UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATHEUS FELIPE MARRA

ULISSES JOSE PERALTA DOS SANTOS

GESTÃO DE ESTACIONAMENTOS ATRAVÉS DE APLICAÇÃO *WEB*

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2017

MATHEUS FELIPE MARRA
ULISSES JOSE PERALTA DOS SANTOS

GESTÃO DE ESTACIONAMENTOS ATRAVÉS DE APLICAÇÃO *WEB*

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Richard Duarte Ribeiro

PONTA GROSSA
2017

TERMO DE APROVAÇÃO

GESTÃO DE ESTACIONAMENTO ATRAVÉS DE APLICAÇÃO *WEB*

por

MATHEUS FELIPE MARRA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 06 de novembro de 2017 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Richard Duarte Ribeiro
Prof.(a) Orientador(a)

_____
Clayton Kossoski
Membro titular

_____
Rafael dos Passos Canteri
Membro titular

_____
Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Mauren Louise Sguario
Coordenadora do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

TERMO DE APROVAÇÃO

GESTÃO DE ESTACIONAMENTO ATRAVÉS DE APLICAÇÃO *WEB*

por

ULISSES JOSE PERALTA DOS SANTOS

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 06 de novembro de 2017 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Richard Duarte Ribeiro
Prof.(a) Orientador(a)

_____
Clayton Kossoski
Membro titular

_____
Rafael dos Passos Canteri
Membro titular

_____
Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Mauren Louise Sguario
Coordenadora do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Dedico este trabalho à minha namorada, que foi minha grande motivadora e a responsável pelo tema. O resultado do presente trabalho deve-se ao seu grande apoio e dedicação.

- Matheus Felipe Marra

AGRADECIMENTOS

Primeiramente agradecemos a Deus, por ter nos dado forças e coragem para chegarmos até aqui.

Aos nossos pais e familiares, por terem investido e acreditado em nós por todos estes anos.

Aos nossos amigos e colegas da universidade, por bons anos de amizade e companheirismo.

Aos professores da universidade, em especial ao nosso orientador, por sua dedicação ao nosso trabalho.

RESUMO

MARRA, Matheus Felipe; SANTOS, Ulisses Jose Peralta dos. **Gestão De Estacionamentos Através De Aplicação *Web*.** 2017. 71 folhas. Trabalho de Conclusão de Curso Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa. 2017.

Este trabalho apresenta a implementação de uma aplicação *Web* para controle de estacionamentos, através do qual é possível manter registros de estacionamentos anteriores, gerar relatórios, manter cadastros de clientes e gerar impressões de cupons de estacionamentos, sendo compatível com grande parte dos dispositivos atuais no mercado. O desenvolvimento de tal aplicação consiste no levantamento de requisitos juntamente a um administrador de uma empresa do ramo de estacionamentos e no desenvolvimento da aplicação, que se divide em duas etapas: servidor e cliente. Para o desenvolvimento do servidor, foi utilizado a linguagem PHP, juntamente com o *framework Laravel*. Para o desenvolvimento do cliente utilizou-se as linguagens HTML, CSS e *Javascript*, através do *framework Bootstrap* e da biblioteca *JQuery.*

Palavras-chave: Estacionamento. Sistema de Gestão. Aplicação *Web*.

ABSTRACT

MARRA, Matheus Felipe; SANTOS, Ulisses Jose Peralta dos. **Parking Management Through Web Application**. 2017. 71 Pages. Work of Conclusion Course (Graduation in Analysis and Systems Development) - Federal Technology University - Paraná. Ponta Grossa, 2017.

This work presents the implementation of a Web application for parking lot control, which can maintain records, generate logs, keep clients data and produce parking receipt print, being compatible with most current devices. The development of this application consists in requirements gathering alongside an branch company administrator and the application development, which can be separated in two stages: server and client. For the server development, the language PHP was used, alongside the framework Laravel. For the Client development, the languages HTML, CSS and Javascript were used, through the framework Bootstrap and the library JQuery.

Keywords: Parking. Management System. Web Application.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

LISTA DE ABREVIATURAS

CMD	*Command*

LISTA DE SIGLAS

AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
ASP	*Active Server Pages*
ASCII	*American Standard Code for Information Interchange*
CEP	Código de Endereçamento Postal
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro de Pessoa Física
CSS	*Cascading Style Sheets*
HTML	*Hypertext Markup Language*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
JSP	*JavaServer Pages*
JSON	*JavaScript Object Notation*
MVC	*Model View Controller*
PHP	*Hypertext Preprocessor*
SGBDR	Sistema De Gerenciamento De Bancos De Dados Relacionais
SQL	*Structured Query Language*
TCO	*Total Cost Of Ownership*
URI	*Uniform Resource Indetifier*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

No contexto do século XXI, as mudanças no mundo dos negócios acontecem com uma velocidade antes inimaginável. Ao mesmo tempo em que surgem novas ameaças, como novos produtos ou novos concorrentes, também é possível identificar novas oportunidades para investimento e crescimento. Com isso, a tecnologia tornou-se uma poderosa ferramenta no mundo da competitividade (DEITOS, 2002).

A informática se tornou uma ferramenta fundamental para os mais diversos tipos de organizações. O acesso mais facilitado à tecnologia fez com que se tornasse inevitável a utilização de *softwares*, os quais vieram a acrescentar muito no atendimento das necessidades das empresas como um todo, inclusive nas micro e pequenas empresas (VIDAL, 1995).

Dentro do setor de serviços, o qual apresenta posição de destaque na economia nacional, pode-se perceber a relevância das micro e pequenas empresas. Essas organizações correspondem a 98% do total de empresas classificadas no setor terciário econômico, segundo dados do Serviço de Apoio Às Micro e Pequenas Empresas (SEBRAE, 2017).

Como afirmado por Rocha et al. (2010), com a ascendente demanda de informações solicitadas pelos gestores e pelo grande fluxo de dados, a utilização de *softwares* para gerenciar, filtrar estes dados e transformá-los em informações relevantes tornou-se uma necessidade. Este processo ocorre com cada vez mais frequência conforme a tecnologia evolui.

Conforme uma pesquisa realizada por Maesta (2012) com empresas de faturamento entre R$ 240.000,00 e R$ 2.400.000,00, relatou-se que 53% delas utilizavam sistemas de gerenciamento, 21% ainda utilizavam a ferramenta *Microsoft Excel*. Cerca de 21% possuíam um controle manual com anotações e 5% trabalhavam sem qualquer tipo de ferramenta.

Empresas do ramo de estacionamentos tem se tornado muito presente na maioria das cidades, onde grande parte das organizações enquadram-se como micro e pequenas empresas, que correspondem à maior parte das instituições do mercado (MAESTA, 2012).

Além disso, observa-se que, dentre os sistemas existentes para este ramo de atividade, a maior parte encontra-se com valores elevados, baixa usabilidade e indisponibilidade para acesso em diversos dispositivos atuais do mercado.

Através destas notações, optou-se pelo desenvolvimento de um sistema de gestão que suprisse os requisitos de valor, usabilidade e facilidade de acesso, não atendidos pelos sistemas já existentes. Foi encontrado um prestador de serviço atuante no ramo e, posteriormente, foi realizada uma entrevista com o mesmo, visando esclarecer as necessidades reais que um estabelecimento do tipo possui.

Com a entrevista, pôde-se estabelecer o objetivo de criar um sistema de aplicação *Web*, que fosse possível ser acessado de qualquer dispositivo que possuísse acesso à *Internet* e tivesse um navegador *Web* atualizado.

Visando o melhor desenvolvimento possível da aplicação *Web*, realizou-se pesquisas para escolher as ferramentas necessárias para tal. Definiu-se que o melhor caminho seria utilizar a linguagem de programação PHP[1] (Hypertext Preprocessor) como estrutura principal do projeto com o auxílio do *framework Laravel*[2]. Como banco de dados, foi optado pela utilização do MySQL[3] *(Structured Query Language)*, por ser gratuito e trabalhar bem com o PHP.

Além disso, também foram usados no projeto a linguagem HTML[4] *(HyperText Markup Language),* para a estruturação da aplicação, o CSS[5] *(Cascading Style Sheets)*, para a formatação do *layout* e o *JavaScript*[6] para trabalhar o comportamento do sistema do lado do cliente. Como ferramentas de auxílio ao *JavaScript*, foram usados os *frameworks Bootstrap*[7] e *jQuery*[8], que possuem ferramentas preparadas para o auxílio de uma aplicação *Web*.

1.1 JUSTIFICATIVA

O ramo de estacionamentos tem crescido juntamente com o desenvolvimento das cidades, motivado pelo grande volume de veículos e a baixa disponibilidade de

[1] http://php.net/
[2] https://laravel.com/
[3] https://www.mysql.com/
[4] https://www.w3.org/html/
[5] https://www.w3schools.com/css/
[6] https://www.javascript.com/
[7] https://getbootstrap.com/
[8] https://jquery.com/

vagas nas ruas dos grandes centros, conforme afirma Garcia (2014). Assim como a grande maioria das empresas, os estacionamentos necessitam de organização e agilidade em seus serviços, que, assim como Carvalho (2000) afirma, podem ser atingidos através de um *software* de gestão.

Os *softwares* disponíveis atualmente para gerenciar esse ramo de atividade possuem, em sua maioria, deficiências como:

- Baixa usabilidade.
- Difícil acesso ao sistema, pois a maior parte somente disponibiliza acesso através do computador em que o sistema se encontra instalado.
- Valores elevados.
- Ambientes antiquados.

Dessa forma, decidiu-se desenvolver um sistema para gerenciamento de estacionamentos que atendesse as necessidades para o correto funcionamento e suprisse as deficiências citadas.

1.2 OBJETIVOS

A seguir, são descritos o objetivo geral e os específicos do presente trabalho.

1.2.1 Objetivo geral

Desenvolver uma aplicação *Web*, capaz de ser acessada por computador, *tablet* ou *smartphone*, para gerenciamento de estacionamentos.

1.2.2 Objetivos específicos

- Analisar as necessidades e características necessárias de um sistema de gerenciamento de estacionamentos;
- Pesquisar técnicas e conceitos de desenvolvimento *Web* relevantes para o desenvolvimento do trabalho;
- Implementação das funcionalidades do sistema em páginas responsivas que possam ser acessadas via computador, *tablet* e *smartphone*.

1.3 ORGANIZAÇÃO DO TRABALHO

A estrutura geral do trabalho divide-se em cinco capítulos. O capítulo 2 trata toda a parte de embasamento teórico necessário para o desenvolvimento da aplicação, bem como as linguagens de programação utilizadas, *frameworks* e ferramentas de auxílio.

O capítulo 3 aborda a respeito da metodologia. Nesse capítulo é apresentado as ferramentas usadas no desenvolvimento do projeto, o levantamento de requisitos e o estudo feito para conhecer as ferramentas existentes no mercado e que atuam no mesmo ramo.

No capítulo 4 é descrito o desenvolvido do projeto e suas principais características.

Por fim, o último capítulo discorre a respeito das considerações finais do desenvolvimento do projeto, bem como as recomendações para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo é destinado para a discussão sobre sistemas de processamento de transações, aplicação *Web*, desenvolvimento da aplicação *Web*, *frameworks* utilizados no projeto, bancos de dados relacional e as ferramentas utilizadas no desenvolvimento do projeto.

2.1 SISTEMAS DE PROCESSAMENTO DE TRANSAÇÕES

Um sistema de processamento de transações tem como foco ser usado por funcionários de nível operacional de uma empresa. Dentre as transações fornecidas por estes sistemas, estão pedidos de clientes, pedidos de compras, recebimentos, cartões ponto, faturas e pagamentos de clientes (STAIR; REYNOLDS, 2012).

As atividades de processamento incluem coleta, edição, correção, manipulação, armazenagem de dados e a produção de documentos. Se a atividade de processamento de transações for devidamente realizada, os registros da empresa são atualizados para refletir o *status* da operação ao mesmo tempo em que a transação é processada (STAIR; REYNOLDS, 2012).

As aplicações de processamento de transações normalmente suportam um ambiente com diversos dispositivos, que podem enviar consultas e atualizar a aplicação. Baseado nisso, pode-se dizer que a aplicação mantém a base de dados atualizada (GRAY; REUTER, 1993).

2.2 APLICAÇÃO *WEB*

Segundo Nations (2016), uma aplicação *Web* utiliza o navegador como cliente. O “cliente”, no modelo “cliente-servidor”, refere-se ao programa utilizado na execução da aplicação pelo usuário. O autor define também que o “cliente” é a aplicação através da qual é possível inserir a informação e “servidor” é o que processa e armazena a informação.

Outra vantagem de uma aplicação *Web* é o fato dela ser compatível com inúmeros navegadores. Além disso, o desenvolvedor controla as atualizações no sistema, podendo essas serem atualizações de manutenção, que possuem pequeno

impacto, ou atualizações críticas, com a vantagem de poderem ser instaladas com prontidão (OFFUTT, 2002).

Segundo Conte et al. (2005), uma aplicação *Web* nada mais é do que um *software* tradicional, o qual necessita da infraestrutura *Web* para ser executado. Essa infraestrutura pode ser designada para atender uma rede local ou para ser acessada pela *Internet* através de um domínio.

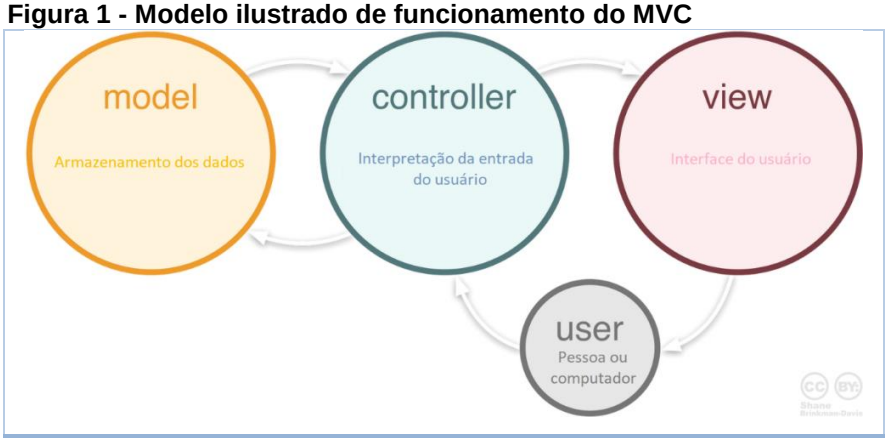
Portanto, pode-se concluir que uma aplicação *Web* possui uma maior facilidade de acesso em relação à programas locais. O usuário não necessita instalar um novo programa, precisaria apenas utilizar o navegador de *Internet* (NATIONS, 2016).

Uma das etapas para a criação de uma aplicação web é a escolha do padrão de arquitetura de software utilizado no desenvolvimento, que será abordado na próxima seção.

2.3 MVC

Burbeck (2017) explica que no modelo MVC é possível separar a interface do usuário, a chamada de um método e a interação com o banco de dados em três classes distintas, chamadas respectivamente de *View, Controller e Model*. As *Views* gerenciam a saída gráfica do sistema, as *Controllers* interpretam as interações do usuário e as *Models* cuidam do comportamento dos dados, respondendo as requisições e instruções (BURBECK, 2017).

Figura 1 - Modelo ilustrado de funcionamento do MVC



Fonte: Adaptado de Zeeshan (2017)

A Figura 1 apresenta um exemplo deste modelo, onde é possível notar que o usuário interage com a *Controller* através da *View*. A *Controller* por sua vez irá interagir com a *Model*, que retorna o resultado para a *Controller*. Por fim, a *Controller,* repassa a alteração para a *View.*

2.4 FERRAMENTAS DE CRIAÇÃO DA APLICAÇÃO *WEB*

Marcondes (2005) afirma que para o desenvolvimento de um site ou aplicação web, faz-se necessário o conhecimento de ao menos uma linguagem de programação voltada à hipertextos. Algumas dessas linguagens são abordadas nas próximas subseções.

2.4.1 HTML

Raggett et al. (2017) definem o HTML, sigla para linguagem de marcação de hipertexto, como sendo o idioma de edição da WWW *(World Wide Web*), referindo-se à versão 4 do HTML, que, tem como adicional de suas características o suporte a páginas de estilo, linguagens de *script*, melhor suporte a impressão e opções de multimídia. Os autores afirmam ainda que, para efetuar a publicação de dados para distribuição global, seria preciso um idioma compreendido por todos os dispositivos. Este idioma é o HTML (RAGGETT et al., 2017).

O HTML foi desenvolvido, originalmente, por Tim Berners-Lee e popularizou-se no navegador *Mosaic*. Durante os anos de 1990, com a grande popularização da *Internet*, o HTML obteve grande destaque (RAGGETT et al, 2017).

Possuindo um grupo de tags predefinidas, com a função de organizar a informação a ser transferida por meio de páginas web, o HTML é uma linguagem que pode ser interpretada por diferentes navegadores de diversas plataformas (W3SCHOOLS, 2017). Raggett et al. (2017) reconhecem a importância dessa interoperabilidade, pois além de homogeneizar o desenvolvimento, evita que hajam linguagens incompatíveis.

Um documento HTML é composto por três partes principais: uma linha contendo a versão do HTML, uma seção de cabeçalho para declaração (delimitado pelas *tags* “<head>” e “</head>”) e o corpo que contém o conteúdo do documento

(delimitado pelas *tags* “<body>” e “</body”>) (RAGGETT et al., 2017). O Quadro 1 exemplifica documento HTML simples:

Quadro 1 - Exemplo de código HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
     <TITLE>My first HTML document</TITLE>
  </HEAD>
  <BODY>
     <P>Hello world!</P>
  </BODY>
</HTML>
```

Fonte: Autoria própria

2.4.2 PHP

O PHP é uma linguagem de *script*, de código aberto, do lado do servidor, desenhada especificamente para a *Web*, onde é possível executar este *script* através de uma página HTML, cada vez que a mesma for visitada (WELLING; THOMPSON, 2001).

Achour et al. (2017) explicam que o código PHP é executado no servidor, de forma a gerar os dados que serão enviados para o navegador, que por sua vez recebe os dados sem saber o código fonte. Os autores afirmam que essa é uma característica diferencial do PHP, pois permite a segurança de que o código fonte do servidor estará ocultado do usuário da página *Web,* garantindo assim a confidencialidade dos códigos do servidor.

O PHP foi concebido por Rasmus Lerdorf, que criou o projeto sozinho em 1994. Mais tarde o projeto foi adotado por outros desenvolvedores e passou por três grandes reescritas até chegar ao produto que se conhece e usa hoje em dia. Em janeiro de 2001, cerca de cinco milhões de domínios já utilizavam o PHP (WELLING; THOMPSON, 2001).

Diversos bancos de dados são suportados pelo PHP, dentre eles, pode-se citar o *PostgreSQL, Sybase, Oracle, SQL Server,* MySQL, dentre outros. Todos os bancos de dados que têm suporte do PHP possuem uma série de comandos específicos para eles, mas é possível utilizar bancos de dados que não tem suporte

no PHP acessando os seus dados via comandos ODBC *(Open Database Connectivity)* (NIEDERAUER, 2004).

O PHP pode ser incorporado ao HTML, isso significa que se pode escrever o código de PHP dentro do código HTML. Converse e Park (2003) afirmam que o PHP é estável e, nesse contexto, significa que o servidor que gerencia o PHP não precisará ser reiniciado frequentemente e que a linguagem não sofre alterações a ponto de ser totalmente incompatível de uma versão para outra.

Ainda segundo os autores, apesar do PHP não ser o ideal para todos os problemas, ele possui um grande número de vantagens, destacando-se por sua leveza, capacidade, facilidade de uso e confiabilidade, além de oferecer o melhor tipo de conectividade para os principais servidores *Web* (CONVERSE; PARK, 2003).

2.4.3 CSS

Uma folha de estilo é formada por uma ou mais regras que definem a forma que um elemento ou conjunto de elementos devem ser apresentados. Para entender o CSS é necessário estar familiarizado com todas as partes de uma regra. Cada regra seleciona um elemento e diz como este deve se apresentar (ROBBINS, 2012).

Como afirma DeBolt (2006), as regras de estilização podem ser integradas de diversas formas com o conteúdo das páginas *Web*. O Quadro 2 contém duas regras, a primeira deixa todos os elementos da *tag* “<h1>” em verde e a segunda diz que o tamanho da fonte deve ser pequeno e o seu estilo “*sans-serif*”:

Quadro 2 - Regras CSS

```
h1 { color: green; }
p  { font-size: small; font-family: sans-serif;
}
```

Fonte: Autoria própria

O CSS possibilita a adaptação das páginas para vários tipos de dispositivos, de diferentes telas e resoluções. Além disso, é independente do HTML, viabiliza que os estilos sejam modificados de forma simplificada e distribuídos entre as páginas (W3SCHOOLS, 2017).

A primeira versão do CSS (o CSS nível 1 ou CSS1) foi lançada oficialmente em 1996 e incluía suporte para alteração de estilo, cor e espaçamento de fonte.

Contudo, a adoção do CSS pelos desenvolvedores demorou alguns anos, devido à falta de suporte confiável dos navegadores (ROBBINS, 2012). O autor ainda afirma que muitos desenvolvedores usam recursos do CSS aprimorados mesmo esses não sendo suportados por todos os navegadores, contanto que o retorno seja aproveitável e não haja perda de conteúdo (ROBBINS, 2012).

Na terminologia do CSS, as duas partes principais do documento são o seletor e a declaração, que podem vir em blocos. O seletor identifica os elementos que serão afetados, enquanto a declaração define as regras de como estes elementos serão renderizados na página. A declaração em si é separada em duas partes: a propriedade e o valor, que são separados por dois-pontos e um espaço (ROBBINS, 2012). Esta definição pode ser observada no Quadro 2.

2.4.4 *JavaScript*

JavaScript pode ser compreendido como uma linguagem de programação *Web*, como define Flanagan (2011). O autor afirma que todos os navegadores, incluindo os de dispositivos móveis e a grande maioria das páginas da *Internet,* fazem uso de *JavaScript*, o que a faz ser a uma das mais populares linguagens de programação.

O JavaScript atua nos navegadores juntamente com os códigos HTML e CSS. O HTML serve para desenvolver o conteúdo da página e o CSS serve para desenvolver a apresentação da página, enquanto *JavaScript* é usado para o desenvolvimento do comportamento da página (FLANAGAN, 2011).

O autor afirma ainda que todos os idiomas precisam possuir uma plataforma, biblioteca ou API *(Application Programming Interface)* que facilite o processo de desenvolvimento. O preceito do *JavaScript* é definir uma API mínima para se trabalhar com textos, datas, matrizes e expressões básicas, porém, não estão inclusas funções de entrada e saída de dados, ficando a cargo do ambiente em que o *JavaScript* está hospedado (FLANAGAN, 2011).

O JavaScript é executado localmente no navegador do usuário, e não em um servidor remoto. Devido a este fato, o navegador pode responder a ações de forma mais rápida. Além disso, o código JavaScript consegue detectar ações de um

usuário que o HTML não pode, tais como teclas pressionadas individualmente. (KYRNIN, 2012).

De acordo com Flanagan (2011), os programas desenvolvidos em *JavaScript* são escritos utilizando o conjunto de caracteres conhecido como *Unicode*, que é capaz de armazenar um grande número de caracteres. Isso faz com que ele virtualmente suporte a maior parte das linguagens usadas atualmente.

Ainda, o autor diz que “os programas de computador funcionam manipulando valores, como o número 3.14 ou o texto ‘Olá Mundo’”. Em *JavaScript*, pode-se dividir o conjunto de tipos de valores em tipos primitivos e tipos de objetos (FLANAGAN, 2011).

Os tipos primitivos são os números, textos, os valores booleanos[9], valor “nulo” e “indefinido”. Tudo o que não se encaixa nisso é considerado um objeto. Um objeto é um conjunto de propriedades, onde cada propriedade possui um nome e um valor, classificando-se como um conjunto não ordenado de valores nomeados. (FLANAGAN, 2011).

2.5 FRAMEWORKS

Riehle (2000, p. 54) define que “*framework* é um modelo de um domínio particular ou um importante aspecto disso”. Um *framework* pode modelar qualquer domínio, desde técnicos até sistemas de aplicação *Web* para empresas, fornecendo *design* e implementações reutilizáveis.

Nesta seção todos os *framework*s utilizados para a construção do projeto são separados e têm o seu funcionamento explicado.

2.5.1 Laravel

De acordo com seu criador, o Laravel foi lançado em junho de 2011 para preencher funcionalidades que o *framework CodeIgniter*[10] (um dos *frameworks* PHP mais populares na época) não possuía, como por exemplo a ORM[11] *(Object-*

[9] Tipo de dado primitivo que possui dois valores, podendo ser 0 e 1.
[10] https://codeigniter.com/
[11] Consiste na manipulação do banco de dados através de uma *Model*,

relational mapping), pacotes de módulos, geração de código e suporte para CLI (*Command-Line Interface*) (SURGUY, 2017).

Desde o seu lançamento, o Laravel continha autenticação construída; *Eloquent* ORM para realizar operações com o banco de dados, localização, modelos e relacionamentos; *cache;* grande extensibilidade através de bibliotecas e módulos, e assistentes de HTML, como é o caso das páginas "php.blade", além de outras funcionalidades, tendo uma grande gama de atratividades para um framework recém lançado (SURGUY, 2017).

O autor complementa que o Laravel ainda não era um *framework* MVC *(Model–View–Controller)*, pois não possuía funcionalidades de controle. Nos meses seguintes do desenvolvimento do Laravel, o seu criador adicionou funcionalidades para validação de métodos, paginação, instalação por linha de comando, expandiu o *Eloquent* ORM e adicionou componentes de testes de unidade fazendo o Laravel ir da versão um para dois em menos de seis meses (SURGUY, 2017).

Yu (2015) explica que as aplicações *Web* cujas arquitetura baseiam-se em Laravel possuem várias camadas. Na estrutura de três camadas da tecnologia *Web*, o banco de dados não é um serviço conectado diretamente a cada cliente, mas conectado ao servidor *Web* para se tornar um serviço dinâmico e em tempo real. O servidor age como se fosse um *proxy* para o cliente e como um cliente para a base de dados, integrando informações de diferentes fontes e formatos em uma interface unificada no navegador.

Yu (2015) afirma que dentro do *framework* Laravel*,* o XML *(Extensible Markup Language)* é adotado para o *design Web* principalmente. XML é uma linguagem de marcação estruturada, auto descritiva e que não depende de uma plataforma (YU, 2015).

Quando o Laravel é utilizado para o desenvolvimento de uma aplicação *Web*, ele pode reduzir o tempo e o esforço demandado pelos desenvolvedores e obter uma arquitetura distribuída de aplicação, além de garantir o melhor processamento de transações e melhorar a segurança do sistema. Ou seja, “em resumo, o método de *design* baseado no *Laravel* que pode ser obtido, é escalável e possui uma escalabilidade poderosa“ (YU, 2015, p. 303).

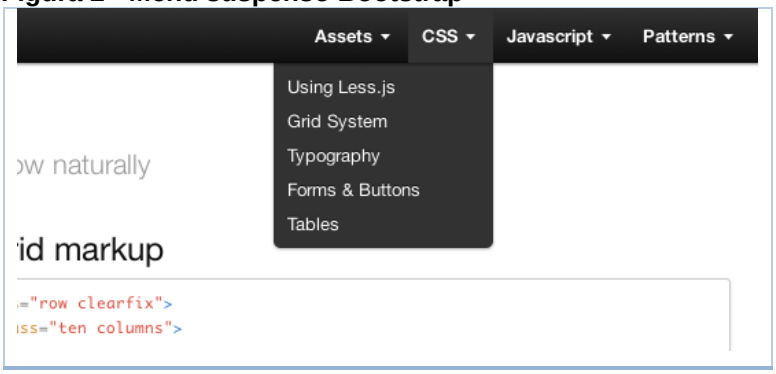
2.5.2 *Bootstrap*

Bootstrap é um conjunto de ferramentas de código aberto feita para ajudar desenvolvedores a criar projetos online de forma mais rápida e eficiente. O objetivo dos criadores é disponibilizar uma biblioteca refinada, bem documentada e abrangente de componentes com *design* adaptativos construídos a partir das linguagens HTML, CSS e *JavaScript* para que, dessa forma, outros tenham a possibilidade de criar e inovar (OTTO, 2017).

O autor ainda traz o exemplo da rede social *Twitter*, onde os desenvolvedores tinham dificuldade para desenvolvimento do *design* da plataforma de forma rápida. A partir do reconhecimento do problema pelas diversas equipes da empresa, foi descoberto a oportunidade para o projeto de criação do *Bootstrap*, juntamente com projetos futuros. Com isso, implementou-se um processo de colaboração entre os departamentos de *Design* e Engenharia (OTTO, 2017).

A Figura 2 representa uma das funcionalidades do *Bootstrap* chamada de menu suspenso, que permite a criação de um menu que exibirá suas subcategorias ao passar o *mouse* na opção do menu.

Figura 2 - Menu suspenso Bootstrap



Fonte: Otto (2017)

Outra vantagem do Bootstrap é seu sistema de *grids*[12], que possibilita ao desenvolvedor projetar suas aplicações em uma resolução padrão, e sem grandes alterações, as telas possam ser visualizadas em dispositivos de resoluções diferentes de tela, como celulares, tablets (FERREIRA, 2013).

[12] Malha formada pela interseção de um conjunto de linhas horizontais e um conjunto de linhas verticais

2.5.3 *JQuery*

O *JQuery* é uma biblioteca de funções de *JavaScript*, que facilita a programação, manipulando detalhes do código. Algumas de suas funcionalidades são: manipular elementos de HTML, encobrir e simplificar operações complexas do Javascript, adicionar efeitos visuais e animações, alterar conteúdos da página, entre outros (CRANLEY et al., 2013).

O *jQuery* oferece técnicas de seleção de elementos usando os seletores do CSS, que consiste em um comando simples que serve para informar ao navegador a qual elemento determinado estilo deve ser aplicado. Um exemplo disso é que enquanto o "*h1*" é compreendido como sendo um seletor de elemento básico, o qual aplica um estilo para cada *tag* '<h1>'; o '*.copyright'*, é entendido como um seletor de classe, que estiliza qualquer *tag* que tenha um atributo de classe '*copyright'*, explica McFarland (2011). É possível notar isso através do trecho de código " Copyright, 2011 ".

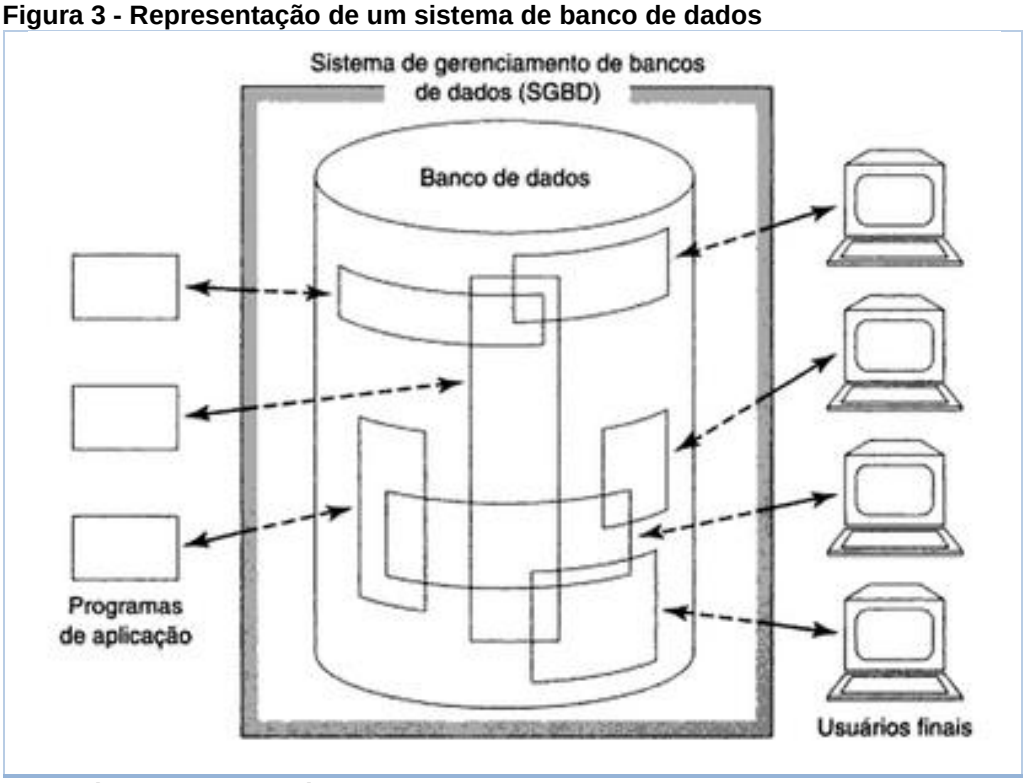
Algumas vezes, quando um desenvolvedor precisa fazer uma série de operações selecionando elemento utilizando o *jQuery*, seria necessário escrever cada linha de código para cada operação. Por exemplo, se o desenvolvedor quiser alterar a altura e largura de um elemento "<*div*>" seriam necessárias duas linhas, mas o *jQuery* possibilita que isso seja feito com uma única linha. O *jQuery* usa um princípio chamado de encadeamento, que permite adicionar uma função após a outra (MCFARLAND, 2011), como no exemplo "$('#popUp').width(300).height(300);".

2.6 BANCO DE DADOS RELACIONAL

Date (2004, p. 6) define um banco de dados como "um sistema computadorizado de manutenção de registro". Isso significa que um banco de dados é um sistema cuja principal função é armazenar informações para que os usuários de uma aplicação possam buscar e atualizar quando necessitarem. Já as informações contidas neste banco de dados podem ser quaisquer necessárias para auxiliar nas atividades de um indivíduo ou organização (DATE, 2004).

Esse comportamento pode-se ser observado na Figura 3, onde os usuários têm acesso ao banco de dados para cadastrar visualizar e editar registros, assim como outros sistemas também podem fazê-lo.

Figura 3 - Representação de um sistema de banco de dados



Fonte: (DATE, 2004, p. 6)

Date (2004) aborda banco de dados como sendo “uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa”. Dados persistentes, por sua vez, podem ser definidos como dados que serão preservados por um período de tempo, diferente dos dados transitórios, como instruções de SQL ou filas de trabalho, que são transitórios e só permanecem guardados enquanto estão em execução. O autor também explica que o termo empresa engloba qualquer organização, seja ela comercial, científica, técnica ou até mesmo, outra organização que seja razoavelmente autônoma.

2.6.1 MySQL

SQL (*Structured Query Language*) é uma linguagem de programação desenvolvida, especificamente, para SGBDR (Sistemas de Gerenciamento de Bancos de Dados Relacionais), como o MySQL. “Todo banco de dados relacional

aplica sua própria versão do padrão SQL", explica Suehring (2004, p. 7). O quadro 2 apresenta a comparação entre os principais sistemas de gerenciamento de bancos de dados relacionais, relacionando seus respectivos TCO's[13] (*Total Cost of Ownership*).

Quadro 3 - Comparação entre SGBDR

SGBDR	Vantagens	Desvantagens
Oracle	Versátil, estável e seguro.	TCO potencialmente alto.
MS Sql Server	Estável e seguro; a Microsoft oferece excelente suporte.	TCO relativamente alto; proprietário.
PostgreSQL	Banco de dados avançado com TCO baixo.	Ainda não deve ser implementado em grande escala para uso comercial.
Informix	Estável; tem um bom suporte disponível.	TCO geralmente maior.
MySQL	Oferece o banco de dados para o melhor dos cenários; baixo TCO; alta estabilidade e segurança; suporte excelente.	Nem todas as versões disponíveis podem oferecer toda a gama de recursos do MySQL.

Fonte: Adaptado de Suehring (2004, p. 8)

MySQL está ganhando espaço no mercado de SGBDR por oferecer um bom suporte ao desenvolvedor, ser estável e ter um custo baixo, além de ter uma excelente documentação. O MySQL oferece inúmeras vantagens, tais como funcionar em muitas plataformas e possuir um TCO estável e baixo (SUEHRING, 2004).

Prates (2006) também cita algumas das vantagens da utilização do MySQL, como a capacidade de manipulação de tabelas com mais de 50.000.000 registros, controle de privilégios e comandos executados com alta velocidade.

O MySQL conta com um site completo, possuindo material de referência, além de um *link* para arquivos de lista de endereços. Ademais, o MySQL oferece suporte altamente qualificado para seus produtos, assim como um serviço que possibilita aos seus desenvolvedores efetuar *login* em seu servidor para efetuar correção de erros e para auxiliar com a sua otimização (SUEHRING, 2004).

[13] Métrica de análise que objetiva calcular os custos de vida e de aquisição de um produto.

2.7 SERVIDOR WEB

O Xampp é a uma plataforma que agrega os servidores *Apache,* Mysql, PHP e *Pearl*, todos em um único programa para ser instalado no computador. É uma simples distribuição Apache cujo intuito visa facilitar a criação de um servidor *Web* local para testes de aplicações. Segundo Walia e Gil (2014, p. 26), “tudo o que você precisa para configurar um servidor Web - servidor de aplicação (Apache), banco de dados (MySQL) e linguagem de script (PHP) - está incluído em um arquivo simples extraível”. O Xampp também é multiplataforma, ou seja, ele funciona tanto no Windows, quanto no Linux e no Mac.

Para principiantes, o desenvolvimento Web é muito facilitado através da utilização da ferramenta Xampp, pois a ferramenta possui as tecnologias mais recorrentes ao tratar-se de desenvolvimento Web. A maioria dos servidores Web lançados atualmente usam os mesmos componentes do Xampp, facilitando a transição dos servidores de testes para o servidor real (WALIA, GIL, 2014).

3 METODOLOGIA

Neste capítulo é apresentado como foi realizado o levantamento dos requisitos para o desenvolvimento do projeto, as escolhas de linguagens de programação, banco de dados e *framework* utilizados no desenvolvimento.

3.1 LEVANTAMENTO DE REQUISITOS

Para nortear o início do projeto, foi realizada uma pesquisa para melhor entender as reais necessidades do dono de um estacionamento. A pesquisa deu-se a partir de uma entrevista na empresa do cliente, onde foram apresentados alguns dos detalhes e processos do dia a dia de um estacionamento, assim como as principais necessidades do cliente e o que o gestor esperava de um sistema de gestão do seu ramo de atuação.

O proprietário do estabelecimento definiu as características básicas que o sistema deveria apresentar, são elas:

- Registro de clientes;
- Registro de estacionamentos;
- Relatório de valores recebidos;
- Impressão de comprovante.

Dentro deste escopo, foi definido, juntamente com o proprietário, que o sistema poderia também possuir o cadastro das vagas, de usuários (por questões de segurança) e de veículos, porém, sem que estas funcionalidades influenciassem nas principais tarefas do sistema. As demais funcionalidades foram implementadas a fim de melhorar a experiência dos usuários sem comprometer os requisitos solicitados. Com os requisitos básicos definidos, pôde-se dar início ao planejamento do projeto.

Na idealização do projeto, optou-se por desenvolvê-lo em linguagem de programação voltada para a *Internet*. Entre as principais linguagens *Web* do mercado (.NET, ASP, JSP, JavaScript, Ruby, Python, Perl e PHP) foi escolhido o PHP pela possibilidade de ser embutida com o código HTML, além de ser uma das linguagens de grande facilidade de acesso à conteúdo de apoio, e ter uma vasta quantidade de *frameworks* que poderiam ser utilizados neste projeto.

Outro motivo para a escolha do PHP como a linguagem de programação utilizada no desenvolvimento do projeto é a sua portabilidade. Segundo Niederauer (2004, p. 26), “podemos executar o PHP no *Linux*, no *Unix* ou no *Windows*. [...] embora haja poucas diferenças em relação ao seu uso nos demais sistemas operacionais”.

O PHP é compatível com os três mais importantes servidores Web: o Apache HTTP Server para *Unix* e *Windows*, o *Microsoft Internet Information Server* e o *NetScape Enterprise Server* (também conhecido como *iPlanet Server*). Além disso, funciona também com vários servidores menos conhecidos, incluindo *fhttpd* do *Alex Belits,* o *Personal Web Server da Microsoft*, o *AOLServer* e o servidor de aplicação *Omniserver* da *Omnicentrix* (CONVERSE; PARK, 2003).

Segundo Welling e Thompson (2001), em comparação com *Perl*, ASP (*Active Server Pages*) e JSP (*JavaServer Pages*), o PHP tem muitos atributos, como: alta *performance*, interface para diferentes sistemas de bancos de dados, bibliotecas integradas para tarefas comuns da *Web*, custo baixo, facilidade de aprendizagem e uso, portabilidade e disponibilidade de código fonte. Os autores afirmam ainda que o PHP “está disponível para *Microsoft Windows*, muitas versões de *Unix* e é funcional com qualquer servidor *Web* e o MySQL tem versatilidade similar” (WELLING; THOMPSON, 2001, p. 04).

Para a escolha do sistema de gerenciamento de banco de dados, foi decidido pelo MySQL (entre opções como *Oracle, Microsoft SQL Server, PostegreSQL, MongoDB*, entre outros) por estar entre os mais populares e por ser rápido, robusto e eficiente na execução de suas tarefas (WELLING; THOMPSON, 2001).

3.2 ANÁLISE DE SISTEMAS EXISTENTES NO MERCADO

Para verificar os produtos do mesmo tipo que já existem no mercado, foi realizada pesquisa na *Internet* para encontrá-los. Os sistemas encontrados foram*: Park Manager, Park Master, Parkeer,* Estacionamento Digital, *E-Car* e *E-Prático.*

Dentre todos os sistemas para gerenciamento de estacionamentos privados pesquisados, apenas o Estacionamento Digital possui funcionamento em algum

sistema de dispositivo móvel ou pelo navegador, sem a necessidade da instalação de um programa.

Além disso, a grande maioria dos sistemas têm a necessidade de fazer a instalação do sistema em um computador com sistema *Microsoft Windows*. A única exceção é o Estacionamento Digital, que pode ser utilizado através de um aplicativo em um dispositivo com sistema *Android* ou pelo navegador de *Internet*.

Outra vantagem do sistema desenvolvido em relação aos sistemas encontrados na *Internet* é a interface para o cliente final do estabelecimento. Nessa interface, o cliente do estacionamento poderá fazer consulta sobre os seus gastos no estacionamento. Nenhum dos sistemas encontrados possuía algum tipo de função para o cliente, contando apenas com funções para o gerenciamento do estabelecimento.

Todos os sistemas testados têm a opção de cadastro de clientes e veículo, mesmo que seja apenas da placa do veículo do cliente. Porém todos eles possuem problemas de usabilidade em sua interface. Além de serem confusas para quem está operando o sistema, possuem problemas como a falta de opções para editar informações relevantes, como o preço a ser cobrado na retirada de um veículo e nas informações do cliente cadastrado.

Todas as opções testadas também fazem a cobrança do veículo automaticamente e fornecem a opção de conceder desconto ao cliente, e com exceção do *E-Prático e do Parkeer,* todos geram relatórios para o administrador do estacionamento.

O Quadro 4 relaciona as principais vantagens e desvantagens de cada um dos sistemas analisados:

Quadro 4 - Quadro comparativo entre sistemas de estacionamento

Programa	Vantagens	Desvantagens
Park Manager	Controle financeiro, opção para utilização de cancelas, várias tabelas de preços, geração de relatórios.	Lentidão para carga inicial do sistema, telas com baixa usabilidade, lista fixa de modelos de carros, acesso somente no computador no qual o sistema está instalado, sistema pago, não possui tela para consulta do cliente.
Park Master	Geração de relatórios, cadastro de convênios, várias tabelas de preços.	Telas com baixa usabilidade, interface ultrapassada, acesso somente no computador no qual o sistema está instalado, sistema pago, não possui tela para consulta do cliente.
Parkeer	Modulo de anotações, modulo de serviços, modulo de convênios, vouchers, várias tabelas de preços, geração de relatórios, backup em nuvem.	Telas com baixa usabilidade, interface ultrapassada, acesso somente no computador no qual o sistema está instalado, sistema pago, não possui tela para consulta do cliente.
Estacionamento Digital	Funcionalidade em dispositivo móvel, exportação de dados, backup em nuvem, modulo de serviços, modulo de convênios, várias tabelas de preços, geração de relatórios, controle financeiro.	Sistema pago, não possui tela para consulta do cliente.
E-car	Exportação de dados, várias tabelas de preços, controle financeiro, geração de relatórios.	Telas com baixa usabilidade, acesso somente no computador no qual o sistema está instalado, sistema pago, não possui tela para consulta do cliente.
E-prático	Modulo de serviços, controle financeiro, exportação de dados, geração de relatórios.	Telas com baixa usabilidade, acesso somente no computador no qual o sistema está instalado, sistema pago, não possui tela para consulta do cliente.

Fonte: Autoria própria

4 DESENVOLVIMENTO

Este capítulo destina-se a descrição dos passos e etapas realizados durante o desenvolvimento da aplicação, assim como discussões acerca das tecnologias utilizadas e principais características do sistema.

4.1 PLANEJAMENTO

O primeiro passo a ser tomado dentro de um projeto é o seu planejamento, de forma a realizar o desenvolvimento visando atingir os objetivos previamente estabelecidos. Dessa forma, a primeira questão a ser definida são as funcionalidades da aplicação.

Conforme citado na seção 3.1, foi realizada uma entrevista com o responsável pelo estacionamento, a fim de obter mais detalhes a respeito das suas rotinas e funcionamentos e, dessa forma, listar as funções primordiais do sistema e as principais características que o produto final deve obter.

Consideramos as funcionalidades primordiais do sistema:

a) Cadastro de clientes.
b) Cadastro de veículos.
c) Cadastro de vagas.
d) Cadastro de usuários.
e) Cadastro da empresa (utilizadora da aplicação).
f) Registro de estacionamentos:
    - Para clientes ocasionais, como é o caso de pessoas que deixam seu veículo estacionado brevemente, mas não tem nenhum tipo de contrato com o estacionamento.
    - Para clientes que possuem contrato mensal, os quais através do pagamento de uma mensalidade, tem direito à estacionamento irrestrito durante o período do mês.
g) Geração dos relatórios:
    - De valores recebidos por período.
    - De cobranças para clientes mensalistas.
    - De quantidade de estacionamentos por cliente.

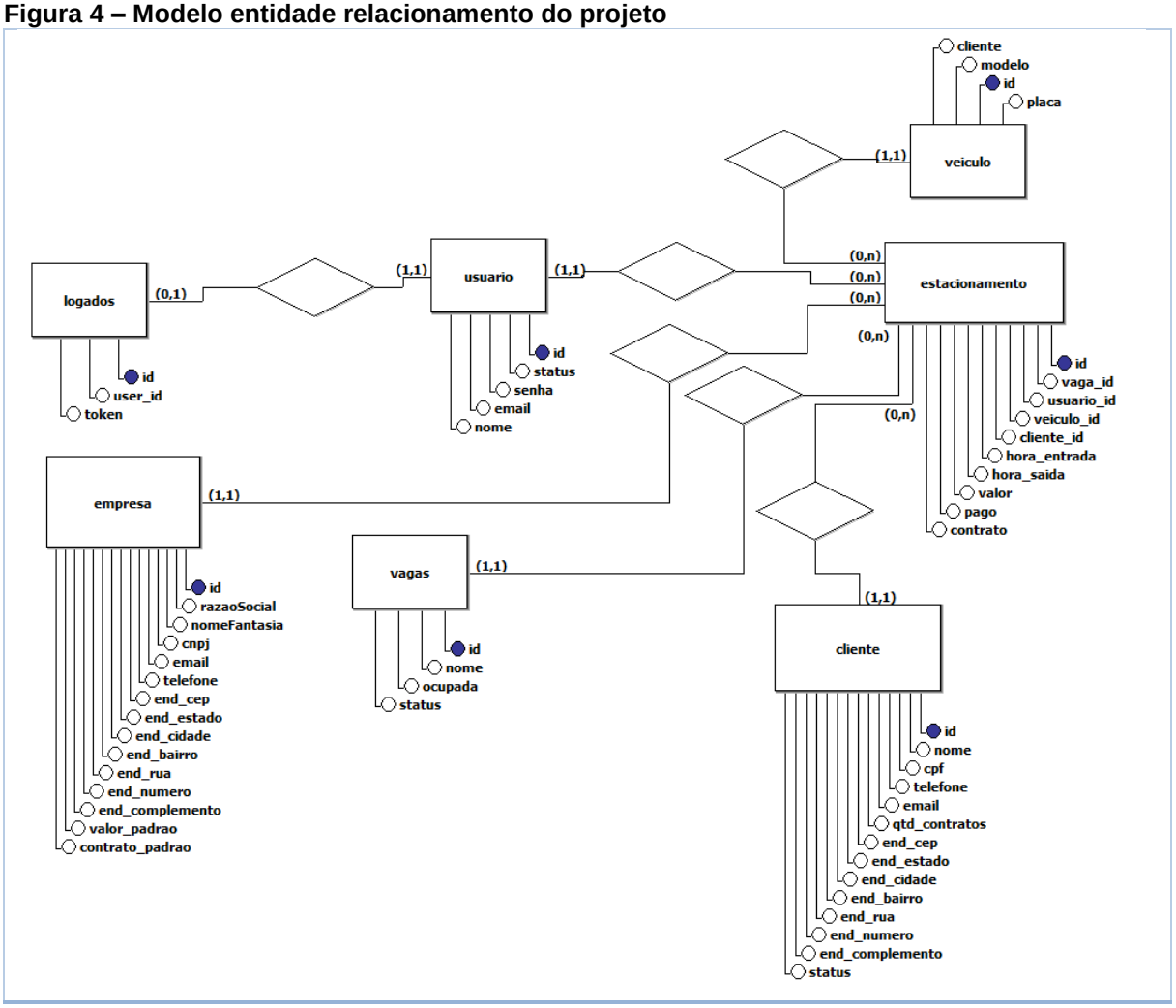
h) Implementação em ambiente *Web* (podendo ser acessado por computadores e *smartphones*).

i) Oferecer uma página de consulta para o cliente final do estacionamento, onde o mesmo pode:

- Consultar seu estacionamento atual de forma simplificada.
- Consultar seu histórico de estacionamentos.
- Consultar o valor de sua mensalidade, no caso de clientes mensalistas.

Dentro deste esboço básico de requisitos, pode-se ser idealizada a estrutura adequada para realizar o armazenamento e tratamento dos dados que viriam a ser computados. Dessa forma, o MER (Modelo Entidade Relacionamento) do sistema obteve o seguinte formato:

Figura 4 – Modelo entidade relacionamento do projeto



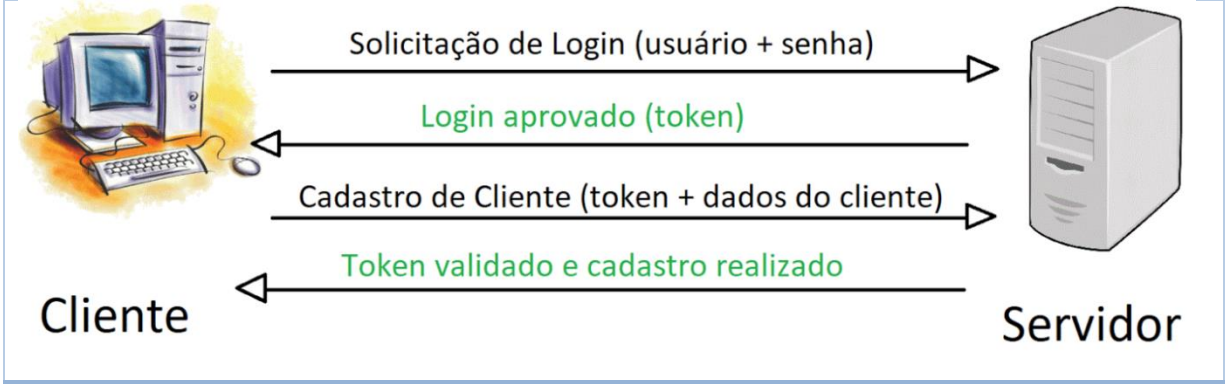
Fonte: Autoria própria

Através do MER, é possível denotar algumas das soluções arquiteturais que foram implementadas para satisfazer as necessidades do modelo de negócios. A seguir, são listados alguns pontos importantes para se observar sobre a estrutura que descreve o banco de dados do sistema:

- Foi definido que as tabelas Cliente, Vaga, Veículo, Usuário, Logados, Estacionamento e Empresa seriam suficientes para o gerenciamento de um estacionamento.
- Dentro das relações entre as classes, observa-se que entre as tabelas Usuário e Logados existe uma relação de um para um, que significa que uma ocorrência da tabela Usuários só pode estar relacionado com uma única ocorrência da tabela Logados. O mesmo se aplica da tabela Logados para a tabela Usuário, obtendo assim a lista dos usuários que estão *online* no sistema.
- Entre as tabelas Cliente e Veículo, existe uma relação de um para muitos, onde um cliente pode possuir vários veículos, mas um veículo pode estar relacionado somente à um cliente.
- Em relação à tabela Estacionamentos, percebe-se que para cada instância de estacionamento existe somente um usuário, um veículo, uma vaga e um cliente, mas cada instância de cliente, vaga, usuário e veículo podem se relacionar com várias instâncias da tabela estacionamento.
- As tabelas Usuário, Cliente e Vaga apresentam a opção “*status*”, que define se o registro está ativo ou inativo. Não apresentando a opção de exclusão para o usuário do produto, mas possibilitando-lhe manter seus registros organizados através da inativação, e ao mesmo tempo, preservando os dados históricos da empresa. A tabela Veículo possibilita a exclusão de seus registros.
- A tabela Logados possui um campo chamado “*token*”, que é de importância vital para o funcionamento e a segurança da aplicação. No momento em que um usuário se autentica no sistema, o id desse usuário é adicionado à tabela Logados e um *token* é gerado para o mesmo. Este *token* é armazenado na mesma tabela e devolvido para o usuário, de forma que na próxima interação, ele só precise fornecer o *token*, evitando o comprometimento de dados

sensíveis como senhas e aumentando a segurança e confiabilidade do sistema. A Figura 5 ilustra detalhadamente este funcionamento:

Figura 5 - Funcionamento do *login* baseado em *token*



Fonte: Autoria própria

- A tabela usuário conta com um campo definido como “administrador”, que permite uma gama de funcionalidades para o usuário que a possuir. Um usuário administrador terá acesso aos relatórios gerados pelo sistema, terá permissão de cadastrar, editar propriedades e inativar outros usuários. Também poderá realizar ajustes no histórico de estacionamentos, enquanto os usuários comuns não terão tal permissão.
- Na tabela de estacionamentos, encontram-se dois campos que possuem funcionalidades específicas. O campo “pago”, que permite diferenciar um registro histórico de um registro ativo, e o campo “contrato”, que define se aquela instância de estacionamento especificamente está qualificada como um estacionamento dentro de um contrato ou um estacionamento à parte. Por exemplo, se um cliente que possui dois contratos ativos, já possuir 2 carros registrados em seu nome estacionados, precisar estacionar um terceiro carro, o sistema irá gerar um alerta para o operador, informando que o número de estacionamentos excedeu o número de contratos, e que este terceiro estacionamento será tarifado normalmente.

4.2 IMPLEMENTAÇÃO

Com as funcionalidades necessárias definidas e a estrutura do banco de dados arquitetada, pôde-se dar início à implementação do sistema. Dentro desta

seção, serão exploradas as fases do desenvolvimento, tanto da parte do servidor quanto da parte do cliente.

Como mencionado no capítulo 1, optou-se por realizar o desenvolvimento dentro das linguagens PHP, HTML, CSS e *JavaScript*, que contemplam o ambiente *Web*. O lado do servidor, também conhecido como *Back-end,* utilizou-se exclusivamente da linguagem PHP, manipulada através do *framework* Laravel, em sua versão 5.2. Para o desenvolvimento da parte do cliente, conhecida como *Front-end*, foram utilizadas as linguagens HTML, CSS e *JavaScript*, juntamente com o *framework* conhecido como *Bootstrap*, manipulador de HTML e CSS, e a biblioteca *JQuery*, a qual possibilita uma maneira mais simples de executar comandos de *JavaScript*. Juntamente com o *JQuery*, foram utilizados alguns de seus *plugins*, que são abordados no decorrer deste capítulo.

Decidiu-se por separar o projeto em duas etapas: desenvolvimento do servidor (modelos das classes, funcionalidades, validações e interação com a base de dados) e cliente (interface *Web* através da qual serão enviadas as requisições para o servidor).

4.2.1 Servidor

Nesta seção, são descritas as atividades realizadas durante a implementação do servidor e os principais conceitos a respeito da estrutura do projeto.

4.2.1.1 Laravel

Para iniciar a implementação, fez-se necessária a instalação do *framework* Laravel, a qual pode ser realizada através da ferramenta chamada *composer*[14].

A primeira alteração a ser realizada no projeto já criado é no arquivo “*.env*”, o qual possui as configurações padrões do banco de dados (IP de onde o banco de dados está hospedado, nome do banco, usuário e senha). Um detalhe importante é que o Laravel possui controle sobre a criação e abastecimento das tabelas do banco

[14] https://getcomposer.org/

de dados, sendo necessário somente existir um banco de dados com o nome descrito nas configurações do arquivo “*.env*”.

Também se utilizou o pacote chamado “*Dingo API*”, que é um complemento para o Laravel e funciona como ferramenta de auxílio ao desenvolvimento de API’s para a chamada das rotas.

Com o ambiente já configurado e o banco de dados criado, pôde-se dar início à codificação. Como citado na seção 2.5.1, o Laravel utiliza o modelo MVC, que objetiva a melhor organização das funções do projeto. O primeiro item a ser desenvolvido são as *Models* das classes. Além das *Models, Views* e *Controllers*, também foram utilizadas no projeto:

- ***Migrations***: Forma através da qual o *framework* consegue realizar a criação e exclusão das tabelas no banco de dados.
- ***Seeders***: Responsável por realizar a pré carga do banco, com dados para testes ou informações estáticas, como por exemplo, a lista dos estados brasileiros.
- ***Routes***: Arquivo onde são descritas as rotas as quais o servidor responde as requisições. São responsáveis por interligar uma requisição feita ao servidor e a função correspondente à esta solicitação.

A junção de todas essas entidades caracteriza a parte a ser desenvolvida no servidor.

4.2.1.2 *Models*

No Laravel, as *Models* têm a função de conter os atributos de uma classe, que será convertida em uma tabela, posteriormente. Nela são descritos todos os atributos de uma determinada classe, mas ainda sem identificar o tipo deste atributo, e a tabela correspondente à esta classe. Também são descritas as ligações entre tabelas relacionadas, como por exemplo, a relação de “um-para-muitos” entre as tabelas Cliente e Veículo.

Para a criação de uma *Model,* é necessário executar o CMD (*Command*), acessar a pasta do diretório do projeto e executar o comando *“php artisan*

make:model NomeDaModel -m “, onde o comando -m irá criar também uma classe *Migration* para esta *Model.* As *MIgrations* são abordadas na próxima seção.

O Quadro 5 apresenta uma das *Models* utilizadas no sistema:

Quadro 5 - Exemplo de Model

```
class Cliente extends Model
{
   protected $table = 'clientes';
   protected   $fillable   =   ['nome',   'cpf',   'telefone',   'email',
'qtd_contratos',  'mensalidade',  'end_cep',  'end_estado',  'end_cidade',
'end_bairro', 'end_rua', 'end_numero', 'end_complemento', 'status'];
   protected $hidden = ['created_at', 'updated_at'];
   public function veiculo()
   {
       return $this->hasMany('App\Veiculo');
   }
   public function estacionamento()
   {
       return $this->hasMany('App\Estacionamento');
   }
}
```

Fonte: Autoria própria

A partir do Quadro 5, pode-se notar que existem dois campos que contém os atributos da tabela desta classe. Os que se encontram na variável “*$fillable*” são retornados normalmente quando esta instância da classe é solicitada pelo servidor, já os atributos da variável “*$hidden*” só são retornados caso sejam solicitados especificamente.

Após a criação de todas as *Models* e o estabelecimento da relação entre as tabelas, pode-se avançar para a próxima etapa do desenvolvimento, onde são criadas as *Migrations.*

4.2.1.3 *Migrations*

As *Migrations* são classes do próprio Laravel, responsáveis por realizar a criação das tabelas e suas respectivas colunas no banco de dados. Para cada tabela do banco de dados, existirá uma classe de *Migration,* que contém as colunas a serem criadas e seus respectivos tipos de dados.

Com as classes de *Migration* criadas para todas as *Models*, é possível criar as tabelas no banco de dados a partir do comando “*php artisan migrate*”, via *prompt*

de comando. Este comando irá realizar a criação de todas as tabelas de uma única vez.

O Quadro 6 representa um exemplo de *Migration:*

Quadro 6 - Exemplo de *Migration*

```
class CreateClientesTable extends Migration
{
   public function up()
   {
       Schema::create('clientes',     function     (Blueprint
$table) {
           $table->increments('id');
           $table->string('nome');
           $table->string('cpf');
           $table->string('telefone');
           $table->string('email');
           $table->integer('qtd_contratos');
           $table->decimal('mensalidade');
           $table->string('end_cep');
           $table->string('end_estado');
           $table->string('end_cidade');
           $table->string('end_bairro');
           $table->string('end_rua');
           $table->string('end_numero');
           $table->string('end_complemento');
           $table->boolean('status');
           $table->timestamps();
       });
   }
   public function down()
   {
       Schema::drop('clientes');
   }
}
```

Fonte: Autoria própria

4.2.1.4 *Seeds*

Uma vez criadas as tabelas no banco de dados, pode-se popular estas tabelas com informações estáticas ou informações para teste. Este processo é possível através das instâncias da classe *Seed*, que através do método *run* executam a criação de uma instância da classe desejada.

Para criar um novo *Seeder,* é necessário executar através do CMD o comando "*php artisan make:seeder ClientesTableSeeder*", um novo arquivo da classe *seeder* será criado e nele é possível especificar qual classe será criada e quais são os valores de seus campos.

O Quadro 7 exemplifica uma instância da classe *Seeder*:

Quadro 7 - Exemplo de Seeder

```
class ClientesTableSeeder extends Seeder
{
   public function run()
   {
       \App\Cliente::create([
             'nome'                =>    'tayrine
freitas',
      'cpf'               => '07979046463',
           'telefone'          => '42999299241',
                  'email'                    =>
'tay@dominio.com.br',
           'qtd_contratos'     => '0',
           'mensalidade'       => '0',
           'end_cep'           => '84010010',
           'end_estado'        => 'PR',
           'end_cidade'        => 'Ponta Grossa',
           'end_bairro'        => 'Centro',
           'end_rua'           => 'Paula Xavier',
           'end_numero'        => '720',
            'end_complemento'    =>  'Próximo  ao
Tozetto',
           'status'            => true
       ]);
   }
}
```

Fonte: Autoria própria

Também é possível criar um arquivo *Seeder* para cada classe que se deseja abastecer no banco de dados e executar todos os *Seeds* de uma única vez, realizando a chamada à cada uma das classes *Seeds* dentro do arquivo "*DatabaseSeeder.php*". Ao executar o comando "*php artisan db:seed*", o abastecimento do banco de dados é efetuado.

O Quadro 8 apresenta a estrutura do arquivo "*DatabaseSeeder.php":*

Quadro 8 - Exemplo do arquivo *DatabaseSeeder.php*

```
class DatabaseSeeder extends Seeder
{
   public function run()
   {
       Model::unguard();
       $this->call(ClientesTableSeeder::class);
       $this->call(EmpresasTableSeeder::class);
       $this-
>call(EstacionamentosTableSeeder::class);
       $this->call(UserTableSeeder::class);
       $this->call(VagaTableSeeder::class);
       $this->call(VeiculosTableSeeder::class);
       Model::reguard();
   }
}
```

Fonte: Autoria própria

4.2.1.5 *Controllers*

As *Controllers* contêm toda a lógica do servidor. É onde são realizadas as validações nos dados antes de realizar as interações com o banco de dados. Como o próprio nome já diz, são elas que realizam o controle das ações do servidor.

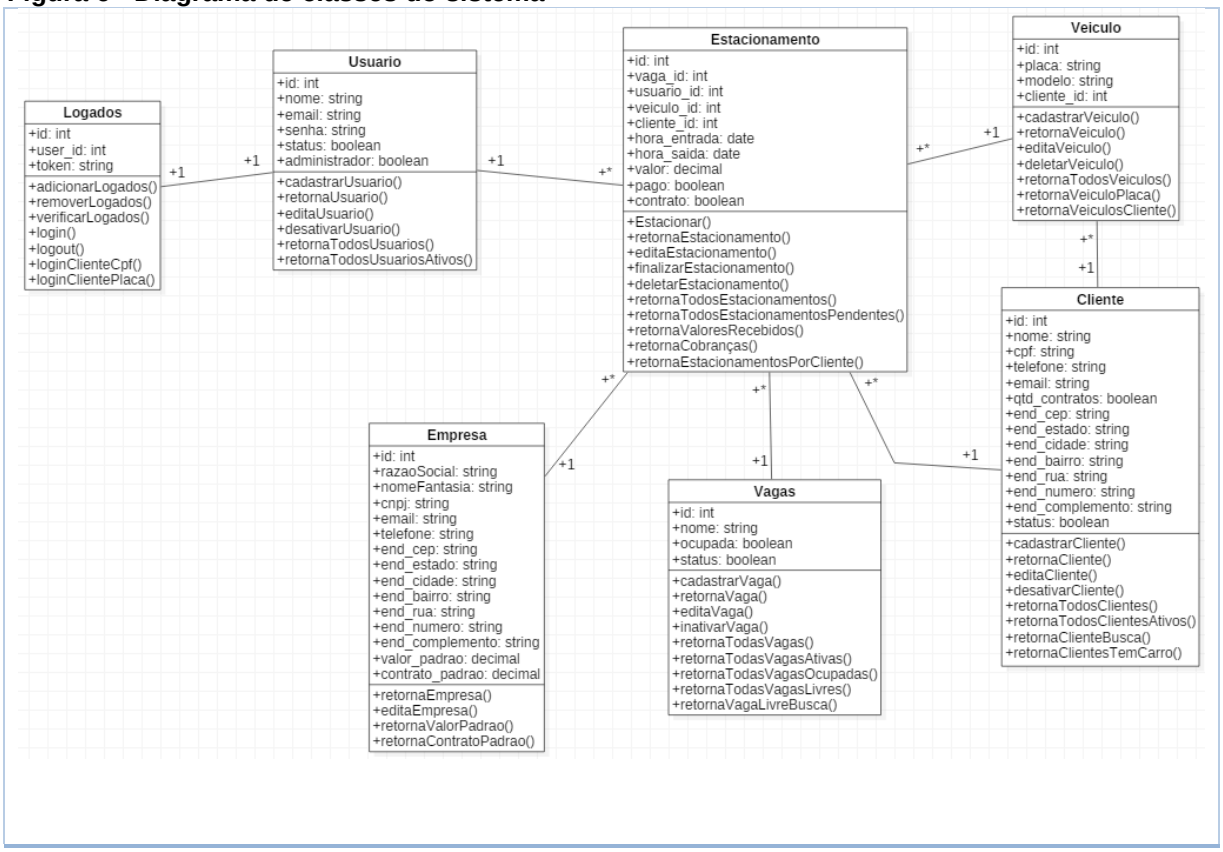
Cada classe do sistema possui o seu conjunto de validações e de funções específicas a serem realizadas, mas todas seguem um mesmo padrão. As requisições realizadas pelos clientes podem ser:

- Para cadastros de novos registros;
- Para consulta de registros existentes;
- Para edição de registros existentes;
- Para exclusão ou inativação de registros existentes;

Estas funcionalidades também podem ser descritas como “CRUD” (*Create, Read, Update* e *Delete*), e são a base de todos os sistemas.

A Figura 6 contêm todas as funções desenvolvidas no sistema.

Figura 6 - Diagrama de classes do sistema



Fonte: Autoria própria

Para cada classe do sistema, existe uma *Controller* correspondente, e em cada *Controller*, existem inúmeras validações para as diversas funções que o sistema disponibiliza, mas de forma abstrata, pode-se dizer que algumas validações são comuns entre as *Controllers*:

- Verificação do usuário *online*, através da autenticação via *token;*
- Validade dos dados (se os dados fornecidos estão corretos);
- Usuário é administrador (para algumas funções).

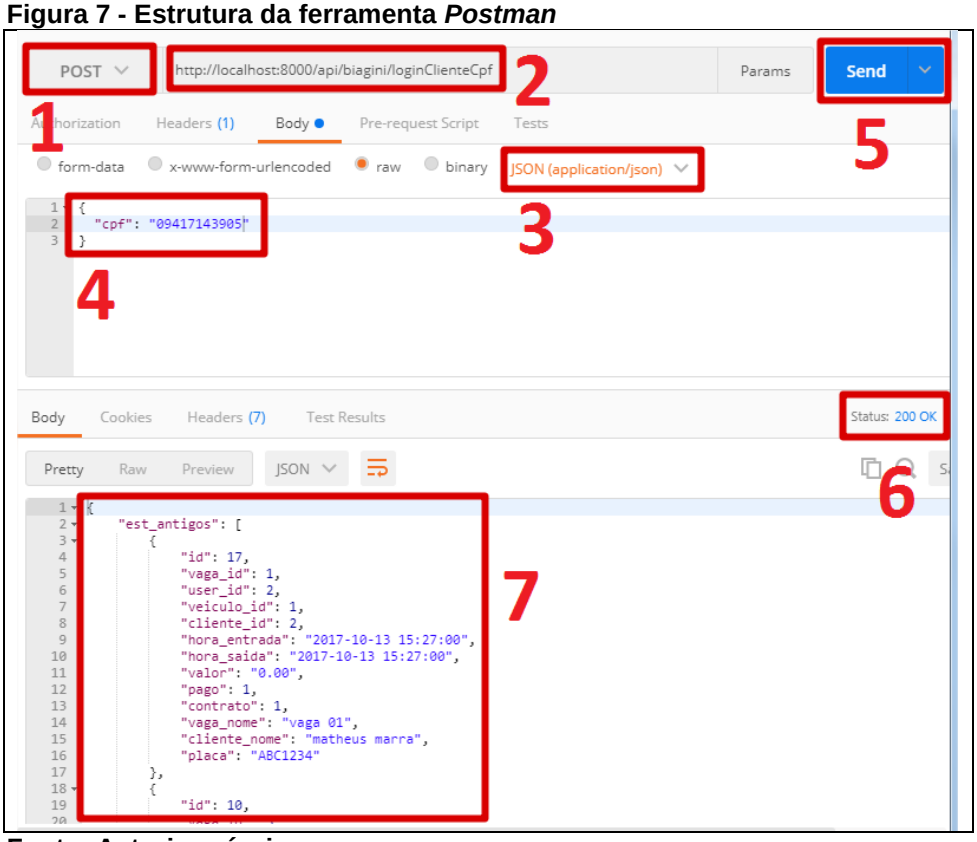
Algumas classes possuem validações mais específicas, pois para efetuar suas ações, precisam que propriedades exclusivas sejam atendidas:

1) Logados:
   a) *Login* pelo CPF do cliente: verifica se o CPF é válido;
   b) *Login* pela placa de veículo: verifica se a placa é válida;
   c) Verificação de *token* ativo: verifica se o *token* informado está na lista dos usuários *online*.
2) Clientes:
   a) Cadastro e edição: valida se o CPF informado é válido e se já está cadastrado.
   b) Inativação: verifica se o cliente possui algum estacionamento ativo no momento.
3) Usuários:
   a) Para realizar alterações, o usuário precisa ser um administrador do sistema.
   b) Cadastro e edição: verifica o nome ou o *e-mail* já estão cadastrados.
4) Veículo:
   a) Cadastro e edição: verifica se a placa já está cadastrada.
   b) Exclusão: verifica se o veículo está estacionado no momento.
5) Vaga:
   a) Inativação: verifica se a vaga está sendo usada no momento.
6) Empresa:
   a) Edição: verifica se os horários de abertura e fechamento são válidos, se os valores padrão para estacionamentos normais, diárias e mensalistas são válidos e se o usuário que está realizando a alteração é um administrador.

7) Estacionamento:
   a) Cadastro: verifica se a vaga é válida, se o veículo está cadastrado e se o veículo não está estacionado em outra vaga e verifica se a quantidade de veículos estacionados do cliente em específico excede o número de contratos que este cliente possui, pedindo uma confirmação, caso o cliente queira estacionar de acordo com a tabela normal de preços.
   b) Edição: verifica se o usuário tem permissão de administrador para alterar o horário de saída (no caso do histórico), se os valores informados são positivos, se a data é válida e se o número de estacionamentos ativos deste cliente excede o número de contratos cadastrados para ele.
   c) Deleção: verifica se o usuário é administrador.
   d) Finalização de estacionamento: verifica se as datas e o valor são válidos.
   e) Para visualização de relatórios é necessário ser administrador e preencher corretamente os filtros.

Para a validação de todas as funções, foi utilizado a ferramenta *Postman*, a qual permite simular requisições HTTP para um servidor especificado. A imagem a seguir explica a estrutura e o funcionamento da ferramenta.

Figura 7 - Estrutura da ferramenta *Postman*



Fonte: Autoria própria

Os itens destacados na imagem correspondem respectivamente a:

1. Método: que pode ser GET, POST, PUT, DELETE, entre outros.
2. URL: *link* para o qual será feita a solicitação ao servidor.
3. Tipo dos dados: informa o tipo dos dados que serão enviados.
4. Requisição: dados que são enviados ao servidor.
5. Botão de envio.
6. Resposta do servidor: código da resposta enviada do servidor para o cliente.
7. Resposta: dados enviados do servidor para o cliente.

Para efetuar a comunicação entre cliente e servidor, foi utilizado o formato de dados JSON (*JavaScript Object Notation*), que é uma linguagem utilizadora do conceito “chave-valor”, onde o primeiramente encontra-se entre parênteses o identificador do campo, seguido de dois pontos e o valor do determinado campo, também entre parênteses.

Enfim, tendo finalizado a criação de todas as *Models, Migrations, Seeders, Controllers,* a etapa de desenvolvimento do servidor foi concluída e deu-se início ao desenvolvimento das *Views,* que são o lado do cliente.

4.2.2 Cliente

Nesta seção, são destacadas as principais etapas do desenvolvimento e características das telas, bem como de suas funções e *plugins*. As linguagens de programação utilizadas para esta finalidade foram HTML, CSS e *JavaScript*, manipuladas através do framework *Bootstrap*, a biblioteca JQuery e com a utilização dos *plugins DataTables, DateTime Picker, Select2, JQuery Masked Input, Moment* e *PrintThis,* que são pacotes de *script* para utilização juntamente com o *JQuery.*

Para a criação das telas do sistema, optou-se por utilizar um *layout* pronto, de acordo com as licenças de uso descritas pelo criador do *layout*. Dentre os *layouts* disponíveis, optou-se pela utilização do *Edgpress*[15]*.*

[15] http://www.os-templates.com/free-Website-templates/edgpress

Conforme expressa a licença do *layout,* foram mantidos os créditos aos seus criadores*,* de forma a cumprir todos os requisitos exigidos pela empresa *OS Templates*.

Todavia, o *template* foi utilizado somente como estrutura para o desenvolvimento do projeto. Dentre os itens mantidos, encontram-se o arquivo CSS original (em sua grande parte), o modelo da parte superior das telas (nome da empresa centralizado, com a barra de navegação logo abaixo contendo os *links* das outras telas) e o rodapé das páginas, contendo os direitos autorais da empresa desenvolvedora do *template*.

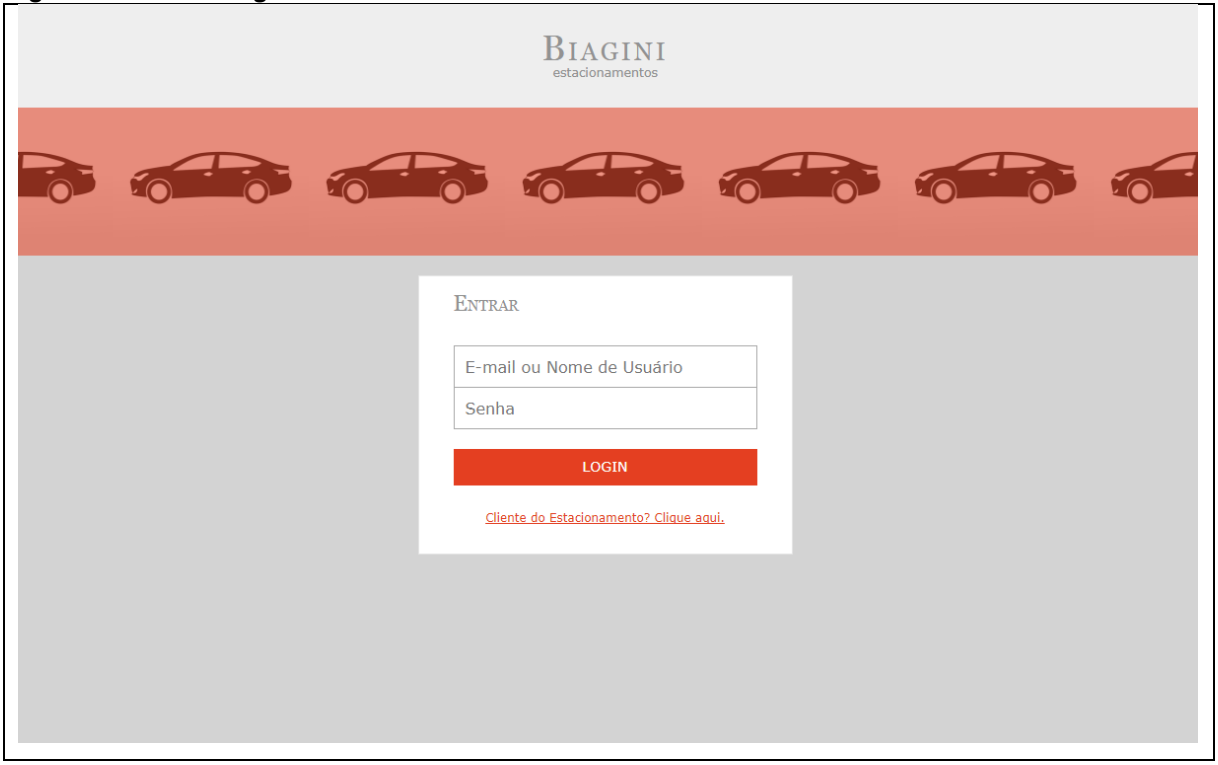
4.2.2.1 Desenvolvimento das telas

Com todas as funções previamente criadas e validadas, a segunda parte do projeto consiste em criar as telas que realizarão as chamadas aos respectivos métodos. As telas contidas neste projeto são:

- *Login;*
- *Home Page;*
- Empresa;
- Usuários;
- Clientes;
- Veículos;
- Vagas;
- Estacionamentos atuais;
- Histórico de estacionamentos;
- Relatórios;
- *Login* dos clientes;
- *Home Page* dos clientes.

Seguindo a ordem de utilização do sistema, a primeira tela a ser desenvolvida foi a tela de *login,* que consiste puramente do nome da empresa, juntamente com os campos para usuário e senha, o botão de *login* e o rodapé, como mostra a Figura 8.

Figura 8 - Tela de *login*

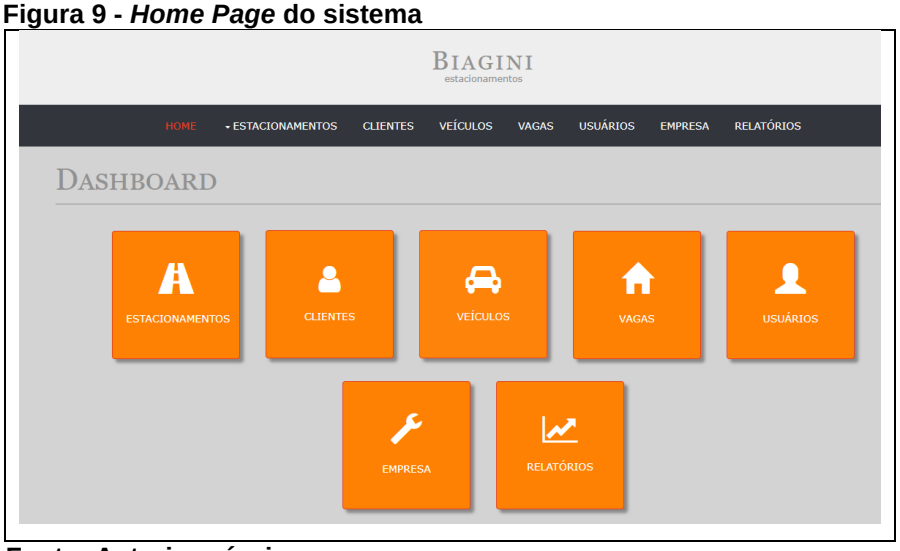


Fonte: Autoria própria

Nesta tela, é possível efetuar a autenticação tanto com o *e-mail* quanto com o nome do usuário, o servidor irá reconhecer qual das duas opções foi utilizada e validar a senha, gerando um *token* e enviando-o ao cliente, para que seja armazenado e utilizado novamente em outras interações, conforme citado na seção 4.1.

A *Home Page* é representada através da Figura 9.

Figura 9 - *Home Page* do sistema



Fonte: Autoria própria

Assim que a autenticação é realizada, o usuário é redirecionado para a *Home Page,* que serve exclusivamente como um ponto de partida, possibilitando o acesso a qualquer outra tela do sistema.

Algumas considerações importantes acerca das telas seguintes do sistema:

- Ao entrar em qualquer tela do sistema, o cliente manda o *token* para ser validado pelo servidor, e, caso o *token* não exista ou esteja expirado, o cliente é desconectado e enviado novamente para a tela de *login.*
- As telas de clientes, veículos, vagas, usuário e estacionamentos conta com as opções de criação, edição, visualização e deleção ou inativação.
- O *design* das telas foi projetado para ser o mais dinâmico, interativo e intuitivo possível, sendo este um dos principais diferenciais dos sistemas já existentes.
- O envio de dados ao servidor, também conhecido como *requests* ou requisições, é realizado a partir do AJAX (*Asynchronous Javascript and XML*), que é uma função do *JavaScript* desenvolvida especificamente para esta finalidade. Como citado na seção 4.2.1.5, o AJAX pode utilizar-se do formato JSON ou XML. Neste trabalho optou-se por utilizar o formato JSON.
- Todas as telas com listagem de dados possuem opções específicas de filtros para buscar os dados de forma eficiente e ao menos uma instância do *plugin DataTables*, responsável por organizar os dados recebidos via AJAX, processá-los e exibi-los em sua respectiva tabela, que conta com:
    1. Paginador: limita a amostragem dos dados de acordo com o número de itens por página, que pode ser definido pelo usuário.
    2. Ordenação de colunas: sendo possível ordenar os dados por qualquer coluna, em ordem crescente e decrescente.
    3. Campo para busca: possibilita realizar uma busca na tabela a partir de um fragmento de informação, podendo ser este, de qualquer das colunas.
    4. Contador de registros: apresenta a contagem dos registros da tabela, sendo alterado dinamicamente quando uma busca é realizada no campo de buscas.

A Figura 10 Ilustra um exemplo de *DataTable:*

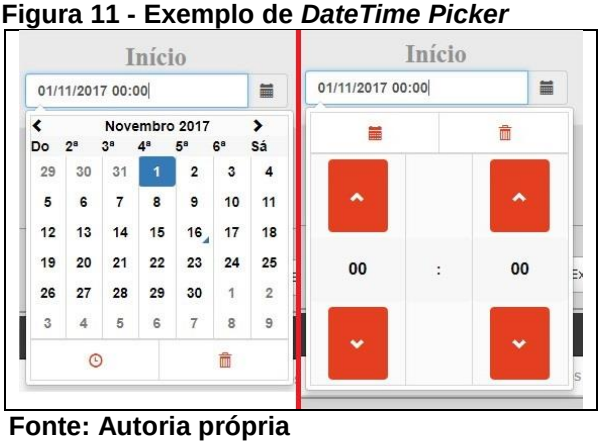
Figura 10 - *DataTables* da tela histórico de estacionamento

Fonte: Autoria própria

Todos os tratamentos de data da parte do cliente são realizados através do *plugin Moment*, que dentre suas funcionalidades possui a formatação de datas, criação de instâncias de datas a partir de *Strings* e tratamento nativo para recuperação de dias, meses, anos, horas, minutos, segundos e até mesmo milésimos de uma de suas instâncias,

Os campos para entrada de datas utilizam-se do *plugin DateTime Picker,* que possui uma interface intuitiva e prática para captação de data e tempo, exemplificada pela Figura 11.

Figura 11 - Exemplo de *DateTime Picker*



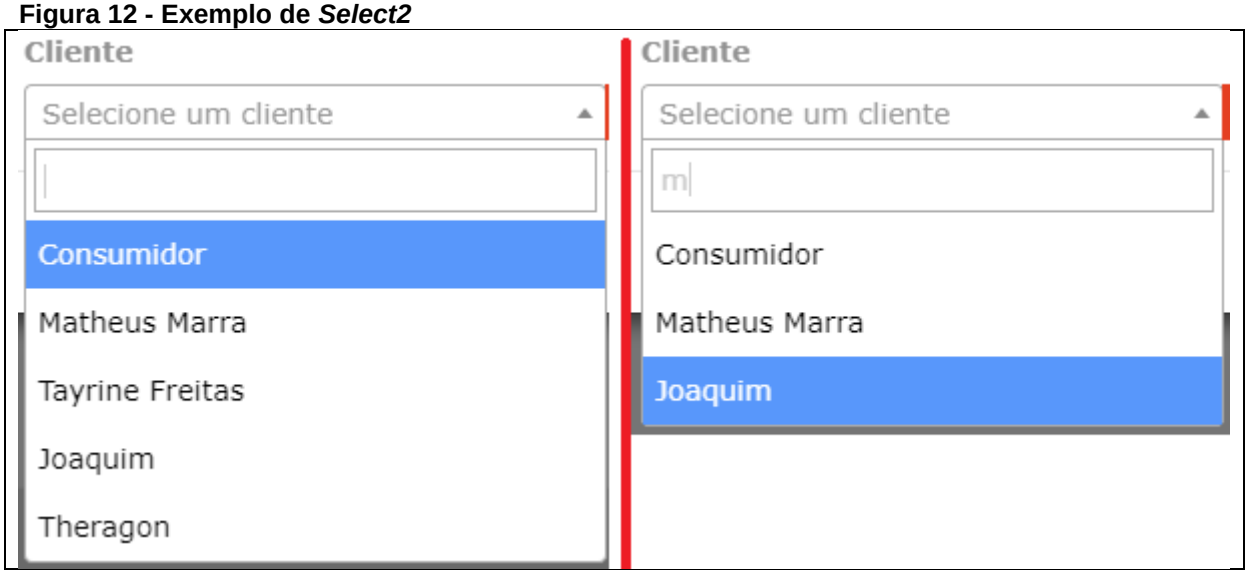
Fonte: Autoria própria

Campos como placa para veículos, telefones, CPF ou CNPJ e CEP utilizam-se do *plugin JQuery Masked Input*, que possibilita a aplicação de máscaras a estes campos de forma simplificada.

Os campos de seleção utilizam-se do *plugin* chamado *Select2,* que possibilita a integração de buscas dentro de sua própria lista. No momento em que o usuário digita um caractere no *Select2,* é enviado um AJAX para o servidor com o conteúdo digitado. O servidor, por sua vez, irá localizar os registros correspondentes da pesquisa e retorná-los para o *Select2,* que irá apresentar as novas opções ao usuário do sistema.

Um exemplo de *Select2* é mostrado na Figura 12:

Figura 12 - Exemplo de *Select2*



Fonte: Autoria própria (2017)

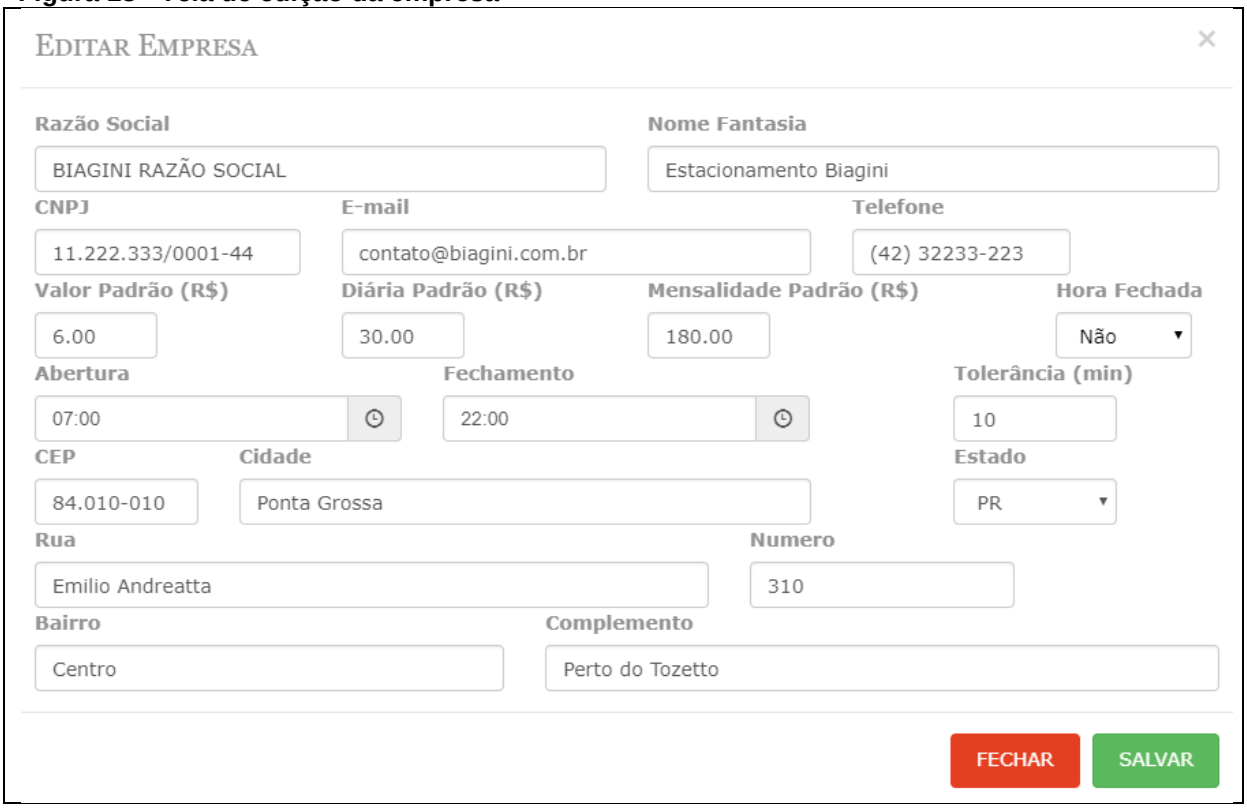
Tendo em vista que as características citadas acima são comuns na maior parte das telas, pode-se tratar individualmente de cada tela.

4.2.2.2 Empresa

A tela de empresa possui somente um registro, por isso, desconsidera a necessidade de filtros de pesquisa. Este registro contém os dados a respeito da empresa utilizadora do *software*, além de algumas configurações padrões a respeito do sistema, como por exemplo, os valores padrão para estacionamentos, diárias e

mensalidades e o valor de tolerância para saída dos veículos sem acréscimo na cobrança, como apresenta a Figura 13.

Figura 13 - Tela de edição da empresa



Fonte: Autoria própria

As *modais* de cadastro, visualização e edição das outras telas seguem este mesmo padrão. Um fator importante a ser lembrado sobre a tela de Empresa é que somente pode ser alterada por um usuário de nível administrativo, de modo a prevenir que usuários comuns possam alterar dados especiais.

4.2.2.3 Usuários

A tela de usuários possui apenas as opções de nome e *e-mail* (lembrando que ambos podem ser utilizados para *login*), senha, a opção de usuário administrador e o status, que indica se o usuário está ativo ou inativo.

Tendo em vista que podem existir registros inativados, foi adicionado um filtro para a inclusão ou não destes registros na tela da lista de usuários. Também é importante destacar que somente usuários administradores têm permissão para criar, alterar ou inativar outros usuários.

4.2.2.4 Clientes

Na tela de clientes, encontram-se os dados pessoais do cliente em questão, a quantidade de contratos e o valor da mensalidade deste cliente. No momento do cadastro, ao informar a quantidade de contratos do cliente, o sistema envia um AJAX para a consulta do valor padrão da mensalidade e calcula este valor vezes a quantidade de contratos informada, preenchendo assim o campo do valor da mensalidade. Vale destacar que este processo serve apenas como facilitador para o usuário, pois caso deseje-se fazer um valor diferenciado, basta sobrescrever o valor preenchido no campo da mensalidade.

Outra funcionalidade da tela do cadastro de clientes é a busca automatizada do endereço a partir do CEP. Uma vez informado um CEP, o sistema consulta o *Webservice* viacep[16]*,* informando o valor digitado pelo usuário. O *Webservice* responde à solicitação com os dados de endereço, e o sistema preenche devidamente os campos com as informações recebidas.

Assim como a tela de usuários, a tela de clientes possui o filtro para incluir registros inativos, visto a necessidade dos usuários de manter uma base de dados integra e ao mesmo tempo evitar a perda de informação.

4.2.2.5 Veículos

A partir da tela de veículos, é possível cadastrar um modelo, uma placa e o cliente ao qual o veículo pertence. O objetivo do campo modelo é servir como um identificador interno para o operador do sistema, sendo um campo auxiliar não obrigatório.

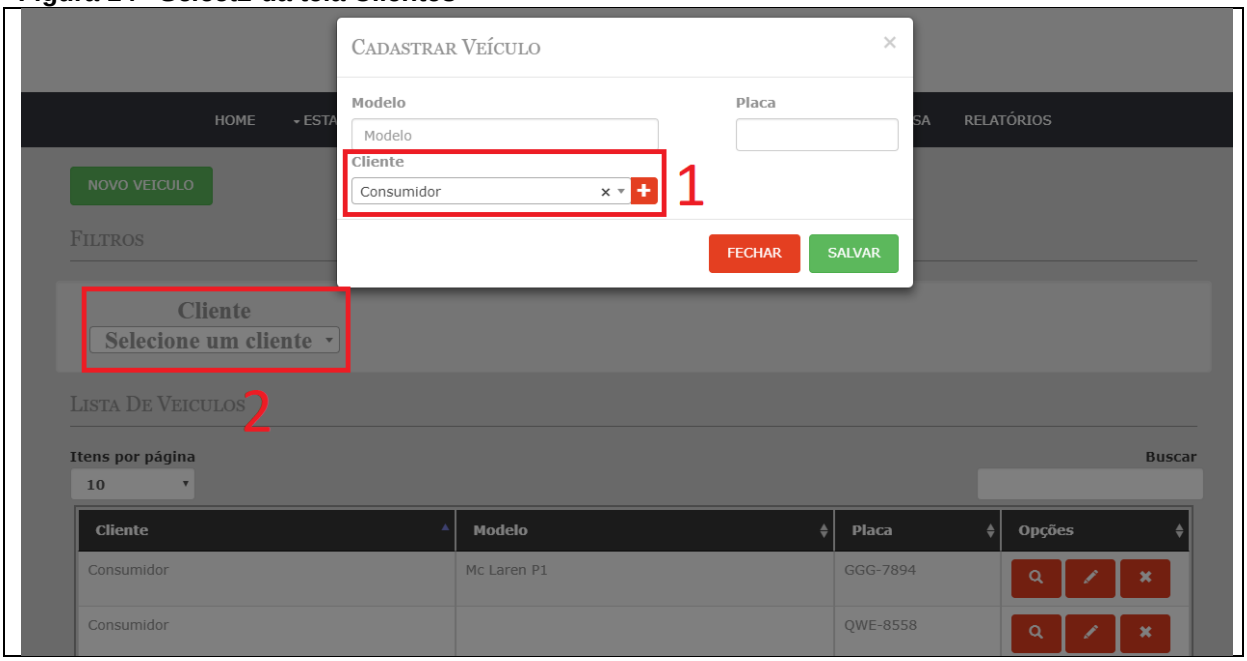
Como esta tela permite que o usuário exclua o registro do veículo, não é necessário um filtro de registros inativos. Contudo, é visto que um filtro de cliente se faz necessário. Ao selecionar um cliente neste filtro, serão exibidos na *DataTable* somente os veículos do cliente em específico.

Esta tela possui dois campos *Select2,* que são o filtro para busca dos veículos do cliente no *DataTable* e o filtro para busca dos clientes na modal de

[16] https://viacep.com.br/

cadastro. Ambos têm resultados diferentes, enquanto no cadastro do veículo o filtro de clientes busca todos os clientes cadastrados, o filtro da busca dos veículos do cliente só irá listar os veículos que possuem carros cadastrados, removendo os clientes desnecessários de sua lista.

Figura 14 - *Select2* da tela Clientes



Fonte: Autoria própria

Na Figura 14, o campo marcado como 1 irá buscar a lista completa de clientes enquanto o campo 2 buscará somente os que tiverem carros cadastrados.

Outro fator importante é que caso o cliente ainda não esteja cadastrado, o botão ao lado do campo de clientes abrirá a em outra guia no navegador a tela de clientes, diretamente na modal de cadastro, sem que o usuário precise fechar a tela de cadastro de veículo.

4.2.2.6 Vagas

Dentre as telas desenvolvidas, a tela de vagas é a mais simples. Contando com apenas dois campos, que são o nome da vaga e o status da mesma, é possível cadastrar uma vaga.

Como existe a opção de vagas ativas e inativas, foi adicionado um filtro para incluir as vagas inativas no *DataTable* da tela de vagas.

4.2.2.7 Estacionamentos atuais

Em contrapartida com a tela de vagas, esta é a tela mais complexa do sistema. Existem diversos casos, possibilidades, validações e transformações de dados nesta tela, a começar pela tela do cadastro do estacionamento.

Figura 15 - Cadastro de estacionamento

Fonte: Autoria própria

Conforme a Figura 15 ilustra, o campo vaga é um *Select2,* que contém apenas as vagas livres, ou seja, as vagas que estão em uso não serão listadas neste campo. Em seguida, temos o campo da data de entrada no estacionamento, que é um *DateTime Picker,* e no momento da abertura da *modal*, este campo é preenchido automaticamente com a data e hora atual, podendo ser alterados pelo usuário. O campo seguinte é um campo para a entrada da placa do veículo, que, no momento em que o usuário digita a placa, um AJAX é disparado para o servidor, que consulta se o carro já está cadastrado e devolve a resposta para o cliente. Neste momento, existem três cenários:

1) O carro já está cadastrado à um cliente. Neste caso, o campo cliente é preenchido automaticamente com o nome do cliente em questão, e é permitido ao usuário salvar o estacionamento.

2) O carro não está cadastrado. No momento que o cliente recebe a resposta informando que o carro não está cadastrado, uma pergunta ao usuário é disparada, questionando se deseja cadastrar o veículo neste momento ou não.
   a) Caso deseje-se cadastrar o veículo, a modal de cadastro de veículo é aberta nesta mesma tela, fazendo com que seja possível cadastrar o veículo sem sair da tela de estacionamento. A modal de cadastro do veículo conta também com o botão para o cadastro do cliente, caso o mesmo ainda não esteja cadastrado.
   b) Caso não se deseje vincular o veículo à um cliente, um AJAX é enviado para o servidor com a placa informada, criando um novo cadastro deste veículo vinculado ao cliente “Consumidor”, que é o cliente genérico do sistema.

Após estas validações, o estacionamento é cadastrado. Enquanto este estacionamento estiver pendente, não será possível cadastrar outro estacionamento na mesma vaga ou cadastrar a mesma placa em outro estacionamento.

A lista com os estacionamentos atuais é apresentada conforme a Figura 16:

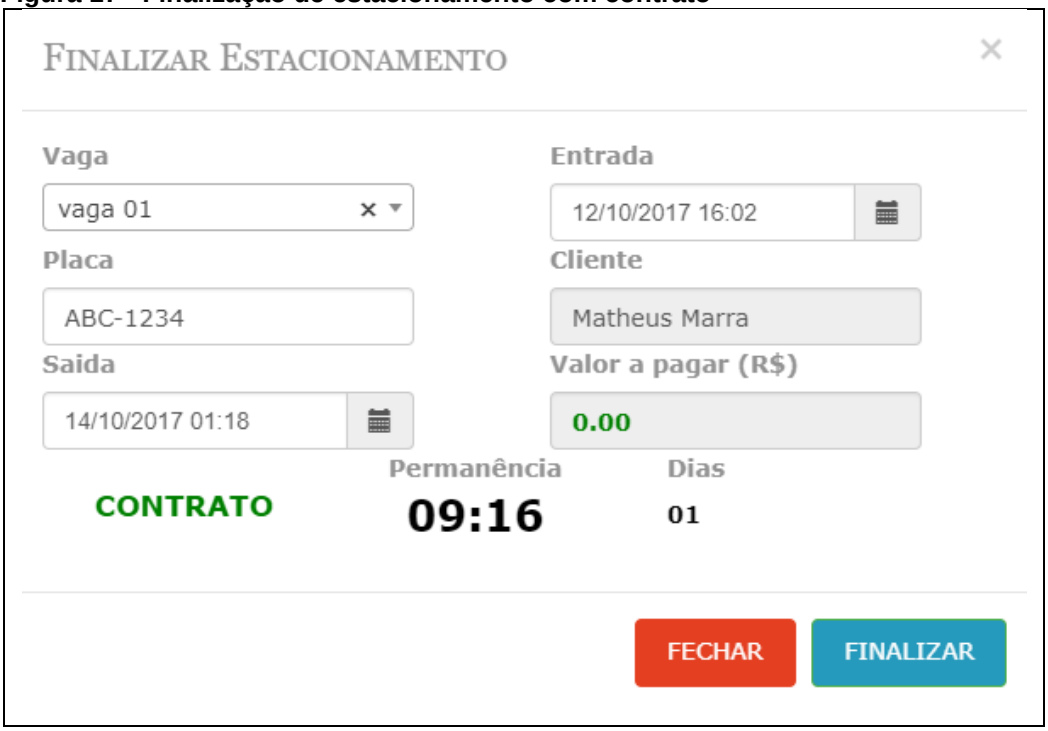
Figura 16 - Lista de estacionamentos

Fonte: Autoria própria

Na tabela contendo os estacionamentos, uma validação é realizada ao carregar a data de entrada. Caso o dia da data de entrada seja diferente do dia atual, a data completa é apresentada no campo. Caso o dia da data de entrada seja o mesmo dia, somente será apresentada a hora.

Com relação aos estacionamentos pendentes, é possível finalizá-los, visualizá-los, editá-los ou excluí-los, através dos botões de opções. A Figura 17 retrata a tela de finalização de um estacionamento:

Figura 17 - Finalização de estacionamento com contrato



Fonte: Autoria própria

Alguns itens importantes de se notar são que na figura 17, o cliente possui um contrato com o estacionamento, então na lateral esquerda inferior da *modal* encontra-se uma notificação em verde, e o campo valor a pagar encontra-se com valor R$ 0,00 e inativo para edição. Já se o cliente não possuísse contrato, o campo valor a pagar seria preenchido com o valor calculado automaticamente, considerando o valor padrão da hora cadastrada na tela empresa e considerando o tempo decorrido do estacionamento em questão.

O tempo de permanência é decorrente do cálculo da data de saída (considerada automaticamente pelo sistema como sendo a data da abertura da modal, e atualizada a cada segundo, caso não seja alterada) menos a data de entrada. Caso qualquer uma das datas seja alterada nos *DateTime Pickers,* o tempo de permanência e o valor são alterados dinamicamente, sem necessidade de salvar o registro para que seja atualizado.

Pressionando o botão finalizar, a vaga torna-se livre para ser utilizada em outro estacionamento, assim como a placa do carro, e a tela de impressão do cupom do estacionamento é apresentada.

Figura 18 - Impressão do cupom do estacionamento

Fonte: Autoria própria

Na impressão do cupom, serão impressos os dias e os meses em que o carro ficou estacionado, caso sejam diferentes de 0. As informações da empresa presentes no cupom são recebidas via AJAX do cadastro da empresa, podendo ser alterados pelo cliente, surtindo efeito imediato para a impressão dos próximos cupons.

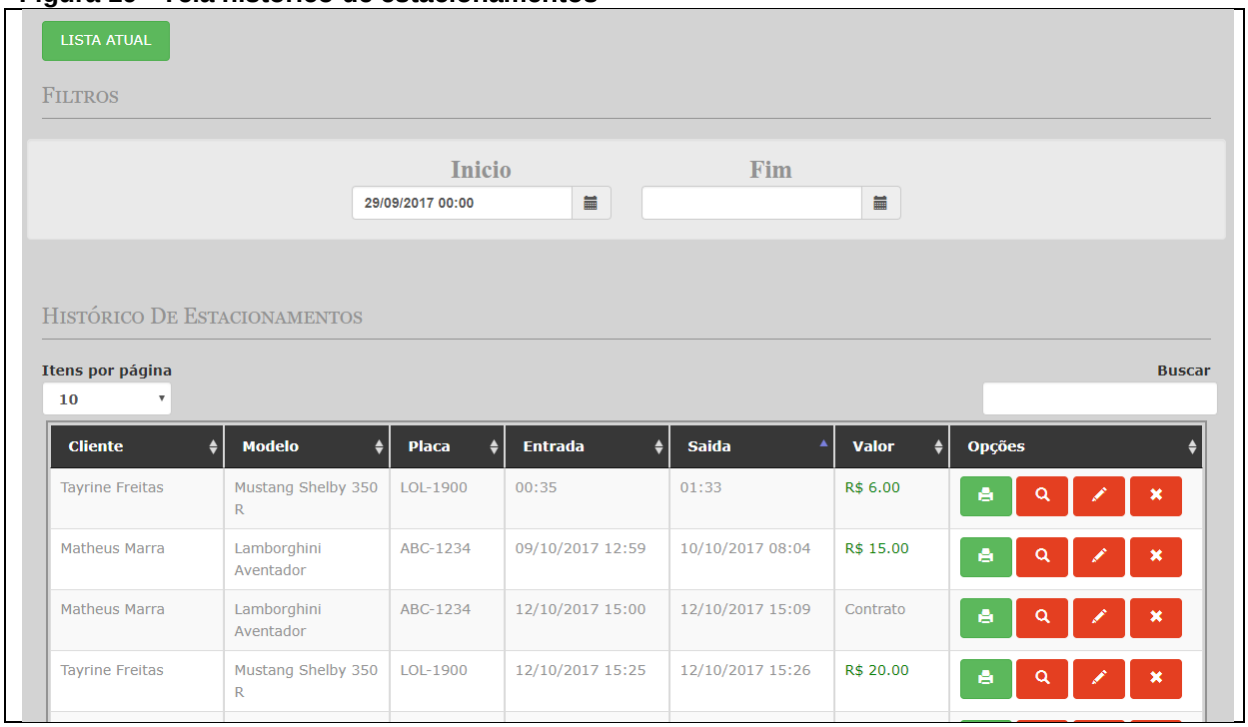
A impressão é feita através do *plugin PrintThis,* sendo somente necessário informar uma div (elemento de agrupamento HTML) para ser impressa. A impressora utilizada para a homologação da impressão foi a impressora térmica *Bematech 2500 TH*, porém, este modelo de impressão também funciona com qualquer tipo de impressora térmica, com bobina de papel no tamanho de 80 milímetros. A impressão também pode ser realizada em folha no tamanho A4.

Outra situação compreendida pela tela de estacionamentos é quando um cliente que possui uma determinada quantidade de estacionamentos e acaba ultrapassando essa quantidade, de forma a estacionar uma quantidade maior de veículos do que possui em contrato. Neste caso, no momento do cadastro o sistema interpreta a quantidade de estacionamentos pendentes do cliente em questão e verifica se o número de contratos é inferior ou igual a este valor. Caso seja igual, o sistema gera um alerta para o usuário informando da situação e solicitando uma confirmação para cadastrar o estacionamento, de forma que seja cobrado do cliente o valor padrão por hora ou diária.

4.2.2.8 Histórico de estacionamentos

Esta tela é muito similar à tela de estacionamentos, sendo suas principais diferenças a apresentação do horário de saída e o valor pago na *DataTable,* e os filtros de início e de fim, para realizar buscas por período, conforme mostra a Figura 19.

Figura 19 - Tela histórico de estacionamentos



Fonte: Autoria própria

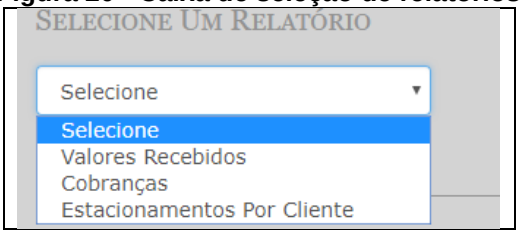
Ao acessar a tela de histórico, a data de início é definida como a data atual menos 15 dias, através do *plugin Moment.* Nesta tela, é possível visualizar e reimprimir o cupom de registros já finalizados, porém, as funções de edição e exclusão dos registros só poderão ser realizadas por usuários com nível administrativo.

4.2.2.9 Relatórios

A tela de relatórios possui uma estrutura diferente das demais. Ao acessá-la, é apresentado somente uma caixa de seleção, solicitando o relatório que se deseja

obter, como apresentado pela Figura 20. Os relatórios só podem ser acessados através de um usuário administrador.

Figura 20 - Caixa de seleção de relatórios



Fonte: Autoria própria

Ao selecionar o tipo do relatório desejado, as opções de filtros e a tabela correspondente são apresentadas. Dentre as três opções de relatórios disponíveis, existem algumas peculiaridades:

1) Valores Recebidos: ao selecionar este relatório, os filtros de seleção de data de início e de fim da consulta são apresentados, juntamente com a lista dos estacionamentos do período, considerando somente os estacionamentos que não possuem contratos. Também é apresentado um campo com o total dos valores do período.
2) Cobranças: selecionando este relatório, os filtros de início e fim são apresentados juntamente com o filtro para inclusão de clientes não presentes no período. Se o filtro estiver desmarcado, será apresentado um *DataTable* com todos os clientes de contrato que estacionaram seus veículos no período informado, e um campo com a soma dos valores apresentados. Caso o filtro “Não presentes” esteja marcado, serão apresentados dois *DataTables*, o primeiro contendo os dados dos clientes que estacionaram no período e o segundo contendo os dados dos clientes que não estacionaram no período informado. São apresentados nas tabelas o nome do cliente, o telefone, a quantidade de contratos do cliente e o valor da mensalidade. Este relatório, diferente dos outros, utiliza-se da data de saída do estacionamento para aplicar os filtros definidos pelo usuário, pois o recebimento pelo tempo estacionado dá-se no momento da saída.
3) Estacionamentos por cliente: os dados deste relatório são similares aos da tela de histórico de estacionamentos, com a diferença que neste relatório existe um

filtro para fazer a busca dos estacionamentos de um cliente específico dentro do período informado e o campo com o total pago pelo cliente ao fim do relatório. O filtro de cliente somente irá apresentar os clientes que possuem carros cadastrados. Neste relatório poderão aparecer registros de estacionamentos ainda pendentes.

Os filtros de início dos três relatórios são preenchidos automaticamente ao entrar na tela, com o valor de 15 dias anteriores ao dia atual. Também é possível reimprimir um cupom de um estacionamento já finalizado a partir desta tela.

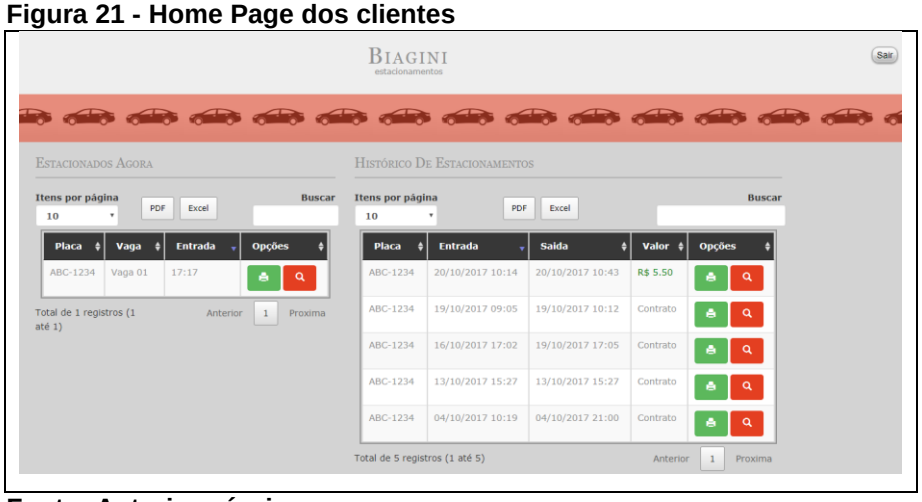
4.2.2.10 *Login* de clientes

Esta é uma tela simples, semelhante à tela de *login* normal do sistema, porém somente com um campo para ser informado o CPF do cliente ou a placa do veículo que se deseja consultar. Ao serem validados os dados, o cliente é redirecionado para a *home page* dos clientes.

4.2.2.11 *Home page* de clientes

O servidor recebe o valor informado na tela de *login* de clientes e verifica que tipo de dado foi informado. Caso tenha sido um CPF válido, o servidor retorna os uma lista com os últimos três estacionamentos do cliente, juntamente com uma lista dos carros estacionados no momento atual e um campo com o valor da mensalidade que o cliente tem cadastrado, caso possua contrato.

Figura 21 - Home Page dos clientes



Fonte: Autoria própria

Como é possível observar na figura 21, a lista dos últimos 3 estacionamentos e dos estacionamentos pendentes dividem-se entre 2 *DataTables*.

Caso o servidor identifique que o valor recebido é referente à uma placa de veículo, serão retornados para o cliente uma lista dos últimos estacionamentos do veículo em questão, e caso ele esteja estacionado no momento, o registro desse estacionamento também é enviado ao cliente. Neste caso, não é apresentado o valor da mensalidade do cliente.

Com todas as telas propostas apresentadas e todas as funções aplicadas, concluiu-se o desenvolvimento do sistema de estacionamento.

4.2.2.12 Observações finais

Tendo cumprido todos os objetivos estabelecidos, ainda restam algumas observações a serem feitas sobre o sistema. Algumas funcionalidades que não podem ser consideradas primordiais, mas que podem ser destacadas são:

Todos os campos para atribuição nominal do sistema, como nome do cliente, modelo do veículo, nome fantasia e nome da vaga são capitalizados automaticamente pelo sistema, para evitar problemas de padronização de fonte, o que é recorrente na maior parte dos sistemas.

Todas as *modais*, sejam elas de cadastro, edição ou finalização, no caso do estacionamento, focam o campo de maior relevância no momento em que são abertas. Esta funcionalidade visa diminuir o máximo possível de interações desnecessárias como mover o mouse até o campo, ou apertar a tecla TAB continuamente até atingir o campo de interesse, visando otimizar as tarefas do operador do sistema.

Todos os dados do sistema são armazenados sem suas máscaras no banco de dados, de forma a fazer o tratamento das máscaras no cadastro, edição, visualização e listagem de suas respectivas telas, a fim de manter uma base de dados organizada e íntegra, de fácil exportação e tratamento de dados, caso necessário.

Por fim, com a maior parte das funcionalidades descritas, é possível observar que o sistema possui uma metodologia de desenvolvimento voltada para a

experiência do usuário, de forma a facilitar e otimizar suas atividades do dia a dia e trazer-lhe segurança quanto a integridade dos dados apresentados pelo sistema.

5 CONCLUSÃO

Com base nos resultados apresentados nesse trabalho, conclui-se que o sistema desenvolvido atende todas as necessidades citadas pelo administrador da empresa solicitante, além de contar com funcionalidades que não estavam descritas no plano inicial do sistema. Este resultado pôde ser obtido através do estudo das atividades executadas durante a operação do estacionamento, as reais dificuldades e idealização da melhor forma para tratar de tais situações, visando aprimorar a experiência do utilizador do software.

A aplicação conta com 12 telas, agrupadas em um *layout* responsivo, que permite fácil acesso em *smartphones* e *tablets*, além de computadores convencionais. Dentre as 12 telas, foram programadas 4633 linhas de códigos somente em *Javascript*, desconsiderando as páginas em HTML, CSS e todo o servidor. O projeto Final conta com a utilização de dois frameworks (*Laravel* e *Bootstrap*), a biblioteca *JQuery* e 6 de seus *plugins,* sendo eles o *DataTables, Masked Input, Select2, Moment, Print This* e *DateTime Picker.*

A utilização do sistema pela empresa contratante pode ser feita através de hospedagem em nuvem, realizando a contratação de um domínio próprio da empresa e locação de um servidor para hospedar o site, ou ainda pode ser utilizado localmente, caso a empresa deseje operar somente em sua rede interna. Para esta opção, somente é necessário um computador que sirva como servidor da aplicação.

5.1 TRABALHOS FUTUROS

Os conceitos e técnicas aplicados durante o desenvolvimento permitem não só uma fácil manutenção, mas possibilidade de expansão, através do desenvolvimento de novos módulos e novos relatórios. Também podem ser feitas integrações com outros Webservices, como consulta dos dados do cliente a partir do CPF, e até mesmo a emissão de nota fiscal de serviços, através da homologação e integração com o Webservice da prefeitura da cidade de Ponta Grossa e das cidades atendidas pelo sistema.

REFERÊNCIAS

ACHOUR, Mehdi et al. **Manual do PHP.** Disponível em: <http://php.net/manual/pt_BR/>. Acesso em: 23 set. 2017.

BEAL, Vangie. **ODBC - Open DataBase Connectivity.** Disponível em: <http://www.Webopedia.com/TERM/O/ODBC.html>. Acesso em: 08 out. 2017.

BURBECK, Steve. **Applications Programming in Smalltalk-80(TM)**: How to use Model-View-Controller (MVC). 1992. Disponível em: <https://Web.archive.org/Web/20120729161926/http://stwww.cs.illinois.edu/users/smarch/st-docs/mvc.html>. Acesso em: 09 out. 2017.

CARVALHO, Rodrigo Baroni de. **Aplicações de softwares de gestão do conhecimento:** tipologia e usos. Belo Horizonte: Escola de Ciência da Informação Universidade Federal de Minas Gerais, 2000.

CONTE, Tayana; MENDES, Emilia; TRAVASSOS, Guilherme. **Processos de Desenvolvimento para Aplicações Web:** Uma Revisão Sistemática. Disponível em: <https://www.researchgate.net/publication/228647730_Processos_de_Desenvolvimento_para_Aplicacoes_Web_Uma_Revisao_Sistematica>. Acesso em: 30 set. 2017.

CONVERSE, Tim; PARK, Joyce. **PHP:** a Bíblia. 2. ed. Rio de Janeiro: Campus, 2003.

CRANLEY, R.; BENEDETTI, R.; COUTO, E. **Use a Cabeça!** JQuery: O Guia Amigo do seu Cérebro. Alta books, 2013. Disponível em: <http://books.google.com.br/books?id=IH0NnQEACAAJ>.

DEBOLT, Virginia. **Integrated HTML and CSS:** A Smarter, Faster Way to Learn. Hoboken, Nova Jersey: John Wiley & Sons, 2006.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. 8. ed. Rio de Janeiro, RJ: Elsevier Brasil, 2004.

DEITOS, Maria Lúcia M. Souza. **A gestão da tecnologia em pequenas e médias empresas:** fatores limitantes e formas de superação. Cascavel: Edunioeste, 2002.

FERREIRA, Érico Dias**. Desenvolvimento De Um Sistema Para O Gerenciamento Do Processo De Trabalho De Conclusão De Curso.** Guarapuava: UTFPR, 2015.

FIELDING, Roy; GETTYS, James; MOGUL, Jeffrey; FRYSTYK, Henrik; MASINTER, Larry, LEACH, Paul; BERNERS-LEE, Tim. **Hypertext Transfer Protocol -- HTTP/1.1**. Disponível em: <https://www.rfc-editor.org/info/rfc2616>. Acesso em: 08 out. 2017.

FLANAGAN, David. **JavaScript**: The Definitive Guide. 6. ed. Estados Unidos da América: O'Reilly Media, 2011.

GARCIA, Elaini Luvisari**. A RESPONSABILIDADE CIVIL OBJETIVA DOS ESTACIONAMENTOS.** Disponível em: <http://faef.revista.inf.br/imagens_arquivos/arquivos_destaque/oUswl9WFGcTeZS2_2014-12-15-19-2-11.pdf>. Acesso em: 27 ago. 2017.

GRAY, Jim; REUTER, Andreas. **Transaction Processing:** Concepts and Techniques. San Francisco: Morgan Kaufmann, 2004.

KYRNIN, Jen. **What is Ajax?**. Disponível em: <http://webdesign.about.com/od/ajax/a/aa101705.htmnifer>. Acesso em: 11 out. 2017.

LÓPEZ, Guillermo. **Sublime Text Unofficial Documentation**. 2016. Disponível em: <http://docs.sublimetext.info/en/latest/index.html>. Acesso em: 08 out. 2017.

MARCONDES, Christian Alfim**. HTML 4.0 fundamental:** A base da programação web, São Paulo, Érica, 2005.

MCFARLAND, David Sawyer. **JavaScript & jQuery: The Missing Manual**. 2. ed. Sebastopol, Ca: O'reilly Media, 2011.

MAESTA, V. P. **Utilização Por Micro E Pequenas Empresas Do Sistema De Informações Gerenciais - Sig**. Apucarana: Faculdade de Apucarana - FAP, 2012. Disponível em:<http://www.classecontabil.com.br/media/trabalhos/UTILIZACAOPORMICROEPEQUENASEMPRESASDESISTEMADEINFORMACOESGERENCIAIS-SIGPARAREVISTA.pdf>. Acesso em: 19 de ago. 2017.

MUTO, Claudio Adonai. **PHP & MySQL:** Guia Introdutório. 3. ed. RJ: Brassport, 2006.

NIEDERAUER, Juliano. **Desenvolvendo Websites com PHP.** 2. ed. São Paulo: Novatec, 2004.

NATIONS, Daniel. **Improve Your Understanding of Web Applications:** What Is a Web Application?. 2016. Disponível em: <https://www.lifewire.com/what-is-a-Web-application-3486637>. Acesso em: 30 set. 2017.

OFFUTT, Jeff. Quality attributes of Web software applications. **Ieee Software,** [s.l.], v. 19, n. 2, p.25-32, 2002. Institute of Electrical and Electronics Engineers (IEEE). Disponível em: <http://dx.doi.org/10.1109/52.991329>. Acesso em: 30 set. 2017.

OTTO, Mark. **Building Twitter Bootstrap.** 2012. Disponível em: <https://alistapart.com/article/building-twitter-bootstrap>. Acesso em: 06 out. 2017.

PRATES, Juliano Niederauer Rubens. **MySQL:** Guia de Consulta Rápida. Novatec, 2000. Disponível em: <http://www.martinsfontespaulista.com.br/anexos/produtos/capitulos/209813.pdf>. Acesso em: 28 out. 2017

RAGGETT, Dave; HORS, Arnaud Le; JACOBS, Ian. **HTML 4.01 Specification.** 1999. Disponível em: <https://www.w3.org/TR/html401/>. Acesso em: 04 out. 2017.

RIEHLE, Dirk. **Framework Design**: A Role Modeling Approach. Ph.D. Thesis, No. 13509. Zürich, Switzerland, ETH Zürich, 2000.

ROBBINS, Jennifer Niederst. **Learning Web Design:** A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. 4. ed. Canadá: O'reilly Media, Inc., 2012.

ROCHA, Roberto; CORREA, Marcos Araújo; FERREIRA, Jane Marry; RIBEIRO, Thiago Vasconcelos; ROCHA, Joice Rodrigues. A utilização dos sistemas de informação como ferramenta efetiva para a gestão empresarial em micro e pequenas empresas do comércio varejista. **Revista Inovação, Gestão e Produção, v**ol. 02, n. 10, p. 97-110, outubro de 2010. Disponível em: <http://www.ingepro.com.br/Publ_2010/Out/320-884-1-PB.pdf>. Acesso em: 19 ago. 2017.

SERVIÇO BRASILEIRO DE APOIO ÀS MICRO E PEQUENAS EMPRESAS. **Participação das Micro e Pequenas Empresas na Economia Brasileira.** Disponível em: <https://m.sebrae.com.br/Sebrae/Portal%20Sebrae/Estudos%20e%20Pesquisas/Par

ticipacao%20das%20micro%20e%20pequenas%20empresas.pdf> Acesso em: 25 ago. 2017.

STAIR, Ralph; REYNOLDS, George. **Fundamentals of Information Systems.** 6. ed. Boston: Course Technology, 2012.

SURGUY, Maks. **History of Laravel PHP framework, Eloquence emerging**. Disponível em: <https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/>. Acesso em: 05 out. 2017.

VIDAL, Antonio Geraldo. **Informática na pequena e média empresa.** São Paulo: Pioneira, 1995.

W3SCHOOLS. **HTML & CSS**. Disponível em: <http://www.w3.org/standards/webdesign/htmlcss#whatcss>. Acesso em: 12 out. 2017.

WALIA, Er Saurabh; GILL, Er Satinderjit Kaur. **A framework for Web based student record management system using PHP**. International Journal of Computer Science and Mobile Computing, v. 3, n. 8, p. 24-33, 2014.

WELLING, Luke; THOMSON, Laura. **PHP and MySQL Web Development.** 5. ed. New Jersey: Pearson Education, 2001.

YU, He Ren. **Design and implementation of Web based on Laravel framework**. Atl. Press, no. Iccset, v. 2014, p. 301-304, 2015.

ZEESHAN, Afzaal Ahmad. **Understanding ASP.NET MVC using real world example, for beginners and intermediate.** Disponível em: <http://www.c-sharpcorner.com/UploadFile/201fc1/understanding-Asp-Net-mvc-using-real-world-example-for-begi/>. Acesso em: 25 set. 2017

APÊNDICE A - PERGUNTAS SOBRE AS PRINCIPAIS NECESSIDADES EM UM SISTEMA DE GESTÃO

QUESTIONÁRIO SOBRE NECESSIDADES DE UM SISTEMA DE GESTÃO

Para dar início ao desenvolvimento do sistema, fez-se necessário realizar uma entrevista sobre as principais dificuldades vivenciadas pela empresa e as principais tarefas a serem realizadas. As perguntas realizadas durante a entrevista encontram-se a seguir:

1. Quais as principais funcionalidades você espera em um sistema?

R: O sistema deve salvar as informações sobre o estacionamento e no momento da saída do cliente, gerar uma impressão destes dados. Precisa manter os cadastros dos clientes mensalistas e gerar um relatório sobre os valores recebidos.

2. O que você sente necessidade de registrar em um sistema?

R: Para os estacionamentos, é necessário salvar a hora de entrada, hora de saída, placa e valor. Para o cadastro de clientes, os dados básicos como nome, CPF ou CNPJ, telefone e endereço.

3. Para todos os estacionamentos, você salva os dados do cliente?

R: Não, na verdade, na maior parte dos casos o cliente somente deixa o carro no estacionamento sem fazer cadastro.

4. Como funciona a questão de preços em relação à uma diária?

R: Na diária, são contadas 24 horas a partir do momento em que o cliente estaciona o carro. Elas têm valor fixo de 30 reais, mas se o cliente passar das 24 horas é cobrado o preço normal por hora.

5. Existe algum tipo de tolerância caso o cliente se atrase para buscar o carro?

R: Sim, em geral nos damos 10 minutos de tolerância, se o cliente passar disso é cobrado o valor da tabela para o tempo que ele ficou extra.