**INSTITUTO FEDERAL DO RIO GRANDE DO NORTE**

**CAMPUS AVANÇADO NATAL – EAD**

**CURSO TÉCNICO DE INFORMÁTICA PARA INTERNET**

**JOSÉ VICTOR GOMES DE FRANÇA**

**ROLIMBERG FURTADO DA SILVA**

**DESENVOLVIMETNO DE UM SISTEMA WEB PARA CONTROLE DOS VEÍCULOS PARA ACESSO AO ESTACIONAMENTO DO IFRN UTILIZANDO RUBY ON RAILS**

**TRABALHO DE CONCLUSÃO DE CURSO**

**NATAL**

**2019**

**JOSÉ VICTOR GOMES DE FRANÇA**

**ROLIMBERG FURTADO DA SILVA**

**DESENVOLVIMETNO DE UM SISTEMA WEB PARA CONTROLE DOS VEÍCULOS PARA ACESSO AO ESTACIONAMENTO DO IFRN UTILIZANDO RUBY ON RAILS**

Trabalho de Conclusão de Curso técnico, apresentado à disciplina de Seminário de Orientação para a Prática Profissional/ Projeto Integrador II, do Curso de

Informática para Internet, do Instituto Federal do Rio Grande do Norte, Campus Avançado natal – EAD, como requisito para obtenção do título de Técnico em Informática para Internet .

Orientador: Prof. Rafael Fernandes de Queiroz

**NATAL**

**2019**

**RESUMO**

GOMES DE FRNAÇA, José Victor; DA SILVA, Rolimberg Furtado. Desenvolvimento de um sistema web para controle dos veículos para acesso ao estacionamento do IFRN utilizando Ruby on Rails. 2019. Monografia de Trabalho de Conclusão de Curso - Curso Técnico de Informática para Internet, Instituto Federal do Rio Grande do Norte, Campus Avançado Natal – EAD. Natal, 2019.

Tendo em vista que a quantidade de vagas no estacionamento do IFRN é limitado e a quantidade de veículos ocupantes está aumentando gradativamente, é necessário controlar os veículos que realmente devem ter acesso ao estacionamento do IFRN, tais como aqueles pertencentes a servidores do instituto. Além da quantidade, o processo para cadastro de um veículo é bastante burocrático, exigindo a comunicação entre várias pessoas (Diretor, Departamento de Segurança, etc.) para autorizar o cadastro do veículo, dificultando a aquisição do adesivo que identifica um veículo autorizado. O sistema deve ser desenvolvido utilizando o framework Ruby On Rails e banco de dados PostgreSQL, utilizando conhecimentos de diversas disciplinas do curso (APOO, PDSI, Banco de Dados, e outras).

**Palavras-chave**: Ruby on Rails. Sistema *web*. Sistema para estacionamento.

**ABSTRACT**

Gomes of Franca, José Victor; DA SILVA, Rolimberg Furtado. Development of a web system for vehicle control to access IFRN parking using Ruby on Rails. 2019. Course Conclusion Work Monograph - Technical Computer Course for Internet, Federal Institute of Rio Grande do Norte, Advanced Campus Natal - EAD. Christmas, 2019.

Given that the number of IFRN parking spaces is limited and the number of occupant vehicles is gradually increasing, it is necessary to control the vehicles that really should have access to the IFRN parking, such as those belonging to IFRN servants. In addition to the amount, the process for registering a vehicle is quite bureaucratic, requiring communication between several people (Director, Security Department, etc.) to authorize the registration of the vehicle, making it difficult to purchase the sticker that identifies an authorized vehicle. The system should be developed using the Ruby On Rails framework and PostgreSQL database, using knowledge of several course subjects (APOO, PDSI, Database, and others)..

**Keywords:** Ruby on Rails. Web system Parking system.

**LISTA DE FIGURAS**

Figura 1 – Componentes do *framework* Ruby on Rails......................................................... 15

Figura 2 – Diagrama de domínio......................................................................................... 24

Figura 3 – Diagrama de entidades e relacionamentos gerado pela *gem* *erd*........................... 29

Figura 4 – Página de *login*.................................................................................................... 30

Figura 5 – Formulário de cadastro de funcionário................................................................. 31

**LISTA DE QUADROS**

Quadro 1 – Tecnologias e ferramentas utilizadas para a modelagem e a implementação....... 20

Quadro 2 – Requisitos funcionais......................................................................................... 25 Quadro 3 – Requisitos não funcionais.................................................................................. 25

Quadro 4 – Diagrama de casos de uso.................................................................................. 25

Quadro 5 – Caso de uso cadastrar funcionário...................................................................... 26 Quadro 6 – Caso de uso registrar agendamento.................................................................... 27

**LISTAGEM DE CÓDIGO**

Listagem 1 – Criação de um novo projeto Rails ................................................................... 36

Listagem 2 – GemFile.......................................................................................................... 37

Listagem 3 – Configurações da base de dados...................................................................... 38

Listagem 4 – Criando a base de dados.................................................................................. 38 Listagem 5 – Construindo a base de dados........................................................................... 39

Listagem 6 – Arquivos criados pelo gerador Scaffold........................................................... 39

Listagem 7 – Realizando migrações..................................................................................... 40

Listagem 8 – Adição da *gem* Bootstrap ao projeto............................................................... 41

Listagem 9 – Leiaute *Template* básico da aplicação.............................................................. 42

Listagem 10 – Geração de arquivos para autenticação.......................................................... 43

Listagem 11 – *Controller* para registro de usuários utilizando a *gem devise* ......................... 43

Listagem 12 – *Models Employee, Activity* e *Comission* ........................................................ 45

Listagem 13 – Aceitar parâmetros de objetos agregados....................................................... 45

Listagem 14 – *Partials* para formulário mestre-detalhe pela *gem* *cocoon* ............................. 47

Listagem 15 – *Model* Scheduling com validações e consultas ao banco de dados................. 48

Listagem 16 – Método para filtragem de agendamentos pelo dia corrente ............................ 49

Listagem 17 – *Partial \_form* para formulário mestre-detalhe de Documento........................ 54

Listagem 18 – *Model* paradocumentos ................................................................................ 56

Listagem 19 – *Model* e método criação do *controller* de *Acquittance*................................... 57

**LISTA DE ABREVIATURAS E SIGLAS**

|  |  |
| --- | --- |
|  |  |
| CRUD | *Create, Read, Update and Delete* |
| CSS | *Cascading Style Sheets* |
| DRY | *Don’t Repeat Yourself* |
| HTML | *HyperText Markup Language* |
| HTTP | *Hypertext Transfer Protocol* |
| JSON | *JavaScript Object Notation* |
| MVC | *Model-View-Controller* |
| ORM | *Object-Relational Mapping* |
| RF | Requisito Funcional |
| RHTML | *Rails and HTML* |
| RNF | Requisito Não Funcional |
| ROR | *Ruby On Rails* |
| RXML | *Rails and XML* |
| SGBDOR | Gerenciador de Banco de Dados Objeto Relacional |
| SMS | *Short Message Service* |
| SQL | *Structured Query Language* |
| URL | *Uniform Resource Locators* |
| WWW | *World Wide Web* |
| YAML | *Ain't Markup Language* |
|  |  |
|  |  |
|  |  |

**SUMÁRIO**

[1 INTRODUÇÃO 10](#_Toc23447782)

[1.1 CONSIDERAÇÕES INICIAIS 10](#_Toc23447783)

[1.2 OBJETIVOS 11](#_Toc23447784)

[1.2.1 Objetivo Geral 11](#_Toc23447785)

[1.2.2 Objetivos Específicos 11](#_Toc23447786)

[1.3 JUSTIFICATIVA 12](#_Toc23447787)

[1.4 ESTRUTURA DO TRABALHO 12](#_Toc23447788)

[2 REFERENCIAL TEÓRICO 14](#_Toc23447789)

[2.1 CONTEXTO 14](#_Toc23447790)

[2.2 RUBY 14](#_Toc23447791)

[2.3 CONVENÇÕES SOBRE CONFIGURAÇÃO 17](#_Toc23447792)

[2.4 DON’T REPEAT YOURSELF 18](#_Toc23447793)

[2.5 MIGRATIONS 18](#_Toc23447794)

[2.6 GEMS 19](#_Toc23447795)

[3 MATERIAIS E MÉTODO 20](#_Toc23447796)

[3.1 MATERIAIS 20](#_Toc23447797)

[3.2 MÉTODO 21](#_Toc23447798)

[4 RESULTADO 21](#_Toc23447799)

[4.1 ESCOPO DO SISTEMA 22](#_Toc23447800)

[4.2 MODELAGEM DO SISTEMA 22](#_Toc23447801)

[4.2.1 Especificação de Requisitos 23](#_Toc23447802)

[4.3 APRESENTAÇÃO DO SISTEMA 30](#_Toc23447803)

[4.4 IMPLEMENTAÇÃO DO SISTEMA 31](#_Toc23447804)

[4.4.1 Configuração da Base de Dados 33](#_Toc23447805)

[4.4.2 Scaffolding 34](#_Toc23447806)

[4.4.3 Definição do Leiaute Básico 36](#_Toc23447807)

[4.4.4 Autenticação de Usuários 38](#_Toc23447808)

[5 CONCLUSÃO 39](#_Toc23447809)

[REFERÊNCIAS 41](#_Toc23447810)

# 1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa do trabalho. No final do capítulo está a apresentação da organização do texto por meio de uma descrição sumarizada dos capítulos subsequentes.

## 1.1 CONSIDERAÇÕES INICIAIS

A Internet, criada durante a guerra fria com o objetivo de possibilitar troca de informações mais rapidamente entre as bases militares, foi expandida e popularizada inicialmente entre as universidades e, posteriormente, no mundo, originando a chamada

*World Wide Web* (WWW) (CAMARIM, 2012). A *web* é caracterizada como uma rede mundial de documentos em hipermídia disponibilizada na Internet. Nos anos 90, a popularização da Internet criou a necessidade de novas linguagens de programação para o desenvolvimento de aplicativos para *web*. Essa necessidade está vinculada aos novos requisitos de interface demandados pelos usuários. As aplicações *web* tradicionais baseadas em *links* entre páginas e formulário simples contrastavam com as aplicações *desktop* repletas de recursos como menus, botões e janelas diferenciados e efeito de arrastar e soltar. Dentre as linguagens que surgiram para o desenvolvimento de aplicações *web* estão PHP, Java, JavaScript, Ruby on Rails, que visam facilitar a implementação de aplicativos para esse ambiente com melhores e diversificados recursos de interação e multimídia.

As tecnologias para desenvolvimento *web* caracterizam-se por oferecer ferramentas para desenvolvimento de interfaces de comunicação entre clientes e servidores utilizando navegadores como aplicações do cliente. O cliente pode ser definido por uma aplicação que realiza requisições ao(s) servidor(es) e/ou alimenta-o(s) com dados. O servidor tem como função gerenciar recursos (bancos de dados, serviços, equipamentos de hardware e outros servidores) e atender requisições dos clientes.

Ruby on Rails, é um *framework* baseado na linguagem de programação Ruby, que foi concebido com o objetivo de facilitar e simplificar o desenvolvimento de aplicativos *web* (SOUZA, 2014). Esse *framework* utiliza recursos de geração de estrutura de código e o conceito DRY (*Don’t Repeat Yourself*), além de outros recursos de configurações prédefinidas que possibilitam o desenvolvimento de aplicações mais rapidamente por meio da geração de código base a partir de diversas ferramentas como *Scaffold* (RUBY, 2014). Além disso, esse *framework* implementa o conceito de *convention over configuration* (convenção sobre configuração) que utiliza convenções de desenvolvimento para gerenciar recursos e facilitar o desenvolvimento sem que seja necessário o desenvolvedor explicitar seu funcionamento.

Um dos grandes desafios para as empresas de software, especialmente as de pequeno porte, no desenvolvimento de software é viabilizar um sistema em tempo hábil, principalmente com um número reduzido de desenvolvedores. A viabilização desse desenvolvimento, mesmo nessas condições, pode ocorrer com a reutilização de código. Esse é um dos objetivos do Ruby on Rails.

Estacionamentos possuem uma grande movimentação de carros, sendo necessário um software para agendamentos e acompanhamento de veículos. Um sistema para *web* para esse segmento facilita o acesso pelo usuário e também pelos proprietários e administradores.

O sistema desenvolvido como resultado deste trabalho é um sistema *web* para estacionamento. O desenvolvimento foi realizado utilizando a tecnologia Ruby on Rails.

## 1.2 OBJETIVOS

A seguir são apresentados os objetivos deste trabalho.

### 1.2.1 Objetivo Geral

Implementar um sistema de gestão de agendamento de parqueamento.

### 1.2.2 Objetivos Específicos

* Apresentar o uso de recursos do *framework* Ruby on Rails na implementação de um sistema gerencial voltado para a prestação de vaga estacional.
* Desenvolver a base de dados do sistema utilizando o PostgreSQL como Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR).
* Apresentar o uso de recursos da tecnologia Ruby on Rails que visam agilizar e prover reuso na implementação de sistema *web*.

## 1.3 JUSTIFICATIVA

O desenvolvimento de sistema *web* vem ganhando força em diversos segmentos de negócio e proporcionando facilidades de acesso decorrente da Internet. Esse é um dos motivos que justificam o amplo uso de tecnologias para desenvolvimento para *web*. Justifica-se a escolha de Ruby on Rails, como forma de utilizar uma linguagem ou tecnologia de desenvolvimento para *web* que tem como um dos seus objetivos agilizar o desenvolvimento.

A linguagem de programação Ruby foi criada para simplificar o desenvolvimento. Ela possui uma sintaxe simples e é inspirada em Eiffel e Ada, apresenta recursos de tratamento de exceção como em Java e Python e é completamente orientada a objetos como SmallTalk (RUBY, 2014). Ruby tem herança única, porém trabalha com o conceito de módulos, que são uma coleção de métodos, sendo possível importá-los. O objetivo desses módulos é diminuir a complexidade que a herança múltipla tem na orientação a objetos (SOUZA, 2013).

O Ruby possui um *meta*-*framework (framework de frameworks)* denominado Ruby on Rails para desenvolvimento *web* rápido e estruturado que traz recursos pré-configurados e suporte nativo aos principais bancos de dados do mercado e ao *framework* JavaScript JQuery. Os *frameworks* que compõemo Rails são, entre outros, *ActionMailer* (envio de e-mails),

*ActionPack* (geração de estrutura *Model-View-Controller* (MVC)), *ActionRecord* (manipulação de banco de dados), *ActiveSuport* (utilidades gerais como tratamentos de erros e extensões de linguagem) (RUBY, 2014). Tais *frameworks* facilitam o desenvolvimento do projeto, principalmente o *ActionRecord*, por meio do qual é possível gerar um *Create, Retrieve, Update and Delete* (CRUD) que é responsável pela maioria de operações com o banco de dados.

## 1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico que está centrado em Ruby e Ruby on Rails. No Capítulo 3 são apresentados os materiais e o método utilizados para o desenvolvimento do trabalho. No Capítulo 4 está o resultado da realização do trabalho que é o sistema desenvolvido. O capítulo inicia com uma apresentação descritiva do sistema, em seguida a sua modelagem e depois as funcionalidades principais do sistema são apresentadas por meio das suas telas. Essa apresentação é complementada pela descrição de funcionalidades desenvolvidas, visando apresentar recursos da linguagem e sua forma de implementação. No Capítulo 5 está a conclusão, seguida das referências bibliográficas.

# 2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho e está centrado em Ruby on Rails por ser a tecnologia utilizada para o desenvolvimento da aplicação.

## 2.1 CONTEXTO

Geer (2006) destaca que muitos programadores querem o melhor da rapidez e da produtividade criando aplicações confiáveis e limpas e desenvolvendo menor quantidade de código. De acordo com esse autor, a linguagem PHP oferece rapidez de desenvolvimento, embora não produza códigos tão limpos e a linguagem Java é de desenvolvimento lento, mas produz código limpo e sólido. Para Geer (2006) Ruby on Rails oferece o melhor dessas duas tecnologias: rapidez e código consistente e limpo.

## 2.2 RUBY

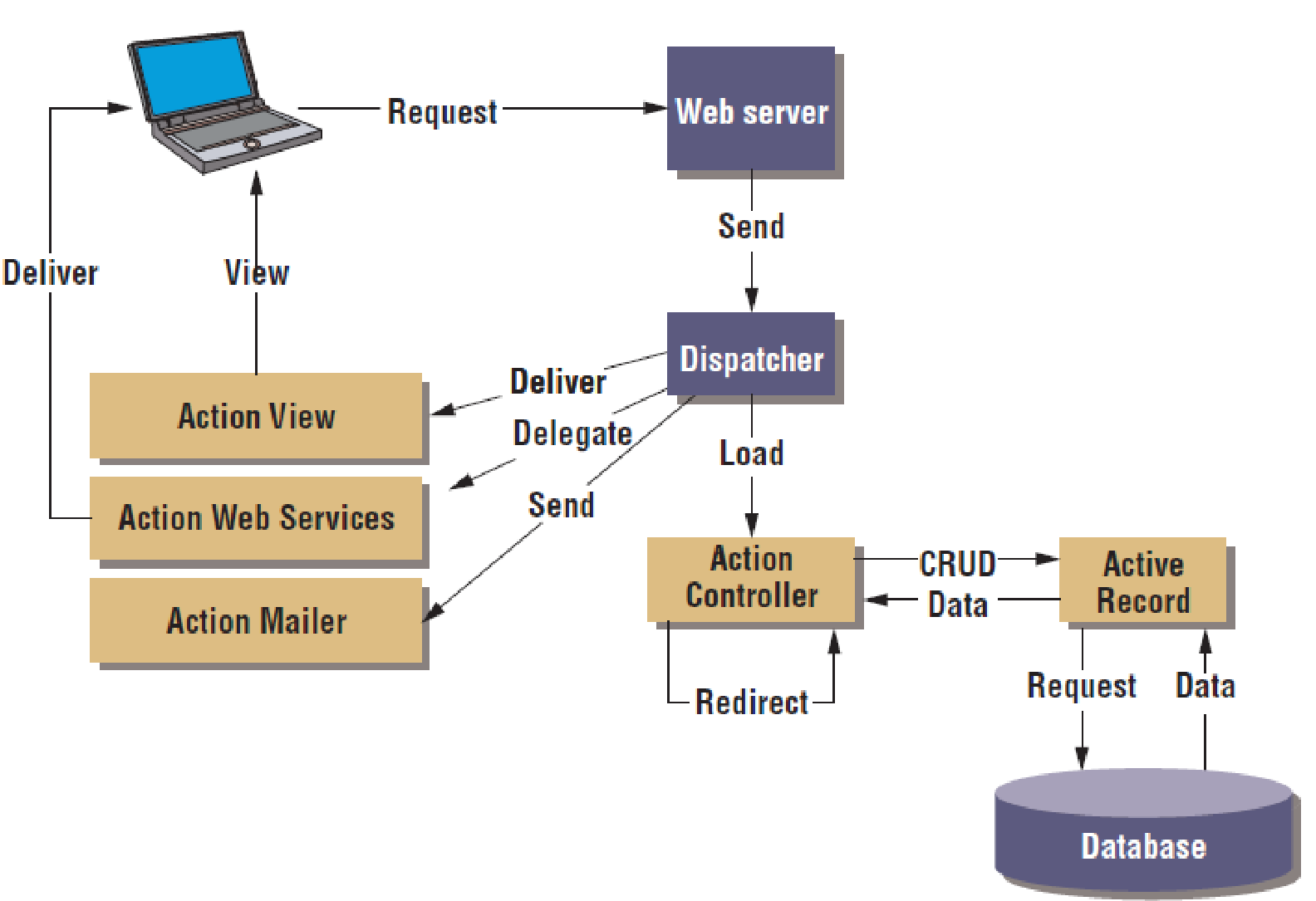
Ruby é uma linguagem dinâmica. Por meio dela ao invés de escrever uma grande quantidade de código, os desenvolvedores podem declarar comandos eficientemente por meio de inclusão de pequenas quantidades de código (GEER, 2006). Ruby pode ser usada para escrever *templates* e convenções que tornam o desenvolvimento mais rápido (GEER, 2006).

Rails, tornado público em 2004, é livre e disponível sob licença MIT[[1]](#footnote-1). Rails se tornou uma linguagem de programação de propósito geral da Ruby para uma solução específica para criar aplicações *web*, gerando assim o *framework* Ruby on Rails (GEER, 2006). De acordo com esse autor, Rails provê as ferramentas necessárias para produzir um modelo de aplicação *web* que é o próprio programa.

2.2 RUBY ON RAILS

O *framework* de desenvolvimento de aplicações Rails é baseado em Ruby que é uma linguagem orientada a objetos baseada em *script* (AN; CHAUDHURI; FOSTER, 2008). Rails é de código fonte aberto e tem como base a linguagem de *scripts* similar a Pearl e Smalltalk (GEER, 2006). Rails utiliza pacotes integrados de programação e código predefinido, conhecido como convenções, projetado para ser completo e pronto para usar, sem configurações (GEER, 2006). Ruby on Rails é um *framework* de desenvolvimento *web* baseado na linguagem orientada a objetos dinâmica, o Ruby (VISWANATHAN, 2008).

Rails é baseado no padrão de projetos MVC para o desenvolvimento das aplicações e esse padrão é usado para vincular ações do usuário, objetos da aplicação e apresentar informações (VISWANATHAN, 2008; GEER, 2006). MVC separa claramente o código de acordo com seus objetivos e três *subframeworks* em Rails desempenham papéis relevantes nesta separação (BÄCHLE; KIRCHBERG, 2007): *Active Record*, *Action View* e *Action Controller.* A Figura 1 apresenta os principais componentes do *framework* Ruby on Rails. Os *subframeworks* *Active Record*, *Action View* e *Action Controller* estabelecem as fundamentações para a arquitetura MVC.



**Figura 1 – Componentes do *framework* Ruby on Rails**

**Fonte: Bächle e Kirchberg (2007, p. 106).**

Vinculando as funcionalidades do *framework* Rails com o padrão MVC, qualquer requisição *web* realizada pelo cliente resulta em uma chamada a algum método em um controle, que por sua vez usa um modelo para realizar os acessos ao banco de dados e retornar para a visão, isto é, o texto da página *web* como resposta. A seguir está a explicação das partes do MVC de acordo com o seu uso em Rails (VISWANATHAN, 2008; GEER, 2006, BÄCHLE; KIRCHBERG, 2007):

#### a) Modelo

O pacote *Active Record* é um mapeador objeto-relacional do Rails que conecta objetos da programação em tabelas de banco de dados permitindo, assim, que o usuário acesse informações do banco de dados.

O *subframework* *Active Record* define as conexões entre os objetos do domínio e o banco de dados. O *Active Record* transforma as funções de CRUD que vem do *Action*

*Controller* em instruções *Structured Query Language* (SQL), envia as requisições para o banco de dados e retorna os resultados recebidos do *Action Controller*. *Active Record* também valida se um usuário possui permissões para acessar ou modificar determinado registro.

#### b) Visão

O componente *Action View* do Rails apresenta os dados por meio de visões, que representam a aparência visual da aplicação em resposta a requisições. Ruby on Rails, tipicamente, representa os dados no formato *HyperText Markup Language* (HTML). Ao processar, realizar, uma ação, a biblioteca *Active View* renderiza *templates* para retornar a resposta para a camada de apresentação. Dentre os *templates* de Rails destacam-se *Rails and*

*HTML* (RHTML) e *Rails and XML* (RXML) (BÄCHLE; KIRCHBERG, 2007). O *template* RHTML é uma junção de código Ruby e HTML. Esse *template* coloca o código Ruby em tags <%...%>. Para criar arquivos RXML, Rails usa bibliotecas próprias.

#### c) Controle

O controle é o mediador entre modelos e visões. Responde às ações do usuário pela interação com os objetos do modelo para obter os dados solicitados e repassa os resultados para a visão apresentá-los.

No Rails, os controles são implementados por meio da biblioteca *Action Controller*. O *Action Controller* recebe as requisições de informações vindas de uma interface *web*, analisa e as decodifica para determinar o que o usuário quer (ou o que é solicitado pela requisição) e então decide qual parte do código da aplicação deve manipular a tarefa.

O *Action Controller* é a unidade central de governo, ou controle, da aplicação. Esse *action* recebe as requisições e devolve os resultados da execução da requisição por meio de uma visão para o cliente. Dentro desses componentes, o desenvolvedor define ações (*actions*) que determinam como o controle reagirá para uma requisição *Hypertext Transfer Protocol* (HTTP) específica. O controle recebe uma requisição de uma *Uniform Resource Locator* (URL), processa a requisição como uma ação ou objeto e envia a visão correta de volta para o navegador.

O Rails permite aos controles selecionar, condicionalmente, visões diferentes de acordo com a necessidade dos dados a serem apresentados.

O Ruby on Rails tem provido suporte para uma variedade de aspectos de desenvolvimento, tais como gerenciamento da apresentação e persistência, permitindo aos desenvolvedores concentrar-se na lógica de negócio da sua aplicação (STELLA; JARZABEK; WADHWA, 2008).

## 2.3 CONVENÇÕES SOBRE CONFIGURAÇÃO

Devido sua essência estar focada em desenvolvimento rápido, o Ruby on Rails (ROR) ou simplesmente Rails, segue a filosofia de *less software* que é definido como (GAMBLE; CARNEIRO; BARAZI, 2013): utilizando convenção sobre configuração, escreve-se menos código, o que põe fim a coisas que desnecessariamente adicionam complexidade a um sistema. Resumidamente, *less software* significa menos código, complexidade e *bugs*.

Esse modelo consiste em utilizar padrões visando diminuir a quantidade de decisões que o desenvolvedor precisa tomar, especificando somente aspectos que realmente são necessários. Por exemplo, a nomenclatura de entidades no banco de dados é o plural (no idioma inglês) grafado em letras minúsculas do nome definido para o *model*, assim, ao ser definido “User”, automaticamente será assumido pelo *framework* que a tabela no banco de dados tem o nome “users” (GAMBLE; CARNEIRO; BARAZI, 2013, p. 5). Entretanto, nenhuma das convenções deve impedir que uma configuração seja definida pelo usuário, assim, a maioria das convenções do *framework* podem ser sobrepostas por configurações, garantindo que a ferramenta tenha a versatilidade necessária.

## 2.4 DON’T REPEAT YOURSELF

O princípio de *Don’t Repeat Yoursef*, em português, não se repita, é um dos princípios que norteiam o Rails, definindo que determinada informação deve ser expressada em apenas um lugar no sistema, ou seja, cada recurso ou ação, deve estar definido em apenas um arquivo, facilitando a manutenção e o entendimento do código. Por exemplo, se é necessário obter informações do usuário no banco de dados, esse recurso deve estar acessível para ser requisitado por qualquer funcionalidade da aplicação que tenha permissão para esse acesso, evitando que ele seja definido novamente, duplicando o código e, consequentemente, dificultando a manutenção (RAILS GUIDE, 2015a).

O Capítulo 3 apresenta os materiais e o método utilizados para implementar o aplicativo utilizando o *framework* Ruby on Rails vinculado a outras tecnologias e ferramentas como de banco de dados e para a interface da aplicação.

## 2.5 MIGRATIONS

*Migrations*, ou migrações definem um modo estruturado de modificar a base de dados da aplicação por meio de arquivos que definem tal alteração. As *migrations* no Ruby On Rails são gerenciadas pela biblioteca ActiveRecord. Sempre que uma modificação se faz necessária, uma nova *migration* é criada. Elas são identificadas por meio da data de criação e de um nome e são anexadas ao projeto em uma ordem específica para garantir a integridade da base de dados de acordo com as modificações efetuadas pelos desenvolvedores. Sempre que uma *migration* é executada, o Rails atualiza o arquivo db/schema.rb(mantido pelo *Active Record* que é um arquivo de configuração contendo a estrutura da base de dados), registrando a última *migration* como versão corrente da aplicação, assim, é possível manter o versionamento da base de dados em paralelo ao do sistema (RAILS GUIDE, 2015b).

A utilização das *migrations* elimina a necessidade de executar comandos SQL diretamente no banco de dados, possibilitando, inclusive, o *rollback* das modificações, uma vez que nenhuma *migration* deve ser excluída do projeto, fornecendo um histórico completo de modificações.

## 2.6 GEMS

O termo *Gem*, é utilizado pela linguagem *Ruby* para designar bibliotecas ou programas em Ruby que podem ser adicionados e utilizados por projetos. O *framework* Ruby on Rails é uma *gem* (*gem* “*rails*”). O arquivo Gemfile contém as *gems* utilizadas por um projeto Rails, definido versões, utilização de *gems* exclusivamente em determinado ambiente de desenvolvimento (desenvolvimento, teste, produção, etc...) (RAILS GUIDE, 2015c).

Para instalar uma *gem*, utiliza-se o gerenciador de dependências *bundle*. Com ele é possível, executando apenas um comando, instalar todas as *gems* definidas no Gemfile, gerenciando inclusive, as dependências de determina *gem*, não sendo necessário explicitá-las no arquivo. A linguagem Ruby possui uma grande quantidade de *gems* disponíveis para os mais diversos fins, bastando adicioná-las ao projeto. É comum que as *gems* possuam documentação específica, facilitando o seu uso.

# 3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para a realização da modelagem e o desenvolvimento do sistema objeto deste trabalho. Os materiais estão relacionados às tecnologias e às ferramentas utilizadas e o método apresenta a sequência das principais atividades desenvolvidas.

## 3.1 MATERIAIS

Para a modelagem e a implementação do sistema serão utilizadas as ferramentas e as tecnologias apresentadas no Quadro 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | **Versão** | **Disponiblização/referência** | **Aplicação no projeto** |
| Astah Professional | 12.0 | http://astah.net/download | Para documentação da modelagem do sistema. |
| Ruby | 2.2.3 | https://www.ruby-lang.org/pt/downloads/ | Linguagem de programação. |
| Ruby on Rails | 4.2.4 | http://rubyonrails.org/download/ | Framework *web* de  desenvolvimento |
| HTML | 5 | http://www.w3.org | Para o desenvolvimento da interface *web.* |
| CSS | 3 | http://www.w3.org | Para o desenvolvimento do leiaute da interface *web.* |
| Jquery | 1.11.0 | http://jquery.com/download | Na implementação da interface *web.* |
| PostgreSQL | 9.4.5 | http://www.pgadmin.org/download/ | Sistema gerenciador de Banco de dados. |
| PgAdmin 3 | 1.20.0 | http://www.pgadmin.org/download/ | Interface para administração do PostgreSQL |
| Virtual box | 5.0.8 | https://www.virtualbox.org/wiki/Downloads | Virtualização do ambiente de desenvolvimento |
| Vagrant | 1.7.4 | https://www.vagrantup.com/downloads.html | Gerenciador do ambiente de desenvolvimento virtualizado |
| Git | 2.6.2 | https://git-scm.com/downloads | Sistema de controle de Versão |
| Putty | 0.64 | http://www.putty.org/ | Emulação de terminal via SSH com o servidor |

**Quadro 1 – Tecnologias e ferramentas utilizadas para a modelagem e a implementação**

No desenvolvimento do sistema, o PostgreSQL foi utilizado como banco de dados por ser um SGBD objeto relacional e o PgAdmin como seu gerenciador. Para modelagem do sistema foi utilizado o Astah Professsional. A implementação realizada com Ruby on Rails juntamente com HTML, *Cascading Style Sheets* (CSS) e Jquery. Durante o desenvolvimento do projeto, para manter o código consistente entre ambos os desenvolvedores, o GIT foi utilizado como sistema de versionamento.

## 3.2 MÉTODO

O método é representado pelas atividades realizadas para o levantamento de requisitos, a modelagem desses requisitos e implementação de funcionalidades definidas para o aplicativo. A seguir está descrita de forma sucinta a realização das etapas de levantamento de requisitos e modelagem dos requisitos e sua implementação. Essas fases têm como base o modelo sequencial linear de Pressman (2006), embora não tenham sido realizadas de forma estritamente sequencial.

#### Levantamento de requisitos

O levantamento de requisitos foi realizado com base nas necessidades percebidas de um estacionamento. Visitas foram realizadas no estacionamento do IFRN. O objetivo das visitas foi verificar a rotina das atividades realizadas e os registros efetuados em relação a essas atividades e como é realizado o controle de carros. O cotidiano e a rotina do estacionamento foram avaliados, mas sem um processo formal de observação ou de coletas de dados. Essa observação permitiu identificar as funcionalidades pretendidas para o sistema e organizá-las sob a forma de requisitos.

#### Análise e projeto

Os requisitos identificados foram modelados como casos de uso e de diagrama de entidades e relacionamentos do banco de dados. Uma visão mais ampla e clara do sistema foi, assim, definida.

#### Desenvolvimento

O desenvolvimento ou implementação do sistema esteve baseado no uso do *framework* Ruby on Rails. Várias dificuldades foram encontradas no uso dessa tecnologia, decorrentes do desconhecimento do seu uso e da relativa falta de material disponível para a versão 4.

# 4 RESULTADO

Neste capítulo é apresentado o resultado da realização do trabalho. O resultado é composto pela definição do escopo do sistema, a modelagem (que inclui a definição dos requisitos e a sua representação), a apresentação das funcionalidades do sistema por meio de suas telas e a implementação do sistema com exemplos dos códigos gerados.

## 4.1 ESCOPO DO SISTEMA

A aplicação desenvolvida é uma ferramenta para gestão de estacionamento fornecendo as funcionalidades comuns a esse tipo de serviço, incluindo o processo de agendamento, gerenciamento de fluxo e controle de carro.

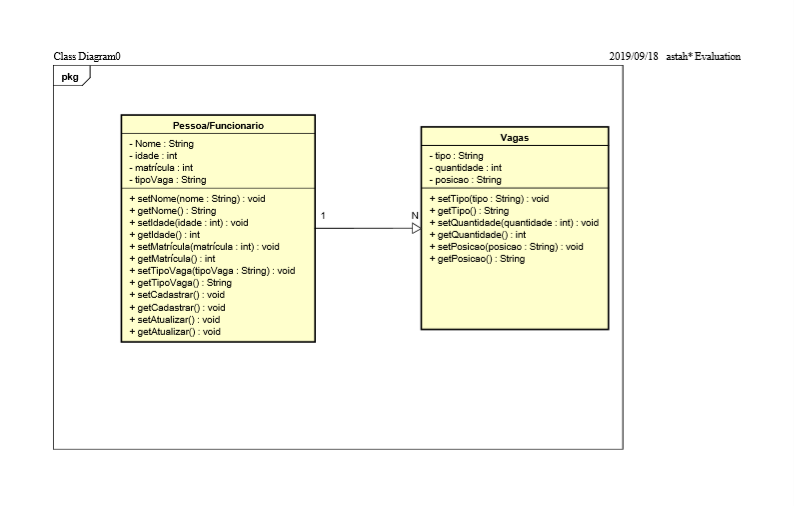
Para facilitar o desenvolvimento e a definição dos requisitos o sistema foi dividido conceitualmente em módulos. O primeiro módulo é o de agendamento de vaga, que consiste em gerenciar o cadastro de visitante, funcionários, atividades, agenda e notificações. O segundo módulo é o gerenciador de permanência.

Para permitir que tanto visitante quanto funcionários tenham a possibilidade de utilizar o sistema, mas sem necessariamente que tenham acesso a dados como os relacionados aos agendamentos e permanência, o ator será unificado no cadastro de Entidade e, assim, posteriormente definido o tipo de acesso ao sistema.

O agendamento de atividades (vagas) permitirá que o usuário selecione um para estacionar.

## 4.2 MODELAGEM DO SISTEMA

O diagrama conceitual de domínio apresentado na Figura 2 apresenta a visão geral do sistema, permitindo identificar, de maneira simplificada, como os dados serão gerenciados pelo sistema. Os conceitos e nomenclatura não representam relações no banco de dados ou classes, eles apenas representam conceitos relacionados entre si que tem o objetivo de definir conceitualmente o sistema.



**Figura 2 – Diagrama de domínio**

Pessoa/Funcionários e vagas são entidades. Pessoa/Funcionários podem registrar um agendamento de estacionamento em uma vaga. Cada Pessoa/Funcionário terá seus atributos registrados na hora do cadastro no sistema. Cada Vaga terá seus atributos também. Na hora do agendamento do serviço, o usurário poderá escolher o tipo de vaga desejada.

### 4.2.1 Especificação de Requisitos

Os requisitos identificados para a aplicação estão listados no Quadro 2. Nesse Quadro RF significa Requisito Funcional.

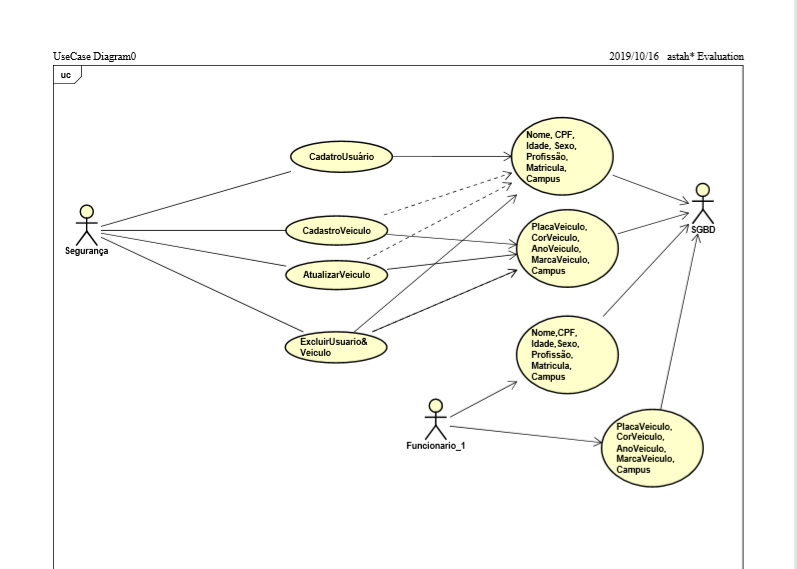
|  |  |  |
| --- | --- | --- |
| **Identificação** | **Nome** | **Descrição** |
| RF01 | Cadastrar Entidade | No cadastro de pessoas serão informados os dados pessoais de uma entidade, podendo ser ela física ou jurídica. |
| RF02 | Definir credenciais de acesso | Administradores concedem acesso a determinados recursos do sistema para uma entidade cadastrada. |
| RF03 | Cadastrar  Pessoa/Funcionário | Um funcionário será vinculado a um cadastro de entidade. |
| RF04 | Cadastrar  Agendamento | Uma atividade pode ser agendada para ser utilizada por uma pessoa/ funcionário para uma vaga. |
|  |  |  |
|  |  |  |
|  |  |  |

Os requisitos não funcionais identificados para o sistema estão listados no Quadro 3. Os requisitos não funcionais, ou suplementares, definem as regras de negócio e estão categorizados em requisitos de segurança, interface e integridade. No Quadro 3 RNF significa Requisito Não Funcional.

|  |  |  |  |
| --- | --- | --- | --- |
| **Identificação** | **Nome** | **Tipo** | **Descrição** |
| RNF01 | Acesso ao sistema | Segurança | Todos os recursos do sistema só poderão ser acessados por usuários através de e-mail e senha. |
| RNF02 | Criptografia | Segurança | A senha do usuário deve ser registrada no banco de dados em forma de "hash". |
| RNF03 | Redefinição de senha | Segurança | No caso do usuário esquecer sua senha, administradores poderão redefini-la. |
| RNF04 | Registro de retribuições | Segurança | Apenas usuários administradores poderão definir retribuições para cada funcionário. |
| RNF05 | Consumar agendamento | Interface | O sistema deve fornecer um meio de gerar um documento a partir dos dados de um agendamento, trazendo a identificação da entidade . |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Quadro 3 – Requisitos não funcionais**

Os casos de uso do sistema são apresentados no Quadro 4 e descreve as principais funcionalidades que o sistema oferece para a gestão do estacionamento pelos atores. Todos os atores são tipos de usuários, sendo eles: segurança e funcionário. Segurança registram agendamentos, documentos e recebimentos. Funcionários têm acesso a todos os recursos que o sistema oferece.



**Quadro 3 – Diagrama de casos de uso**

Os casos de uso “Cadastrar veículos” e “Cadastrar usuário” requerem que uma entidade seja referenciada, portanto estes casos serão complementações do caso de uso “Cadastrar Entidade”. No caso de uso “Cadastrar veículo” é possível registrar as características que o veículo tem. No Quadro 5, este caso de uso está descrito com maiores detalhes.

|  |  |
| --- | --- |
| **Caso de uso:**  Cadastrar Funcionário **Descrição**:  O segurança registra uma entidade cadastrada como funcionário. **Evento Iniciador**:  Cadastro de entidade **Atores**:  Usuário Segurança.  **Pré-condição**:  A entidade com os dados da pessoa que será o funcionário deverá estar registrada.  **Sequência de Eventos**:  1. Ator acessa a tela para cadastrar funcionário | |
| 1. O sistema apresenta uma lista de entidades cadastradas para seleção. 2. Ator seleciona a entidade. 3. O ator lança os dados que compõe o cadastro de pessoa/funcionário. 4. Ator confirma os dados inserindo-os no banco de dados. 5. Sistema informa que os dados foram incluídos no banco de dados e redireciona para a página de listagem de funcionários.   **Pós-Condição**:  Dados do funcionário inseridos no banco de dados. | |
| Nome do fluxo alternativo (extensão) | Descrição |
| Linha 4: O segurança deseja definir as características que o veículo tem. | 4.1 O Ator adiciona uma característica.  4.2 O Ator seleciona o local da vaga.  4.3 O Ator informa a existência da vaga.  4.4 Retorna ao passo 4 e o caso de uso prossegue com fluxo normal. |
| Linha 5: Dados informados são inválidos | 5.1 No momento de salvar, o sistema verifica a validade de todos os dados informados, armazenando quais são inválidos;  5.2 Retorna para o formulário de cadastro informando cada um dos campos contendo dados inválidos. |

**Quadro 4 – Caso de uso cadastrar funcionário**

O caso de uso “Registrar Agendamento” é apresentado no Quadro 6.

|  |  |
| --- | --- |
| **Caso de uso:**  Registrar Agendamento **Descrição**:  O usuário registra um agendamento de vaga de estacionamento.  **Evento Iniciador**:  Contato de uma pessoa/funcionário para agendamento de atividade.  **Atores**:  Usuário Atendente ou Segurançar. **Pré-condição**:  A entidade com os dados da pessoa para o qual a atividade será registrada deve estar  cadastrada no sistema.  A atividade deve estar cadastrada no sistema.  Tipo de vaga deve estar registrado no sistema.  **Sequência de Eventos**:   1. Ator acessa a página de calendário com todos as vagas agendadas no período exibido. 2. O Ator seleciona um horário no calendário. 3. O sistema apresenta um formulário com a data e horário selecionado, uma lista de entidades cadastradas e uma lista de vagas disponíveis para aquele horário. 4. O ator seleciona a entidade, vaga e informações adicionais. 5. Ator confirma os dados inserindo-os no banco de dados. 6. Sistema informa que os dados foram incluídos no banco de dados e retorna para a página | |
| de calendário.  **Pós-Condição**:  Dados do tipo de vaga inseridos no banco de dados. | |
| Nome do fluxo alternativo (extensão) | Descrição |
| Linha 3: Não há vagas disponíveis no horário selecionado | 3.1 O sistema emite uma mensagem.  3.2 O Ator é retorna ao passo 2 e o caso de uso prossegue com fluxo normal. |
| Linha 5: Dados informados são inválidos | 5.1 No momento de salvar, o sistema verifica a validade de todos os dados informados, armazenando quais são inválidos;  5.2 Retorna para o formulário de cadastro informando cada um dos campos contendo dados inválidos. |

**Quadro 5 – Caso de uso registrar agendamento**

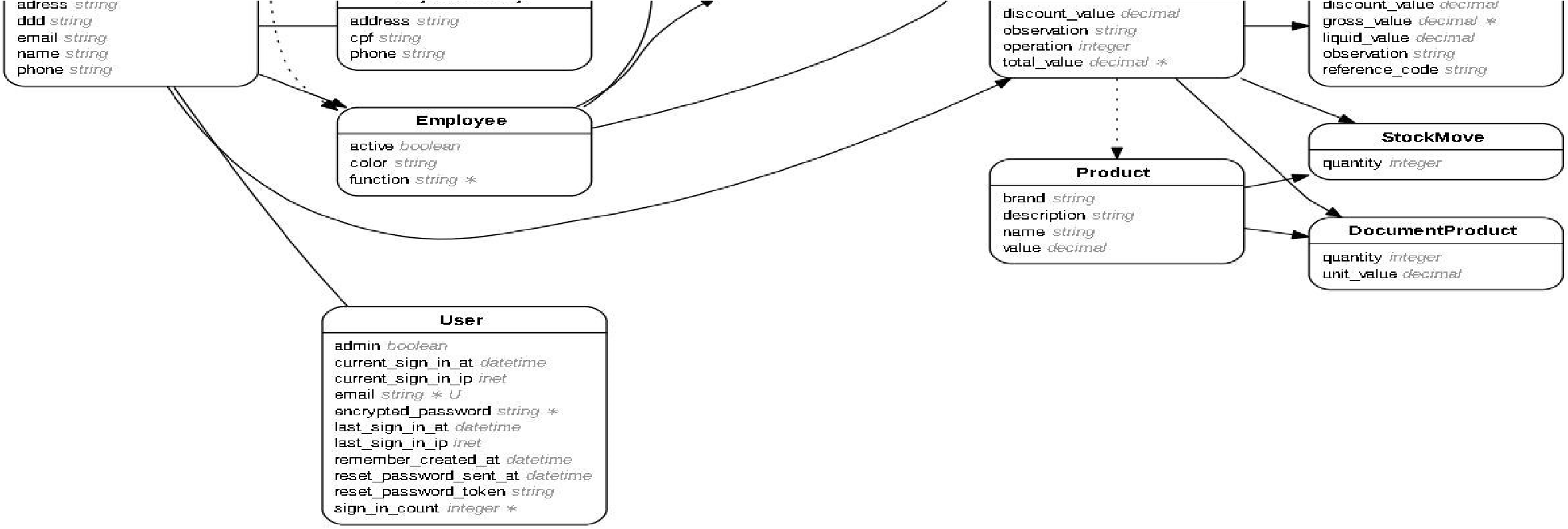
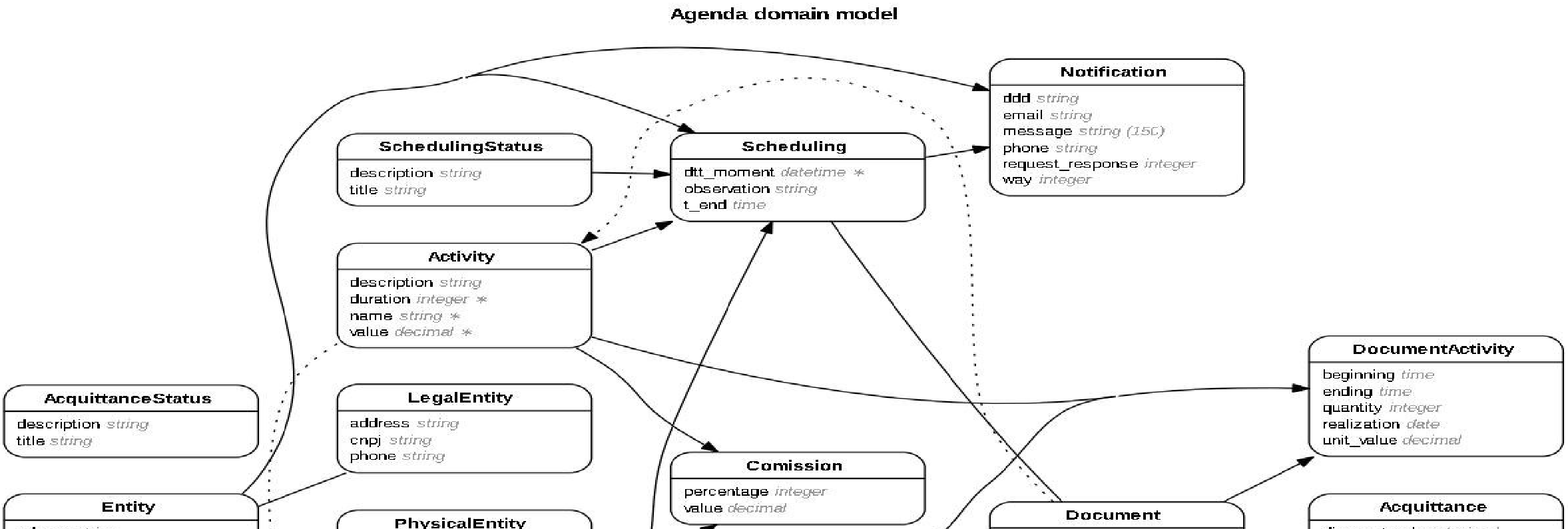
Devido ao *framework* Ruby on Rails trabalhar com mapeamento objeto-relacional

(*Object-Relational Mapping* (ORM)) por meio da biblioteca *Active Record*, as classes do sistema não necessitam que os atributos de instância sejam especificados na classe para serem referenciados nas operações de CRUD.

A criação e manutenção da estrutura da base de dados é efetuada por meio de *migrations*, gerenciadas pels biblioteca *Active Record*, não sendo necessária a inclusão manual das tabelas do banco de dados diretamente por linguagem SQL. Essas implementações do *Active Record* além de facilitar o desenvolvimento de diversas formas, como agilidade e organização nas modificações da estrutura da base de dados, eliminam a necessidade da criação do diagrama de entidades e relacionamentos do sistema, uma vez que tanto as tabelas como suas restrições são criadas com base nos *models*.

A Figura 3 apresenta o diagrama de entidades e relacionamento do sistema gerado por meio da *gem* *rails-erd*. Após o desenvolvimento do sistemaapresentando os *models* e suas respectivas relações. Essa *gem* utiliza os *models* para mapear a estrutura da base de dados do sistema e gera uma representação gráfica para facilitar a visualização.

29

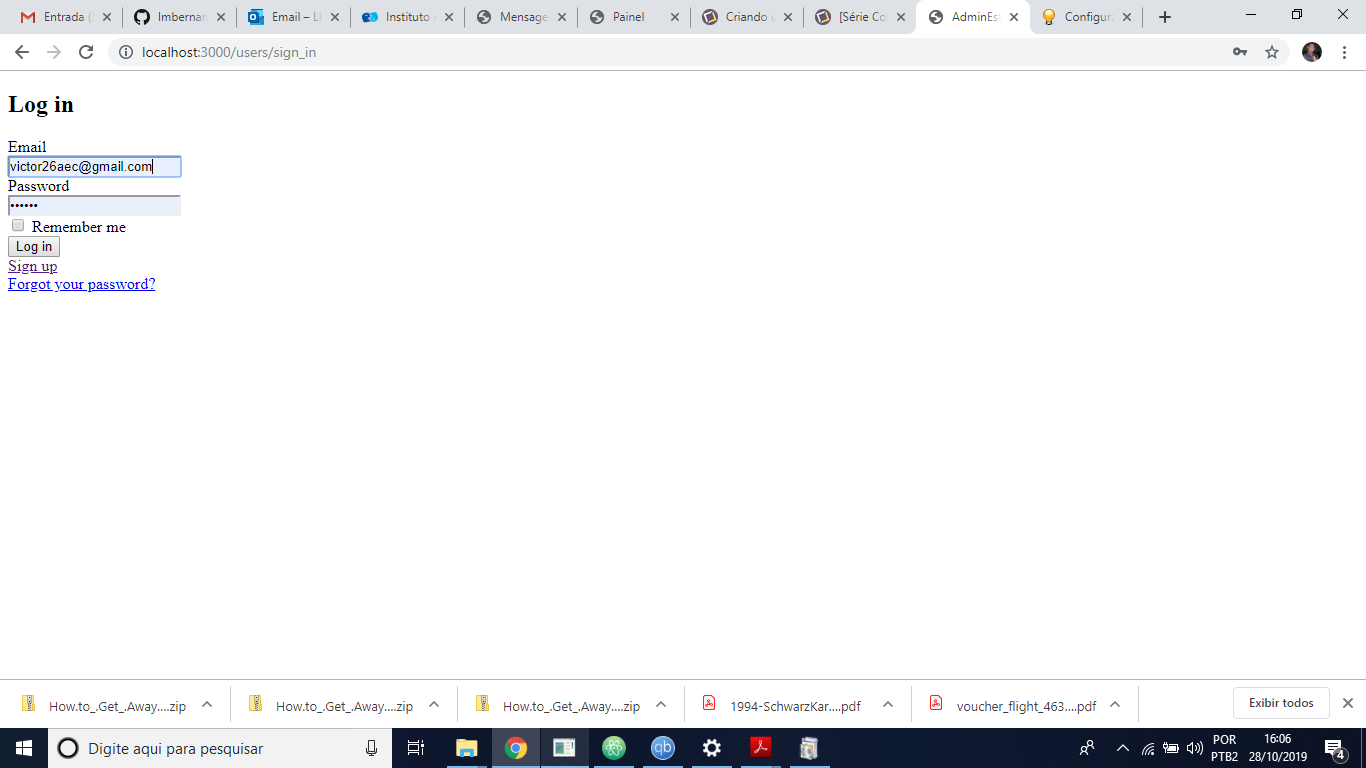


**Figura 3 – Diagrama de entidades e relacionamentos gerado pela *gem* *erd***

Para aproveitar ao máximo as convenções do *framework*, pela filosofia *Convention over configuration*, todos os recursos de texto e mensagens para o usuário foram desenvolvidos utilizando a língua inglesa. O Rails possui uma biblioteca que é utilizada por todos os módulos denominada “*pluralize*”, cuja função é traduzir os termos e métodos automaticamente para o plural quando necessário. Porém, essa funcionalidade está apenas disponível para língua inglesa. Além disso, evita-se a nomeação de elementos na base de dados com caracteres especiais, como “ç”, “ã”, entre outros. Para que o sistema possa ser utilizado sem a necessidade de o usuário conhecer a língua inglesa, o Rails disponibiliza uma gem denominada “I18n” (abreviação para “*Internacionalization*”), em que é possível modificar todos os textos da aplicação criando arquivos de tradução. Essa funcionalidade poderá ser utilizada como uma opção de implementação futura.

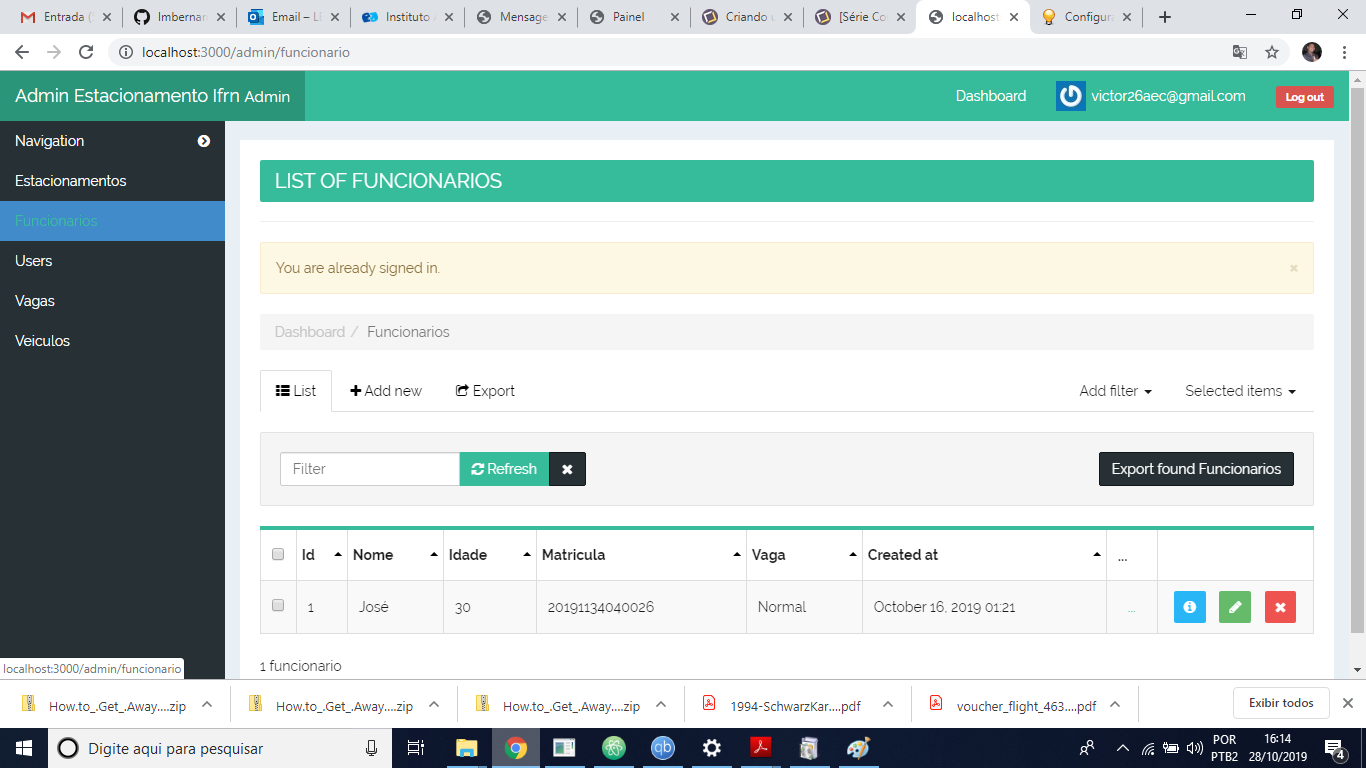
## 4.3 APRESENTAÇÃO DO SISTEMA

Ao acessar o sistema o usuário deverá informar suas credenciais de acesso, qualquer *link* para recursos do sistema será bloqueado, redirecionando o usuário para a página de *login*.

**Figura 04.**

Na página de *login*, apresentada na Figura 4 o usuário tem a opção cadastrar-se no sistema (caso ainda não tenha a sua conta) ou entrar com os seus dados de acesso pelo formulário de *login*.

No formulário de funcionário é possível escolher a vaga, o estacionamento e o veículo.



## 4.4 IMPLEMENTAÇÃO DO SISTEMA

O Ruby on Rails por ser um *framework* e ter em sua concepção a ideia de desenvolvimento rápido, utilizando a filosofia Less, como mencionado na secção 2.3, toda estrutura do projeto será a já definida por convenção do *framework*. Para gerar a estrutura do sistema para o projeto foi utilizado o comando apresentado na Listagem 1.

rails new IFRN\_Estacionamento\_Admin --database=postgresql

**Listagem 1 – Criação de um novo projeto Rails**

O comando “rails” é reservado para operações do *framework*. O parâmetro “new” especifica que se deseja criar um novo projeto Rails, em seguida, deve ser definido o nome do projeto a ser criado, nesse caso, “IFRN\_Estacionamento\_Admin”. Após a execução do comando, um diretório é criado com o nome do projeto, contendo toda a estrutura do *framework*.

O Rails possui uma vasta quantidade de *gems* disponíveis para utilização que facilitam, ou até, liberam o desenvolvimento de determinadas funcionalidades das aplicações. A lista de *gems* utilizadas pelo projeto está disponível no arquivo “/GemFile”, cujo conteúdo é apresentado pela Listagem 2.

|  |
| --- |
| source 'https://rubygems.org' gem 'rails', '4.2.4' gem 'pg' gem 'sass-rails', '~> 5.0' gem 'therubyracer' gem 'twitter-bootstrap-rails' gem 'uglifier', '>= 1.3.0' gem 'coffee-rails', '~> 4.1.0' gem 'jquery-rails' gem 'turbolinks' gem 'jbuilder', '~> 2.0' gem 'devise' gem 'ransack' gem 'pundit' gem 'will\_paginate', '~> 3.0.6' gem 'simple\_form' gem 'cocoon' gem 'sdoc', '~> 0.4.0', group: :doc group :development, :test do gem "rails-erd" gem 'byebug' end  group :development do gem 'web-console', '~> 2.0' gem 'spring' end |

**Listagem 2 – GemFile**

A primeira linha da Listagem 2 apresenta o local de *download* das *gems* pelo *bundle* (gerenciador de dependência do Ruby On Rails). A *gem* "rails” é o *framework* Ruby; “pg” é a *gem* para conexão do SGBD PostgreSQL; “twitter-bootstrap-rails” adiciona o *framework* CSS/JavaScript para estilização das páginas; “devise” é uma biblioteca para autenticação de usuários; “ransack” é uma biblioteca para busca; “will\_paginate” é uma biblioteca para paginação; “cocoon” é uma API para geração de formulários para objetos agregados;

Sempre que uma *gem* for adicionada ou removida, basta utilizar o comando “bundle install” para que as dependências sejam mapeadas e seu *download* efetuado se for o caso.

### 4.4.1 Configuração da Base de Dados

A configuração do banco de dados é especificada no arquivo /config/database.yml. Por convenção, o banco de dados vem pré-configurado com três ambientes: desenvolvimento, teste e produção. Cada ambiente pode possuir suas próprias configurações de base de dados utilizada, usuários, portas e diversas outras opções. A extenção “.yml” refere-se a um arquivo que utiliza o formato *Ain't Markup Language* (YAML) de serialização, capaz de ser mapeado para obtenção de dados diretamente do arquivo. As configurações definidas para o ambiente de desenvolvimento estão apresentadas na Listagem 3.

|  |
| --- |
| default: &default adapter: postgresql encoding: unicode pool: 5 development: <<: \*default  database: IFRN\_Estacionamento\_Admin\_development username: vagrant password: |

**Listagem 3 – Configurações da base de dados**

As definições incluídas em “default” são aplicadas para todos os ambientes. Em “development” está definido que o banco utilizado será “IFRN\_Estacionamento\_Admin\_development”, utilizando o usuário “vagrant” sem especificar senha. Para que o sistema possa conectar na base de dados é necessário criar a *role* de acordo com a configuração especificada na Listagem 3. Após a definição das configurações da base de dados é necessário executar o comando que criará as bases de acordo com o especificado, para isso utiliza-se o comando “rake”, que é um programa de construção que executa diversas funcionalidades para o *framework*. Entre essas funcionalidades está a execução de *migrations*, que são os arquivos cuja execução realiza as modificações na estrutura da base de dados da aplicação. Para criar a base de dados e já tornála disponível é executado o comando da Listagem 4.

rake db:create

**Listagem 4 – Criando a base de dados**

### 4.4.2 Scaffolding

O desenvolvimento do sistema é gerado por meio de *Scaffolding* (termo em inglês que denota estruturação de andaimes temporários). O gerador *Scaffold* do Rails cria diversos arquivos com base nos parâmetros informados. Esta técnica de desenvolvimento não visa isentar o desenvolvedor de desenvolver a estrutura, mas somente utilizar as convenções para agilizar este trabalho.

A Listagem 5 apresenta que o comando Rails deve gerar *scaffolding* o *model* Entity (Entidade), com campos de nome, *email*, cpf, endereço e telefone. Outros campos podem ser adicionados posteriormente utilizando *migrations*.

rails g scaffold Entity name:string email:string cpf:string adress:string phone:string

**Listagem 5 – Construindo a base de dados**

Ao executar, todos os componentes MVC necessários para o funcionamento básico das operações de CRUD serão criados com base nas convenções do *framework*. A lista de arquivos gerados é apresentada na Listagem 6.

invoke active\_record create db/migrate/20151111043919\_create\_entities.rb create app/models/entity.rb invoke test\_unit create test/models/entity\_test.rb create test/fixtures/entities.yml invoke resource\_route route resources :entities invoke scaffold\_controller create app/controllers/entities\_controller.rb invoke erb create app/views/entities create app/views/entities/index.html.erb create app/views/entities/edit.html.erb create app/views/entities/show.html.erb create app/views/entities/new.html.erb create app/views/entities/\_form.html.erb invoke test\_unit create test/controllers/entities\_controller\_test.rb invoke helper create app/helpers/entities\_helper.rb invoke test\_unit invoke jbuilder create app/views/entities/index.json.jbuilder create app/views/entities/show.json.jbuilder invoke assets invoke coffee create app/assets/javascripts/entities.coffee invoke scss create app/assets/stylesheets/entities.scss invoke scss create app/assets/stylesheets/scaffolds.scss

**Listagem 6 – Arquivos criados pelo gerador Scaffold**

O principal arquivo criado é “/app/models/entity.rb”, que é basicamente uma classe que herda *Active Record*, que realizará toda a interface entre aplicação e banco de dados, inclusive mapeando os atributos da entidade no banco de dados. A criação da entidade na base de dados é feita por meio da *migration* “/db/migrate/20151111043919\_create\_entities.rb”, contendo os atributos requisitados no comando de geração da Listagem 7.

$ rake db:migrate

**Listagem 7 – Realizando migrações**

A execução da *migration* é feito pelo comando *rake*, como apresentado na Listagem 7. Com a *migration* gerada pelo *scaffold* executada, todos os recursos de CRUD já estão disponíveis. As operações são atendidas pelos controladores, neste caso, o

“/app/controller/entities\_controller.rb” que atende as requisições dos usuários. Por padrão, as quatro operações básicas já são suportadas pelo *controller* e estão prontas para serem utilizadas pelos quatro *templates* criados pelo gerador: Listagem dos registros (/app/entities/index.html.rb), cadastro de um novo registro (/app/entities/new.html.rb), edição de um registro (/app/entities/edit.html.rb) e exibição detalhada de um registro (/app/entities/show.html.rb).

Para que o *controller* receba as informações e renderize as páginas é necessário que rotas sejam definidas no arquivo /config/routes.rb. O comando “rake routes” apresenta as rotas configuradas na aplicação, como é apresentado no Quadro 8.

|  |  |  |  |
| --- | --- | --- | --- |
| **Prefixo** | **Ação**  **HTTP** | **URL Modelo** | **Controller#Ação** |
| Entities | GET | /entities(.:format) | entities#index |
|  | POST | /entities(.:format) | entities#create |
| new\_entity | GET | /entities/new(.:format) | entities#new |
| edit\_entity | GET | /entities/:id/edit(.:format) | entities#edit |
| Entity | GET | /entities/:id(.:format) | entities#show |
|  | PATCH | /entities/:id(.:format) | entities#update |
|  | PUT | /entities/:id(.:format) | entities#update |
|  | DELETE | /entities/:id(.:format) | entities#destroy |

**Quadro 7 – Rotas configuradas para a aplicação**

Rotas funcionam baseadas em correspondência de padrão e podem incluir variáveis. O padrão é composto por segmentação em árvore, particionando esses segmentos com barras (/) com as variáveis prefixadas por dois-pontos (:). No Quadro 8, a coluna “prefixo” apresenta o apelido para a rota para não ser necessário especificar todo o caminho a cada necessidade, referenciando-o como um método sempre que seu uso for requisitado. Os métodos *resources* criam todas as rotas básicas para o CRUD do Rails para o recurso em questão, as rotas contidas dentro dos blocos “*controller”* sobrescrevem as rotas já definidas e especificam que apontam para o *controller* do recurso informado pelo *symbol*.

### 4.4.3 Definição do Leiaute Básico

O leiaute do sistema foi estilizado utilizando *twitter* Bootstrap por meio da *gem*

“bootstrap-sass”. A instrução da Listagem 8 adiciona ao projeto a *gem* Bootstrap com motor “sass”. Uma vez instalada e configurada, já será possível utilizar os geradores da *gem* que estilizam a página para utilização do Bootstrap.

$ rails g bootstrap:themed entities

**Listagem 8 – Adição da *gem* Bootstrap ao projeto**

O gerador Bootrstap criará as *views* para o recurso nomeado utilizando o mesmo padrão do *scaffold*, porém, adicionando a estilização básica do Bootstrap, sendo necessário apenas aprimorar o leiaute.

Toda requisição feita para uma aplicação Rails passará pela biblioteca *Action Pack*, que é composto de dois módulos: *Action Controller* e *Action View*. Esses dois módulos trabalham em conjunto para atender as requisições à aplicação. *Action Controller* as recebem, realizam a ação requisitada resultando em uma resposta que é resolvida por *Action View* formatando da forma com que o remetente possa entender, renderizando uma página para o navegador ou retornando o resultado em outros formatos, como *JavaScript Object Notation* (JSON).

Quando o *controller* gerar uma resposta para uma requisição na qual uma página será renderizada, o Rails buscará por um *layout template* para construí-la nomeado-a segundo o nome do recurso, como no exemplo de “Entity”, em “/app/view/layouts/entities.html.erb”. Se este leiaute não estiver definido, *Action View* buscará pela o leiaute nomeado de acordo com a classe progenitora (*Application Controller*), “/app/views/layout/application.html.rb”. Devido às *views* sempre serem renderizadas a partir dos *templates* de leiaute, os itens padrões para toda a aplicação, como menu do topo da página e mensagens *flash*, serão definidas neste local. A Listagem 9 apresenta o conteúdo de “application.html.rb”.

# /app/view/layout/application.html.erb

# (...)

<body>

<header>

<nav class="navbar navbar-primary navbar-static-top">

#(...) html code of top menu

</nav>

</header>

<main>

<div class="container">

<div class="row">

<%= bootstrap\_flash %>

</div>

<div class="row-fluid">

<%= yield %>

</div>

</div><!--/row-->

</main>

<footer>

</footer>

</body>

# (...)

**Listagem 9 – Leiaute *Template* básico da aplicação**

Na Listagem 9, o menu será fixado no topo do site e conterá *links* para as funcionalidades do sistema. As *tags* estão estilizadas utilizando as regras do Bootstrap, mencionando anteriormente. Quando o método ***yield*** é chamado, outro *template* é obtido para a montagem da página. O *template* a ser utilizado para a geração é identificado pelo nome do método (Ação) requisitada ao *controller*. Portanto, quando uma a rota “(...)/entities” é requisitada, o método chamado (de acordo com o que foi definido nas rotas) será no *controller* “***entities***”, método “***index*** e o leiaute renderizado é

“/app/views/layout/application.html.rb”, incluindo, pelo método ***yield*** o conteúdo do *template* “/app/views/entities/index.html.rb”.

### 4.4.4 Autenticação de Usuários

Para autenticar usuários a *gem* *Devise* foi utilizada. Essa *gem* possui um gerador que cria o *model* para usuário, *migration* e as *views* para *login*, cadastro, edição, modificação e solicitação de nova senha. Porém é necessário gerar o *controller* para ter acesso às ações que o *devise* possui e modificá-las. O código para autenticação da *gem* *Devise* está na Listagem

10.

$ rails g devise:install

$ rails g devise User entity:references

$ rails g devise:views

**Listagem 10 – Geração de arquivos para autenticação**

A Listagem 10 apresenta os comandos executados para geração dos arquivos para autenticação de usuários utilizando a *gem* *devise*. O primeiro comando adiciona os recursos da *gem* como arquivos de configuração e rotas. Em seguida, o *model* User é gerado com uma referência à Entidade, os parâmetros para *login* que são definidos por padrão no *devise* são *email*, senha e diversos *timestamps* que marcam cadastro, edição de senha, último *login,* entre outros. O próximo comando gera as *views* para as diversas funcionalidades que o *devise* traz pronta, como cadastro, edição de dados, *login*, recuperação de senha e confirmação de *email*, e outros recursos. A geração das *views* é necessária para personalização das páginas, caso contrário, o *controller* da *gem* renderizaria *views* padrão, sem estilização alguma. Apesar do gerador disponibilizado pelo *devise* criar a *migration* adicionando os parâmetros adicionais, as *views* e o *controller* não fornecem meios de adicionar a informação da entidade, sendo necessário criar um *controller* próprio para atender as necessidades específicas do projeto. O código fonte do *controller* que herda *RegistrationController* do *devise* é apresentado na Listagem 11.

|  |
| --- |
| # /app/controller/registrations\_controller.rb class RegistrationsController < Devise::RegistrationsController prepend\_before\_filter :authenticate\_scope!, only: [:edit, :update, :destroy]  def new  @user = User.new if params[:entity\_id]  @user.entity\_id = params[:entity\_id] @user.email = @user.entity.email unless @user.entity.email.blank?  end  end private    def account\_update\_params params.require(:user).permit(:entity\_id, :email, :password,  :password\_confirmation, :current\_password) end end |

**Listagem 11 – *Controller* para registro de usuários utilizando a *gem devise***

Herdando de RegistrationsController do *Devise* é possível modificar o comportamento das ações do *controller* via polimorfismo adicionando o campo entity\_idpara registro e edição do cadastro de usuário. Sobrescrevendo, assim, os métodos que tratam os parâmetros que cada ação recebe, neste caso, os métodos de cadastro e atualização de dados da conta.

# 5 CONCLUSÃO

O objetivo deste trabalho foi implementar um sistema para gerenciamento de um estacionamento, registrando agendamentos vagas. A plataforma *web* foi escolhida para o desenvolvimento da aplicação devido à amplitude do alcance e crescente quantidade de meios de acesso, além da facilidade com relação a aplicações embarcadas na manutenção, implantação e disseminação do sistema.

A principal tecnologia utilizada para a implementação foi o *framework* Ruby On Rails. Baseado na linguagem Ruby, e criado com o objetivo de ser uma alternativa mais versátil, estruturada e com diversas filosofias que visam facilitar o trabalho do desenvolvedor sem sacrificar estabilidade e escalabilidade, como é o caso da filosofia *less*, princípios como DRY e convenções sobre configurações, fazem desta ferramenta uma promissora opção para desenvolvimento *web*.

Em comparação ao desenvolvimento de aplicativos em PHP sem a utilização de *frameworks*, a diferença de tempo de desenvolvimento, apesar de não ser quantificada precisamente no decorrer do desenvolvimento, estabeleceu-se em aproximadamente metade do tempo necessário para o desenvolvimento dos mesmos recursos. Com relação à quantidade de código digitado, tomando como exemplo o recurso de agendamentos, utilizando-se o Ruby On Rails foram necessárias um quinto (1/5) de linhas de código com relação a mesma funcionalidade utilizando PHP. Ressalta-se que essas informações foram levantadas superficialmente por meio do gerenciador de versões Git.

Apesar da comunidade do Ruby on Rails não ser tão abrangente como outras linguagens já estabelecidas, mostrou-se participativa. As dúvidas são prontamente respondidas, artigos e tutoriais estão constantemente sendo publicados. Uma excelente característica desta tecnologia é sua constante evolução, novas versões, correções de *bugs* e a facilidade de desenvolvimento de *gems* para enriquecimento do material disponível. Por outro lado, a complexidade da ferramenta, seu paradigma diferenciado (a linguagem Ruby) aliados a sua rápida evolução, fazem com que todo o material disponível torne-se obsoleto relativamente rápido comparado às linguagens PHP, Java ou C, prejudicando a curva de aprendizagem.

O gerenciador de banco de dados PostgreSQL, mostrou-se estável e atendeu as necessidades do projeto. Contudo, a utilização desta ferramenta não foi explorada em todos os aspectos pela estrutura do *framework*, que dispensa a linguagem SQL para a maioria das operações com a base de dados.

Para controle de versões, o Git foi uma surpresa agradável, possibilitando a reestruturação de parte do projeto quando necessário e principalmente o gerenciamento de modificações de um mesmo arquivo por diferentes desenvolvedores, eliminando problemas para pareamento de códigos fonte.

Uma grande dificuldade encontrada durante o desenvolvimento do projeto, além da ausência de bons livros atualizados sobre Ruby On Rails em português, foi a quebra de paradigmas do conhecimento adquirido nas diversas outras linguagens devido a peculiaridade da linguagem Ruby e a complexidade do *framework* Rails.

As dificuldades foram superadas com a utilização da própria documentação do *framework* Ruby On Rails que possui guias excelentes para praticamente todos os recursos fundamentais da ferramenta. Além disso, é necessário ressaltar a atenção prestada e a disponibilidade em auxiliar da comunidade Rails em redes sociais e fóruns de programação, que, apesar de relativamente pequena comparada a outras linguagens já estabelecidas, como Java ou PHP, mostrou-se presente e genuinamente disposta a ajudar.

Necessário para implementações futuras a utilização mais abrangente do *framework* Jquery para automatização e desenvolvimento de interfaces mais avançadas, possibilitando ao usuário uma melhor experiência de utilização da ferramenta em seu dia a dia.

# REFERÊNCIAS

AN, Jong-hoon David; CHAUDHURI, Avik; FOSTER, Jeffrey S. **Static Typing for Ruby on Rail**s. 2009. In: IEEE/ACM International Conference on Automated Software Engineering, 2008, p. 98-106.

BÄCHLE, Mchael; KIRCHBERG, Paul. **Ruby on Rails**. IEEE Software,

November/December 2007, p.105-108.

CANARIM, Patrícia. **O nascimento da internet começou na 2ª Guerra Mundial**. 2012. Disponível em: < http://webinsider.com.br/2012/04/07/o-nascimento-da-internet-comecou-na2a-guerra-mundial/>. Acesso em: 22 ago. 2014.

FERNANTEZ, Obie. FAUSTINO, Kevin. KUSHNER, Vitaly. **The Rails 4 Way.** Leanpub, Vancouver, 2014.

GAMBLE, Adam; CARNEIRO, Cloves Jr.; BARAZI, Rida Al. **Begining Rails 4. Apress**.

Disponível em:

<http://file.allitebooks.com/20150617/Beginning%20Rails%204,%203rd%20Edition.pdf>. Acesso em: 07 nov. 2015.

GEER, David. Will software developers ride Ruby on Rails to success? Technology News,v. 39, n. 2, 2006, p. 18 – 20.

MASCHIETTO, Leandro Cesari. **Banco de dados orientados a objetos**. 2012. Disponível em: < http://lmtelecom.blogspot.com.br/2012/04/banco-de-dados-orientados-objetos.html>. Acesso em: 11 jan. 2015.

RAILS GUIDE. **Active record basics**. 2015ª. Disponível em:

**<**http://guides.rubyonrails.org/active\_record\_basics.html#convention-over-configuration-inactive-record>. Acesso em: 07 nov. 2015.

RAILS GUIDE. **Migrations**. 2015b. Disponível em:

<http://guides.rubyonrails.org/v3.2.21/migrations.html>. Acesso em: 29 nov. 2015.

RAILS GUIDE. **Gems**. 2015c. Disponível em: <https://rubygems.org/gems>. Acesso em: 29 nov. 2015.

RUBY ON RAILS. 2015. Disponível em: <https://www.ruby-lang.org/pt/>. Acesso em: 28 jul. 2014.

SOUZA, Lucas. **Ruby**. Casa do código, 2013. Disponível em:

<http://www.casadocodigo.com.br/products/livro-ruby> . Acesso em: 20 fev. 2015.

STELLA, Lok Fang Fang; JARZABEK, Stan; WADHWA, Bimlesh. **A Comparative Study of Maintainability of Web Applications on J2EE, .NET and Ruby on**

**Rails.** 10th International Symposium on Web Site Evolution (WSE 2008), 2008, p. 93 – 99.

VISWANATHAN, **Viswa.** Rapid Web Application Development: a Ruby on Rails Tutorial. IEEE Software, v. 25, n. 6, 2008, p. 98-106.

WORLD WIDE WEB CONSORTIUM (W3C). Disponível em: <http://www.w3.org/>.

Acesso em: 25 ago. 2014.

1. www.opensource.org/licenses/mit-license.php [↑](#footnote-ref-1)