

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**  
**DEPARTAMENTO ACADÊMICO DE INFORMÁTICA**  
**CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**FELIPE SCHROLL LOSS**  
**MAICO SILVINO DAL PONTE**

**DESENVOLVIMENTO DE UM APLICATIVO WEB PARA SALÕES DE**  
**BELEZA UTILIZANDO RUBY ON RAILS**

**TRABALHO DE CONCLUSÃO DE CURSO**

**PATO BRANCO**

**2015**

**FELIPE SCHROLL LOSS**  
**MAICO SILVINO DAL PONTE**

**DESENVOLVIMENTO DE UM APLICATIVO WEB PARA SALÕES DE  
BELEZA UTILIZANDO RUBY ON RAILS**

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

Orientador: Profa. Beatriz Terezinha Borsoi

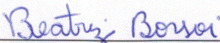
**PATO BRANCO**

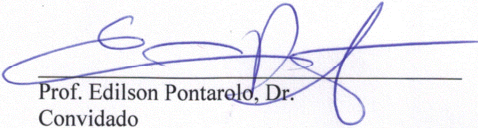
**2015**

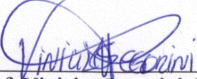
ATA Nº: 276

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DOS ALUNOS MAICO SILVINO DAL PONTE e FELIPE SCHROLL LOSS.

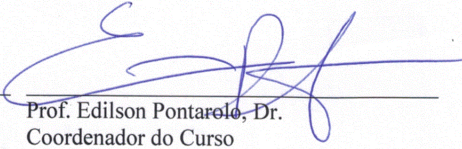
Às 14:40 hrs do dia 27 de novembro de 2015, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Edilson Pontarolo (Convidado) e Vinicius Pegorini (Convidado), para avaliar o Trabalho de Diplomação do aluno Maico Silvino Dal Ponte, matrícula 1209159 e do aluno Felipe Schroll Loss, matrícula 1210068, sob o título **Desenvolvimento de um aplicativo web para salões de beleza utilizando Ruby on Rails**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação os candidatos foram entrevistados pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 15:30 hrs foi encerrada a sessão.

  
\_\_\_\_\_  
Profa. Beatriz Terezinha Borsoi, Dr.  
Orientadora

  
\_\_\_\_\_  
Prof. Edilson Pontarolo, Dr.  
Convidado

  
\_\_\_\_\_  
Prof. Vinicius Pegorini, M.Sc.  
Convidado

  
\_\_\_\_\_  
Profa. Soelaine Rodrigues Ascari, M.Sc.  
Coordenador do Trabalho de Diplomação

  
\_\_\_\_\_  
Prof. Edilson Pontarolo, Dr.  
Coordenador do Curso

## RESUMO

LOSS, Felipe Schroll; DAL PONTE, Maico Silvino. Desenvolvimento de um aplicativo web para salões de beleza utilizando Ruby on Rails. 2015. 62f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Salões de beleza prestam serviços relacionados à estética. E como esses serviços podem ser diversificados (relacionados a cabelo, tratamento de pele, maquiagem, unhas e outros) é natural que sejam realizados por profissionais distintos e podendo haver diversos profissionais para uma mesma especialidade, dependendo do tamanho do estabelecimento. Para o gerenciamento dos serviços, controle de contas e de estoque e outros um sistema *desktop* pode ser suficiente. Contudo, quando enviar mensagens para clientes, seja para lembrá-los de agendamentos e promoções, por exemplo, é uma estratégia de negócio utilizada pelo salão, sistemas para Internet são mais indicados. Considerando esse escopo e contexto de aplicabilidade de desenvolvimento de sistemas *web* para salões de beleza, por meio da realização deste trabalho um aplicativo *web* para salões de beleza foi desenvolvido. A tecnologia principal utilizada foi o *framework* Ruby on Rails, que foi escolhida pelos recursos que oferece e pelo tratamento da orientação a objetos, visando reuso e simplificação do desenvolvimento. Assim, além do sistema em si, neste texto é apresentado o uso desse *framework* no desenvolvimento de sistema *web*, com foco em produtividade pelo reuso e uso de recursos da própria tecnologia.

**Palavras-chave:** Ruby on Rails. Aplicativo *web*. Sistema para salões de beleza.

## ABSTRACT

LOSS, Felipe Schroll; DAL PONTE, Maico Silvino. Development of a web application for beauty salons using Ruby on Rails. 2015. 62f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Beauty salons render aesthetic-related services. As diversified this services can be (related to hair, skin treatment, makeup, nails and others) it is natural to be carried out by different professionals and there can be several to the same specialty, depending of the establishment size. For the services management, account and inventory control and other, a desktop system may be sufficient. However, when sending messages to customers, either to remind them of appointments and promotions, for example, is a business strategy used by the salon, Internet systems are best suited. Considering this context of web systems development applicability for salons, through this work a web application for salons was developed. The main technology used was the Ruby on Rails framework, which was chosen by the features it offers and the treatment of object orientation, aiming to reuse and simplify the development. Thus, besides the system itself, in this text display the use of framework to web system developing, focusing on productivity through reuse and resource usage of the technology itself.

**Palavras-chave:** Ruby on Rails. Web application. Beauty salons system.

## LISTA DE FIGURAS

Figura 1 – Componentes do <i>framework</i> Ruby on Rails.....	15
Figura 2 – Diagrama de domínio .....	24
Figura 3 – Diagrama de entidades e relacionamentos gerado pela <i>gem erd</i> .....	31
Figura 4 – Página de <i>login</i> .....	32
Figura 5 – Formulário de cadastro de funcionário.....	33
Figura 6 – Cadastro de agendamento.....	34
Figura 7 – Lista de Agendamentos .....	34
Figura 8 – Formulário de cadastro de Documento .....	35
Figura 9 – Nova quitação (baixa) .....	36

## 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 .....	26
Quadro 4 – Diagrama de casos de uso .....	27
Quadro 5 – Caso de uso cadastrar funcionário.....	28
Quadro 6 – Caso de uso registrar agendamento .....	29
Quadro 7 – Caso de uso registrar documento.....	30
Quadro 8 – Rotas configuradas para a aplicação.....	40

## 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 <i>gem</i> Bootstrap ao projeto.....	41
Listagem 9 – Leiaute <i>Template</i> básico da aplicação.....	42
Listagem 10 – Geração de arquivos para autenticação.....	43
Listagem 11 – <i>Controller</i> para registro de usuários utilizando a <i>gem devise</i> .....	43
Listagem 12 – <i>Models Employee, Activity e Comission</i> .....	45
Listagem 13 – Aceitar parâmetros de objetos agregados.....	45
Listagem 14 – <i>Partials</i> para formulário mestre-detalle pela <i>gem cocoon</i> .....	47
Listagem 15 – <i>Model</i> Scheduling com validações e consultas ao banco de dados .....	48
Listagem 16 – Método para filtragem de agendamentos pelo dia corrente .....	49
Listagem 17 – <i>Partial _form</i> para formulário mestre-detalle de Documento .....	54
Listagem 18 – <i>Model</i> para documentos .....	56
Listagem 19 – <i>Model</i> e método criação do <i>controller</i> de <i>Acquittance</i> .....	57



## LISTA DE ABREVIATURAS E SIGLAS

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

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>10</b>
1.1 CONSIDERAÇÕES INICIAIS .....	10
1.2 OBJETIVOS.....	11
<b>1.2.1 Objetivo Geral.....</b>	<b>11</b>
<b>1.2.2 Objetivos Específicos.....</b>	<b>12</b>
1.3 JUSTIFICATIVA .....	12
1.4 ESTRUTURA DO TRABALHO .....	13
<b>2 REFERENCIAL TEÓRICO .....</b>	<b>14</b>
2.1 CONTEXTO.....	14
2.2 RUBY.....	14
2.3 CONVENÇÕES SOBRE CONFIGURAÇÃO.....	17
2.4 DON'T REPEAT YOURSELF .....	18
2.5 MIGRATIONS .....	18
2.6 GEMS.....	19
<b>3 MATERIAIS E MÉTODO .....</b>	<b>20</b>
3.1 MATERIAIS.....	20
3.2 MÉTODO .....	21
<b>4 RESULTADO .....</b>	<b>22</b>
4.1 ESCOPO DO SISTEMA.....	22
4.2 MODELAGEM DO SISTEMA.....	23
<b>4.2.1 Especificação de Requisitos .....</b>	<b>25</b>
4.3 APRESENTAÇÃO DO SISTEMA .....	32
4.4 IMPLEMENTAÇÃO DO SISTEMA .....	36
<b>4.4.1 Configuração da Base de Dados .....</b>	<b>37</b>
<b>4.4.2 Scaffolding.....</b>	<b>38</b>
<b>4.4.3 Definição do Leiaute Básico.....</b>	<b>41</b>
<b>4.4.4 Autenticação de Usuários.....</b>	<b>42</b>
<b>4.4.5 Funcionários e Comissões .....</b>	<b>44</b>
<b>4.4.6 Validações e Consultas à Base de Dados .....</b>	<b>48</b>
<b>4.4.7 Documentos .....</b>	<b>49</b>
<b>4.4.8 Quitações .....</b>	<b>56</b>
<b>5 CONCLUSÃO.....</b>	<b>59</b>
<b>REFERÊNCIAS.....</b>	<b>61</b>

## 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 pré-

definidas 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.

Salões de beleza de médio e grande porte possuem uma grande movimentação de pessoas, sendo necessário um software para agendamentos e acompanhamento de serviços. Um sistema para *web* para esse segmento de mercado facilita o acesso pelo cliente e também pelos proprietários e administradores.

Os salões de beleza trabalham com agendamento de serviços para clientes. Para o gerenciamento da agenda é necessário um controle de horário para cada funcionário que realiza serviços no salão. Em sendo o sistema de gerenciamento do salão para *web*, torna-se possível o envio de mensagens para o cliente via *Short Message Service* (SMS) e/ou e-mail para lembrá-lo de serviços agendados, assim como notificá-lo de promoções, eventos ou outros tipos de mensagens do interesse da empresa.

O sistema desenvolvido como resultado deste trabalho é um aplicativo *web* para salões de beleza. 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 serviços, acompanhamento de execução e registro de pagamento de atividades de um salão de beleza.

### 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 serviços em salões de beleza.
- 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 aplicativos *web*.

### 1.3 JUSTIFICATIVA

O desenvolvimento de aplicações *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 compõem o Rails são, entre outros, *ActionMailer* (envio de e-mails), *ActionPack* (geração de estrutura *Model-View-Controller* (MVC)), *ActionRecord* (manipulação de banco de dados), *ActiveSupport* (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<sup>1</sup>. 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.

---

<sup>1</sup> [www.opensource.org/licenses/mit-license.php](http://www.opensource.org/licenses/mit-license.php)

## 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.

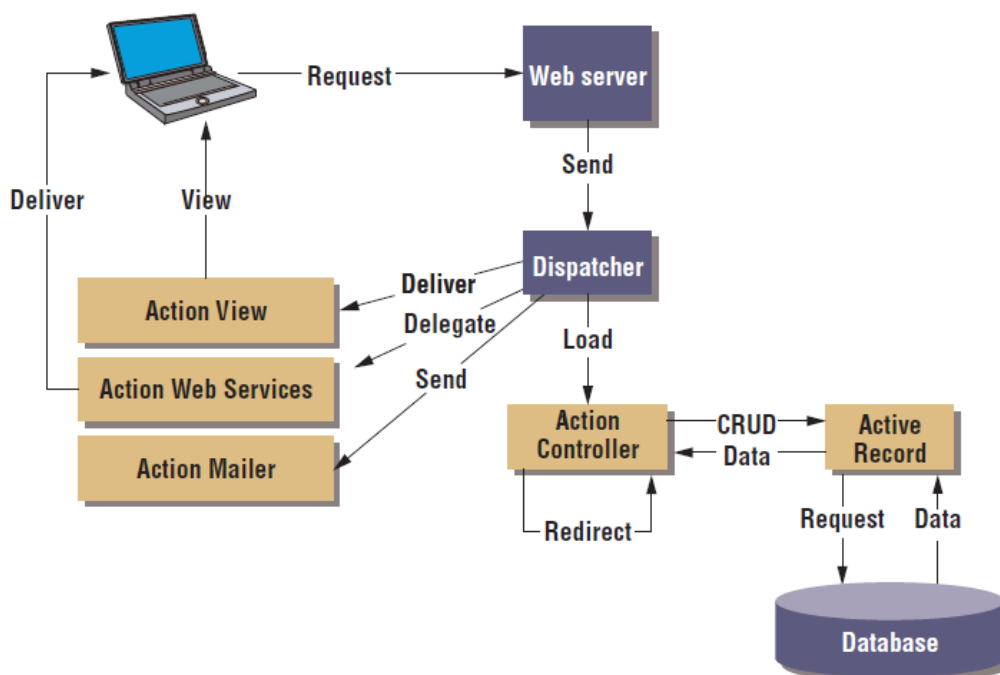


Figura 1 – Componentes do *framework* Ruby on Rails

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 Yourself*, 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, conseqüentemente, 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	Disponibilização/referência	Aplicação no projeto
Astah Professional	12.0	<a href="http://astah.net/download">http://astah.net/download</a>	Para documentação da modelagem do sistema.
Ruby	2.2.3	<a href="https://www.ruby-lang.org/pt/downloads/">https://www.ruby-lang.org/pt/downloads/</a>	Linguagem de programação.
Ruby on Rails	4.2.4	<a href="http://rubyonrails.org/download/">http://rubyonrails.org/download/</a>	Framework <i>web</i> de desenvolvimento
HTML	5	<a href="http://www.w3.org">http://www.w3.org</a>	Para o desenvolvimento da interface <i>web</i> .
CSS	3	<a href="http://www.w3.org">http://www.w3.org</a>	Para o desenvolvimento do leiaute da interface <i>web</i> .
Jquery	1.11.0	<a href="http://jquery.com/download">http://jquery.com/download</a>	Na implementação da interface <i>web</i> .
PostgreSQL	9.4.5	<a href="http://www.pgadmin.org/download/">http://www.pgadmin.org/download/</a>	Sistema gerenciador de Banco de dados.
PgAdmin 3	1.20.0	<a href="http://www.pgadmin.org/download/">http://www.pgadmin.org/download/</a>	Interface para administração do PostgreSQL
Virtual box	5.0.8	<a href="https://www.virtualbox.org/wiki/Downloads">https://www.virtualbox.org/wiki/Downloads</a>	Virtualização do ambiente de desenvolvimento
Vagrant	1.7.4	<a href="https://www.vagrantup.com/downloads.html">https://www.vagrantup.com/downloads.html</a>	Gerenciador do ambiente de desenvolvimento virtualizado
Git	2.6.2	<a href="https://git-scm.com/downloads">https://git-scm.com/downloads</a>	Sistema de controle de Versão
Putty	0.64	<a href="http://www.putty.org/">http://www.putty.org/</a>	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 Professional. 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 salões de beleza. Visitas foram realizadas em alguns desses estabelecimentos da cidade de Pato Branco. 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 estoque dos produtos. O cotidiano e a rotina desses estabelecimentos 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 salões de beleza fornecendo as funcionalidades comuns a esse tipo de negócio, incluindo o processo de agendamento de serviços com profissionais específicos, gerenciamento de fluxo de caixa e controle de estoque. O sistema permitirá definir retribuições personalizadas para cada tipo de serviço, de acordo com o profissional que o realiza.

Para facilitar o desenvolvimento e a definição dos requisitos o sistema foi dividido conceitualmente em módulos. O primeiro módulo é o de agendamento de serviços, que consiste em gerenciar o cadastro de clientes, usuários, funcionários, atividades, agenda e notificações. O segundo módulo é o gerenciador de prestação de serviços, retribuições, fluxo de caixa e controle de estoque.

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

O agendamento de atividades (serviços) permitirá que o usuário selecione um funcionário para executar o referido serviço, porém não necessariamente será seu realizador, podendo lançar o documento de venda com o serviço realizado por outro colaborador.

Nos salões de beleza de médio e grande porte é comum definir uma porcentagem de retribuição (comumente conhecida por comissão) para o realizador das atividades. Nesse caso, é possível que cada colaborador negocie sua parcela de recebimentos, sendo assim, para cada funcionário deve ser possível definir uma porcentagem de retribuição específica para cada atividade, quando este recurso não se fizer necessário, o sistema deve utilizar valores definidos por padrão pelo administrador no cadastro de atividades.

É comum que salões de beleza incluam o custo dos produtos utilizados no serviço realizado. Porém, o sistema deverá prover uma maneira de especificar no documento de venda um produto utilizado. Para isso, é necessário manter um controle de estoque, registrando entradas e saídas de produtos para que o saldo de estoque seja atualizado.

Para o desenvolvimento de fluxo de caixa serão consideradas apenas movimentações de crédito e débito. O registro de contas a receber e a pagar ocorrerá por meio de um documento, visto que ambas as movimentações possuem o mesmo processo de quitação. O sistema possibilitará parcelar um documento, porém isso não restringirá o recebimento fora do valor das parcelas. O objetivo é permitir que o valor do documento seja recebido de diversas formas diferentes, dentre elas, boletos, cheques, transferência bancária e dinheiro. Assim como qualquer prestação de serviços ou venda de produtos, é possível que em alguns casos o consumidor requisite reembolso, independentemente do motivo. Para esses casos, a opção de estorno de recebimentos estará disponível.

## 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.



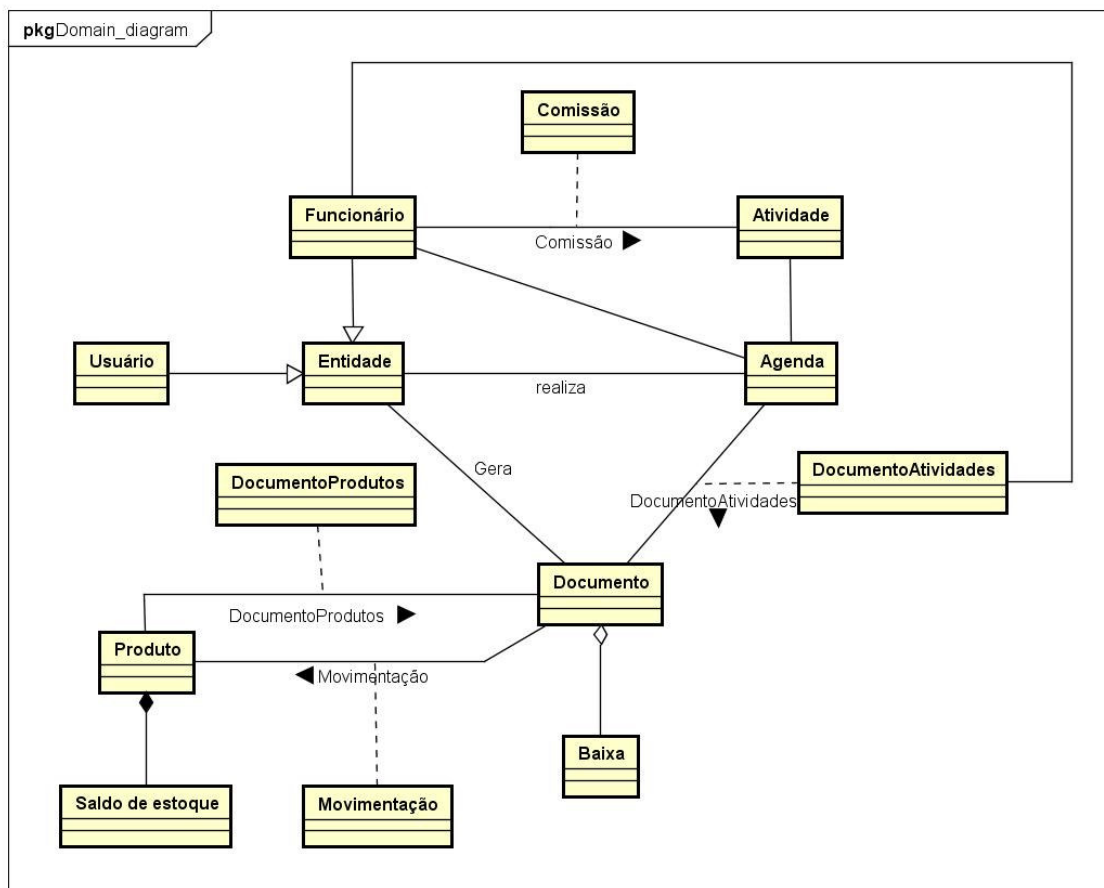


Figura 2 – Diagrama de domínio

Funcionários e usuários são entidades. Usuários podem registrar um agendamento de uma atividade com um funcionário para uma entidade (cliente). O cliente (entidade) pode ser notificado via SMS ou e-mail da atividade agendada. Quando um serviço é efetivamente realizado, um usuário registra um documento de venda que agrega atividades e o funcionário que a realizou. Documentos podem ou não agregar produtos utilizados. As atividades que cada funcionário realiza podem render retribuição, definida por um usuário administrador. Um documento de compra pode ser emitido para registrar um passivo. Quando um documento possui itens, o sistema automaticamente registra a movimentação de estoque e atualiza o saldo de cada produto. Ao receber ou realizar um pagamento os usuários registram a baixa do valor recebido/pago para o documento. Cada documento pode gerar baixa com *status* de pago, cancelado, estornado ou vencido.

#### 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 Funcionário	Um funcionário será vinculado a um cadastro de entidade. Um funcionário possui uma função e retribuições específicas para cada atividade.
RF04	Cadastrar Atividades	Atividades possuem uma descrição, tempo estimado para realização e valor a ser cobrado. Apenas usuários administradores podem cadastrar atividades.
RF05	Definir retribuições	Administradores podem definir no cadastro de um funcionário as retribuições em porcentagem que será recebida e valores que cada realização de atividade custará.
RF06	Cadastrar Agendamento	Uma atividade pode ser agendada para ser realizada por um funcionário para uma entidade. Apenas usuários podem registrar um agendamento.
RF08	Registrar Documento	Quando uma ou mais atividades são realizadas para uma entidade, um usuário deve registrar sua execução informando o funcionário que a realizou e os produtos utilizados.
RF09	Registrar Baixa	Após o registro de um documento de execução de serviço, um usuário registra o recebimento de todo ou parte do valor definido no documento.
RF11	Cadastrar Produto	Os dados do produto poderão ser informados para serem utilizados em atividades, referenciando-os em documentos.
RF12	Registrar movimentação de estoque	Usuários poderão informar entrada ou saída no estoque de produtos, informando a quantidade para cada produto movimentado.
RF13	Emitir relatório de documentos	Emitir um relatório de documentos por período de datas e por funcionário.
RF14	Emitir relatório de contas a receber	Emitir um relatório com os documentos cujo valor recebido não corresponde ao valor do documento.

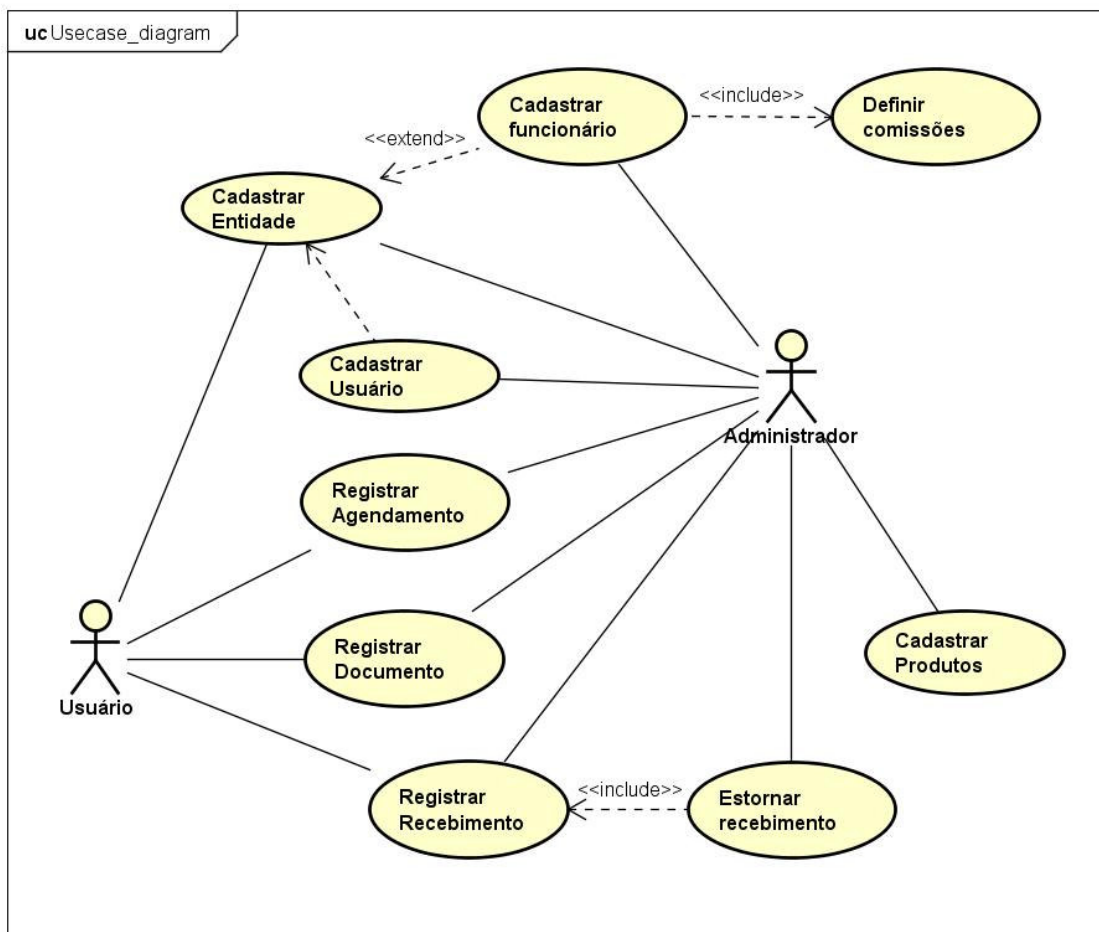
**Quadro 2 – Requisitos funcionais**

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.

<b>Identificação</b>	<b>Nome</b>	<b>Tipo</b>	<b>Descrição</b>
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 e do funcionário que realizou a atividade, automaticamente adicionando a atividade agendada com o valor definido como retribuição para o funcionário que a realizou se esta estiver definida.
RNF06	Valor do documento	Integridade	O valor do documento é calculado com base no valor das atividades e dos itens que o compõe. O usuário tem a opção de especificar um valor de desconto que será subtraído do valor do documento. O valor de desconto não pode ser maior que o valor do documento.
RNF07	Valor da baixa	Integridade	O valor somado de todas as baixas do documento não podem ser maiores que o valor do documento.
RNF08	Movimentação de estoque	Integridade	Quando itens são registrados para um documento de entrada, o sistema automaticamente registra uma movimentação com quantidade positiva, se o documento for de saída, a movimentação é registrada com quantidade negativa
RNF09	Saldo de estoque	Integridade	Sempre que uma movimentação de estoque for registrada, o saldo de estoque do produto é atualizado, somando o saldo atual com o valor da movimentação.

**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 salão de beleza pelos atores. Todos os atores são tipos de usuários, sendo eles: administrador e atendente. Atendentes registram agendamentos, documentos e recebimentos. Administradores têm acesso a todos os recursos que o sistema oferece, incluindo as funcionalidades disponíveis para atendentes além das funcionalidades de gestão financeira e de usuários, estornar recebimento, emitir relatórios, definir retribuições para funcionários, cadastro de atividades e produtos.



Quadro 3 – Diagrama de casos de uso

Os casos de uso “Cadastrar funcionário” 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 funcionário” é possível registrar as retribuições que o funcionário receberá ao realizar atividades. No Quadro 5, este caso de uso está descrito com maiores detalhes.

**Caso de uso:**

Cadastrar Funcionário

**Descrição:**

O administrador registra uma entidade cadastrada como funcionário.

**Evento Iniciador:**

Cadastro de entidade

**Atores:**

Usuário Administrador.

**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

<p>2. O sistema apresenta uma lista de entidades cadastradas para seleção.</p> <p>3. Ator seleciona a entidade.</p> <p>4. O ator lança os dados que compõe o cadastro de funcionário.</p> <p>5. Ator confirma os dados inserindo-os no banco de dados.</p> <p>6. Sistema informa que os dados foram incluídos no banco de dados e redireciona para a página de listagem de funcionários.</p> <p><b>Pós-Condição:</b> Dados do funcionário inseridos no banco de dados.</p>	
Nome do fluxo alternativo (extensão)	Descrição
<p>Linha 4: O administrador deseja definir as retribuições que o funcionário receberá para cada atividade</p>	<p>4.1 O Ator adiciona uma retribuição.</p> <p>4.2 O Ator seleciona a atividade que deseja definir a retribuição.</p> <p>4.3 O Ator informa os valores da atividade e seu percentual de retribuição.</p> <p>4.4 Retorna ao passo 4 e o caso de uso prossegue com fluxo normal.</p>
<p>Linha 5: Dados informados são inválidos</p>	<p>5.1 No momento de salvar, o sistema verifica a validade de todos os dados informados, armazenando quais são inválidos;</p> <p>5.2 Retorna para o formulário de cadastro informando cada um dos campos contendo dados inválidos.</p>

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

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

<p><b>Caso de uso:</b> Registrar Agendamento</p> <p><b>Descrição:</b> O usuário registra um agendamento de atividade com um funcionário para uma entidade.</p> <p><b>Evento Iniciador:</b> Contato de um cliente para agendamento de atividade.</p> <p><b>Atores:</b> Usuário Atendente ou Administrador.</p> <p><b>Pré-condição:</b> 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. Um funcionário deve estar registrado no sistema.</p> <p><b>Sequência de Eventos:</b></p> <ol style="list-style-type: none"> <li>1. Ator acessa a página de calendário com todos as atividades agendadas no período exibido.</li> <li>2. O Ator seleciona um horário no calendário.</li> <li>3. O sistema apresenta um formulário com a data e horário selecionado, uma lista de entidades cadastradas e uma lista de funcionários disponíveis para aquele horário.</li> <li>4. O ator seleciona a entidade, funcionário e informações adicionais.</li> <li>5. Ator confirma os dados inserindo-os no banco de dados.</li> <li>6. Sistema informa que os dados foram incluídos no banco de dados e retorna para a página</li> </ol>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

de calendário. <b>Pós-Condição:</b> Dados do funcionário inseridos no banco de dados.	
Nome do fluxo alternativo (extensão)	Descrição
Linha 3: Não há funcionários 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**

O caso de uso “Registrar Documento” é apresentado no Quadro 7

<p><b>Caso de uso:</b> Registrar Documento</p> <p><b>Descrição:</b> O usuário registra um documento que representa uma conta a pagar ou a receber.</p> <p><b>Evento Iniciador:</b> Realização de atividade para um cliente, venda ou aquisição. Consumação de uma atividade agendada</p> <p><b>Atores:</b> Usuário Atendente ou Administrador.</p> <p><b>Pré-condição:</b> 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.</p> <p><b>Sequência de Eventos:</b></p> <ol style="list-style-type: none"> <li>1. O ator acessa a página de registro de novo documento</li> <li>2. O ator seleciona o tipo de documento e a entidade representada.</li> <li>3. O ator adiciona atividades e/ou produtos</li> <li>4. Ator confirma os dados inserindo-os no banco de dados.</li> <li>5. Sistema informa que os dados foram incluídos no banco de dados e redireciona para a página de recebimento, referenciando o documento recém registrado.</li> </ol> <p><b>Pós-Condição:</b> Dados do funcionário inseridos no banco de dados.</p>	
Nome do fluxo alternativo (extensão)	Descrição
Linha 3: Não há produtos em estoque	3.1 O sistema emite uma mensagem solicitando confirmação de movimentação inválida. 3.2 O autor retorna ao passo 3 e o caso de uso prossegue com fluxo normal.
Linha 5: Dados informados são inválidos	4.1 No momento de salvar, o sistema verifica a validade de todos os dados informados, armazenando quais são inválidos;

	4.2 Retorna para o formulário de cadastro informando cada um dos campos contendo dados inválidos.
--	---------------------------------------------------------------------------------------------------

**Quadro 6 – Caso de uso registrar documento**

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 pela 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 sistema apresentando 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.

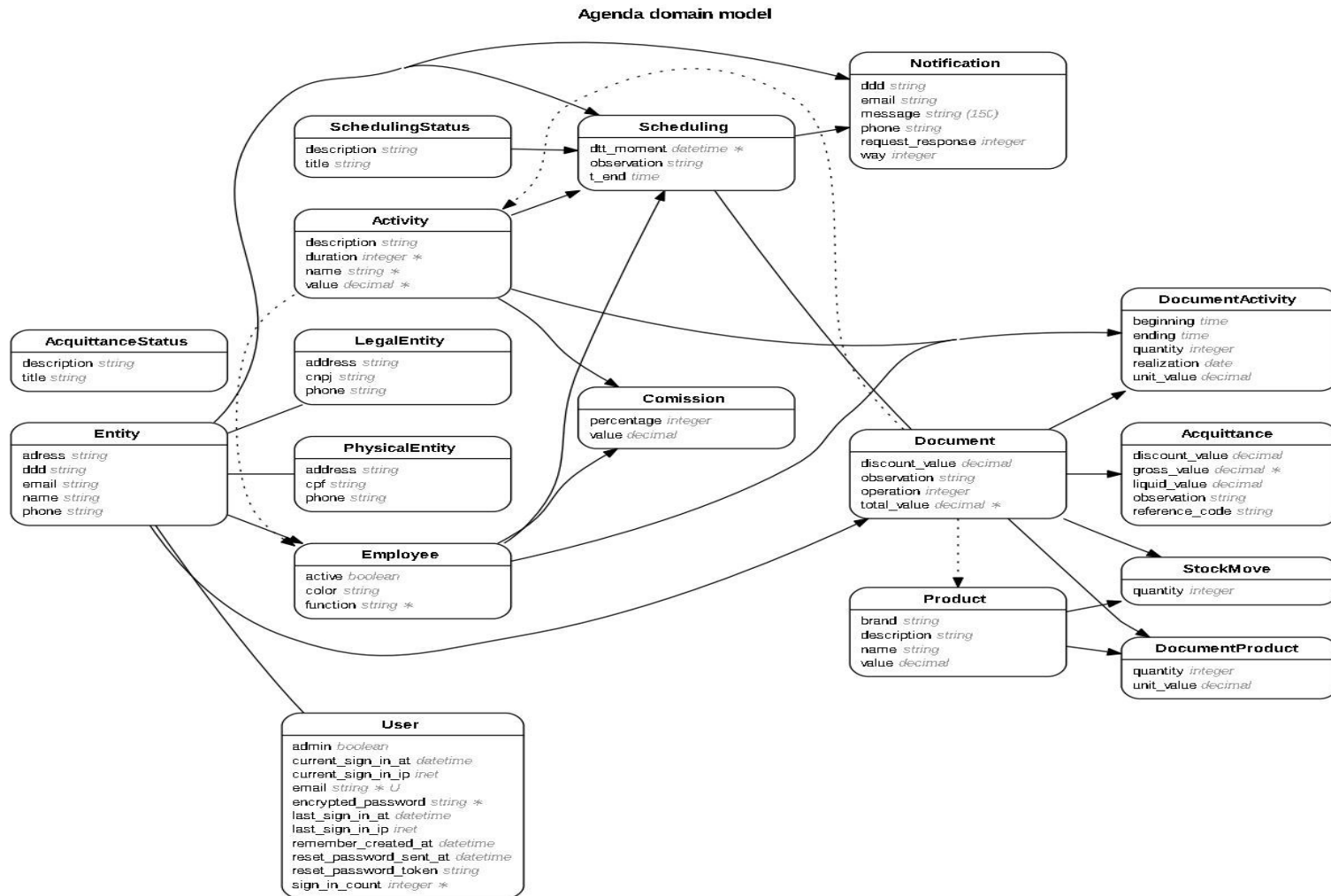


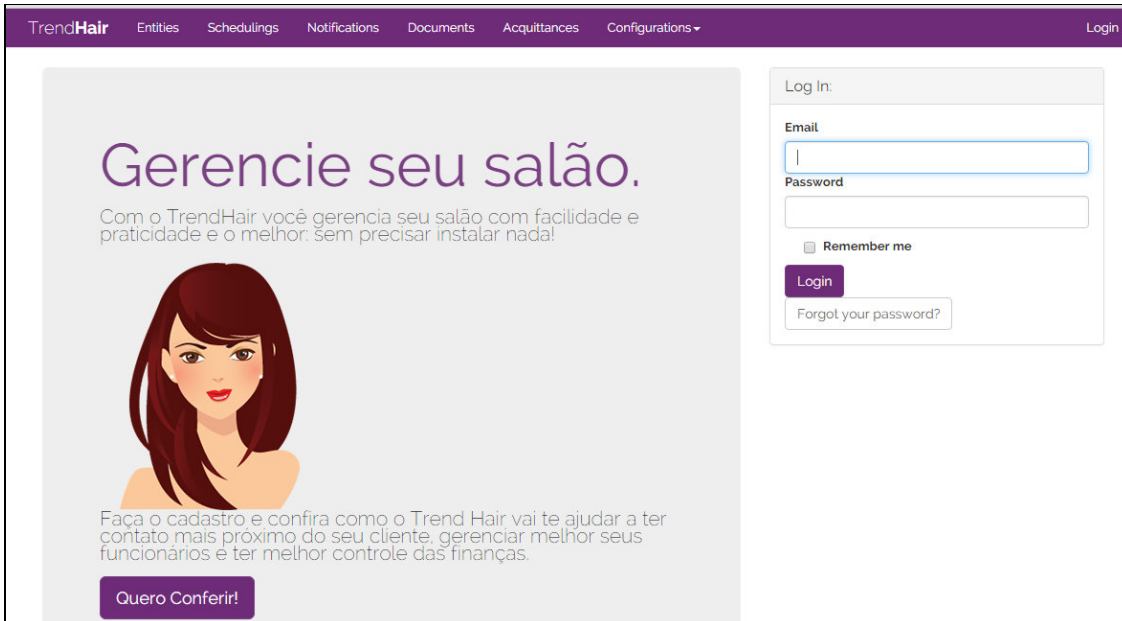
Figura 3 – Diagrama de entidades e relacionamentos gerado pela gem erd



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*.



TrendHair Entities Schedulings Notifications Documents Acquittances Configurations Login

## Gerencie seu salão.

Com o TrendHair você gerencia seu salão com facilidade e praticidade e o melhor: sem precisar instalar nada!



Faça o cadastro e confira como o Trend Hair vai te ajudar a ter contato mais próximo do seu cliente, gerenciar melhor seus funcionários e ter melhor controle das finanças.

Quero Conferir!

Log In:

Email

Password

Remember me

Login

[Forgot your password?](#)

**Figura 4 – Página de login**

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 modificar dados básicos como nome, função, atribuir uma cor para o mesmo e definir os valores que deverão ser cobrados pelo serviço deste funcionário, assim como o percentual de retribuição para tal atividade. A Figura 5 apresenta a *view* de edição de funcionário com o formulário de cadastro visualizado.

The screenshot shows the 'Edit Employee' interface. At the top, there's a navigation bar with 'TrendHair' and various menu items. The main title is 'Edit Employee' with sub-options 'Options', 'New', 'List', and 'Detail'. The form contains several sections:

- Entity:** A dropdown menu showing 'Maico Dal Ponte' and a 'New Entity' button. To the right, 'Id' is 3 and 'E-mail' is 'dalpontemaico@gmail.com'.
- Function:** A dropdown menu showing 'Atividades diversas' and a 'Color' input field.
- Active:** A checked checkbox.
- Commissions:** A table with three rows:
 

Activity	Value	Percentage	Action
Atividade demorada	1352,0	100	Remove
Corte de cabelos	486,35	40	Remove
Fazer barba	50	10	Remove

Buttons for 'Add Comission' (orange) and 'Update Employee' (purple) are located at the bottom of the form.

**Figura 5 – Formulário de cadastro de funcionário**

No Cadastro de agendamentos é possível escolher entre os clientes pré-cadastrados, a atividade que deseja e o profissional que a realizará, agendando o momento de início e final da atividade a fim de que não seja possível criar outro agendamento concomitante. A Figura 6 apresenta a *view* de novo agendamento.

**New Scheduling** Options

**Entity**: Juca Bala

**Activity**: Atividade demorada

**Employee**: Funcionário do mês

**Dtt moment**: 2015 November 27 14:30

**T end**: 15:52

**Observation**: Atividade agendada

**Scheduling status**: Scheduled

**Id**: 4 **E-mail**: juca@balas.com

**Id**: 1 **Value**: 1254.0

**Id**: 1 **Function**: Guerriá Tudo

**Figura 6 – Cadastro de agendamento**

Os usuários podem visualizar os agendamentos na página de lista de agendamentos, exibida na Figura 7. É possível filtrar os agendamentos cadastrados pela data e status facilmente utilizando os botões no topo da página.

**Schedulings** Options

Id	Status	Entity	Activity	Employee	Dtt moment	Actions
2	Scheduled	Tilápia de toca	Atividade demorada	Maico Dal Ponte	2015-11-24 21:00:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
3	Scheduled	Felipe Scrül	Apresentação de TCC	Funcionário dedicado	2015-11-27 14:30:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
4		Maico Dal Ponte	Fzer Unhas	Maico Dal Ponte	2015-11-26 22:06:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
5	Cancelled	Juca bala 2	Apresentação de TCC	Funcionário do mês	2015-11-26 21:24:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
6	Notified	Edilson	Fzer Unhas	Maico Dal Ponte	2015-11-27 13:50:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
7	Cancelled	Bentivi de capacete	Fazer barba	Funcionario dedicado	2015-11-27 18:50:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
8	Consummate	Juca bala 2	Apresentação de TCC	Funcionário do mês	2015-11-27 12:51:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
9	Scheduled	Vinicius	Fazer barba	Funcionário do mês	2015-11-22 16:55:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>
10	Scheduled	Juca Bala	Atividade demorada	Funcionário do mês	2015-11-27 14:30:00 UTC	<input type="button" value="Notify"/> <input type="button" value="Make Document"/> <input type="button" value="Edit"/>

**Figura 7 – Lista de Agendamentos**

Na tela de cadastro de Documentos (operações de compra e venda) apresentada na Figura 8 é possível selecionar Produtos e Serviços, informar qual funcionário efetuou o serviço além de possíveis quantidades e descontos. O valor das atividades é definido com base nas informações de retribuições para cada funcionário. Caso não haja valor de retribuição

definido para o funcionário selecionado, o sistema utilizará o valor registrado no cadastro da atividade.

The screenshot displays the 'New Document' form in the TrendHair system. The form is organized into several sections:

- Header:** TrendHair logo and navigation menu (Entities, Schedulings, Notifications, Documents, Acquittances, Configurations) with a user greeting 'Hello Maico Dal Ponte'.
- Form Fields:**
  - Operation:** Dropdown menu set to 'Sale'.
  - Scheduling:** Input field with value '10'.
  - Scheduling date:** Input field with value '27/11/2015 14:02'.
  - Entity:** Dropdown menu set to 'Juca Bala'.
  - New Entity:** Button.
  - Id:** Input field with value '4'.
  - E-mail:** Input field with value 'Juca@balas.com'.
- Atividades do documento (Activities):**
  - Activity:** Dropdown menu set to 'Atividade demorada'.
  - Employee:** Dropdown menu set to 'Funcionário do mês'.
  - Unit value:** Input field with value '1000.0'.
  - Quantity:** Input field with value '1'.
  - Buttons:** 'Add activity' (orange) and 'Remove' (red).
- Produtos do documento (Products):**
  - Product:** Dropdown menu set to 'Shampoo francês'.
  - Unit value:** Input field with value '570.0'.
  - Quantity:** Input field with value '2'.
  - Buttons:** 'Add product' (orange) and 'Remove' (red).
- Summary:**
  - Total value:** Input field with value '2140'.
  - Discount value:** Input field with value '410'.
  - Final value:** Input field with value '1730'.
- Footer:** 'Create Document' (purple) and 'Cancel' (grey) buttons.

**Figura 8 – Formulário de cadastro de Documento**

Na página para nova quitação (ou recebimento) (Figura 9) o usuário pode registrar uma baixa, ou seja, quitar todo ou parte do valor de um documento. O valor que será considerado como recebimento é o que é informado pelo campo “*Gross value*” (valor bruto), porém, o usuário tem a liberdade de definir um valor de desconto, para qualquer situação que esteja fora do escopo do sistema, como por exemplo, arredondamento de troco ou pagamentos a vista. Apenas para registro, também é possível informar um código de referência, podendo ser um número de boleto ou cheque.

**Figura 9 – Nova quitação (baixa)**

#### 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 agenda --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, “agenda”. 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 'uglifyer', '>= 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 extensão “.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: agenda_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 “Agenda\_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 torná-la 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.erb), cadastro de um novo registro (/app/entities/new.html.erb), edição de um registro (/app/entities/edit.html.erb) e exibição detalhada de um registro (/app/entities/show.html.erb).

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 Bootstrap 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\_id* para 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.

#### 4.4.5 Funcionários e Comissões

De acordo com os requisitos identificados para retribuições, cada funcionário pode receber uma retribuição específica para cada atividade que realiza, assim, o formulário de funcionário foi desenvolvido como mestre-detalle, pelo qual um usuário administrador poderá definir as retribuições para cada atividade. Toda a estrutura básica do código para esses recursos foi gerada utilizando *scaffolding*, como mencionado na seção 4.4.2. A *gem cocoon* foi utilizada para construção dos itens por meio das relações entre os *models Employee, Activity e Comissions*. A Listagem 12 apresenta os códigos fonte dos *models* com suas devidas relações.

```
# /app/models/employee.rb
class Employee < ActiveRecord::Base
  belongs_to :entity
  has_many :comissions
  has_many :activities, through: :comissions
  accepts_nested_attributes_for :comissions, allow_destroy: true
  validates :function, presence: true
  validates :entity, on: :create, presence: true, :uniqueness => true
  validates :entity, presence: true, if: :entity_employee?
  # validates_associated :activities # Verify the method work
  def entity_employee?
    errors.add(:entity_id, "This entity already are a employee") if
self.persisted? and Employee.where("entity_id = #{self.entity_id} and
id != #{self.id}").any?
  end
  def employee_name
    self.entity.name
  end
end
# /app/models/activity.rb
class Activity < ActiveRecord::Base
  has_many :comissions
  has_many :employee, through: :comissions
  validates :name, :value, :duration, presence: true
  def final_value(employee_id = nil)
    if employee_id
      Comission.find_by(employee_id: employee_id).value
    else
      self.value
    end
  end
end
# /app/models/comission.rb
```

```

class Comission < ActiveRecord::Base
  belongs_to :employee
  belongs_to :activity
  validates_presence_of :activity
end

```

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

O método `has_many` explicita a relação de um para muitos entre *employee* e *activity* com *comission*, o parâmetro `through` explicita que a relação ocorre por meio do *model* *comissions*. Dessa forma, o *Active Record* é capaz de definir métodos utilizando as convenções para acesso mútuo entre objetos que compõe essas relações. O método `accepts_nested_attributes_for` define que o *model* em questão é capaz de gerenciar também o recebimento dos atributos de objetos agregados, bastando explicitar no *controller* que tais atributos serão recebidos, como mostra o código apresentado na Listagem 13. Caso contrário, qualquer parâmetro não especificado será ignorado.

```

# /app/controller/employee_controller.rb
class EmployeesController < ApplicationController

  # (...) todos os outros métodos do controller

  def employee_params
    params.require(:employee).permit(:entity_id, :function, :color,
    :active, comissions_attributes: [:id, :activity_id, :value,
    :percentage, :_destroy])
  end

end

```

**Listagem 12 – Aceitar parâmetros de objetos agregados**

A adição de novas retribuições realizada por meio do *cocoon* é feita adicionando campos com os atributos deste *model* no formulário de funcionário. A Listagem 14 apresenta o código fonte da *partial* formulário de funcionário e a *partial* contendo o conteúdo a ser adicionado pelo *cocoon* quando o usuário requisitar uma nova retribuição.

```

# /app/view/employee/_form.html.erb
<%= simple_form_for @employee, :html => { :class => 'form-horizontal' }
do |f| %>
<div class="form-group">
  <div class="col-lg-6">
    <%= f.association :entity, autofocus: true, required: true,
    :value_method => :id, :label_method => :name, :default =>
    @employee.entity ? @employee.entity.id : nil, :input_html => {:id =>
    "entity", :class => "form-control"} %>
    <%= error_span(@employee[:entity_id]) %>
  </div>

  <div class="col-sm-2">
    <%= link_to t('.new_entity', :default =>

```

```

t("helpers.links.new_entity")), new_entity_path, :class => 'btn btn-
default btn-new-resource' %>
</div>
<div class="col-sm-0 col-md-1">
  <%= label_tag "ID", nil, class: '' %>
  <%= label_tag "", nil, id: "entity-id" , class: 'form-control',
:readonly => "readonly" %>
</div>
<div class="col-sm-0 col-md-3">
  <%= label_tag "E-mail", nil, class: '' %>
  <%= label_tag "", nil, id: "entity-info", class: 'form-control',
:readonly => "readonly" %>
</div>
</div>
<div class="form-group">
  <div class="col-lg-6">
    <%= f.input :function, :input_html => {:class => "form-control"} %>
    <%= error_span(@employee[:function]) %>
  </div>
  <div class="col-lg-6">
    <%= f.input :color, :input_html => {:class => "form-control"} %>
    <%= error_span(@employee[:color]) %>
  </div>
</div>
<div class="form-group">
  <div class="col-lg-6">
    <div class="checkbox">
      <%= f.input :active, :input_html => { :checked => true, :class =>
""} %>
      <%= error_span(@employee[:active]) %>
    </div>
  </div>
</div>
<div class="row">
  <div class="col-sm-12">
    <div class="panel panel-default">
      <div class="panel-heading">
        Comissões
      </div>
      <div class="panel-body" id="employee-comissions">
        <%= f.simple_fields_for :comissions do |comission| %>
          <%= render 'comission_fields', :f => comission %>
          <% end %>
        <div class="links col-sm-12">
          <%= link_to_add_association 'Add Comission', f, :comissions,
class: 'btn btn-warning add-button' %>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="row-fluid">
  <div class="col-sm-12">
    <%= f.button :submit, :class => 'btn-primary' %>
  </div>
</div>
<% end %>

```

```

<script>
$(document).ready(function(){
  $('#entity').change(function () {
    var element = $(this);

    $.ajax({
      type: 'GET',
      remote: true,
      url: '/entities/' + element.val(),
      success: function(data){
        $("#entity-id").html(data.id);
        $("#entity-info").html(data.email);
      },
      error: function(data){
        console.log('error: ' + data);
        $("#entity-id").html("");
        $("#entity-info").html("");
      },
      dataType: 'JSON'
    });
  });
  $('#employee-comissions').on('change', '.idactivity', function () {
    var element = $(this);

    $.ajax({
      type: 'GET',
      remote: true,
      url: '/activities/' + element.val(),
      success: function(data){
        console.log(data);
        element.closest('.employee-
comission').find('.percentage').val(10);
        element.closest('.employee-
comission').find('.value').val(data.value);
      },
      error: function(data){
        console.log('error: ' + data);
      },
      dataType: 'JSON'
    });
  });
});
</script>

```

**Listagem 13 – *Partials* para formulário mestre-detalhe pela *gem cocoon***

No carregamento da página o Rails buscará os objetos agregados para exibi-los utilizando o conteúdo da partial *comissions\_fields*. Da mesma forma, o *cocoon* adiciona via JavaScript seu conteúdo quando o usuário solicita por meio do *link* criado pelo método *link\_to\_add\_association*.



#### 4.4.6 Validações e Consultas à Base de Dados

Os usuários podem registrar um agendamento para qualquer pessoa cadastrada no sistema, selecionando o funcionário que o atenderá, a atividade que deseja e o momento em que deseja realizá-la. Não deve ser possível registrar o agendamento de mais de uma atividade simultaneamente para o mesmo funcionário. Portanto, no momento do cadastro, o sistema verifica a existência de um agendamento concomitante, não finalizando o registro caso seja encontrado. A Listagem 15 apresenta o código fonte do *model Scheduling*. Nesse código é possível visualizar a validação do registro.

```
# /app/model/scheduling.rb
class Scheduling < ActiveRecord::Base
  belongs_to :entity
  belongs_to :activity
  belongs_to :employee
  belongs_to :scheduling_status

  has_one :document
  has_many :notifications

  validate :picked_moment, on: :create
  validate :available_moment, on: :update
  validates :entity_id, presence: true
  validates :employee_id, presence: true
  validates :dtt_moment, presence: true, allow_blank: false

  scope :today, ->{where("DATE(dtt_moment) = DATE(NOW())")}
  scope :status, ->(status) {where("scheduling_status_id = ?", status)}
  scope :pending, ->{where("scheduling_status_id != 2 AND
scheduling_status_id != 4")}

  def picked_moment
    errors.add(:dtt_moment, "Date already Picked") if
Scheduling.where("employee_id = ? AND dtt_moment = ?",
self.employee_id, self.dtt_moment).any?
  end
  def available_moment
    errors.add(:dtt_moment, "Date already Picked") if
Scheduling.where("employee_id = ? AND dtt_moment = ? AND id <> ?",
self.employee_id, self.dtt_moment, self.id).any?
  end
end
```

**Listagem 14 – Model Scheduling com validações e consultas ao banco de dados**

O *Active Record* verifica no momento de persistir as informações no banco de dados se o atributo *dtt\_moment* não está nulo ou vazio, chamando em seguida o método *picked\_moment?* para realizar a validação. Utilizando-se de métodos da interface de *query* do *Active Record* em cascata é possível construir uma consulta SQL sem ser necessário criá-la

diretamente. No final da execução, o método *any?* informa se algum registro foi encontrado, adicionando uma descrição do erro para ser retornada automaticamente para a página informando o usuário.

Para facilitar o trabalho do usuário, o sistema disponibiliza uma página que exibe somente os agendamentos para o dia corrente. Para essa funcionalidade a rota de filtro */schedulingstoday* (Listagem 15) foi criada chamando o método *today* de *SchedulingController*. A Listagem 16 apresenta o método que realiza esta ação.

```
# /app/controller/schedulingstoday_controller.rb
class Scheduling < ActiveRecord::Base
  (. . .)
  def today
    @schedulingstoday = Scheduling.today
    @active_alltoday = "active"
    render action: :index end
  (. . .)
end
```

**Listagem 16 – Método para filtragem de agendamentos pelo dia corrente**

Ao receber a requisição da rota, o *model* disponibilizará a consulta definida pelo método *scope* como um método que realiza a busca e retorna todas as ocorrências, armazenando na variável compartilhada *@schedulingstoday*. Seguindo o conceito DRY, seu conteúdo será enviado para a *view index*, uma vez que o conteúdo a ser exibido não é diferente da ação “index”, evitando que uma nova página seja necessária. A variável *@active\_alltoday* é utilizada apenas para marcar o botão deste filtro como ativo na *view*.

#### 4.4.7 Documentos

Um documento pode explicitar uma venda ou uma compra, efetivamente para o sistema não há nenhuma diferença funcional, exceto pelos relatórios e cálculos de caixa. As páginas foram geradas utilizando *Scaffold*. Portanto, todo o seu funcionamento básico ocorre da mesma forma que nos outros *models*. No formulário de registro e edição, para adição dos produtos e serviços a *gem Cocoon* foi utilizada. Quando o usuário seleciona um serviço ou um produto, o sistema automaticamente buscará o valor do mesmo para valorizar os campos relativos a ele. Esse recurso foi desenvolvido utilizando um script *Jquery* que realiza uma requisição via *Ajax* para o *controller* do recurso em questão, simplesmente requisitando o objeto selecionado pelo usuário pelo mesmo método de exibição, que é automaticamente

serializado para leitura no *script*. A Listagem 17 apresenta o código da partial *\_form* do recurso documento.

```
# /app/view/documents/_form.html.erb
<%= simple_form_for @document, :html => { :class => 'form-horizontal',
id: :document_form} do |f| %>
<div class="form-group">
  <div class="col-lg-6">
    <%= f.input :operation, :selected => 1, :input_html => {:id =>
"document", :class => "form-control"} %>
    <%= error_span(@document[:operation]) %>
  </div>
  <div class="col-lg-1">
    <%= f.input :scheduling_id, :input_html => {:class => "form-
control", :readonly => "readonly", :value => @document.scheduling_id}
%>:
    <%= error_span(@document[:scheduling_id]) %>
  </div>
  <div class="col-lg-5">
    <%= label_tag "Scheduling Date", nil, class: '' %>
    <%= label_tag @document.scheduling_date, nil, class: 'form-
control', :readonly => "readonly" %>
  </div>
</div>
<div class="form-group">
  <div class="col-lg-6">
    <%= f.association :entity, autofocus: true, required: true,
:value_method => :id, :label_method => :name, :input_html => {:id =>
"entity", :class => "form-control" } %>
    <%= error_span(@document[:entity_id]) %>
  </div>

  <div class="col-sm-2">
    <%= link_to t('.new_entity', :default =>
t("helpers.links.new_entity")), new_entity_path, :class => 'btn btn-
default btn-new-resource' %>
  </div>
  <div class="col-sm-0 col-md-1">
    <%= label_tag "ID", nil, class: '' %>
    <%= label_tag f.object.entity_id, nil, id: "entity-id", class:
'form-control', :readonly => "readonly" %>
  </div>
  <div class="col-sm-0 col-md-3">
    <%= label_tag "E-mail", nil, class: '' %>
    <%= label_tag f.object.entity.email, nil, id: "entity-info", class:
'form-control', :readonly => "readonly" %>
  </div>
</div>
<div class="row">
  <div class="col-sm-12">
    <div class="panel panel-default">
      <div class="panel-heading">
        Atividades do documento
      </div>
      <div class="panel-body" id="document-activities">
        <%= f.simple_fields_for :document_activities do
```

```

|document_activity| %>
  <%= render 'document_activity_fields', :f => document_activity
%>

  <% end %>
  <div class="links col-sm-12">
    <%= link_to_add_association 'Add activity', f,
:document_activities, class: 'btn btn-warning add-button' %>
  </div>
</div>
<div class="panel-heading">
  Produtos do documento
</div>
<div class="panel-body" id="document-products">
  <%= f.simple_fields_for :document_products do
|document_product| %>
  <%= render 'document_product_fields', :f => document_product %>
  <% end %>
  <div class="links col-sm-12">
    <%= link_to_add_association 'Add product', f,
:document_products, class: 'btn btn-warning add-button' %>
  </div>
</div>
<div class="panel-footer">
  <div class="row">
    <div class="col-lg-4">
      <%= f.input :total_value, :input_html => { :id =>
"document-total-value", value: f.object.total_value || '0', min: 0,
:class => "form-control"}, :readonly => true %>
      <%= error_span(@document[:total_value])%>
    </div>
    <div class="col-lg-4">
      <%= f.input :discount_value, :input_html => { :id =>
"document-discount-value", value: f.object.discount_value || '0', min:
0, :class => "form-control"} %>
      <%= error_span(@document[:discount_value]) %>
    </div>
    <div class="col-lg-4">
      <%= f.input :final_value, :input_html => { :id =>
"document-final-value", value: f.object.final_value, min: 0, :class =>
"form-control"} %>
    </div>
  </div>
</div>
</div>
</div>
<div class="row-fluid">
  <div class="col-sm-12">
    <%= f.button :submit, :class => 'btn-primary' %>
    <%= link_to t('.cancel', :default => t("helpers.links.cancel")),
documents_path, :class => 'btn btn-default' %>
  </div>
</div>
<% end %>

<script>
$(document).ready(function(){

```

```

$('#entity').change(function () {
    var element = $(this);

    $.ajax({
        type: 'GET',
        remote: true,
        url: '/entities/' + element.val(),
        success: function(data){
            $("#entity-id").html(data.id);
            $("#entity-info").html(data.email);
        },
        error: function(data){
            console.log('error: ' + data);
            $("#entity-id").html("");
            $("#entity-info").html("");
        },
        dataType: 'JSON'
    });
});

$('#document-activities').on('change', '.idemployee', function () {
    var element = $(this).closest('.document-activity').find('.idactivity');
    selectActivityValue(element);
});
$('#document-activities').on('change', '.idactivity', function () {
    var element = $(this);

    $.ajax({
        type: 'GET',
        remote: true,
        url: '/activities/' + element.val(),
        success: function(data){
            element.closest('.document-activity').find('.quantity').val(1);
            element.closest('.document-activity').find('.unit_value').val(data.value);

            element.removeData('comissions');
            element.data('comissions', data.comissions);
            element.data('value', data.value);
            selectActivityValue(element);
            calculateValues();
        },
        error: function(data){
            console.log('error: ' + data);
        },
        dataType: 'JSON'
    });
});

$('#document-products').on('change', '.idproduct', function () {
    var element = $(this);
    $.ajax({
        type: 'GET',
        remote: true,
        url: '/products/' + element.val(),
        success: function(data){
            console.log(data);
        }
    });
});

```

```

        element.closest('.document-product').find('.quantity').val(1);
        element.closest('.document-
product').find('.unit_value').val(data.value);
        calculateValues();
    },
    error: function(data){
        console.log('error: ' + data);
    },
    dataType: 'JSON'
});
});

$('#document-discount-value').on('change', function () {
    calculateValues();
});
$('.btn-rm-association').on('click', function () {
    calculateValues();
});
$('#document-activities').bind('DOMSubtreeModified', function () {
    calculateValues();
});
$('#document-products').bind('DOMSubtreeModified', function () {
    calculateValues();
});
$('#document-activities').on('change', '.quantity', function () {
    calculateValues();
});
$('#document-products').on('change', '.quantity', function () {
    calculateValues();
});

function calculateValues(){
    var activitiesList = $("#document-activities .document-
activity").map(function () {
        return this;
    }).get();
    var productsList = $("#document-products .document-
product").map(function () {
        return this;
    }).get();

    var activitiesValue = 0;
    var productsValue = 0;

    if (activitiesList.length > 0) {
        $.each(activitiesList , function (key, item) {
            if ( $(item).is(':visible')) {
                quantity = $(item).find('.quantity').val();
                value = $(item).find('.unit_value').val();

                activitiesValue += value * quantity;
            }
        });
    }
    if (productsList.length > 0) {
        $.each(productsList , function (key, item) {
            if ( $(item).is(':visible')) {

```

```

        quantity = $(item).find('.quantity').val();
        value = $(item).find('.unit_value').val();
        productsValue += value * quantity;
    };
});
};

$('#document-total-value').val(activitiesValue + productsValue);
$('#document-final-value').val((activitiesValue + productsValue) -
$('#document-discount-value').val());
}

function selectActivityValue(activity_element) {
    comissions = activity_element.data('comissions');
    employee_id = parseInt(activity_element.closest('.document-
activity').find('.idemployee').val());
    has = false;
    if (comissions.length > 0 && employee_id > 0) {
        for (var i = comissions.length - 1; i >= 0; i--) {
            if (comissions[i].employee_id == employee_id) {
                activity_element.closest('.document-
activity').find('.unit_value').val(comissions[i].value);
                has = true;
            }
        }
    };
    if (!has) {
        activity_element.closest('.document-
activity').find('.unit_value').val(activity_element.data('value'));
    };
};
}

calculateValues();

});
</script>

```

**Listagem 15 – Partial *\_form* para formulário mestre-detalle de Documento**

A *gem Cocoon* encarrega-se de gerenciar os itens de detalhe para atividades e produtos, enquanto o *script JQuery* no final da página realiza a obtenção dos dados do item selecionado via *controller* e exibe nos componentes *html* da página. A requisição Ajax é realizada na mesma rota utilizada para renderizar a *view* de detalhes do documento, isso porque o *Active Controller* tem a capacidade de identificar a forma com que a requisição foi feita, realizando o retorno com a codificação adequada para leitura via JavaScript, dispensando completamente a necessidade de um método exclusivo no *controller*.

Para impedir que o usuário registre documentos com valor negativo, uma validação foi criada no model *Document*, cujo conteúdo é apresentado pelo código da Listagem 18.

```

# /app/model/document.rb
class Document < ActiveRecord::Base
  belongs_to :entity
  belongs_to :scheduling
  before_create :validate_values

  has_many :document_activities
  has_many :activities, through: :document_activities
  accepts_nested_attributes_for :document_activities, reject_if:
:all_blank, allow_destroy: true
  has_many :document_products
  has_many :products, through: :document_products
  accepts_nested_attributes_for :document_products, reject_if:
:all_blank, allow_destroy: true

  has_many :acquittances

  enum operation: {'Sale' => 1, 'Purchase' => 2}

  validates :entity_id, presence: true
  validates :total_value, presence: true, :numericality =>
{:greater_than => 0 }
  validates :discount_value, :numericality => {:less_than_or_equal_to
=> :total_value }

  default_scope { order(id: :desc) }
  scope :pending, ->{having(
  'SUM(acquittances.gross_value) < (documents.total_value -
documents.discount_value) OR SUM(acquittances.gross_value) IS NULL')}
  scope :payed, ->{having(
  'SUM(acquittances.gross_value) >= (documents.total_value -
documents.discount_value)')}
  scope :with_status, ->{select("documents.*",
  "SUM(acquittances.gross_value) as received_value",
  "COUNT(acquittances.document_id) as quantity").joins(
  "LEFT JOIN acquittances ON acquittances.document_id =
documents.id").group("documents.id")}

  def validate_values
    if not self.total_value > 0
      self.total_value = 0
    end
    if self.discount_value == nil
      self.discount_value = 0;
    elsif self.total_value - self.discount_value < 0
      self.discount_value = self.total_value
    end
  end

  def load_from_scheduling(scheduling)
    self.scheduling = scheduling
    self.scheduling_id = scheduling.id
    self.entity_id = scheduling.entity_id
    self.discount_value = 0
    self.total_value = scheduling.activity.value

    docAct = DocumentActivity.new(

```



```

        activity_id: scheduling.activity_id,
        employee_id: scheduling.employee_id,
        unit_value: scheduling.activity.value,
        quantity: 1
    )

    self.document_activities.append(docAct)
end

def scheduling_date
  if self.scheduling
    self.scheduling.dtt_moment.strftime('%d/%m/%Y %H:%I')
  else
    nil
  end
end

def final_value
  self.total_value - self.discount_value unless self.new_record?
end

def get_pending_value
  self.final_value - self.get_received_value unless self.new_record?
end

def get_received_value
  self.acquittances.sum(:gross_value) unless self.new_record?
end
end

```

#### Listagem 16 – Model para documentos

O método *before create* executará o método *validate\_values* que verifica os valores informados pelo usuário, não permitindo valores negativos. Para esse recurso, duas consultas estão disponíveis por meio de *scope*. A primeira filtra os documentos, trazendo todos os que não possuem seu valor registros em baixas. A segunda obtém todos os registros de documento que possuem todo o seu valor registrado em baixas. O terceiro *scope* traz todos os registros de documentos campos adicionais de total do valor registrado em baixas e a quantidade de baixas registradas. Esses recursos estão disponíveis no menu “venda” e/ou pelas rotas “/documents/pending” e “/documents/payed”, respectivamente.

#### 4.4.8 Quitaões

O registro de pagamento de um documento é realizado pelo lançamento de quitaões. Um documento pode receber diversas quitaões, podendo ser parcelas ou diferentes formas de pagamento. Para registrar as quitaões é necessário selecionar um documento no próprio

formulário do recurso ou utilizar o *link* disponibilizado pela *view* “index” do recurso “documento”, que adiciona à rota de novo recebimento o identificador do documento. Ao recebê-lo, o *controller* carrega suas informações preliminarmente no formulário de quitação, não sendo necessário ao usuário seu preenchimento por completo. A Listagem 19 apresenta o código fonte que realiza o carregamento dos dados do documento para esta funcionalidade.

```
# /app/model/acquittance.rb
class Acquittance < ActiveRecord::Base
  belongs_to :document

  validates :document_id, presence: true
  validates :gross_value, presence: true, :numericality => true
  validates :discount_value, :numericality =>
{:greater_than_or_equal_to => 0}
  validate :valid_value, on: [:create, :update]

  default_scope { order(id: :desc) }
  scope :today, ->{where("DATE(created_at) = DATE(NOW())")}

  def valid_value
    pending_value = self.document.get_pending_value
    received_value = self.document.get_received_value
    if pending_value <= 0 && self.gross_value >= 0
      errors.add(:gross_value, 'This document already was full
paid')
    elsif received_value + self.gross_value < 0
      errors.add(:gross_value, "Can't reverse document with no
acquittances")
    elsif self.gross_value > pending_value
      self.gross_value = pending_value
      self.liquid_value = self.gross_value - self.discount_value
    end
  end

  def load_from_document(document)
    self.document = document
    self.gross_value = document.get_pending_value
    self.discount_value = 0
    self.liquid_value = self.gross_value
  end

  def get_reversed
    acquittance = Acquittance.new
    acquittance.document_id = self.document_id
    acquittance.gross_value = self.gross_value * -1
    acquittance.discount_value = 0
    acquittance.liquid_value = acquittance.gross_value
    acquittance.observation = "Reverse acquittance"
    acquittance.reference_code = self.id
    return acquittance
  end
end
```

**Listagem 17 – Model e método criação do controller de Acquittance**

Ao acessar uma rota com o parâmetro adicional de identificação do documento o *controller* realiza a chamada do método definido no *model* para obter os dados do documento e criar o objeto para recebimento. Para manter a simplicidade no esquema de quitação, o usuário pode informar qualquer valor para recebimento. Em análise posterior, o campo *gross\_amount* será o utilizado como referência para calcular os valores recebidos por cada venda.

## 5 CONCLUSÃO

O objetivo deste trabalho foi implementar um sistema para gerenciamento de um salão de beleza, registrando agendamentos com um funcionário para realização de atividades, registrando sua execução, recebimento e controle de retribuições para cada funcionário. 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 Rails**. 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-na-2a-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://lntelecom.blogspot.com.br/2012/04/banco-de-dados-orientados-objetos.html>>. Acesso em: 11 jan. 2015.
- RAILS GUIDE. **Active record basics**. 2015<sup>a</sup>. Disponível em: <[http://guides.rubyonrails.org/active\\_record\\_basics.html#convention-over-configuration-in-active-record](http://guides.rubyonrails.org/active_record_basics.html#convention-over-configuration-in-active-record)>. Acesso em: 07 nov. 2015.
- RAILS GUIDE. **Migrations**. 2015<sup>b</sup>. Disponível em: <<http://guides.rubyonrails.org/v3.2.21/migrations.html>>. Acesso em: 29 nov. 2015.
- RAILS GUIDE. **Gems**. 2015<sup>c</sup>. 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.