Instituto Federal de Educação, Ciência e Tecnologia Rio Grande do Norte

Projeto Final da Disciplina de Projeto de Desenvolvimento de Sistemas para Internet - Curso Técnico em Informática para Internet

Desenvolvimento de um sistema web para controle de veículos e acesso aos estacionamentos dos campi do IFRN

José Victor Gomes de França

Natal-RN, Novembro de 2019

Lista de ilustrações

Lista de tabelas

Sumário

Parte I

Preparação da pesquisa

1 INTRODUÇÃO

Quando a demanda por vagas de estacionamento aumentam demasiadamente, inevitavelmente se estabelece um reconhecido problema de congestionamento. Atualmente, é comum situações em que, por exemplo, nos campus universitários e regiões próximas aos centros das grandes cidades, a capacidade dos estacionamentos se esgotem facilmente. Geralmente dimensiona-se o número de vagas alocadas para estacionamentos no plano diretor das instituições públicas ou privadas. Estas vagas podem ser, à princípio, suficientes para atender a demanda de estacionamento dos veículos dos professores, funcionários, estudantes e visitantes no anos iniciais. Porém, a previsão de crescimento do número de usuários, pode ser, por algum motivo, subestimada. Nas situações em que a demanda pelo estacionamento continua a crescer, o espaço reservado para acomodação dos veículos torna-se limitado. Outra situação a ser considerada é a de que alguma área livre do campus, geralmente seja utilizada na construção de novas salas de aula, escritórios e laboratórios, ou seja, as infraestruturas que são consideradas mais importantes. Em vista destas situações, recorrem-se às estratégias de gestão e a Sistemas de Transporte Inteligentes (Intelligent Transportation Systems, ITS) para lidar com o problema de congestionamento do estacionamento, em vez da capacidade de expansão. Pode-se argumentar que o acesso ao estacionamento é um fator que determina se e como os usuários chegam aos campus das universidades e institutos federais. Normalmente, nestes estabelecimentos, a maioria dos usuários possuem seu próprio horário de trabalho ou estudo e a demanda de estacionamento é relativamente inelástica. Algumas alternativas para mitigar o problema do congestionamento nos estacionamentos podem ser antipáticas aos usuários, dentre estas, pode-se citar a administração do estacionamento por meio de uma combinação de cobrança pelo direito de estacionar ou fornecer modos de transporte alternativos. Outra abordagem admitida para resolver o problema de congestionamento do estacionamento do campus é resolver o problema pelo lado da oferta; isto é, implementar políticas para melhor controle do uso do estacionamento e usar o ITS para tornar o uso do estacionamento mais eficiente. A dificuldade de encontrar uma vaga de estacionamento não apenas contribui para o congestionamento do tráfego dentro da repartição ou estabelecimento, mas também nas ruas circundantes, ao procurar uma vaga vazia para estacionar, os veículos em circulação utilizam a marcha lenta e podem aumentar as emissões de gases nocivos resultantes da queima de combustível, afetando negativamente a saúde da comunidade e prejudicando o meio ambiente. Deve-se aumentar a conscientização dos tomadores de decisão das instituições públicas e privadas sobre a importância de evitar o congestionamento dos estacionamentos e aumentar a compreensão dos impactos dos estacionamentos no meio ambiente e na saúde da comunidade. O ideal a ser alcançado deve, preferencialmente, contar com projetos de estacionamentos que

consigam evitar seu congestionamento.

1.1 Tema

Alguns campus do IFRN, notadamente, o de Natal Central, vêm enfrentando situações semelhantes as descritas anteriormente. No intuito de organizar e registrar o ingresso de veículos as suas dependências, este trabalho de conclusão de curso propõe o desenvolvimento de um sistema web para controle dos veículos e acesso ao estacionamento do Instituto Federal de Educação, Ciência e Tecnologia Rio Grande do Norte - IFRN, visando controlar os veículos que podem ter acesso ao estacionamento da instituição.

1.2 Objetivo Geral

A proposta deste trabalho de conclusão de curso consiste em desenvolver um sistema web administrativo completo, com autenticação, para controle de veículos e acesso aos estacionamentos dos campi do IFRN.

1.3 Objetivos Específicos

- Utilizar a estrutura Ruby on Rails (RoR) (HARTL; FERNANDEZ, 2011) para desenvolvimento do sistema web.
- Utilizar o banco de dados PostgreSQL.
- Apresentar um caso de uso com o sistema elaborado.

1.4 Delimitação do Problema

Apresentar um sistema web administrativo completo, com autenticação, para controle de veículos e acesso aos estacionamentos dos campi do IFRN.

1.5 Motivações e Justificativas

Tendo em vista que a quantidade de vagas no estacionamento do campus Natal Central do IFRN é limitado e que a quantidade de veículos ocupantes está aumentando gradativamente, é necessário controlar o acesso dos veículos que realmente possuem prerrogativa para sua utilização, tais como aqueles pertencentes a servidores do instituto e visitantes. Além da quantidade, o processo para cadastro de um funcionário, que possua um veículo, é bastante burocrático, exigindo a comunicação entre um certo número

de funcionários, por exemplo, diretor e departamento de segurança, para autorização do registro, dificultando a aquisição do adesivo que identifica um veículo habilitado. A abordagem de uma solução para o problema de congestionamento e dificuldades para organização e liberação de autorizações, culmina na necessidade do desenvolvimento de um sistema que gerencie esse, e, possivelmente, outros estacionamentos. Esta é uma grande oportunidade de aplicar os conceitos apresentados nas diversas disciplinas estudadas no curso técnico de informática para internet, ofertado pelo IFRN, na modalidade de ensino à distância. Particularmente, no presente trabalho, optou-se por desenvolver o referido sistema utilizando a(o) estrutura/framework Ruby On Rails (RoR) e banco de dados PostgreSQL. Nesse sentido, o desafio que se apresentou é de certa maneira complexo e instigante, pois tratam-se de ferramentas elaboradas e que exigem considerável nível de conhecimento para sua utilização adequada.

1.6 Método de Trabalho

O presente trabalho de conclusão de curso utilizou as funcionalidades do Rails Admin, um mecanismo, disponibilizado na internet, que fornece uma interface fácil de usar para gerenciamento de dados (SCORZA, 2018). Adicionalmente, utilizou-se como Sistema de Gerenciamento de Banco de Dados (SGBD), o PostgreSQL. Identificou-se como uma necessidades muito importante para a comunidade acadêmica do IFRN, propor soluções para o problema de desorganização do acesso dos usuários aos estacionamentos da instituição. Dessa forma, o presente trabalho propõe um meio de gerenciar de maneira simples os estacionamentos da instituição. Inicialmente, analisou-se as causas mais prováveis da má utilização dos estacionamentos. Pode-se citar, por exemplo, o acesso de pessoas não autorizadas, que não possuem vínculo com o IFRN.

Concomitantemente, estudou-se mais aprofundadamente a linguagem de programação Ruby, e a utilização da estrutura Ruby on Rails. Com essas ferramentas principais implementou-se um sistema administrativo em conjunto com a base de dados. Além disso foram empregados conhecimentos em HTML, CSS e Javascript. Para a criação da plataforma que irá gerenciar o estacionamento de veículos, foi utilizada `gems` específicas, que simplificam a criação de um sistema administrativo. Dessa forma, criou-se os seguintes modelos: `estacionamento`, `veiculo`, `funcionario` e `vaga`. Os parâmetros de cada modelo, bem como suas relações e as `gems` utilizadas, serão descritas mais à frente.

1.7 Organização do Trabalho

Em sua organização, o presente trabalho possui capítulos apresentados da seguinte maneira:

- Desenvolvimento:
  No segundo capítulo são apresentadas as ferramentas tecnológicas empregadas na construção do software e os requisitos funcionais e não-funcionais. Nessa oportunidade descrevem-se em detalhes cada aplicação realizada. Mostram-se, adicionalmente, a maneira de integração com o banco de dados, diagramas para o entendimento do sistema criado, as categorias de usuários e as atribuições que esses tem acesso.

- Avaliação:
  No terceiro capítulo, caracteriza-se a verificação do sistema e apresentam-se os resultados de seu funcionamento.

- Conclusão:
  No quarto e último capítulo, argumenta-se conclusivamente a respeito do software construído. Igualmente, pondera-se sobre possíveis melhorias em trabalhos futuros.

1.8 Equipe e Infraestrutura

O presente TCC foi concebido e escrito individualmente pelo autor que contou com uma infraestrutura simples constituída por seu notebook pessoal *idepad*^{S145}, core `i5`, marca lenovo e acesso à intenet.

1.9 Cronograma de Acompanhamento

O cronograma do trabalho encontra-se na Tabela 1, onde as etapas são:

- 1º) **Iniciação:** Definição do problema; Apresentação de proposta de solução com os requisitos funcionais e suplementares; Levantamento bibliográfico; Feedback do orientador.

- 2º) **Elaboração:** Protótipo de Arquitetura; Análise inicial; Estudo e utilização da `gem` 'rails_admin'; `gem` 'Devise'; Estudo aprofundado de Relacionamentos em Ruby; Estudo detalhado em gerenciamento de banco de dados utilizando PostgreSQL; Feedback do orientador.

- 3º) **Construção:** Implementação e verificação dos requisitos que foram definidos; Feedback do orientador.

- 4º) **Transição:** Conclusão do protótipo e documentação; Feedback do orientador.

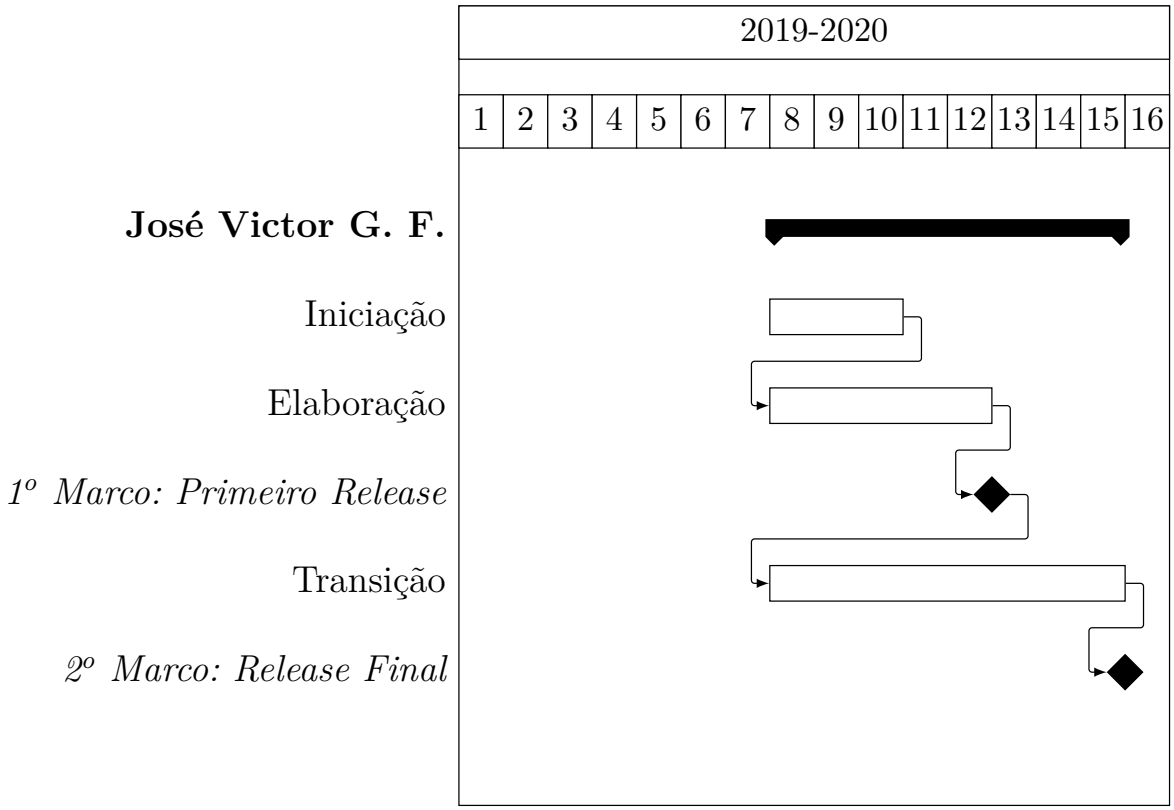
Tabela 1 – Cronograma

Etapa	ago.	set.	out.	nov.	dez.	jan. (2020).	fev. (2020)	mar. (2020)
1	x	x	x					
2	x	x	x	x	x	x	x	
3		x	x	x	x	x	x	x
4						x	x	x

1.10 Marcos do Projeto

A Figura 2 mostra os marcos significativos do projeto, indicando os artefatos que serão entregues.

Tabela 2 – Marcos do Projeto



1.11 Gerência de Riscos

Para o desenvolvimento da plataforma de gerenciamento do presente trabalho buscou-se controlar, na medida do possível, alguns fatores. As Fig. 1 e 2 mostram, respectivamente, os riscos externos e internos presumidos.

Figura 1 – Riscos Externos.

Nº	Identificação
1	causas/fenômenos naturais
2	crises políticas de uma região ou país
3	crises econômicas
4	doenças
5	problemas do financiador do projeto ou do fornecedor de insumos/recursos;
6	alterações na legislação
7	pressões da organização, da sociedade, do cliente
8	alterações no mercado quanto às suas necessidades ou capacidade de compra

Figura 2 – Riscos Internos.

Nº	Identificação
1	equipamentos defeituosos
2	equipe despreparada para trabalho em grupo
3	falta de domínio tecnológico
4	gastos excessivos
5	estouro de prazo devido a falhas de desenvolvimento
6	estouro de prazo devido a erros no gerenciamento necessidades tecnológicas desconsideradas durante o planejamento das etapas
7	a complexidade do sistema, não devidamente percebida nas etapas iniciais
8	alterações no escopo do projeto.

1.12 Qualidade do Sistema

O presente projeto levou em consideração o desenvolvimento de um software de qualidade primando por sua confiabilidade e usabilidade. Atualmente, trabalha-se para que a plataforma de gerenciamento execute completamente suas funcionalidades. O emprego das `gems` mencionadas na seção 1.9 proporciona ao sistema fácil manutenção, mantendo a integridade dos dados e evitando possíveis falhas. A referência básica que levou-se em consideração foi a ISO/IEC 9126(NBR13596) que pode ser consultada em (DEVMEDIA, 2008). Para se garantir a qualidade do produto durante seu desenvolvimento seguiu-se um conjunto de métodos e técnicas cuja implementação têm acontecido ao longo das etapas. As atividades de validação, verificação e testes contarão com revisões de software e é

apresentada na seção 2.8

2 Estado da Arte

No presente capítulo serão apontados os fundamentos teóricos nos quais o desenvolvimento do sistema web de gerenciamento foram embasados, ou seja, como se constituem os padrões de organização do RoR e o comportamento de aplicações web. Adicionalmente, mostra-se uma descrição cuidadosa e sucinta de trabalhos relacionados à área do sistema desenvolvido, que mostram um recorte adequado para visualização do universo de estudo. Em seguida, apresentam-se os atores, isso é, os usuários finais do sistema. Como mencionado anteriormente, o software é direcionado à comunidade acadêmica do IFRN. Também são apresentadas, detalhadamente, todas as funcionalidades e delimitações do software. Na última seção, serão mostrados o diagrama de casos e uso. Uma perspectiva comparativa será realizada, relacionando as possibilidades possíveis de cadastramento .

2.1 Desenvolvimento web

Desenvolvimento web é o termo utilizado para descrever a criação de sites, na Internet ou numa intranet. Atualmente, alguns sites são aplicativos complexos capazes de realizar transações, apresentar dados em tempo real, e fornecer experiências interativas ao usuário. Os software baseados na web estão se tornando tão poderosos e importante quanto software de mesa. A grande maioria dos aplicativos da web produzidos fornecem funcionalidades avançadas, e isso é uma tarefa complexa que envolve um considerável número de conjuntos de ferramentas, e muitas opções. As estruturas da web criadas com tais tecnologias necessitam de um gerenciamento de conteúdo. Para esse objetivo, costuma-se utilizar ferramentas que construam uma estrutura web, permitindo escolher entre diferentes tipos de elementos que melhor atendam as necessidades (HACKER, 2008). Em sua investigação, (PLEKHANOVA, 2009), avaliou três estruturas de desenvolvimento web principais de código fonte abertas: Django, Ruby on Rails (RoR) e CakePHP, escritos em três linguagens diferentes - Python, Ruby e PHP, respectivamente. Todas as três estruturas citadas têm arquiteturas semelhantes e reivindicam ter características parecidas, como produtividade aprimorada e reutilização de código. Em sua conclusão, a autora classificou as ferramentas segundo critérios que verificaram:

- Os componentes front-end e back-end (desenvolvimento da interface do usuário, gerenciamento e migração de dados).
- O desempenho total das estruturas (manutenibilidade, testabilidade).
- A popularidade, difusão na comunidade, maturidade e comercialização.

A conclusão geral dessa autora foi de que todas as três estruturas são ferramentas robustas e a escolha qual utilizar precisa considerar o contexto de aplicação. As empresas e os profissionais de desenvolvimento web necessitam individualizar o processo avaliativo de escolha ao seu contexto único.

(LEI; MA; TAN, 2014), realizou uma investigação concentrando-se no impacto do desempenho de três diferentes tecnologias da Web: Node.js, PHP e Python-Web. No entanto, negligenciou os problemas de segurança e escalabilidade. Nessa perspectiva, desenvolveu um método universal de avaliação da técnica de desenvolvimento web com base na comparação do desempenho das ferramentas analisadas. Em sua conclusão verificou que o Node.js tem um desempenho muito melhor que o tradicional PHP em situação de alta simultaneidade e que o Python-Web também não é adequado para o uso intensivo em computação.

Em síntese, pode-se argumentar que o gerenciamento de conteúdo da web é o resultado da entrega de material para a web. O crescimento do número de páginas da web tornou-se popular nas últimas décadas e os padrões Model-View-Controller (MVC) facilitaram bastante seu desenvolvimento (MCKEEVER, 2003). Eles ocultam a complexidade, dão estrutura e consistência e promovem as melhores práticas.

A partir de sua experiência, (TAPIADOR; SALVACHÚA, 2012) explora a criação de sistemas de gerenciamento de conteúdo da web com Ruby on Rails, uma estrutura popular de desenvolvimento web produzida para aumentar a produtividade. Em sua investigação o autor explica as vantagens da implementação da arquitetura MVC baseando-se nos príncipios de “convention over configuration” (convenção sobre configuração) e “don’t repeat yourself” (não se repita) (BÄCHLE; KIRCHBERG, 2007).

Diante dessa realidade, no presente trabalho de conclusão de curso, optou-se, como componente orientador do processo de desenvolvimento do sistema de gerenciamento de estacionamentos, aderir à doutrina Rails (TAYLOR, 2010), uma vez que partindo desta ferramenta é possível estabelecer as consistências e as principais dependências do sistema (ASTELS; MILLER; NOVAK, 2002).

2.2 Descrição da arquitetura Model-View-Controller

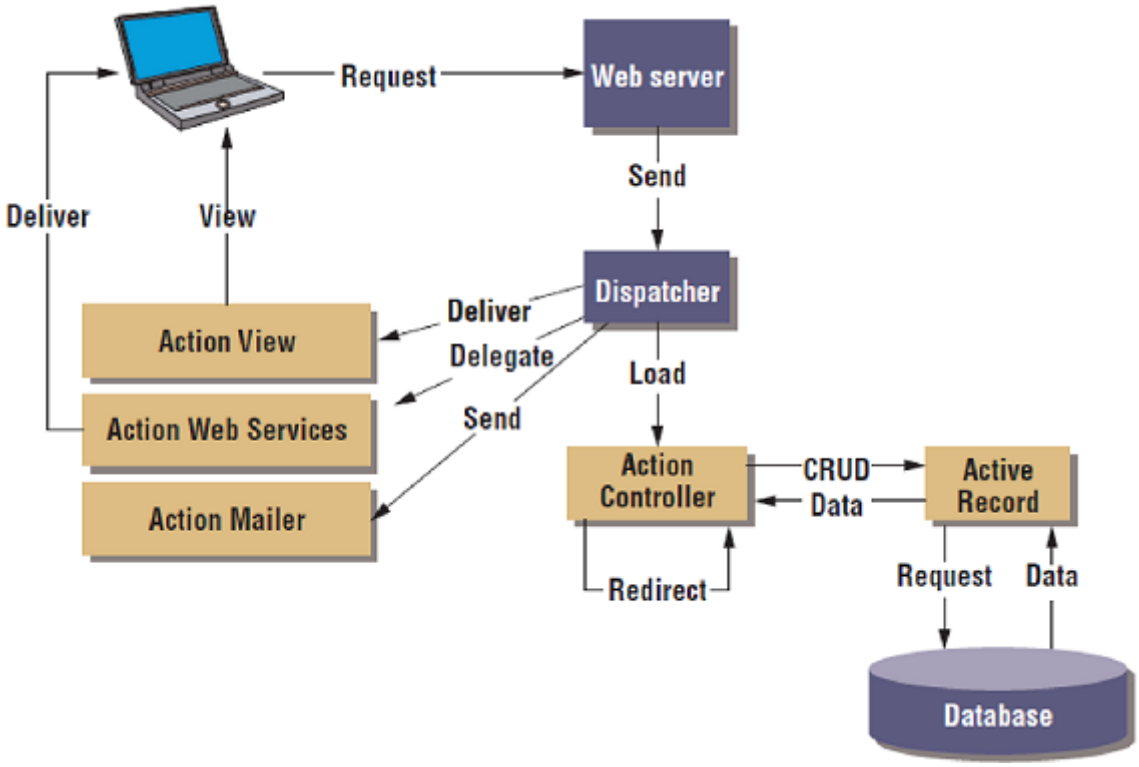
O Rails possiu diversas características interessantes. Dessa propriedades, pode-se destacar algumas considerações bastante sérias, por exemplo as restrições sobre como se estrutura a construção dos aplicativos da web. Surpreendentemente, essas restrições, na realidade, facilitam a criação dos aplicativos. Elaborado em 1979 por Trygve Reenskaug, essa forma de organização é aplicada no desenvolvimento de funcionalidades interativas, dividindo-se em modelos, visões e controladores (RUBY; THOMAS et al., 2009). O modelo é responsável por manter o estado do aplicativo. As vezes esse estado é transitório, durando

apenas algumas interações com o usuário. Às vezes, o estado é permanente e é armazenado fora do aplicativo, frequentemente em um banco de dados.

A visualização é responsável por gerar uma interface com o usuário, normalmente baseados em dados do modelo. Embora a visualização possa apresentar ao usuário várias maneiras de inserir dados, a própria exibição nunca lida com dados recebidos. O trabalho da visualização é concluído depois que os dados são exibidos. Os controladores recebem eventos do mundo exterior (normalmente, a entrada de utilizador), interagem com o modelo, e apresentam uma pespectiva apropriada para o usuário. Na atualidade, o padrão de organização MVC prevalece como aspecto de criação dos frameworks.

Em geral, os aplicativos da web guardam suas informações em um banco de dados relacional. Os sistemas de entrada de solicitações armazenam informações e detalhes do cliente em tabelas de banco de dados. As consultas ao banco de dados relacionais são fundamentadas matematicamente na teoria de conjuntos. Porém, existe uma certa dificuldade em combinar bancos de dados relacionais com linguagens de programação orientadas a objetos (OO). Os objetos lidam com dados e operações, e os bancos de dados são todos sobre conjuntos de valores. Muitas vezes, torna-se complicado expressar termos relacionais codificando-os em um sistema OO. O contrário também é verdade.

Figura 3 – Estrutura MVC do RoR.



Na seção seguinte serão apresentados os atores, requisitos funcionais e não funcionais, o diagrama de caso de uso e informações sobre o banco de dados utilizado.

2.3 Banco de Dados

Para o desenvolvimento desse projeto, implementou-se um banco de dados PostgreSQL, para armazenamento de informações sobre os usuários. A construção do Sistema de Gerenciamento de Banco de Dados do PostgreSQL foi iniciada em 1986. Os conceitos iniciais foram apresentados em (STONEBRAKER; ROWE, 1986). A primeira demonstração do sistema em operação foi realizada em 1987 e o modelo de dados inicial apareceu em (STONEBRAKER et al., 1988).

2.4 Requesitos do Sistema

2.4.1 Ator

Os atores que poderão acessar o sistema são os seguintes:

Segurança: É o funcionário do IFRN responsável por realizar o cadastramento dos demais servidores ou alunos.

Administrador: É o funcionário do setor de TI responsável pela administração da plataforma. Esse ator tem as permissões necessárias para gerenciar a plataforma, realizando atualizações e melhorias, se necessárias.

2.4.2 Requisitos Funcionais

Na descrição desse tópico serão descritas as aplicações dos usuários definidos anteriormente. As ações dos atores estão relacionadas aos requisitos estabelecidos pelo sistema (SOMMERVILLE, 2011), (PRESSMAN, 2016).

1º Requisito Funcional
Cadastro de Usuários: No software desenvolvido é utilizada a `gem` 'devise' para cadastrar novos usuários.
2º Requisito Funcional
Autenticação: A `gem` 'devise' proporciona ao software a geração de uma tela de login onde o usuário informa suas habilitações obtendo acesso à plataforma de gerenciamento.
3º Requisito Funcional
Redefinição de Senha: A `gem` 'devise' prepara a página de cadastramento de maneira que possa enviar um link para o e-mail do usuário para redefinição de senha, caso seja solicitada.
4º Requisito Funcional
Cadastramento dos elementos do sistema: A `gem` 'rails_admin' permite a geração de modelos sem a necessidade da criação de *Controllers* ou rotas. Dessa maneira, gerou-se

os modelos para o cadastramento do **estacionamento**, **veículo**, **tipo de vaga** e **funcionário**.
5º Requisito Funcional
Edição dos cadastros: A `gem` 'Admin' prepara automaticamente a possibilidade de edição dos cadastros realizados. Nesse caso é possível atualizar, remover e exportar.

2.4.3 Requisitos Não-Funcionais

Define-se requisitos não-funcionais como aquele que não possuem relacionamento direto com as atividades do software. Contudo, são limitações dos serviços oferecidos pelo software (REZENDE, 2006).
1º Requisito Não-Funcional
Incompatibilidade com dispositivos móveis: O software não é compatível com dispositivos móveis.
2º Requisito Não-Funcional
Acessos Simultâneos: O banco de dados `postgresql` não possui limitação. No entanto, para proteção de acessos simultâneos, configurou-se um limite, restrito a uma faixa de IPs. Caso seja necessário uma maior capacidade, existe a possibilidade de modificação.

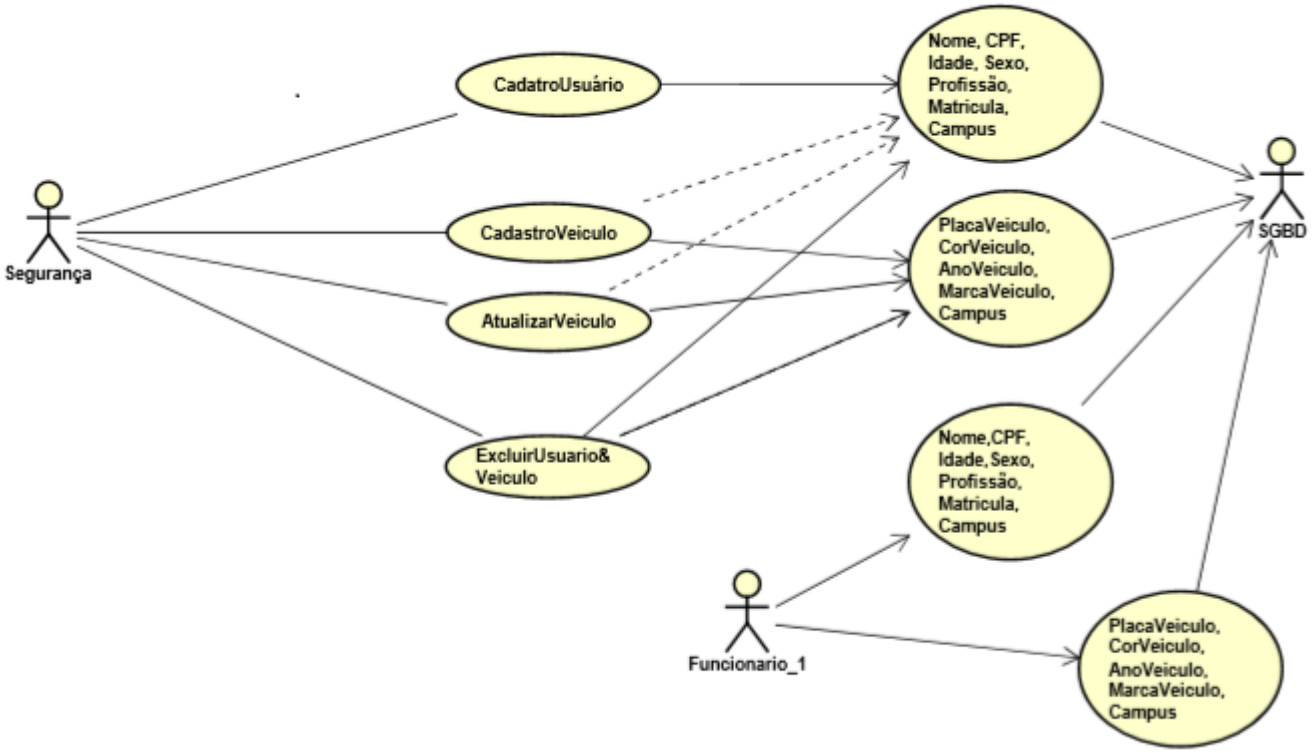
2.5 Diagrama de Casos de Uso

Levando-se em consideração as premissas anteriormente explicadas, pode-se modelar as funções do sistema proposto. Na Fig. 4 é apresentado o diagrama de casos de uso do sistema de gerenciamento de estacionamentos. Idealizou-se que o ator *segurança* é o responsável pelo cadastramento, atualização e exclusão do ator *Funcionário_1*, que por sua vez, fornece seus dados pessoais e de seu veículo.

2.6 Relacionamentos no RoR

Comumente, sistemas que empregam orientação a objetos utilizam classes que retratam elementos reais do domínio de negócio. O sistema de gerenciamento de estacionamentos desenvolvido no presente trabalho possui as classes cujos modelos foram mencionados anteriormente (seções 1.6 e 2.4.2. Essas classes de entidades em geral possuem relacionamentos que devem ser formalizados em código, isso quer dizer que essas classes precisam conter as informações de como devem atuar entre si. O framework Rails possui recursos para essa atividade. O tópico seguinte mostra alguns dos tipos mais impostantes de relacionamentos que podem existir entre as classes do modelo de uma aplicação.

Figura 4 – Diagrama de Caso de Uso



2.6.1 Tipos de relacionamentos

Em (SILVA, 2015) se lê:

> Existem diversas formas de dois objetos se associarem entre si, os relacionamentos mais comuns entre classes são:

- Um para um. Onde um objeto pode estar associado a apenas outro objeto de um determinado tipo.
- Um para muitos. Neste caso um objeto poderá se associar com um ou vários objetos de outra classe.
- Muitos para muitos. Um objeto pode participar de vários relacionamentos com outros objetos de determinado tipo.

A aplicação desses relacionamentos na estrutura Rails segue as seguintes formas: `belongs_to` (pertence a), `has_one` (tem um), `has_many` (tem muitos), `has_and_belongs_to_many` (tem e pertence a muitos), entre outras. Cada método possui recursos que trazem efeitos ou modificações no banco de dados. No próximo tópico essas características serão discutidas.

2.6.2 Associação `belongs_to`

Essa associação é geralmente denominada de *um para um* e é utilizada para mostrar que um certo modelo *pertence a* outro.

2.6.3 Associação `has_one`

Essa ligação é do tipo *um para um* e possui alguma semelhança com `belongs_to`. No entanto, preferivelmente, deve ser empregada em situações diferentes causando um efeito diferenciado. Tal associação é bidirecional a `belongs_to`. A diferença mais significativa é que não adiciona chave estrangeira ao modelo que a declara.

2.6.4 Associação `has_many`

Usa-se essa ligação entre modelos para indicar que o modelo em questão tem nenhum ou muitos elementos de outro modelo da aplicação. É denominada, frequentemente, de *um para muitos*. Semelhante a `has_one` ela não adiciona nenhuma coluna no modelo que a utiliza. Seu emprego deve ser em conjunto com `belong_to` para indicar por meio de chave estrangeira com quem o objeto filho se relaciona.

2.6.5 Associação `has_and_belongs_to_many`

Nessa opção é possível gerar uma ligação do tipo *muitos para muitos* entre dois modelos da aplicação.

2.7 Diagrama de Classes e seus Relacionamento

Num diagrama de classes UML, as associações possibilitam a descrição de informações nos extremos da linha que representa a ligação entre as classes. A Fig. 5 mostra os relacionamentos e os limites superiores (ou máximos) que representam a multiplicidade entre as classes.

2.8 Testes do Sistema

Nessa seção serão apresentadas as telas da plataforma de gerenciamento de estacionamentos seguidas de descrições das funcionalidades.

Na tela mostrada na Fig. 6 o usuário insere as credenciais para ser admitido no sistema. A Fig. 7 mostra a tela de confirmação de cadastramento do usuário. Ele deve cadastrar o estacionamento, indicando o campus, o funcionário, que possui direito a um tipo de vaga no estacionamento e seu veículo. As telas com esses direcionamentos são exibidas na sequência.

Atualmente trabalha-se na definição dos relacionamentos entre os modelos. Foi possível, até o momento, realizar o cadastramento de estacionamentos, conforme mostra a Fig. 8.

Figura 5 – Diagrama de Classes

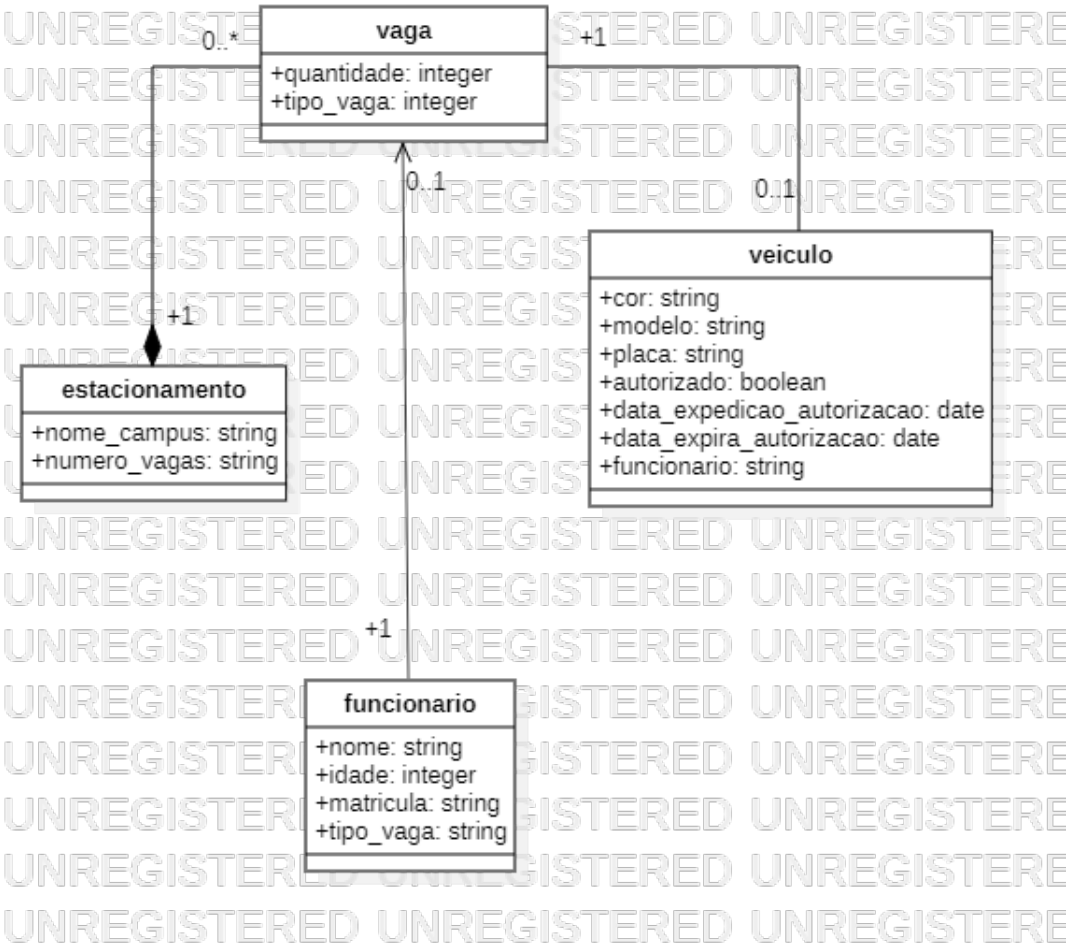


Figura 6 – Tela de Cadastramento do Usuário.

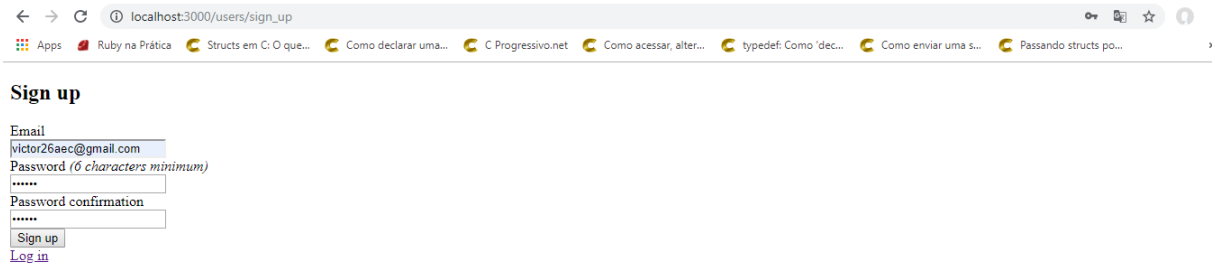


Figura 7 – Tela de Confirmação de Cadastramento do Usuário.

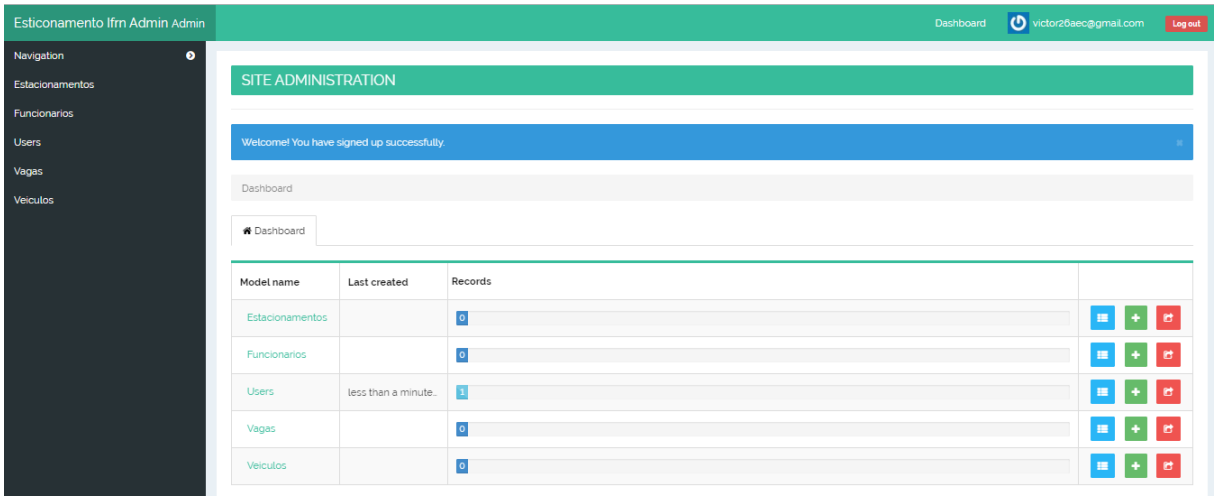
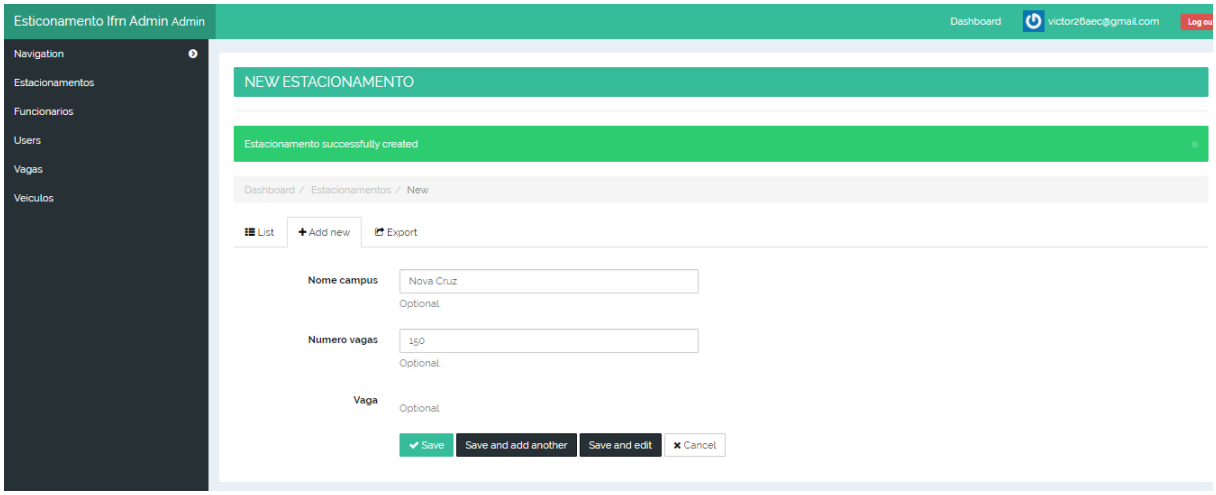


Figura 8 – Tela de Confirmação de Cadastramento de Estacionamento.



Referências

AN, J.-h.; CHAUDHURI, A.; FOSTER, J. S. Static typing for ruby on rails. In: IEEE. *2009 IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2009. p. 590–594. Nenhuma citação no texto.

ASTELS, D.; MILLER, G.; NOVAK, M. Extreme programming: guia prático. *Rio de Janeiro: Campus*, 2002. Citado na página 12.

BÄCHLE, M.; KIRCHBERG, P. Ruby on rails. *IEEE software*, IEEE, v. 24, n. 6, p. 105–108, 2007. Citado na página 12.

DEVMEDIA. 2008. Online; accessed 21 nov. 2019. Disponível em: <https://www.devmedia.com.br/qualidade-de-software/9408>. Citado na página 8.

HACKER, S. 2008. Online; accessed 04 November 2019. Disponível em: <http://birdhouse.org/blog/2008/11/19/notes-on-a-django-migration>. Citado na página 11.

HARTL, M.; FERNANDEZ, O. O. *The Ruby on Rails 3 Tutorial and Reference Collection (Collection)*. [S.l.]: Addison-Wesley, 2011. Citado na página 4.

LEI, K.; MA, Y.; TAN, Z. Performance comparison and evaluation of web development technologies in php, python, and node. js. In: IEEE. *2014 IEEE 17th international conference on computational science and engineering*. [S.l.], 2014. p. 661–668. Citado na página 12.

MCKEEVER, S. Understanding web content management systems: evolution, lifecycle and market. *Industrial management & data systems*, MCB UP Ltd, v. 103, n. 9, p. 686–692, 2003. Citado na página 12.

PLEKHANOVA, J. Evaluating web development frameworks: Django, ruby on rails and cakephp. *Institute for Business and Information Technology*, 2009. Citado na página 11.

PRESSMAN, R. S. Engenharia de software, uma abordagem profissional–8ª ed–amgh editora ltda. *Porto Alegre–RS-2016*, 2016. Citado na página 14.

REZENDE, D. A. *Engenharia de software e sistemas de informação*. [S.l.]: Brasport, 2006. Citado na página 15.

RUBY, S.; THOMAS, D. et al. *Agile web development with rails*. [S.l.]: Raleigh, NC: Pragmatic Bookshelf, 2009. Citado na página 12.

SCORZA, L. 2018. Online; accessed 17 nov. 2019. Disponível em: <https://onebitcode.com/rails-admin-tutorial-1/>. Citado na página 5.

SILVA, J. D. T. 2015. Online; accessed 20 nov. 2019. Disponível em: <https://www.devmedia.com.br/introducao-a-relacionamentos-no-ruby-on-rails/33860>. Citado na página 16.

SOMMERVILLE, I. *Engenharia de Software*. 3. ed. [S.l.]: Pearson Education, 2011. Acesso em: 18 nov 2019. Citado na página 14.

STONEBRAKER, M. et al. The design of xprs. In: *VLDB*. [S.l.: s.n.], 1988. p. 318–330. Citado na página 14.

STONEBRAKER, M.; ROWE, L. A. *The design of Postgres*. [S.l.]: ACM, 1986. v. 15. Citado na página 14.

TAPIADOR, A.; SALVACHÚA, J. Content management in ruby on rails. *arXiv preprint arXiv:1209.3878*, 2012. Citado na página 12.

TAYLOR, B. *Rails: a guide to rails, crakes, gallinules and coots of the world*. [S.l.]: Bloomsbury Publishing, 2010. Citado na página 12.