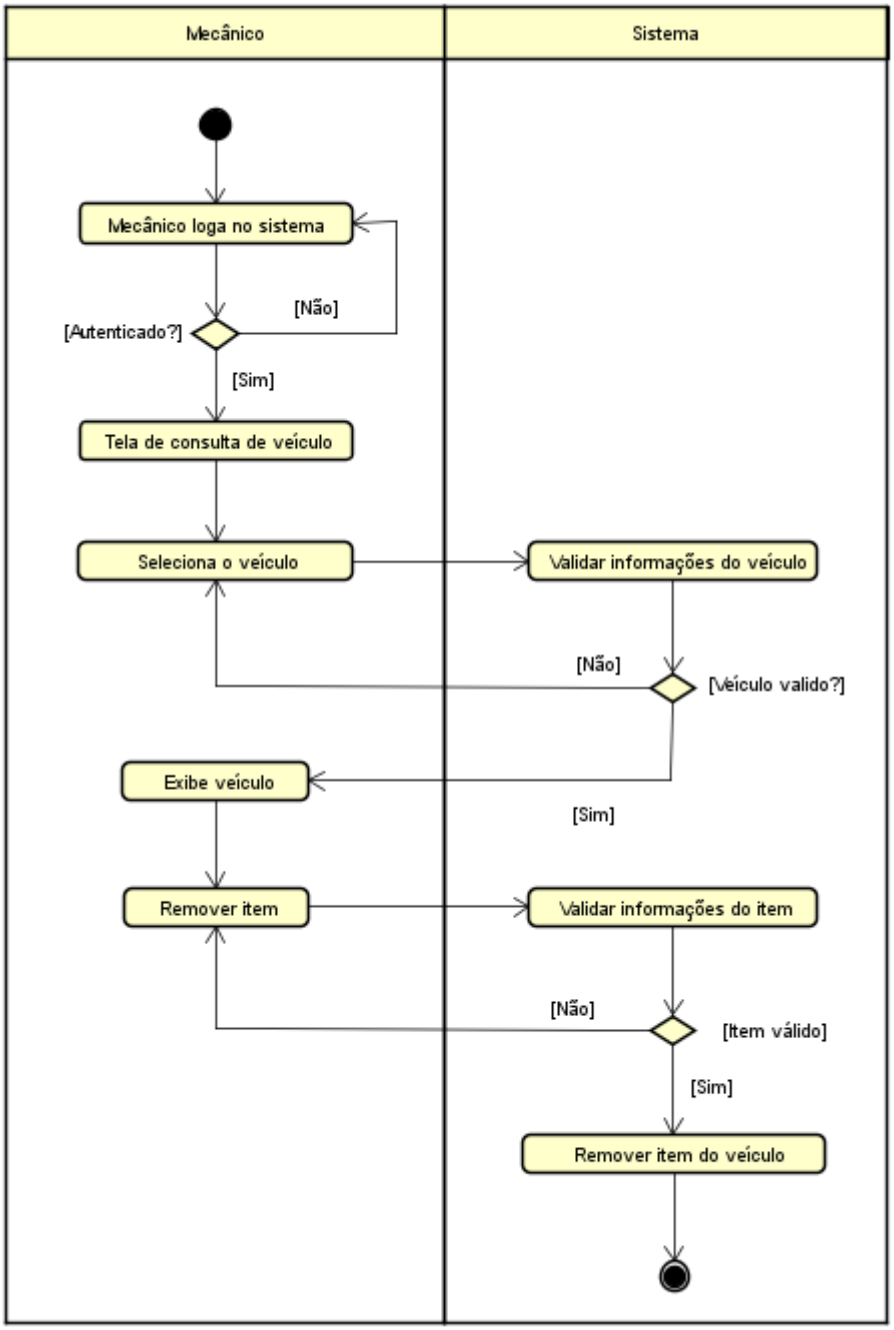
Fonte: João Paulo Frezzarin

Figura 7 - DIAGRAMA DE ATIVIDADES (Consulta de veículos)



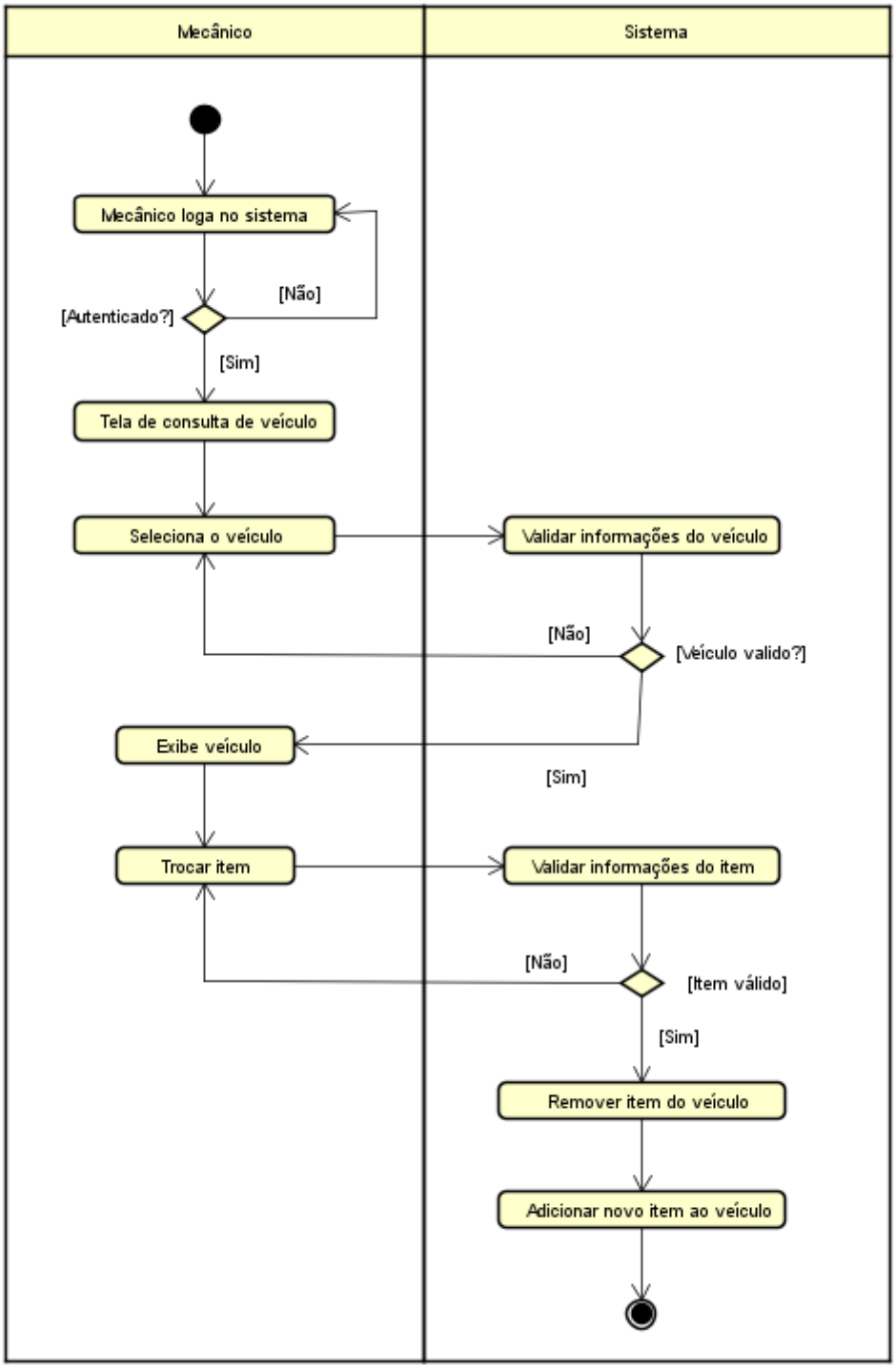
Fonte: João Paulo Frezzarin

Figura 8 - DIAGRAMA DE ATIVIDADES (Remover item do veículo)



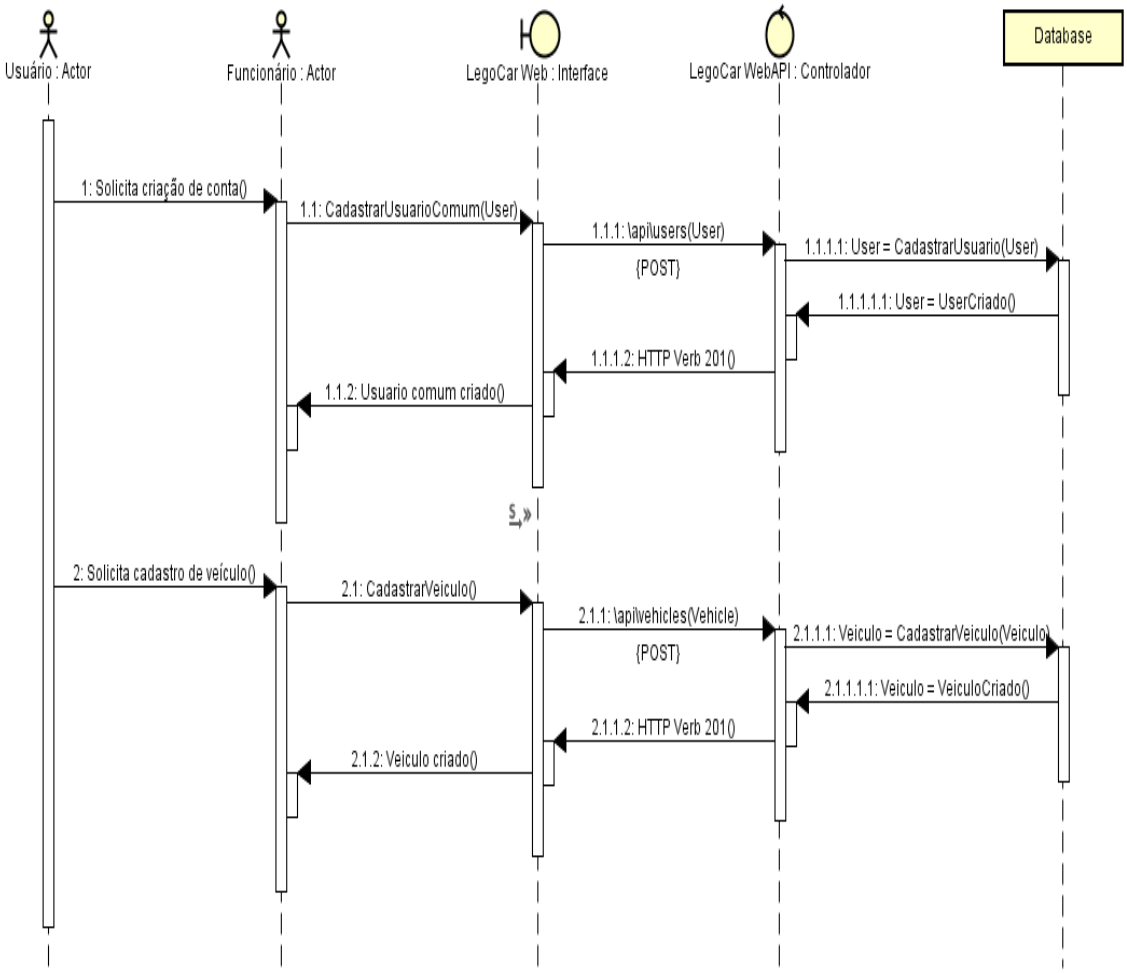
Fonte: João Paulo Frezzarin

Figura 9 - DIAGRAMA DE ATIVIDADES (Trocar item do veículo)



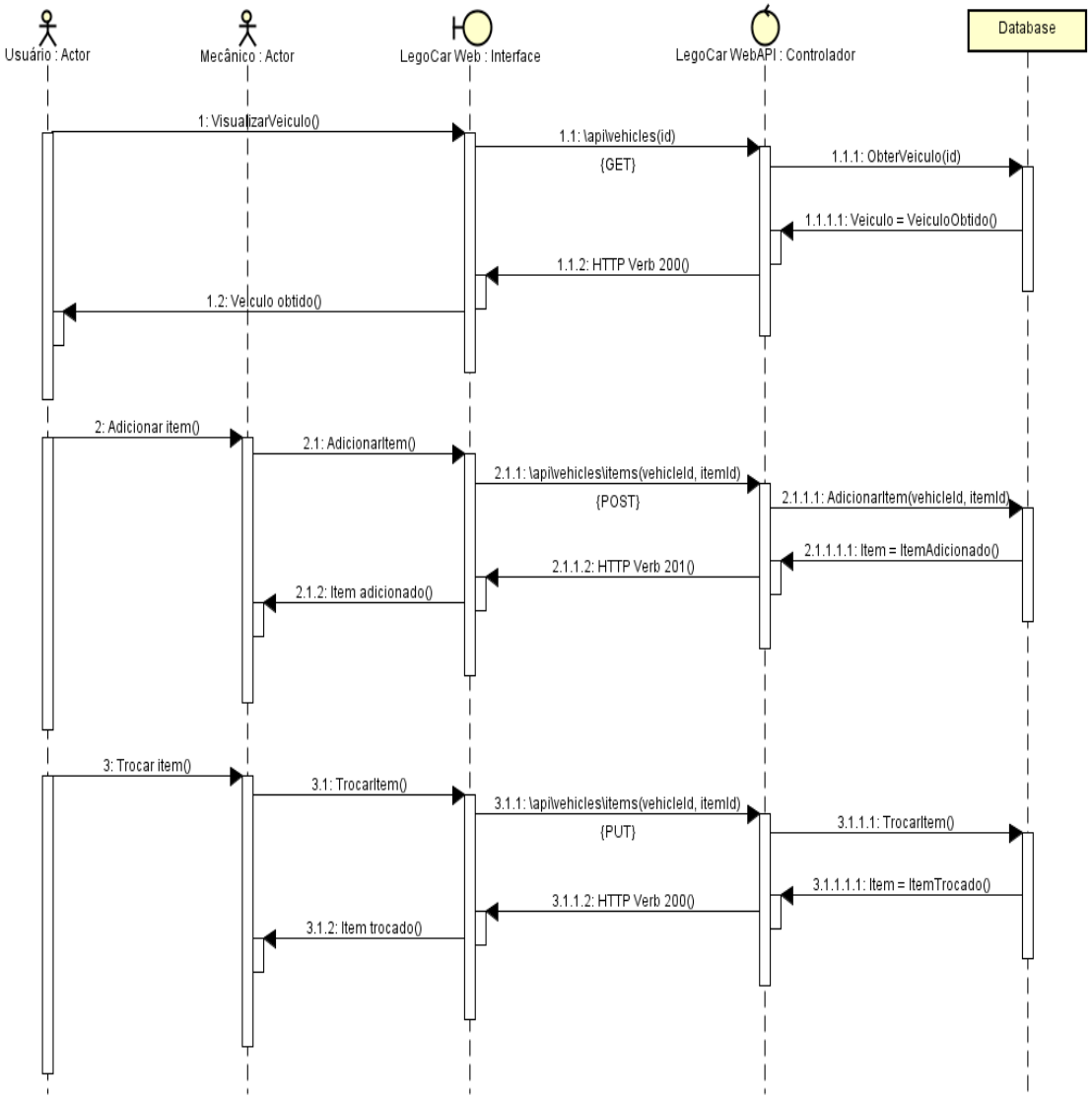
Fonte: João Paulo Frezzarin

Figura 10 - DIAGRAMA DE SEQUÊNCIA (Criação da conta e cadastro de veículos)



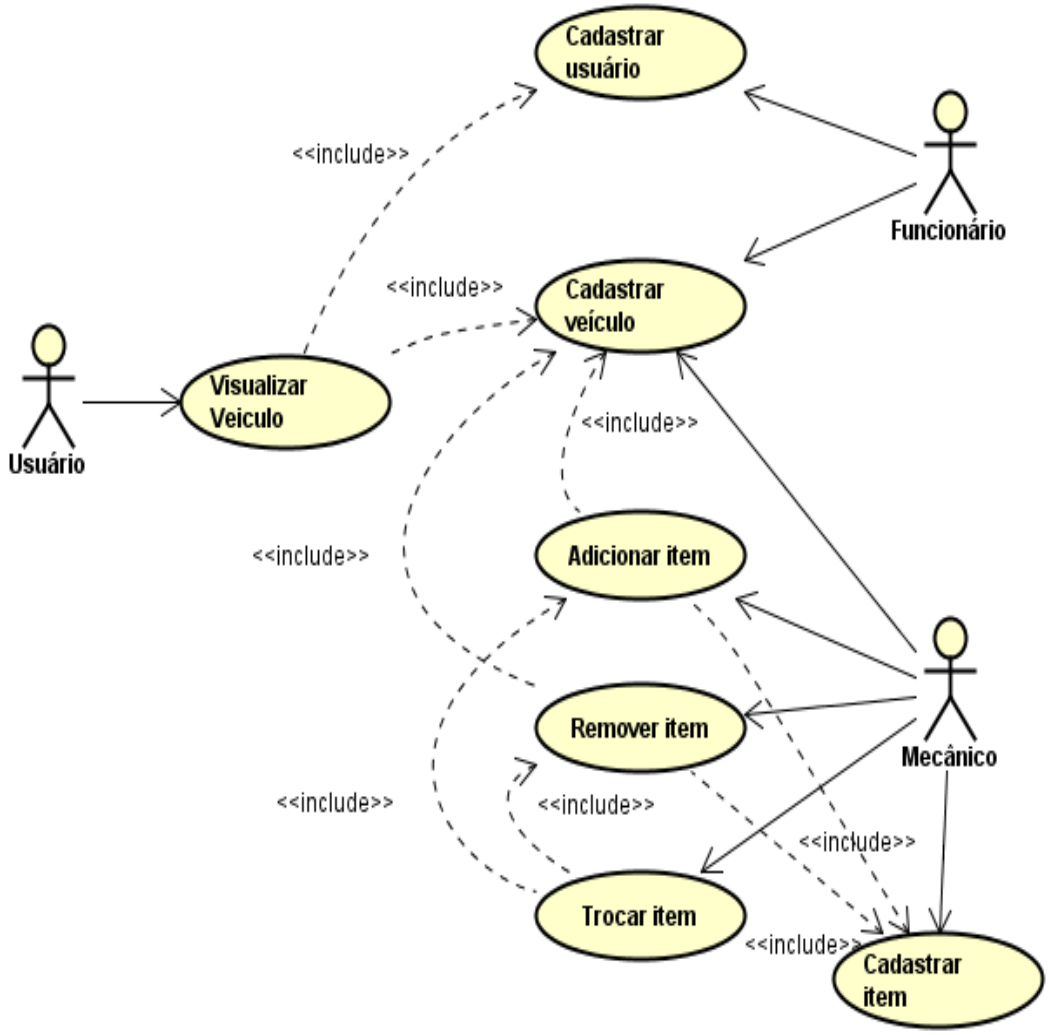
Fonte: João Paulo Frezzarin

Figura 11 - DIAGRAMA DE SEQUÊNCIA (



Fonte: João Paulo Frezzarin

Figura 12 - DIAGRAMA DE CASO DE USO



Fonte: Pedro Guilherme de Almeida Leme

4. MODELAGEM DO BANCO DE DADOS

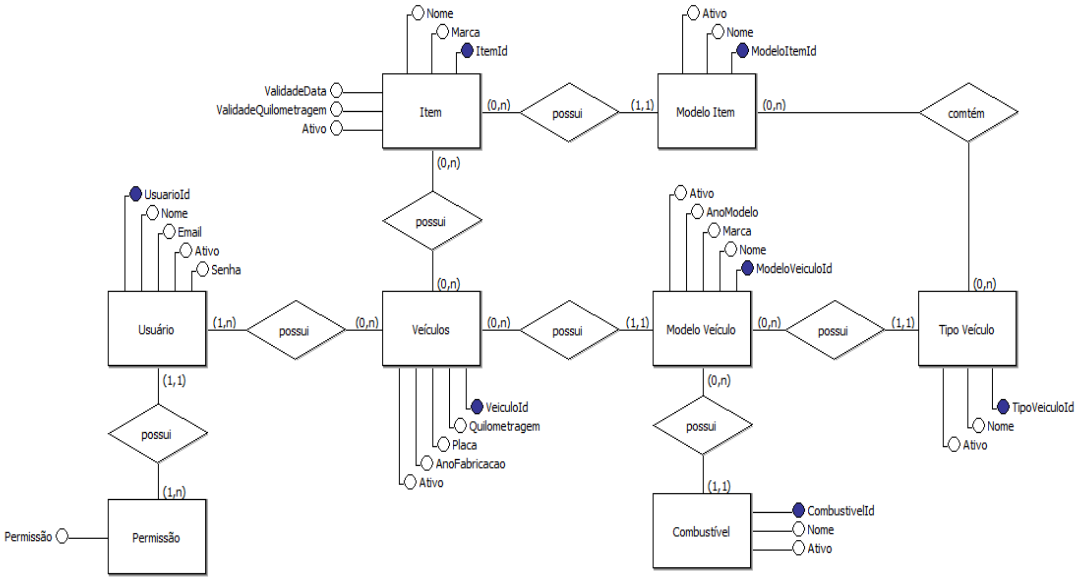
Podemos entender a modelagem como um dos momentos mais críticos no processo de desenvolvimento de um software. É nessa fase que deve-se entender de maneira precisa as necessidades do cliente/requisitante.

Todo e qualquer erro durante a modelagem pode comprometer a usabilidade do sistema, podendo causar na necessidade de um possível retrabalho, aumentando o custo do processo de desenvolvimento.

A modelagem só pode acontecer depois que os requisitos são estabelecidos. Com isso temos a definição do Modelo Conceitual e do Modelo Lógico.

O modelo conceitual desenha as relações entre as áreas ou usuários do sistema para um modelo teórico. Nessa fase entra em cena o uso dos diagramas da UML como auxílio.

Figura 13 - Modelo Conceitual para o sistema LegoCar

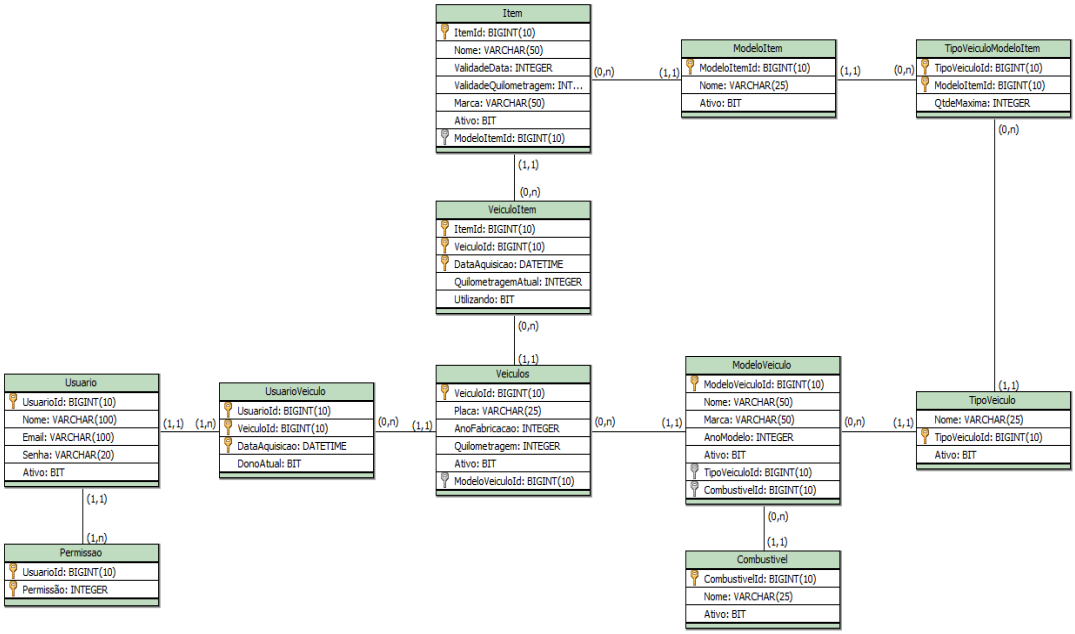


Fonte: João Paulo Frezzarin

Por sua vez, o modelo lógico implementa algumas regras ou restrições ao desenho, como as chaves primárias, que não podem ter duplicatas.Um cadastro de RG de clientes por exemplo, não podemos ter dois clientes com um mesmo Registro.

Temos também os relacionamentos entre as entidades, que são chamados de chaves estrangeiras. Podemos citar como exemplo um cadastro de vendas.

Figura 14 - Modelo Lógico do sistema LegoCar



Fonte: João Paulo Frezzarin

5. DESENVOLVIMENTO

Para o desenvolvimento desse trabalho foram utilizadas algumas ferramentas, tais como HTML, CSS, JavaScript, Bootstrap, .NET, JSon.

Vamos entender os conceitos utilizados para o desenvolvimento do sistema:

Front-end

É a primeira camada que nos deparamos ao acessar um site ou até mesmo um sistema web. Refere-se às linguagens client-side, que são linguagens onde apenas o navegador é capaz de 'entender', e são processadas pelo browser.

A estrutura do Front-End é composta geralmente por HTML, CSS, JavaScript e Flash.

HTML

Foi criado por Tim Berners-Lee, com o intuito da comunicação ser expandida entre ele e o seu grupo de colegas. A linguagem passou a ser utilizada para formar uma espécie de rede pública de sua época.

HTML é uma linguagem de marcação, ou seja, nós marcamos os elementos para mostrar o que a página deve exibir e o que não deve.

CSS ou CascadingStyleSheets

Define como os elementos que compõe determinada página serão exibidos. Tem como princípio fazer separação do conteúdo, da interatividade ou da apresentação de um site ou aplicação web. Tem um papel importante na camada da apresentação, controlando fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos entre outros.

Os benefícios do CSS são o controle da interface em documentos diferentes, controle de diferentes interfaces para diferentes dispositivos, precisão para manter a mesma interface para diferentes navegadores, menor consumo de banda para usuário e servidor, entre outros.

JavaScript

É uma linguagem orientada a objetos que pode ser processada pelo próprio navegador. Com o JavaScript é possível criar efeitos especiais para as páginas da web, além de ser mais interativa com os usuários. Porém não devemos confundir JavaScript com a linguagem Java. O nome original era LiveScript, porém quando a Netscape adicionou suporte à tecnologia Java em seu navegador (Applets), o nome foi trocado para JavaScript.

Bootstrap

Provê de componentes, tais como o css, sass, less e js, prontos para você utilizar na sua aplicação.

O Bootstrap depende basicamente de 2 arquivos:

bootstrap.css

bootstrap.js

O bootstrap.js é baseado em jQuery, então para que ele funcione você precisa ter o jQuery incluído no projeto.

O framework é utilizado por ter muitos plugins e blibliotecas customizadas, facilitando muito nos desenvolvimentos.

Back-end

O profissional trabalha na "parte de trás", nos bastidores da aplicação, ao lado do servidor. Enquanto o front-end trabalha no visual, o back-end age

nos bastidores, buscando informações, dados e arquivos que serão exibidos por exemplo. É conhecido também como serve-side.

JSon

É basicamente um formato de troca de dados entre sistemas. Significa JavaScriptObjectNotation, porém isso não significa que possa ser utilizado apenas com JavaScript. Pode ser comparado com o formato XML.

Podemos visualizar a diferença entre XML e JSon abaixo.

XML

1id>

Alexandre Gamanome>

R. Qualquerendereco>

JSon

{"id":1, "nome":"Alexandre Gama", "endereco":"R. Qualquer"}

API (ApplicationProgramming Interface)

A Interface de Programação de Aplicações, em inglês, ApplicationProgramming Interface (API) é uma interface que proporciona a integração entre sistemas que possuem linguagem totalmente distintas, de maneira ágil e segura, segundo FERNANDES, André, 2018

Fernandes ainda reforça: “Elas [APIs] são uma forma de integrar sistemas, possibilitando benefícios como a segurança dos dados, facilidade no intercâmbio entre informações com diferentes linguagens de programação e a monetização de acessos”

Foi utilizado o padrão de programação API visando o benefício de comunicação facilitada, dando a possibilidade de reutilizar as mesmas funcionalidades do sistema para o desenvolvimento de outras frentes, tais como um aplicativo mobile do LegoCar.

.NET Core

O .NET Core é a mais nova plataforma de desenvolvimento criada pela Microsoft, anunciada em 2014. Diferentemente do .NET Framework, o Core veio com conceito de ser "open source" e "crossplatform" (SANTOS, Ciro, 2017), onde a plataforma é mantida atualmente pela Microsoft e pela comunidade em geral e consegue ser desenvolvida para ambientes Windows, Linux, Mac e até mesmo Mobile.

Segundo a Microsoft, 2018, existem alguns requisitos que fazem com que seja ideal desenvolver com o .NET Core ao invés do .NET Framework, tal como se sua aplicação necessite de alto desempenho e que seja escalonável. O sistema LegoCar necessita de alto desempenho e futuramente será um requisito que a aplicação se torne escalonável caso atingir um alto consumo de processamento.

REST (RepresentationalStateTransfer)

A representação de estado de transferência, em inglês RepresentationalStateTransfer (REST) é um modelo arquitetural de software que visa facilitar a comunicação entre sistemas distribuídos.

Alexandre Afonso, 2018, menciona que o REST é independente de tecnologias, os seja, é possível implementar este modelo arquitetural em qualquer linguagem, framework ou plataforma.

Fora isto ele também menciona que há alguns requisitos do REST, tais como que a aplicação deve possuir uma interface uniforme, que basicamente é um conjunto de operações bem definidas e padronizadas, ser *stateless,* que diz

que a aplicação não deve manter estado, toda e qualquer operação deve conter toda informação necessária para realizar o processamento, entre outros.

Todos estes requisitos acima foram seguidos no desenvolvimento da API do LegoCar, o que o denomina de API RESTful, que nada mais é que uma API que engloba os modelos arquiteturais REST.

Rodrigo Ferreira, 2017, menciona algumas boas práticas que podem ser seguidas para a construção de um Web Service REST tais como a utilização de URIs* legíveis, utilização de métodos do protocolo HTTP (GET, POST, PUT e DELETE) e representação de recursos globais como XML ou JSON.

Embora Ferreira menciona que estas boas práticas são para construção de Web Services, elas também foram utilizadas na construção da API RESTful do LegoCar, visto que elas são universais e podem ser seguidas para qualquer tipo de desenvolvimento.

*URIs: Utilizado para endereçar recursos (entidades a nível de código) dentro de uma aplicação RESTful.

Entity Framework

O Entity Framework (EF), segundo Macoratti, 2017, é uma ferramenta ORM (Object-relationalmapping, em português, Mapeamento de objeto relacional) que permite ao desenvolvedor trabalhar com dados relacionais na forma de objetos específicos do domínio.

Este framework foi desenvolvido pela Microsoft em 2008, fazendo parte do *servicepack* do .NET Framework 3.5. Ele veio como uma alternativa a utilização do ADO.NET *, visando trazer maior produtividade no desenvolvimento de aplicações, segundo Priscilla, 2013.

Esta ferramenta auxilia e facilita o desenvolvedor, fazendo automaticamente o mapeamento dos elementos da base de dados para entidades da aplicação. “*O uso de ORMs … auxilia na produtividade e o Entity Framework é um dos melhores frameworks neste quesito. Com o uso do*

mesmo você poderá aproveitar as facilidades do mapeamento objeto-relacional em sua aplicação, obtendo o máximo de produtividade na persistência e recuperação de dados" (Priscilla, 2013)

Existem três metodologias que podem ser seguidas ao desenvolver uma aplicação utilizando o EF: DatabaseFirst, ModelFirst e CodeFirst. No projeto do LegoCar foi utilizado a metodologia CodeFirst, que basicamente consiste somente ao desenvolvedor codificar as classes das entidades, e toda a criação da base e os relacionamentos das entidades fica sob responsabilidade do framework (Priscilla, 2013).

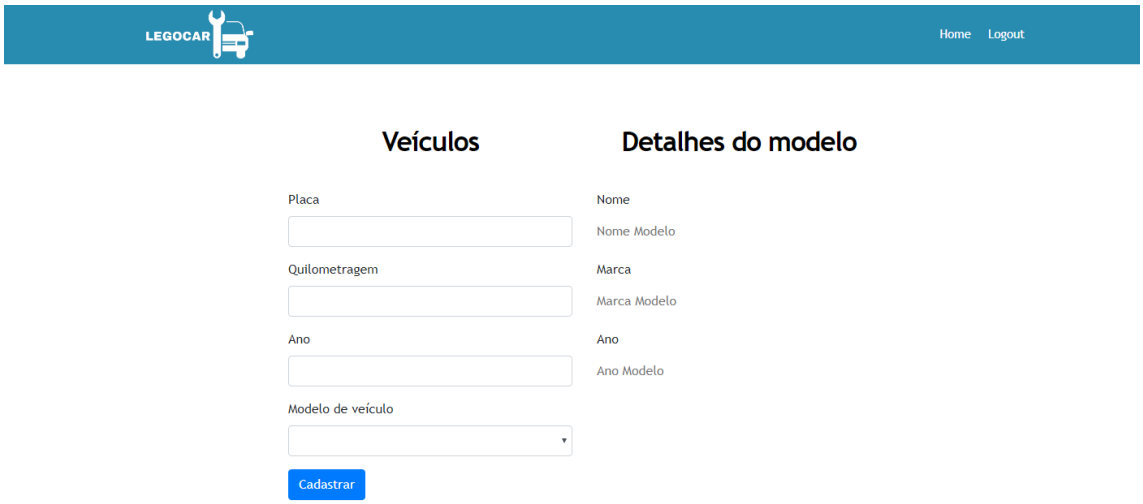
*ADO.NET: O ADO.NET fornece acesso consistente a fontes de dados como o SQL Server e o XML, e a fontes de dados expostas através do OLE DB e do ODBC.

5.1 TELAS DO SISTEMA

Figura 15 - Tela de login do sistema

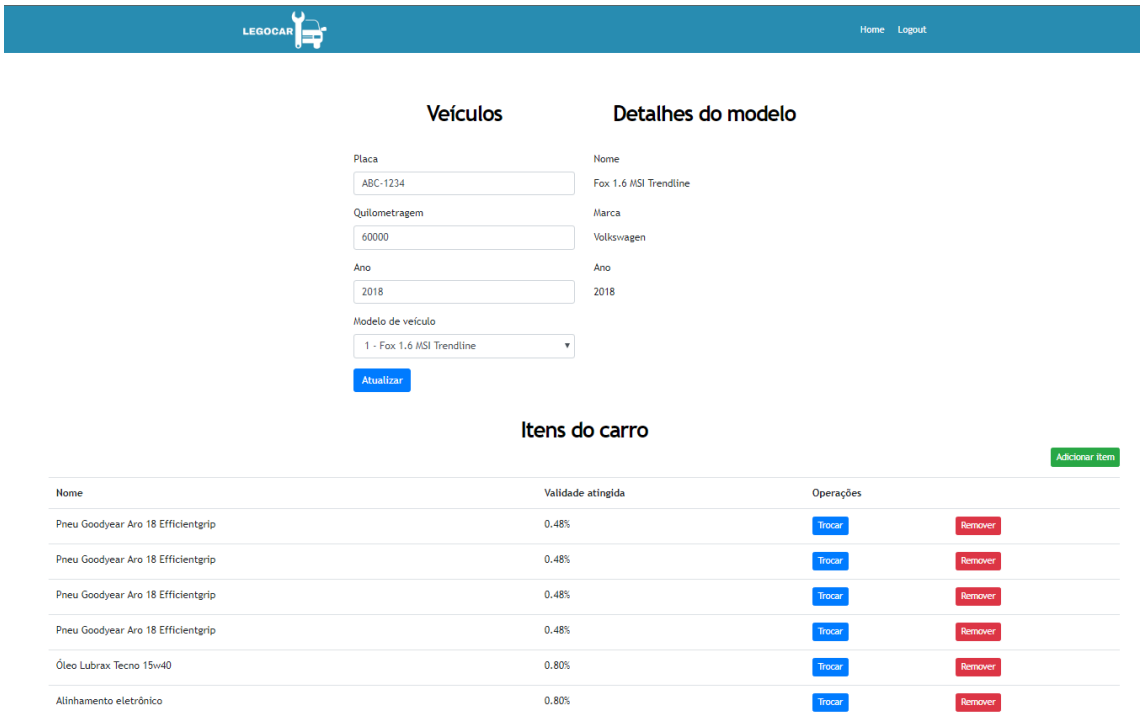
Fonte: Pedro Guilherme de Almeida Leme

Figura 16 - Tela de cadastro de veículo



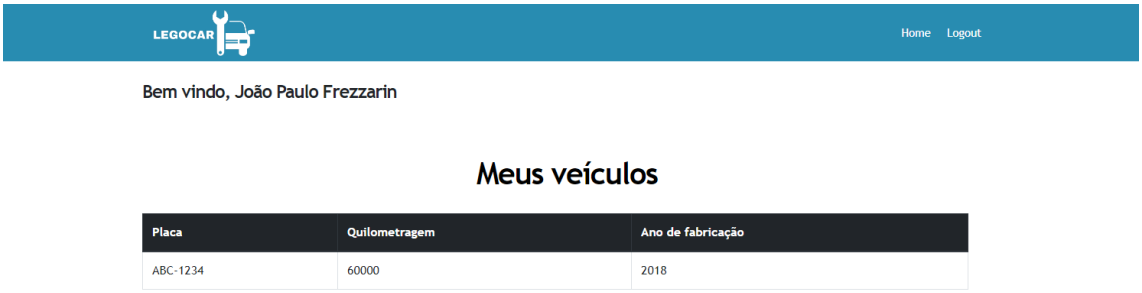
Fonte: Pedro Guilherme de Almeida Leme

Figura 17 - Tela de detalhes do veículo



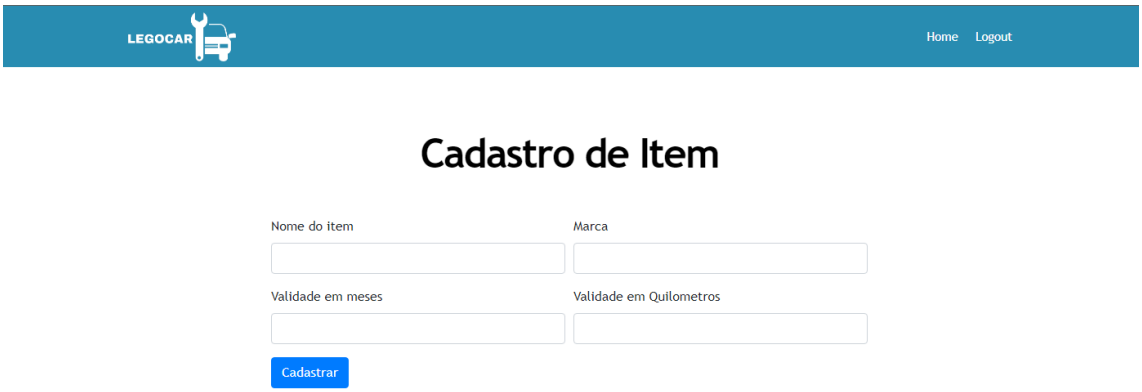
Fonte: Pedro Guilherme de Almeida Leme

Figura 18 - Tela Home do usuário



Fonte: Pedro Guilherme de Almeida Leme

Figura 19 - Tela de cadastro de itens



Fonte: Pedro Guilherme de Almeida Leme

5.2 PROBLEMAS ENCONTRADOS

Durante o desenvolvimento deste trabalho, algumas dificuldades foram encontradas, como quando tentava vincular um item a um veículo e o mesmo era cadastrado, porém os demais itens eram removidos. Outro erro encontrado era quando tentava realizar a troca de um item para um determinado veículo esse item não era adicionado porém o item antigo era removido. Ao deletar um item a operação não era concluída, logo a mesma ocasionava um erro ao adicionar um novo item.

Foi desenvolvido o sistema, e em alguns testes foi possível identificar que no momento de adicionar um item a determinado veículo ocorria uma remoção dos demais itens já cadastrados no mesmo. Isso ocorria pois para vincular um objeto em alguma classe é necessário que a classe exista. O erro foi sanado após notar que a verificação não estava sendo feita de maneira correta.

6. INFRAESTRUTURA DA TECNOLOGIA DA INFORMAÇÃO

Pode ser compreendida como o alicerce interno de uma empresa, sendo a base do negócio. Existe para dar assistência ao negócio em um âmbito global, lembrando que é apenas o alicerce e não a empresa em si.

As empresas precisam de competitividade no mercado, e para isso é necessário ter uma estratégia em que a TI terá a iniciativa de criar e melhorar a relação com o cliente.

Grandes empresas, por exemplo possuem informações de fornecedores, funcionários e clientes, armazenadas e confiadas a segurança da infraestrutura de TI. As decisões dessa infraestrutura são as mais importantes dentro das decisões relacionadas a tecnologia da informação.

6.1 INFRAESTRUTURA DE REDES

É o meio responsável por conectar todos os equipamentos de tecnologia de uma empresa, desde as estações de trabalho dos usuários até os servidores mais avançados, permitindo acesso aos recursos e sistemas instalados nos equipamentos de um datacenter.

Faz parte dessa infraestrutura todo o cabeamento que serve para interligar os equipamentos entre os diversos ambientes, tais como switches, roteadores e equipamentos de rede sem fio. Esses recursos são responsáveis pela segurança e também pelo desempenho de um ambiente de TI.

Se instalada corretamente a infraestrutura de redes garante retorno sobre o investimento aplicado na mesma.

6.2 LINK DE INTERNET

Na busca pela melhor performance das empresas especialmente devido à competitividade entre as organizações, as soluções para uma conexão mais veloz e segura se tornaram fundamentais.

Estar conectado é essencial para que os negócios mantenham-se de pé, bem como para o crescimento contínuo dos mesmos. Exatamente por essa maneira é preciso escolher qual a melhor estratégia e tipo de tecnologia deve ser empregada.

Com isso temos o link, que é desenvolvido para que as conexões ocorram de forma eficaz, dando uma qualidade de navegação melhor. A companhia não precisa competir pelo tráfego de dados com outros usuários da rede, isso dá um melhor rendimento.

Os links também facilitam na comunicação, fazendo com que todos os serviços possam fluir com alta performance.

6.3 IMPORTÂNCIA DA LICENÇA DE SOFTWARE

O licença do software nada mais é do que a autorização para o uso do mesmo. Ou seja, quando uma empresa compra licenças, ela está adquirindo autorização para utilizá-lo, assim como as especificações do que os usuários podem fazer com determinado software. Existem vários tipos de licenças de software, podendo variar desde o número de máquinas no qual o sistema pode ser executado, até no número de usuários.

Para as empresas, o mais comum é o licenciamento visando o número de usuários que irão desprover das funcionalidades do software adquirido.

Utilizar um sistema original garante segurança a empresa em si, pois softwares piratas podem conter vulnerabilidades consequentes da falta de atualizações e suporte.

Para adquirir a licença certa de determinado software é necessário que antes saiba quais módulos e funcionalidades do mesmo farão parte do gerenciamento do negócio. Certificar que a licença foi feita para organizações que tenham comoobjetivo fins lucrativos também é uma boa ideia.

7. CONSIDERAÇÕES FINAIS

Neste trabalho foi abordada uma forma para o gerenciamento de veículos, e com base na análise levantada foram encontrados alguns problemas em relação a forma que a maioria das pessoas cuida ou até mesmo deixa de cuidar de seu meio de transporte. Como solução foi desenvolvido um sistema (Legocar) que gerencia de forma totalmente prática e fácil as manutenções obrigatórias que devem ser feitas no automóvel.

Esse trabalho teve como foco principal o desenvolvimento de uma aplicação web, no qual foi possível aplicar as técnicas e linguagens de programação para internet e dispositivos móveis, no sistema desenvolvido o acesso a todos os recursos se efetiva através de aplicações que possibilitam a navegação na internet.

7.1 TRABALHOS FUTUROS

Como trabalhos futuros, pode-se apontar:

º Fazer com que o sistema torne-se uma rede social. Isso ajudaria na troca de informações entre as pessoas, seja sobre peças, modelos de veículos e até mesmo sobre onde fazer a manutenção com um preço mais acessível.

º Desenvolver um aplicativo mobile para o Legocar.

º Migrar o site para Angular.

REFERÊNCIA BIBLIOGRÁFICA

GUEDES, Gilleanes T. A. **UML Uma abordagem prática**. [*S. l.*]: Novatec, 2018.

SOMMERVILLE, Ian. **Engenharia de Software. 8. ed**. São Paulo: Pearson Addison Wesley, 2007.

SANTOS, Ciro. **.NET CORE, UMA VISÃO PRÁTICA**. [*S. l.*], 2017. Disponível em: https://www.concrete.com.br/2017/08/23/net-core-uma-visao-pratica. Acesso em: 1 fev. 2019.

RIBEIRO, Gabriella Fonseca. **Os principais diagramas da UML**. [*S. l.*], 2011. Disponível em: https://www.profissionaisti.com.br/2011/07/os-principais-diagramas-da-uml-resumo-rapido/. Acesso em: 23 jan. 2019.

Entenda o que é a modelagem de banco de dados. [*S. l.*], 2018. Disponível em: https://www.impacta.com.br/blog/2017/07/28/entenda-o-que-e-a-modelagem-de-banco-de-dados/. Acesso em: 27 dez. 2018.

Escolhendo entre o .NET Core e .NET Framework para aplicativos de servidor. [*S. l.*], 2018. Disponível em: https://docs.microsoft.com/pt-br/dotnet/standard/choosing-core-framework-server. Acesso em: 30 jan. 2019.

FERNANDES, André. **O que é API? Entenda de uma maneira simples**. [*S. l.*], 2018. Disponível em: https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples. Acesso em: 25 dez. 2018.

SATO, Priscila Mayumi. . *In*: SATO, Priscila Mayumi. **Entity Framework Tutorial**. [*S. l.*], 2013.Disponível em: https://www.devmedia.com.br/entity-framework-tutorial/27764. Acesso em: 21 dez. 2018.

INFRA-ESTRUTURA de Redes. [*S. l.*], [entre 2016 e 2019]. Disponível em: http://www.psmi.com.br/servicos/infraestrutura-de-ti/engenharia-de-redes. Acesso em: 18 jan. 2019.

APÊNDICE - CÓDIGO FONTE

Podemos visualizar o código javascript onde é feito a chamada do endpoint de obter os veículos do usuário, localizado na tela home do sistema. Ao obter os veículos do usuário, é construído o corpo de uma tabela e adicionado na elemento “tbBody”.

```
function carregarMeusVeiculos() {

    let userId = localStorage["userId"];

    fetch(`https://localhost:5001/api/users/${userId}/vehicles`, getFetchGetParam())
    .then((res) => res.json())
    .then((data) => {
        let output = '';
        data.forEach(function(veiculo){
            output += `
                <tr>
                    <td onclick='detail(${veiculo.vehicle.vehicleId})'>${veiculo.vehicle.licensePlate}</
                    <td onclick='detail(${veiculo.vehicle.vehicleId})'>${veiculo.vehicle.mileage}</td>
                    <td onclick='detail(${veiculo.vehicle.vehicleId})'>${veiculo.vehicle.manufactureYear
                </tr>
            `;
        });
        document.getElementById('tbBody').innerHTML = output;
    });
}
```

Abaixo segue o código da API em C# do próprio endpoint de obter os veículos do usuário. Nota-se que o método do endpoint “GetVehicles” obtém alguns Data Annotations como por exemplo “[HttpGet]”, “[Route]” e “[Authorize]” que definem, respectivamente, qual é o verbo HTTP a ser utilizado no endpoint (que neste caso é GET), a rota que irá compor a URI do recurso e a necessidade de estar autenticado no sistema para chamar o endpoint.

```
/// <summary>
/// Obtem-se os veículos do usuário.
/// </summary>
/// <param name="id">Identificador do usuário.</param>
/// <param name="currentOwner">Flag informando se é somente os veículos no qual o user é dono.</param>
/// <returns></returns>
[HttpGet]
[Route("{id}/vehicles")]
[Authorize("Bearer")]
[ProducesResponseType(typeof(UserVehicleJsonModel), 200)]
[ProducesResponseType(typeof(Error), 400)]
[ProducesResponseType(typeof(Error), 500)]
0 references
public async Task<IActionResult> GetVehicles([FromRoute]long id, [FromQuery]bool? currentOwner)
{
    if (!ModelState.IsValid || id <= 0)
    {
        return BadRequest(new Error { Message = "Atributos inválidos" });
    }

    try
    {
        var vehicles = await _userBusiness.GetVehicles(id, currentOwner);
        var vehiclesJsonModels = vehicles.Select(v => _mapper.Map<UserVehicleJsonModel>(v));
        return Ok(vehiclesJsonModels);
    }
    catch (LegocarException ex)
    {
        return UnprocessableEntity(new Error { Message = ex.Message });
    }
    catch (Exception ex)
    {
        return InternalServerError(new Error { Message = ex.Message });
    }
}
```

O endpoint irá buscar na base de dados todos os veículos referente ao usuário informado no parâmetro. Porém, antes de buscar os dados, o sistema fará algumas verificações básicas, como por exemplo, se o usuário realmente existe. Estas validações são feitas nas classes *Business*, que são responsáveis por validar as entidades e dizer se estão condizentes ou não para a operação ser processada, onde cada entidade possui sua própria classe.

```
1 reference
public async Task<IEnumerable<UserVehicle>> GetVehicles(long userId, bool? currentOwner)
{
    // Verifica se o usuário é válido.
    var user = await Get(userId);

    return await _userVehicleBusiness.GetVehicles(userId, currentOwner);
}
```

No método abaixo, ele irá buscar o usuário pelo método “Get()”, caso o usuário não existir, ele irá lançar uma exceção. Caso o usuário for obtido, é chamado o método “GetVehicles()” da *Business* da entidade “UseVehicle”.

```
5 references
public async Task<User> Get(long id)
{
    return await _userRepository.Get(id) ??
        throw new LegocarException("Usuário não encontrado");
}
```

Por fim, se a operação estiver válida, ele irá buscar na base pelos veículos daquele usuário. A função “GetVehicles()” da entidade “UserVehicle” irá montar uma expressão C# que será utilizada pelo framework do Entity Framework, que automaticamente irá montar uma query que será utilizada na consulta do banco de dados, como é exibido na imagem abaixo.

```
1 reference
public async Task<IEnumerable<UserVehicle>> GetVehicles(long userId, bool? currentOwner)
{
    // Expressão da query:
    // Obter todos os veículos onde o identificador do usuário seja igual a variável userId e
    // o que o dono atual seja igual a variável currentOwner
    Expression<Func<UserVehicle, bool>> where = uv =>
        uv.User.UserId == userId &&
        (currentOwner == null || uv.CurrentOwner == currentOwner.Value);

    var vehicles = await _userVehicleRepository.GetAll(where);

    // Removo toda referência de usuário nas entidades obtidas,
    // visto que o mesmo não será necessário.
    vehicles.AsParallel().ForAll(uv => uv.User = null);

    return vehicles;
}
```

A busca dos dados será realizada na classe *Repository*, que é responsável por exclusivamente por obter as funções básicas de DML realizadas internamente na base de dados. A imagem abaixo mostra o função “GetAll()”, que retorna uma listagem de entidade de acordo com a expressão informada por parâmetro.

```
5 references
public async Task<IEnumerable<TEntity>> GetAll(Expression<Func<TEntity, bool>> where)
{
    return await Context
        .Set<TEntity>()
        .Where(where)
        .ToListAsync();
}
```