Universidade Federal
do Rio de Janeiro
Escola Politécnica

DESENVOLVIMENTO ÁGIL E MODELO DE NEGÓCIOS VOLTADOS PARA APLICAÇÕES WEB

Ariel Schvartz
Leonardo Gardel Valverde

Projeto de Graduação apresentado ao Curso de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientador: José Ferreira de Rezende

Rio de Janeiro
Setembro de 2013

DESENVOLVIMENTO ÁGIL E MODELO DE NEGÓCIOS VOLTADOS PARA APLICAÇÕES WEB

Ariel Schvartz
Leonardo Gardel Valverde

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Prof. José Ferreira de Rezende, Dr.

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2013

Schvartz, Ariel
Valverde, Leonardo Gardel

Desenvolvimento Ágil e Modelo de Negócios voltados para Aplicações Web/Ariel Schvartz, Leonardo Gardel Valverde. – Rio de Janeiro: UFRJ/POLI – COPPE, 2013.

XIII, 58 p.: il.; 29, 7cm.

Orientador: José Ferreira de Rezende

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Computação e Informação, 2013.

Referências Bibliográficas: p. 55 – 58.

1. Sistemas Web. 2. Ruby on Rails. 3. MVC. 4. TDD. 5. RESTful. I. de Rezende, José Ferreira. II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Computação e Informação. III. Título.

Agradecimentos

- Por Ariel Schvartz

  Agradeço à toda minha família, principalmente meus pais que sempre estiveram presentes e me apoiaram em todos os momentos, tanto bons quanto ruins. Agradeço especialmente ao meu avô que hoje não está mais aqui, porém estaria muito feliz de estar vivendo esse momento ao meu lado. Agradeço a todos os meus amigos e à minha namorada, que me trazem ânimo e felicidade todos os dias. Por fim, agradeço ao Leo, parceiro no trabalho final que certamente será um parceiro para toda a vida.

- Por Leonardo Gardel Valverde

  Agradeço à minha mãe, que sempre esteve presente física e mentalmente em todos os momentos importantes da minha vida. Agradeço ao meu pai e minha irmã, que, apesar do recente distanciamento, me ensinaram muito e encheram meus dias de lembranças e saudade. Agradeço aos meus amigos, que fazem a vida valer a pena. Por fim, agradeço ao meu amigo e companheiro de projeto, Ariel, por sua excelente companhia e seu grande empenho em fazer tudo dar certo.

- Por Ambos

  Agradecemos ao nosso orientador, que é um exemplo como profissional e como pessoa, dedicado e companheiro.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

DESENVOLVIMENTO ÁGIL E MODELO DE NEGÓCIOS VOLTADOS PARA APLICAÇÕES WEB

Ariel Schvartz
Leonardo Gardel Valverde

Setembro/2013

Orientador: José Ferreira de Rezende

Curso: Engenharia de Computação e Informação

O sucesso e rentabilidade de diversos sistemas web instigam o surgimento de novos empreendimentos na área de computação. Porém, é importante saber como fazê-lo diante de uma infinidade de alternativas para o desenvolvimento e planejamento de tais aplicações por suas equipes. Este trabalho visa orientar tais empreendimentos, apresentando conceitos e métodos, que além de modernos, são frequentemente requisitados no mercado de software atual para o desenvolvimento ágil. Por fim, será apresentada uma implementação onde tais conceitos são aplicados, além de um modelo de negócios do projeto.

Palavras-Chave: Sistemas Web, Ruby on Rails, MVC, TDD, RESTful.

Abstract of the Undergraduate Project presented to Poli/COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

AGILE DEVELOPMENT AND BUSINESS MODEL FOR WEB APPLICATIONS

Ariel Schvartz
Leonardo Gardel Valverde

September/2013

Advisor: José Ferreira de Rezende

Course: Computer and Information Engineering

The success and profitability of many web systems instigates the emergence of new ventures in computing. However, it is important to know how to do it, in front of a plethora of alternatives for the development and planning of such applications by their teams. This work aims to guide these ventures, presenting modern concepts and methods that are often required in the software market today for agile development. Lastly, will be presented an implementation where this concepts are applied, aside from a project business model.

Keywords: Web Systems, Ruby on Rails, MVC, TDD, RESTful.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xi**

Lista de Abreviaturas **xii**

1 Introdução **1**

1.1 Motivação 1
1.2 Objetivo 3
1.3 A ideia 4
1.4 Estruturação do documento 5

2 Ferramentas e Metodologias Utilizadas **6**

2.1 RESTful 6
2.2 *Frontend* 7
2.2.1 HTML 7
2.2.2 CSS 8
2.2.3 Javascript 8
2.2.4 AJAX (Asynchronous JavaScript and XML) 8
2.2.5 JQuery 8
2.3 *Backend* 9
2.4 A escolha do Ruby on Rails 9
2.4.1 Arquitetura MVC 10
2.4.2 Sistema de rotas independente 12
2.4.3 Incentivo à programação RESTful 12
2.4.4 Geradores automáticos de código 13
2.4.5 Ferramentas de automatização de design de *frontend* 13
2.4.6 Ferramenta para envio de e-mails 14
2.4.7 A linguagem Ruby 15
2.4.8 Boa abstração do BD 15
2.4.9 Suporte à autenticação e autorização terceirizado 18
2.4.10 Segurança 18

2.4.11 Eficiente . . . 19
2.4.12 Suporte a Internacionalização . . . 20
2.4.13 Incentivo à Implementação de Testes . . . 21
2.5 TDD (Test Driven Development) . . . 22
2.6 Desenvolvimento Colaborativo . . . 23

3 A Implementação 25
3.1 Funcionalidades . . . 25
3.1.1 Cadastro e Sessão de Usuários . . . 27
3.1.2 Integração com Redes Sociais . . . 28
3.1.3 Eventos e Lista de Convidados . . . 28
3.1.4 Lista de Itens . . . 29
3.1.5 Pagamento . . . 30
3.1.6 Amizades . . . 30
3.1.7 Notificações e Alertas . . . 31
3.1.8 Ranqueamento de Eventos . . . 31
3.2 Modelos . . . 32
3.2.1 Usuário . . . 34
3.2.2 Evento . . . 34
3.2.3 Usuário no Evento . . . 34
3.2.4 Notificação . . . 35
3.2.5 Outras Tabelas . . . 35
3.2.6 Extras . . . 36
3.3 Controladores . . . 36
3.4 Visões . . . 37
3.5 Teste . . . 38
3.5.1 Teste dos Modelos . . . 38
3.5.2 Teste dos Controladores . . . 39
3.6 Ambientes . . . 40
3.6.1 Desenvolvimento . . . 40
3.6.2 Produção . . . 41

4 Modelo de Negócios 42
4.1 Plano de Negócios . . . 42
4.1.1 Vantagens . . . 43
4.1.2 Desvantagens . . . 43
4.2 BMC . . . 43
4.2.1 Vantagens . . . 43
4.2.2 Desvantagens . . . 44
4.3 Blocos do BMC . . . 44

4.3.1 Principais Atividades 44
4.3.2 Principais Recursos 44
4.3.3 Rede de Parceiros 45
4.3.4 Proposição de Valor 45
4.3.5 Segmentos de Clientes 45
4.3.6 Canais 46
4.3.7 Relacionamento com o Cliente 46
4.3.8 Estrutura de Custos 46
4.3.9 Fluxos de Receita 46
4.4 Modelo de Negócios do EVNTs 47

5 Conclusão e Trabalhos Futuros **52**
5.1 Conclusão 52
5.2 Trabalhos Futuros 53
5.2.1 Pagamento 53
5.2.2 Segurança 53
5.2.3 Parcerias e Sociedade 53

Referências Bibliográficas **55**

Lista de Figuras

2.1 *Curva de Interesse sobre os frameworks ao longo dos anos.* 10
2.2 *Arquitetura MVC do framework Ruby on Rails.* 11

3.1 *Fluxo de ações do usuário no sistema.* 26
3.2 *Modelo de Dados - Tabelas e Relacionamentos.* 33
3.3 *Página de Visualização de Evento.* 37

4.1 *BMC do projeto* 47

Lista de Tabelas

2.1 Tabela de rotas automáticas para recursos no Ruby on Rails. 12

Lista de Abreviaturas

AJAX	Asynchronous JavaScript and XML, p. 8
BDD	Behavior Driven Development, p. 22
BD	banco de dados, p. 2
BMC	Business Model Canvas, p. 42
CRUD	Create, Retrive, Update, Destroy, p. 7
CSS	Cascading Style Sheets, p. 2
DDoS	Distributed Denial of Service, p. 18
ERB	Embedded Ruby, p. 11
FAQ	*Frequently Asked Questions*, p. 49
HTML	HyperText Markup Language, p. 1
HTTP	Hypertext Transfer Protocol, p. 1
Haml	HTML Abstraction Markup Language, p. 11
IRB	*Interactive Ruby Shell*, p. 40
JSON	JavaScript Object Notation, p. 8
MVC	Model-View-Controller, p. 10
ORM	Object-relational mapping, p. 2
REST	Representational State Transfer, p. 6
SGBD	Sistema de Gerência de Banco de Dados, p. 15
SSL	Secure Sockets Layer, p. 18
TDD	Test Driven Development, p. 22

URI	Uniform Resource Identifier, p. 7
XHR	XML HTTP Request, p. 8
XML	Extensible Markup Language, p. 8
XSS	Cross-Site Scripting, p. 18

Capítulo 1

Introdução

O protocolo HTTP (Hypertext Transfer Protocol), além de linguagens como o HTML (HyperText Markup Language), permitiram o surgimento dos primeiros navegadores web, marcando o início da chamada Web 1.0. Nesta primeira versão, que vigorou ao longo dos anos 90, seu objetivo era simplesmente prover informação, já que o conteúdo presente em sua maioria era do tipo "somente leitura" [1].

Surge então a Web 2.0, por volta de 1999, onde o conteúdo da rede passa a ser largamente preenchido pelos usuários, através de blogs ou redes sociais, por exemplo [2]. Desde então, aplicações web extremamente rentáveis (como o Facebook, que registrou um lucro líquido no segundo trimestre de 2013 de cerca de 333 milhões de dólares [3]) instigam o surgimento de novos empreendimentos. Isso aliado com a crescente difusão dos conhecimentos necessários ao desenvolvimento de aplicações web, tornou o mercado extremamente competitivo. Tal competitividade torna a agilidade característica essencial aos novos empreendedores, na implantação e manutenção de seus negócios.

1.1 Motivação

Para os novos empreendimentos em computação, não faltam oportunidades ou ideias, especialmente no ambiente universitário. Esta área é extremamente favorável a criação destes, pois as possibilidades para a inovação são inúmeras, além do baixo capital inicial demandado por tais iniciativas.

Entretanto, para que consigamos colocar muitas ideias em prática, além de garantir que cada ideia seja desenvolvida em pouco tempo (essencial para garantir competitividade no mercado), os esforços no desenvolvimento das mesmas devem ser reduzidos. Para isso, é extremamente interessante que se utilize um bom *framework* de desenvolvimento.

Utilizar um *framework* é simplesmente reutilizar código desenvolvido por terceiros, porém existe uma grande diferença entre este e as chamadas bibliotecas. Uma

biblioteca é utilizada para uma função específica, complementando o fluxo de controle da sua aplicação, ao passo que os *frameworks* definem este fluxo, mas permitem personalização para se atingir a funcionalidade desejada. Trata-se da chamada inversão de controle. Sendo assim, o uso de *frameworks* é muito interessante para sistemas web, uma vez que o fluxo de controle destes em geral é igual ou muito semelhante [4].

Para que possamos entender melhor os *frameworks* para sistemas web, vamos focar em suas atividades mais gerais. A grande maioria de seus fluxos se baseia, respectivamente, na recepção de requisições HTTP, no acesso ao BD (banco de dados), no tratamento dos dados recebidos pelas requisições ou pelo BD e na resposta ao usuário através de uma página HTML pertinente. Sendo assim, é interessante tornar genérica essa lógica de controle, evitando o retrabalho dos desenvolvedores. Uma vez que a "espinha dorsal" do sistema já está a cargo do *framework*, todo o esforço de implementação fica diretamente relacionado às especificidades do projeto.

Entretanto, a tarefa de escolher um *framework* não é trivial, diante das inúmeras possibilidades existentes. Podemos optar pelo *framework* somente com base nas linguagens a serem utilizadas, mas, a longo prazo, dificilmente esse critério resultará melhor escolha. Para fazer uma boa escolha, é interessante saber as características desejáveis em um *framework* para sistemas web. É interessante que este possua: [5]

- uma abstração interessante como interface com o BD

  O código escrito na aplicação não deve ser específico para o BD a ser utilizado. Isso requer uma interface de acesso ao mesmo que abstraia as suas especificidades, permitindo que um mesmo código seja utilizado para acesso a diversos tipos de bancos de dados. Também é muito interessante a utilização de ORM (Object-relational mapping) para tornar mais amigável o manuseio de dados do BD. Além disso também é importante que o *framework* permita a validação dos dados a serem persistidos de forma manutenível.

- ferramentas facilitadoras para a construção de páginas web

  É interessante que o *framework* facilite a criação de páginas web, de forma que seja necessário pouco ou nenhum entendimento sobre linguagens como HTML ou CSS (Cascading Style Sheets). Para isso, devem ser fornecidas funções que automatizem a criação do código de estilo.

- um sistema eficaz de autenticação

  Diversos sites permitem aos usuários o acesso de forma anônima, mas boa parte deles requer autenticação. Por ser uma necessidade comum a muitos sistemas, deve ser oferecida pelo *framework* de forma opcional. Dentro deste tópico estão

incluídas diversas funcionalidades, como a criação de contas, recuperação das mesmas, abertura de sessão, entre outras.

- segurança

  Autenticação e autorização andam sempre lado a lado. Após autenticado, é necessário ter o controle sobre as operações (como leitura ou escrita) realizadas pelos usuários. O conjunto formado por usuário, informação e operação deve ser autorizado antes da ação ser executada. Uma falha de segurança pode ser, por exemplo, permitir que um usuário altere dados de algum outro, ou que acesse dados confidenciais.

- eficiência

  As operações realizadas pelo *framework* devem ser otimizadas, especialmente no acesso aos dados do BD e no carregamento das páginas web, visando um menor tempo total de resposta do servidor. Uma das maneiras de alcançar este objetivo é permitir que os dados sejam armazenados em cache.

- método simples para criação e utilização de bibliotecas

  Para o desenvolvimento ágil, é essencial a reutilização de código. Sendo assim, é importante que a inclusão ou criação de bibliotecas não seja uma operação custosa.

- uma grande comunidade que o utiliza

  Quanto mais pessoas utilizando o *framework*, melhor. Isso faz com que as bibliotecas sejam melhor revisadas e, consequentemente, atualizadas. Além disso, a busca por informação é facilitada, uma vez que a quantidade de informação disponível a respeito de um *framework*, em geral, é proporcional ao tamanho da sua comunidade.

É importante não se ater apenas às questões relativas à implementação. Um empreendimento perfeitamente implementado, porém com baixa aceitação por seu público ou pouco rentável, representa apenas tempo e dinheiro perdidos. Para que isso não aconteça, é necessário avaliar o seu empreendimento enquanto negócio, preferencialmente antes mesmo de ser escrita a primeira linha de código.

1.2 Objetivo

O objetivo deste trabalho é apresentar um método moderno e ágil para planejamento e implementação de sistemas web. Entretanto, não terá um cunho puramente

teórico. A partir de uma ideia, detalhada na próxima seção, será realizada a implementação (apresentada no capítulo 3) e o planejamento do negócio (apresentado no capítulo 4), até que um protótipo seja desenvolvido.

1.3 A ideia

Dado que será implementada uma aplicação web real, que seja algo interessante e possivelmente rentável, sendo assim passível de planejamento enquanto negócio. Observando alguns dos aplicativos web existentes para a gerência de eventos, isto é, aqueles que auxiliam os seus usuários a administrar e planejar de forma organizada eventos (como shows, festas, entre outros), é possível notar que nenhum possui todas as funcionalidades demandadas por seus usuários, obrigando-os a recorrer a sistemas complementares. Surgiu então a ideia de criar uma plataforma que auxilie de forma plena seus usuários nesse tipo de demanda.

Para avaliar como e porque não existe uma plataforma completa com esta finalidade, vamos refletir sobre os tipos de eventos possíveis. Sabemos que, quanto à privacidade, os eventos podem ser abertos (públicos) ou fechados (privados), porém, quanto ao pagamento, um evento pode ser:

- fixo, onde são vendidos ingressos a preços predefinidos (ou gratuito)

  Esses eventos são implementados em diversos aplicativos web, como o Facebook, Lista Amiga ou Eventick, porém os dois primeiros não possuem uma funcionalidade de pagamento. Neste caso, em geral os usuários informam que o pagamento deve ser realizado através de outros sites, como o Ingresso Rápido ou o Ingresso Certo. O Eventick implementa um sistema de pagamento e possui integração com redes sociais, no entanto os usuários não podem ser convidados para um evento, eles precisam se inscrever voluntariamente para fazer parte dos mesmos.

- compartilhado, onde o custo total é dividido igualmente entre os convidados

  Existem alguns aplicativos simples e que dão suporte a esse tipo de eventos, porém funcionam apenas localmente, como calculadoras.

- colaborativo, onde ocorre a divisão de responsabilidades sobre itens

  Neste tipo de evento, o organizador define uma lista de itens a serem levados, com suas respectivas quantidades. Cada participante se responsabiliza por levar um ou mais itens no dia do evento. É importante que o valor total dos itens a serem levados por cada participante seja aproximadamente igual.

Seria interessante uma ferramenta que auxiliasse neste tipo de evento por ser de difícil gerenciamento, mas não é implementado por nenhum aplicativo conhecido.

Além de permitir a criação e acesso à eventos, é interessante que se possa buscar por eventos de forma eficiente. Diversos algoritmos de busca podem ser implementados para realizar um bom ranqueamento dos eventos. Esses algoritmos podem ser baseados em relacionamentos (como o Facebook), em popularidade (como o Lista Amiga) ou ate por geolocalização.

Nossa ideia é reunir a interação social do Facebook, o bom ranqueamento do Lista Amiga e o sistema de pagamento do Eventick, para construir um sistema único e auto suficiente. Serão implementadas também outras funcionalidades como: eventos colaborativos, eventos compartilhados, notificações por e-mail, entre outros.

1.4 Estruturação do documento

No capítulo 2, veremos as metodologias utilizadas no desenvolvimento do *software* e os motivos de sua utilização. Em seguida, o capítulo 3 detalha a implementação através das metodologias elucidadas. O capítulo 4 apresenta o modelo de negócios desenvolvido, enquanto o capítulo 5 visa concluír o trabalho e apontar melhorias necessárias em trabalhos futuros.

Capítulo 2

Ferramentas e Metodologias Utilizadas

Obviamente, assim como todas as outras aplicações web, foi utilizado o protocolo HTTP para comunicação entre cliente e servidor. Essa comunicação se inicia quando o cliente (no caso, o navegador do usuário) realiza uma requisição HTTP. Ao receber a requisição, o servidor web a processa, gera uma resposta pertinente e a devolve ao cliente [6].

O código de aplicações web é dividido em duas principais categorias: o *frontend*, responsável pela interação com o usuário, e o *backend*, responsável por todo o processamento realizado no servidor. A compreensão desta distinção é essencial, pois existem linguagens e metodologias específicas, tanto para o desenvolvimento de cada uma delas, como para o planejamento da comunicação entre esses dois mundos. Mais adiante, as duas categorias serão explicadas em mais detalhes.

2.1 RESTful

O protocolo HTTP foi desenvolvido em paralelo com a arquitetura REST (Representational State Transfer), que serve de molde para o desenvolvimento de aplicativos web, pois ilustra como eles devem se comportar [7]. Uma aplicação que é desenvolvida segundo a arquitetura REST, é dita RESTful. Seguindo esta arquitetura no desenvolvimento da aplicação, uma série de vantagens são obtidas, como uma melhora na escalabilidade da mesma. Isto ocorre pois esta arquitetura segue uma lógica *stateless*, isto é, o servidor não armazena estados, sendo necessária a transmissão de todas as informações inerentes ao processamento das requisições por meio das mesmas.

Para que a aplicação seja RESTful, devemos nos preocupar com alguns detalhes. É importante que cada recurso seja representado por uma única URI (Uniform

Resource Identifier), mas para que isso seja possível, é necessário explorar ao máximo os seus métodos. Como a maioria dos recursos precisam ser relacionados a cada uma das ações do CRUD (Create, Retrive, Update, Destroy), é interessante que tais ações sejam relacionadas diretamente com os métodos do protocolo HTTP, visando manter inviolável a relação entre a URI e o recurso. Sendo assim, os quatro principais métodos do protocolo HTTP, que serão invocados no *frontend*, devem ser relacionados aos quatro métodos do CRUD, que serão invocados no *backend*. Em geral essa relação ocorre da seguinte forma:

- C (Create) - método POST
- R (Retrive) - método GET
- U (Update) - método PUT
- D (Destroy) - método DELETE

2.2 *Frontend*

O *frontend* da aplicação é todo o código encarregado de realizar a interação com o usuário. No caso, trata-se de tudo que será enviado ao navegador. Esse código tem a finalidade de exibir de forma inteligível e interessante o resultado da requisição tratada pelo *backend*, além de lidar com as entradas de dados (*inputs*) do usuário. Para a construção do *frontend* utilizamos algumas linguagens e bibliotecas, que serão elucidadas a seguir.

2.2.1 HTML

O HTML é a linguagem base do *frontend*, pois as demais operam sobre seus elementos. Ela torna possível a interpretação das páginas web pelos navegadores. Esta linguagem sofreu algumas atualizações, porém, após mais de dez anos sem novas versões, ficou bastante defasada. Com os avanços da Internet, passaram a ser utilizados inúmeros *plug-ins* para que páginas web fossem capazes de realizar algumas atividades essenciais, como reprodução de vídeos ou de áudio.

Esse atraso se prolongou até 2008, quando foi publicada uma nova versão, o HTML5, que ainda não esta completamente em vigor, porém cada vez mais em uso [8]. Essa nova versão dá suporte a diversas funcionalidades sem que sejam necessários *plug-ins*. Dentre as funcionalidades implementadas estão a reprodução de vídeos , áudios, o *drag and drop* e a exibição de imagens vetorizadas [9].

2.2.2 CSS

O CSS é uma linguagem criada para estilização de componentes HTML. Podemos dizer que os navegadores interpretam o CSS como adjetivos dos componentes HTML. É uma linguagem voltada para o "como exibir", enquanto o HTML se preocupa com "o que exibir". O CSS3, sua versão mais recente que trouxe diversas novas funcionalidades, vem sendo cada vez mais utilizado.

2.2.3 Javascript

O Javascript é uma linguagem de programação que, apesar de ser utilizada no desenvolvimento do *frontend* da aplicação, se assemelha às linguagens de *backend*, pela liberdade oferecida ao programador. É utilizada para dar interatividade às páginas web, visto que, apesar de essenciais, o HTML e o CSS não fornecem muita dinamicidade à mesma. Através da utilização de Javascript é possível alterar de forma livre o conjunto composto pelo HTML e CSS, mesmo após o carregamento completo da página. Inúmeras bibliotecas são construídas através da linguagem Javascript. Muitas delas também são utilizadas no *frontend* de aplicações web, como a JQuery.

2.2.4 AJAX (Asynchronous JavaScript and XML)

O AJAX é um conjunto de técnicas que permite a criação de requisições assíncronas em páginas web. A sua utilização é interessante quando é necessária a troca de dados sem que seja preciso recarregar a página por inteiro. Resumidamente, uma implementação de AJAX é um trecho de código em Javascript, que faz com que seja realizada uma requisição XHR (XML HTTP Request), que será tratada e respondida pelo servidor. Essa resposta, que em geral é no formato XML (Extensible Markup Language) ou JSON (JavaScript Object Notation), é recebida pelo Javascript, podendo ser utilizada para atualizar elementos.

2.2.5 JQuery

A JQuery é talvez a biblioteca mais utilizada em Javascript no desenvolvimento de aplicações web. Diversas funcionalidades interessantes são implementadas por esta biblioteca, dentre elas, uma interface amigável para customização de requisições AJAX. É muito utilizada também para criar animações, como a inserção de elementos na tela de forma gradual.

2.3 *Backend*

O *backend* da aplicação é todo o código responsável pelo processamento da requisição até o momento da geração da resposta. Para exemplificar, pode-se pensar em uma requisição de abertura de sessão (*login*) por meio da aplicação. Neste caso, o *backend* é o código responsável por atividades como a verificação da existência do usuário ou a validação da senha inserida pelo mesmo. O *backend* determina também qual o tipo da resposta que será devolvida ao usuário (falha ou sucesso na abertura de sessão, segundo o exemplo). Observando por outro prisma, o *backend* pode ser encarado como o código responsável pela tomada de decisão.

O *backend* pode ser implementado em inúmeras linguagens. Como é muito comum e recomendada a utilização de *frameworks*, as linguagens relativas a estes acabam sendo as mais utilizadas. Linguagens como o SQL também são utilizadas no *backend* da aplicação, já que o BD também o compõe.

Existem diversos *frameworks* nas mais variadas linguagens, porém o escolhido para este trabalho foi o Ruby on Rails. A seguir serão explicados com detalhes os motivos da escolha, através de funcionalidades deste *framework*.

2.4 A escolha do Ruby on Rails

O desenvolvimento ágil de aplicações web teve início com o surgimento do Ruby on Rails. Esta agilidade na implementação se deve a uma série de princípios adotados no desenvolvimento deste *framework*, como o DRY (Don't Repeat Yourself) e o CoC (Convention over Configuration) [10]. O DRY é um princípio de desenvolvimento de software que introduz a ideia de que uma mesma informação não deve ser repetida mais de uma vez ao longo do código. Este princípio simplifica o trabalho do desenvolvedor, evitando redundâncias e os problemas causados por elas. O CoC implica em adotar por padrão uma mesma configuração. Caso o desenvolvedor queira fugir do padrão, deve escrever código para isso, porém não deve escrever uma única linha para seguir os padrões.

Por ser pioneiro em desenvolvimento ágil de aplicações web, o Ruby on Rails teve o seu grande momento logo após o seu lançamento, e, por volta de 2006, atingiu o seu pico de interesse. Até hoje existe uma grande comunidade que o utiliza, porém, após o surgimento de diversos outros *frameworks* para desenvolvimento ágil, ocorreu significativa queda no interesse. Estabilizou-se então dentre os *frameworks* mais utilizados, como podemos notar através da figura 2.1, que indica a curva de interesse sobre os *frameworks* ao longo dos anos.

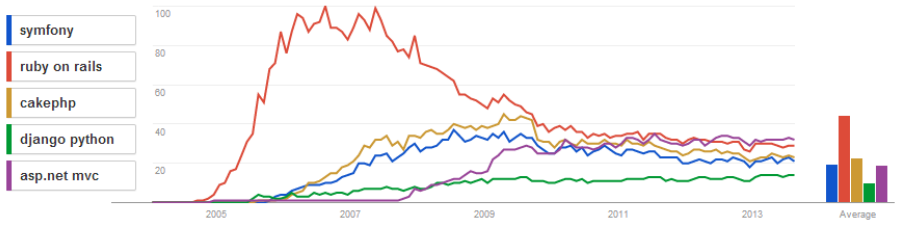

Figura 2.1: Curva de Interesse sobre os *frameworks* ao longo dos anos [11].

O Ruby on Rails é um *framework* dito *full-stack*, por ser utilizado obrigatoriamente por inteiro (ao contrário de outros *frameworks*, onde seus módulos podem ser utilizados separadamente, como o Zend). Possui uma arquitetura MVC (Model-View-Controller), muito comum em diversos *frameworks*, que será melhor detalhada a seguir. Esta arquitetura aplicada em um *framework full-stack* e aliada a conceitos como o DRY e o CoC, permite que em pouquíssimas linhas de código, enormes funcionalidades sejam implementadas, como veremos mais a frente.

2.4.1 Arquitetura MVC

Esta arquitetura realiza uma separação entre a parte que interage com o usuário (a visão) e a informação (o modelo), permitindo que esta seja acessada por meio de métodos em classes específicas (os controladores). Esta forma modularizada de desenvolvimento provê um excelente isolamento entre essas camadas com funções tão distintas, viabilizando a reutilização de código e tornando o mesmo mais legível, e, consequentemente, manutenível.

A figura 2.2 apresenta a arquitetura MVC aplicada ao *framework* Ruby on Rails [12]. O servidor web repassa as requisições recebidas a um sistema de rotas, que é encarregado de enviá-las ao controlador correspondente. O sistema de rotas muitas vezes é definido no próprio servidor web, porém, o Rails possui sistema de rotas independente, como será elucidado na subseção 2.4.2. Os demais componentes, relativos ao MVC, serão melhor detalhados a seguir.

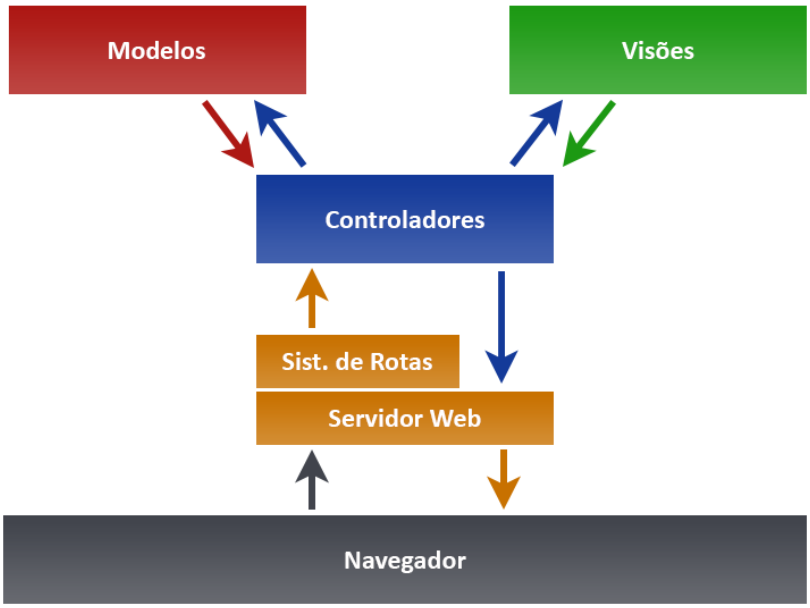

Figura 2.2: Arquitetura MVC do *framework* Ruby on Rails.

- O Modelo

  O modelo é o módulo responsável pela gerência do conhecimento da aplicação e sua manipulação segundo métodos específicos. Em geral, as aplicações utilizam BD, porém existem exemplos de aplicações em que não é interessante o seu uso. No primeiro caso, os modelos em geral estabelecem uma relação de um para um com as estruturas de objetos (as tabelas do BD). Códigos especialistas, como buscas eficientes em BD ou manipulação e validação de dados inseridos pelos usuários, devem ser pertencentes aos modelos.

- A Visão

  A visão é, em geral, a representação visual do seu modelo. Todo o código do *frontend* da aplicação é parte das visões. Comumente são utilizadas linguagens para mesclar dados variáveis (resultado das requisições) com as visões, além de fazer o chamado *templating*, que será melhor elucidado ainda neste capítulo. No Ruby on Rails, são utilizadas linguagens como o ERB (Embedded Ruby) e o Haml (HTML Abstraction Markup Language) para esta finalidade.

- O Controlador

  O controlador é responsável por conectar os usuários ao sistema em si. Informações presentes nas requisições HTTP realizadas e informações extraídas dos modelos, servem de base para os controladores, que, segundo uma lógica definida pelo implementador, definem o fluxo resultante da requisição.

Para exemplificar o funcionamento geral dos módulos elucidados, será utilizado novamente o caso do usuário que tenta realizar uma abertura de sessão. Quando a requisição HTTP é disparada, por meio do mecanismo de rotas (que será detalhado mais a frente), esta requisição chega a um controlador. Este, por sua vez, acessa o modelo e verifica a validade dos dados, como a existência do usuário e a corretude da senha inserida. O controlador decide então por abrir a sessão do usuário e redirecioná-lo para a visão correspondente ou por recarregar a página de tentativa de abertura de sessão, exibindo uma mensagem de erro pertinente.

2.4.2 Sistema de rotas independente

Uma aplicação em Rails implementa o seu próprio sistema de rotas, que é independente do servidor web que está sendo utilizado. O servidor padrão oferecido é o WEBrick, em Ruby, porém outros servidores podem ser utilizados sem que o esquema de rotas precise ser alterado. Trata-se de um arquivo onde, basicamente, é feita a correspondência entre os controladores e o conjunto de recursos e métodos das requisições HTTP.

2.4.3 Incentivo à programação RESTful

Além dos princípios citados, o Ruby on Rails incentiva a programação RESTful através de algumas de suas configurações padrão. Para a criação de rotas RESTful em sua aplicação, basta apenas uma linha de código. Esta linha define que deve ser gerada a correlação entre o CRUD de um recurso e os métodos HTTP sobre uma URI referente ao recurso. Para ser mais claro, essa única linha de código resulta na criação de rotas como indica a tabela 2.1.

Tabela 2.1: Tabela de rotas automáticas para recursos no Ruby on Rails

Método HTTP	**Endereço**	**Ação**
GET	/usuarios	index
GET	/usuarios/new	new
POST	/usuarios	create
GET	/usuarios/:id	show
GET	/usuarios/:id/edit	edit
PUT	/usuarios/:id	update
DELETE	/usuarios/:id	destroy

Com exceção do método *show* (que é equivalente ao método *retrieve* no CRUD), os outros três métodos do CRUD possuem nomes equivalentes. Entretanto, podemos notar que existem três métodos que não se encaixam no CRUD. Vamos analisar primeiramente o *new* e o *edit*. Eles foram criados visto que, para a realização de tarefas como *update* ou *create*, são necessários formulários, já que não é racional exigir que usuários comuns realizem manualmente requisições HTTP, fornecendo os parâmetros corretos para cadastro ou atualização. Por isso foram criados esses dois métodos, que exibem ao usuário páginas contendo formulários de cadastro e de atualização, viabilizando as operações de *update* e de *create*. Além destes foi criado o *index*, que foi feito para dar lugar a uma página contendo (todos ou alguns) objetos equivalentes àquele recurso. Resumidamente, trata-se de um *retrieve* em grupo.

2.4.4 Geradores automáticos de código

O Ruby on Rails possui outra característica interessante que provê agilidade no desenvolvimento: os geradores automáticos de código. Com um simples comando é possível evitar algum tempo de desenvolvimento utilizando esses geradores. Eles podem criar modelos, controladores, visões, migrações ao BD, ou até, todos ao mesmo tempo, como é o caso do gerador *scaffold*.

Esta funcionalidade também é muito útil para instalação de bibliotecas externas (as chamadas gemas, como veremos a seguir). Muitas bibliotecas necessitam da criação de tabelas no BD para seu funcionamento pleno, ou oferecem visões prontas para seus primeiros usos. Desta forma, são criados instaladores automáticos para as bibliotecas, utilizando os geradores automáticos de código.

2.4.5 Ferramentas de automatização de design de *frontend*

Ao programar em Rails, assim como em diversos outros *frameworks* para desenvolvimento ágil, é possível implementar o design de suas visões de forma automatizada, ou resumida. Esta é uma importante tarefa dos *frameworks*, pois faz com que seja necessário saber pouco ou nada sobre linguagens como CSS ou Javascript, para que seja desenvolvida uma página visualmente agradável. Apesar de o Rails não auxiliar na produção de estilos ou interatividade para a aplicação, como estes *frameworks* de *frontend*, ele fornece diversos recursos que facilitam enormemente a produção do conteúdo da aplicação (o código HTML).

Através das linguagens ERB e HAML, como dito anteriormente, é possível a utilização de operações em Ruby dentro dos arquivos de visões. Suponhamos que é necessário que a visão contenha 20 elementos HTML idênticos ou extremamente semelhantes. Esta tarefa seria extremamente penosa e repetitiva, além de produzir código redundante. Entretanto, com o Rails, é possível escrever um único elemento

dentro de um iterador na linguagem Ruby, para obter o mesmo resultado. A linguagem HAML vai mais além. Sua utilização torna a produção de código mais legível e organizada, reduzindo boa parte do código inserido para produção de elementos HTML.

É comum que precisemos exibir os mesmos componentes em mais de uma visão da aplicação. Se tratando do *framework* Ruby on Rails, um mesmo trecho de código replicado diversas vezes ao longo de toda a aplicação indica que é necessário um replanejamento do mesmo. Um iterador resolve o problema para o caso de código replicado em cadeia, porém existe uma boa solução para replicação de código de visões em geral: os parciais. Um parcial é um arquivo de visão auxiliar que pode ser invocado dentro de outras visões, ou até mesmo dentro de outros parciais. Ao invocar um parcial, o código de invocação é substituído pelo seu conteúdo. É possível que parciais recebam parâmetros no momento de sua invocação, possibilitando a personalização do resultado final em cada chamada.

Além disso, o Rails possui funções ajudantes (do inglês *helpers*), que automatizam procedimentos que são comumente realizados pelos usuários. Enquanto os parciais auxiliam na criação de elementos de visão (em HTML), evitando a replicação de código, as funções ajudantes permitem que código Ruby seja inserido, com a mesma finalidade.

Outro mecanismo que dá praticidade à produção de código de visão são os *templates*. Estes possibilitam que código seja adicionado sem que seja preciso sequer invocá-los. É muito comum a sua utilização quando é necessária a produção de um mesmo código de visão para diversas páginas, como em cabeçalhos. É comum que um aplicativo web possua diferentes estilos de cabeçalhos, como para usuários autenticados ou não autenticados. Apesar de serem distintos, estes cabeçalhos se repetirão ao longo de diversas visões. Neste caso é interessante que sejam incluídos em *templates* que serão adotados pelos arquivos de visões.

2.4.6 Ferramenta para envio de e-mails

O Rails permite que, através do módulo *Action Mailer*, sejam enviados e-mails com extrema praticidade pela aplicação. A formatação dos e-mails é realizada através dos *mailers*, que são semelhantes aos controladores, e dos arquivos de visão relacionados. Os *mailers* possuem funções que representam tipos diferentes de e-mails a serem enviados, cada um com seu respectivo arquivo de visão. Em cada função é definido o conteúdo dos campos que preenchem o e-mail, porém a descrição do mesmo é preenchida através do arquivo de visão, utilizando código HTML. Os e-mails podem possuir conteúdo dinâmico, assim como os demais arquivos de visão.

2.4.7 A linguagem Ruby

Ruby é a linguagem utilizada ao trabalhar com o *framework* Ruby on Rails. Trata-se de uma linguagem totalmente orientada a objetos. Nela, até mesmo os números e classes são objetos. É extremamente dinâmica, permitindo inclusive que métodos sejam implementados em tempo de execução (é o chamado *monkey patching*) [13]. Inspirada em linguagens como o SmallTalk, o Ruby é extremamente legível e manutenível, já que o código pode ser escrito sem operadores como parênteses ou colchetes (é o chamado *poetry mode*, que funciona desde que não existam ambiguidades no código produzido). A implementação de código se aproxima cada vez mais da escrita livre de textos, abrindo mão de redundâncias e declarações desnecessárias, que representam grande parte dos caracteres escritos em grande parte das linguagens de programação. Novamente podemos observar que essa é uma característica que dá agilidade ao processo de desenvolvimento.

Outro ponto extremamente positivo do Ruby é a existência de um gerenciador de pacotes próprio, o RubyGems. Ele facilita de maneira excepcional a utilização de bibliotecas externas, aliado com os geradores automáticos de código. Em outras palavras, com duas linhas é possível incluir e instalar uma biblioteca para utilização. Com apenas mais uma linha é possível gerar (quando necessário) as modificações que configuram sua aplicação, de forma a se adequar à utilização da nova biblioteca. As bibliotecas são as chamadas gemas. Torna-se também trivial a atualização das versões instaladas das bibliotecas, sendo necessário apenas um comando para buscar atualizações e instalá-las.

2.4.8 Boa abstração do BD

Qualquer um que está habituado a armazenar dados em um BD sabe que não é uma tarefa fácil. Um dos motivos é a necessidade de se preocupar com sintaxe das consultas em SQL (linguagem dos bancos de dados relacionais) realizadas. É comum nos depararmos com operações que são normalmente interpretadas por um SGBD (Sistema de Gerência de Banco de Dados), mas que causam erros sintáticos em outros.

O Active Record

O Rails oferece por padrão uma biblioteca chamada ActiveRecord que possui uma série de módulos importantes que são, sem dúvida, uma das principais ferramentas do *framework*. Esta biblioteca estabelece uma associação entre uma classe de modelo a uma tabela do BD. Com ela é possível lidar com os dados de forma muito mais amigável. Para executar consultas ao banco não é mais necessário ter o domínio da

linguagem SQL, pois esta biblioteca cria uma abstração da interface com o BD, provendo uma outra interface muito mais amigável e em Ruby. Para que esta biblioteca consiga se relacionar corretamente com o banco, basta instalar a gema correspondente ao seu SGBD. Desta forma, o seu código não se altera ao se alternar entre os diferentes SGBDs.

O ActiveRecord simplifica também o acesso a objetos por meio de relacionamentos. Isso implica na realização de consultas automáticas utilizando a chave estrangeira pertencente a um objeto, permitindo o acesso de entidades relacionadas com a mesma facilidade de acesso dos atributos locais. Também é possível a utilização de transações para prover às operações propriedades como atomicidade, e, aliado com o sistema de travas (do inglês *lock*), propriedades como isolamento e consistência.

O Active Model

Existe também uma outra biblioteca chamada ActiveModel que possui módulos essenciais para o desenvolvimento de aplicações de qualidade, que serão detalhados a seguir.

Validação

Com este módulo é possível garantir que os dados que serão persistidos no banco sejam válidos ou consistentes. Existe uma série de funções que automatizam os processos de validação mais comuns, como a *validates_presence_of*, que garante que não sejam válidos registros onde os campos passados como parâmetro são nulos. Porém, é possível implementar suas próprias funções de validação, que podem ser exportadas para validadores externos às classes dos modelos, para que sejam reutilizadas em mais de um deles.

Tais validações ocorrem entre o instante em que o sistema requisita que um registro seja persistido no banco e o momento em que ele é persistido de fato. Caso a validação invalide o registro, é adicionado um erro ao registro (atraves do módulo de erros) e o mesmo não é persistido, garantindo a integridade do banco.

Chamadas de Retorno

É extremamente importante que possamos realizar ações em momentos oportunos, como após a atualização de determinado registro, ou antes do momento em que são realizadas as validações. Para isso foi criado o módulo das chamadas de retorno (do inglês *callbacks*). Com estas chamadas é possível executar procedimentos em momentos específicos e muito oportunos.

Para exemplificar o uso deste módulo, pode-se imaginar que é necessário incrementar um contador em um registro do banco toda vez que ele for atualizado. Existe

uma solução redundante e trabalhosa para isso, que exige que o implementador lembre de incrementar este contador em todos os locais do código onde o registro é atualizado Porém, é muito mais interessante realizar essa tarefa utilizando as chamadas de retorno. Para isso, basta implementar uma função no modelo que realize esse incremento, e passar essa função como parâmetro para o método *before_ save* do módulo de chamadas de retorno. Desta forma, função sempre será executada logo antes da atualização do registro. Note que, neste caso, a função só será invocada caso o registro passe por todas as validações.

Observação

O módulo de observação possibilita que sejam implementadas chamadas de retorno sobre uma classe, porém executadas fora da mesma. Na prática, criamos um observador que "vigia" uma ou mais classes de modelos, podendo executar chamadas de retorno sobre elas simultaneamente.

Trapaça

Outro módulo importante é o módulo de trapaça (do inglês *dirty*). Pode ser aplicado para a seguinte situação: deseja-se incrementar o contador citado apenas para o caso de um determinado atributo ter sido alterado. Para isso, também há soluções análogas à explicada no exemplo das chamadas de retorno. Porém tais soluções não são escaláveis. Em conjunto com as chamadas de retorno, o módulo de trapaça pode ser utilizado para prover a solução ideal. Este módulo permite acessar informações sobre atributos alterados desde a sua última versão persistida em banco. O nome tão peculiar deste módulo se deve ao fato de que ele nos dá o poder de "bisbilhotar" as modificações realizadas e tomar decisões baseadas nelas.

Nomeação

Este módulo lida com a pluralização das palavras, uma vez que o *framework* convenciona que, por exemplo, nomes de classes são no singular, porém suas tabelas correspondentes são nomeadas no plural. Isso torna o código mais legivel, porém pode causar problemas, principalmente em expressões que não são pluralizadas de forma regular. Para isso existe um arquivo de inflexões, onde devem ser informadas as pluralizações não regulares a serem utilizadas pelo sistema.

Tradução

Outro módulo semelhante é o de tradução, que se comunica com as ferramentas de internacionalização do sistema, elucidadas a mais a frente.

Serialização

Este módulo permite a transformação de registros de modelos em tabelas *hash* serializáveis ou arquivos de texto plano, como XML ou JSON, para que possam ser exportados ou persistidos.

2.4.9 Suporte à autenticação e autorização terceirizado

O Rails por padrão não implementa sistemas de autenticação e autorização, porém existem gemas que desempenham estas funções com maestria, como veremos adiante. Apesar disso, existem sistemas que possibilitam a verificação de condições antes da realização das ações, são os chamados filtros.

Os filtros são uma funcionalidade que permite que em pouquíssimas linhas sejam realizadas verificações que, muitas vezes, estão ligados à autorização. Um exemplo disso são os *before_filters*, que muitas vezes são utilizados para avaliar se um usuário está autenticado ao sistema, antes de permitir a realização das ações correspondentes.

2.4.10 Segurança

Inúmeras questões referentes à segurança ficam a cargo do servidor web, como a prevenção contra ataques DDoS (Distributed Denial of Service) por exemplo, porém, após o momento em que as requisições chegam ao sistema de rotas do Rails, é responsabilidade do *framework* e do programador localizar todas as brechas do sistema. Esporadicamente são necessárias atualizações de segurança ao Ruby on Rails, porém a maioria esmagadora dos ataques bem sucedidos são ocasionados por descuido dos desenvolvedores da aplicação. O *framework* divulga uma página alertando sobre falhas comuns e suas medidas preventivas [14].

Uma destas medidas oferecidas pelo Rails é a utilização de conexões SSL (*Secure Sockets Layer*). A aplicação pode ser configurada para exigir do usuário que seja estabelecida uma conexão SSL antes de permitir a utilização do sistema. Esta medida pode prevenir a chamada *Session Hijacking*, que permite a um usuário atacante a utilização do sistema através de uma conta com sessão previamente iniciada e em atividade.

Ataques de injeção também são muito comuns em aplicações web. Elas podem ser de diversos tipos. Um dos mais utilizados é o XSS (Cross-Site Scripting), que é um ataque de injeção de textos capaz de prejudicar o usuário, seja simplesmente para tornar pior a sua experiência ao utilizar o sistema ou para fazer *Session Hijacking*. É possível também fazer injeções a nível de BD, para outras finalidades. Caso o sistema não seja imune à injeção de dados de uma forma geral, as possibilidades são

infinitas para os atacantes. Uma medida preventiva para ataques de injeção é filtrar os dados inseridos pelo usuário, sempre que necessário.

2.4.11 Eficiente

O Rails implementa duas funcionalidades que visam prover eficiência aos aplicativos: o *Assets Pipeline* e o sistema de *cache*.

Assets Pipeline

O *Assets Pipeline* é um módulo que auxilia na compilação de todos os arquivos que devem ser servidos ao navegador. Isto inclui os arquivos CSS, Javascript, fontes e imagens. Este módulo realiza atividades como:

- Concatenação de arquivos

  Essa é uma característica muito importante, principalmente no ambiente de produção. Ela é capaz de reduzir o número de requisições necessárias para que o navegador seja capaz de exibir a página. Todos os arquivos CSS são concatenados em um único arquivo (também em CSS), chamado arquivo de manifesto. O mesmo ocorre para os arquivos Javascript, totalizando dois arquivos de manifesto. Para evitar que o navegador tenha que carregar estes dois arquivos sempre que ocorrerem novas requisições, o Rails utiliza uma estratégia de "impressão digital". Isso faz com que o navegador só requisite novamente os arquivos de manifesto que contenham impressões digitais diferentes das versões presentes em seu cache.

- Compressão e minimização de arquivos

  Esta atividade minimiza os arquivos CSS e Javascript. No caso dos arquivos CSS, todos os espaços e comentários são excluídos do arquivo. Em arquivos Javascript processos mais complexos são aplicados, porém foge ao escopo deste trabalho explicitá-los. Essas lógicas para minimização e compressão podem ser customizadas de forma a atender caso a caso da melhor forma possível.

- Código de Alto Nível

  O *Assets Pipeline* permite a utilização de ERB dentro do código CSS ou Javascript, permitindo um mundo de possibilidades que antes não era possível. No momento de compilação, esse código é transformado de forma a gerar um arquivo simples CSS. Além disso, permite a utilização de linguagens que vêm sendo cada vez mais utilizadas, como Sass (uma linguagem de CSS que permite a utilização de variáveis e pseudo-funções) e CoffeeScript (uma forma de escrever Javascript com menos linhas de código).

- Pré-compilação

  A pré-compilação acelera muito a exibição da página. Ela permite que todo o código CSS e Javascript seja compilado apenas uma vez, para que seja servido repetidamente em todas as requisições recebidas. Por padrão, os arquivos de manifesto CSS e Javascript são pré-compilados, mas é fácil configurar para que outros arquivos também sejam.

Cache

Existem diversos níveis de cache implementados no Rails, visando acelerar o acesso à informação. Os principais são: o cache de página, de ação e de fragmento.

O cache de página permite que as requisições sejam preenchidas diretamente pelo servidor web, sem que seja necessária a interferência da aplicação Rails em si. Dessa forma, o tempo de acesso é muito reduzido, porém, em muitos casos é necessário que o fluxo chegue ao controlador, como em páginas que exigem autenticação.

Para atender estes casos com um bom desempenho, foi criado o cache de ações do controlador. Neste caso, a requisição deve passar por filtragens, como verificações de autenticação, existentes no método controlador correspondente, antes de ser respondida por este nível de cache.

Por último, o cache de fragmentos permite o armazenamento de pequenas informações específicas sob demanda. O Rails provê funções para criar e expirar fragmentos de cache separadamente.

Além disso, o Rails por padrão faz o cache de consultas ao BD, de forma que elas só precisam ser realizadas uma única vez por requisição.

2.4.12 Suporte a Internacionalização

O Ruby on Rails permite internacionalização de textos de forma extremamente fácil. Todas as configurações de internacionalização já são padronizadas. Basta escrever os textos em arquivos no formato YAML, e utilizá-los ao escrever as visões. Como existem inúmeros idiomas e, conforme a sua aplicação cresce torna-se mais difícil a manutenção dos textos, pode ser uma tarefa complicada lidar com os arquivos de configuração YAML. Neste caso, podem ser utilizados outros métodos de internacionalização. Para isso, basta alterar o *backend* de internacionalização da sua aplicação, que passa a procurar por traduções utilizando outros métodos, como uma busca em BD por exemplo. Através deste segundo método, com algum código extra, é possível criar recursos para edição de traduções, permitindo que seja possível alterar qualquer texto da aplicação com seu respectivo idioma utilizando a própria aplicação web.

2.4.13 Incentivo à Implementação de Testes

Cada vez mais, testes em projetos de software são mais valorizados e utilizados. As principais vantagens da criação de testes são:

- Reduzem bugs

  Como o nome já diz, testes têm por objetivo verificar a corretude da implementação dos casos de uso do sistema, de forma que todas as possibilidades sejam contempladas. Desta maneira, bugs são encontrados e corrigidos.

- Proveem documentação

  Em geral, testes têm nomes bem claros e de acordo com a funcionalidade a ser testada. Sendo assim, torna-se muito mais clara a relação entre o código da aplicação e suas funcionalidades, facilitando a produção de uma documentação de qualidade.

- Automatizam o processo de busca por erros

  Ao adicionar funcionalidades, é comum a inserção de erros em trechos de código que estavam funcionais. Isso é especialmente ruim para aplicações em produção. Sem testes automatizados, detectar tais erros seria uma tarefa extremamente penosa, pois seria necessária a reavaliação de todas as funcionalidades do sistema por testadores. Utilizando testes automáticos no Rails, com uma única linha de comando é possível realizar todos os testes programados.

- Melhoram o design da aplicação

  Testes possibilitam ao desenvolvedor uma noção macro do resultado final. Isso ocorre pois eles forçam os desenvolvedores a pensar sobre todas as funcionalidades a serem desenvolvidas e no modo como serão desenvolvidas, oferecendo uma melhor visualização sobre os componentes da aplicação de forma conjunta.

Os testes podem englobar diversas áreas do software, e por isso, foram divididos em alguns tipos. Os principais são:

- Testes de Unidade

  Visam testar as menores unidades passíveis de testes do software, sendo assim, estes são independentes dos demais. No Rails, estes testes são diretamente relacionados aos modelos, suas validações e chamadas de retorno.

- Testes Funcionais

  Visam testar as funcionalidades do software, verificando se todos os requisitos estão sendo atendidos. Como o bom funcionamento de cada uma das funcionalidades é dependente da corretude das unidades, os testes funcionais dependem

diretamente dos testes unitários. No Rails, testes funcionais estão atrelados aos controladores, visto que estes são os responsáveis por receber requisições que acionam as funcionalidades.

- Testes de Integração

  Visam verificar a integração entre as funcionalidades do sistema. Por testarem a aplicação de forma geral, o seu desempenho está diretamente relacionado ao desempenho dos testes funcionais e de unidade. No Rails, estão associados a testes em visões e, em parte, em controladores.

- Testes de Aceitação

  Testam a aceitação dos usuários finais do sistema. Em geral, é realizado em cima de ambientes controlados e abertos para um grupo não muito grande de usuários, avaliando questões de funcionamento, design e experiência do usuário.

O Rails é uma das comunidades de programação que mais valoriza os testes. Não é a toa que qualquer tutorial, vídeo aula ou qualquer curso de Rails tem uma parcela importante sobre testes. O próprio *framework* foi construído com o auxílio de testes. Para facilitar ainda mais a utilização dos mesmos, os geradores de código padrões do Rails criam automaticamente arquivos de teste com estruturas básicas para iniciar a programação, incentivando sua implementação.

Por padrão, o Rails fornece uma ferramenta de testes bastante robusta, funcional e de fácil utilização. Entretanto, um outro *framework* de testes ganhou muito espaço, e é um dos mais utilizados atualmente: o RSpec [15]. Foi construído para utilizar a metodologia TDD (que será elucidada a seguir) de forma bastante produtiva. A metodologia utilizada pelo RSpec se baseia na descrição de problemas e na programação de testes de forma mesclada. Utilizando a descrição como elemento da linguagem de programação, é possível incrementá-la a medida que a lógica dos testes é incrementada, de forma a compor descrições cada vez mais específicas. Desta forma são produzidas mensagens de erro bastante claras, facilitando o trabalho dos programadores em encontrar os pontos de falha na implementação. Além disso, a linguagem do RSpec é muito simples e tenta utilizar operadores da língua inglesa, como o operador *should*, que verifica se uma variável possui um determinado valor esperado para um caso de teste especifico. [16]

2.5 TDD (Test Driven Development)

Existem dois métodos extremamente populares e recomendados para o desenvolvimento de software: o BDD (Behavior Driven Test) e o TDD. Neste projeto, será

aplicada a metodologia TDD, porém são metodologias semelhantes, pois ambas se baseiam no desenvolvimento de testes antes da implementação da aplicação propriamente dita, possibilitando aos desenvolvedores uma noção macro do resultado final. Isso ocorre pois os testes os forçam a pensar sobre todas as funcionalidades a serem desenvolvidas (e no modo como serão desenvolvidas) antes de implementadas, evitando que o código da aplicação se torne uma "colcha de retalhos".

O TDD sugere que os testes devem ser programados do micro ao macro, isto é, começando por partes unitárias e rumando para testes cada vez mais abrangentes, enquanto o BDD propõe o caminho inverso. É normal que programadores que não estão habituados a estas práticas se sintam desconfortáveis em um primeiro momento, mas, uma vez percebidas as vantagens em suas utilizações, tornam-se cada vez mais interessados. Vale ressaltar que testes demandam tempo e, em projetos muito simples ou pequenos, pode não ser interessante realizá-los, seja por questões financeiras ou prazos. Sendo assim, é sempre importante avaliar até que ponto o sistema deve ser testado.

2.6 Desenvolvimento Colaborativo

Como este projeto foi desenvolvido em dupla, foi necessária a criação de um ambiente de desenvolvimento colaborativo. Isso implicou na utilização de ferramentas e repositórios. Uma das ferramenta utilizadas foi o Git [17], com a finalidade de realizar o controle de versão da aplicação. Com essa ferramenta, todas as modificações submetidas são salvas como versões do sistema, que podem ser recuperadas. Existe ainda uma ferramenta de acoplamento de versões, necessária para fundir modificações simultâneas realizadas por diferentes usuários. Esta ferramenta não é perfeita, porém na maioria dos casos, isenta os usuários da preocupação de unir os trabalhos realizados.

Para que o controle de versão possa ser utilizado por qualquer desenvolvedor (independente de qualquer máquina utilizada no desenvolvimento), foi utilizado um repositório *online* gratuito chamado Bit Bucket [18]. A grande vantagem deste repositório é a privacidade do conteúdo, que torna necessária a autenticação por meio de senha para acesso ao mesmo (até mesmo para projetos hospedados gratuitamente no repositório).

Durante o desenvolvimento, é interessante que haja interações em tempo real entre os desenvolvedores. Para isso foram utilizadas algumas ferramentas. Uma delas foi o Skype [19], para conferências em áudio e vídeo. Além disso, é interessante que um desenvolvedor possa avaliar o andamento da aplicação, até mesmo em momentos onde esta não esteja estável, sem que seja necessário submetê-las ao repositório. Para isso foi utilizada uma gema chamada Local Tunnel [20]. Esta

biblioteca permite que uma aplicação Rails que está sendo executada em ambiente de desenvolvimento (localmente na máquina do desenvolvedor), seja acessível por qualquer usuário da Internet, por meio de uma URI. É uma biblioteca extremamente útil para desenvolvimento colaborativo e de fácil instalação e utilização.

Capítulo 3

A Implementação

Conforme mencionado na seção 1.3, como aplicação prática para demonstrar os novos métodos para desenvolvimento ágil de sistemas web, foi implementado um sistema robusto para criação e manutenção de eventos batizado de EVNTs. Sempre com o objetivo de acelerar o processo de produção, foram utilizadas gemas, as quais serão explicadas com mais detalhes adiante.

3.1 Funcionalidades

A figura 3.1 visa apresentar de forma geral (através de um fluxo de ações do usuário) as funcionalidades do sistema, que serão detalhadas através das subseções a seguir.

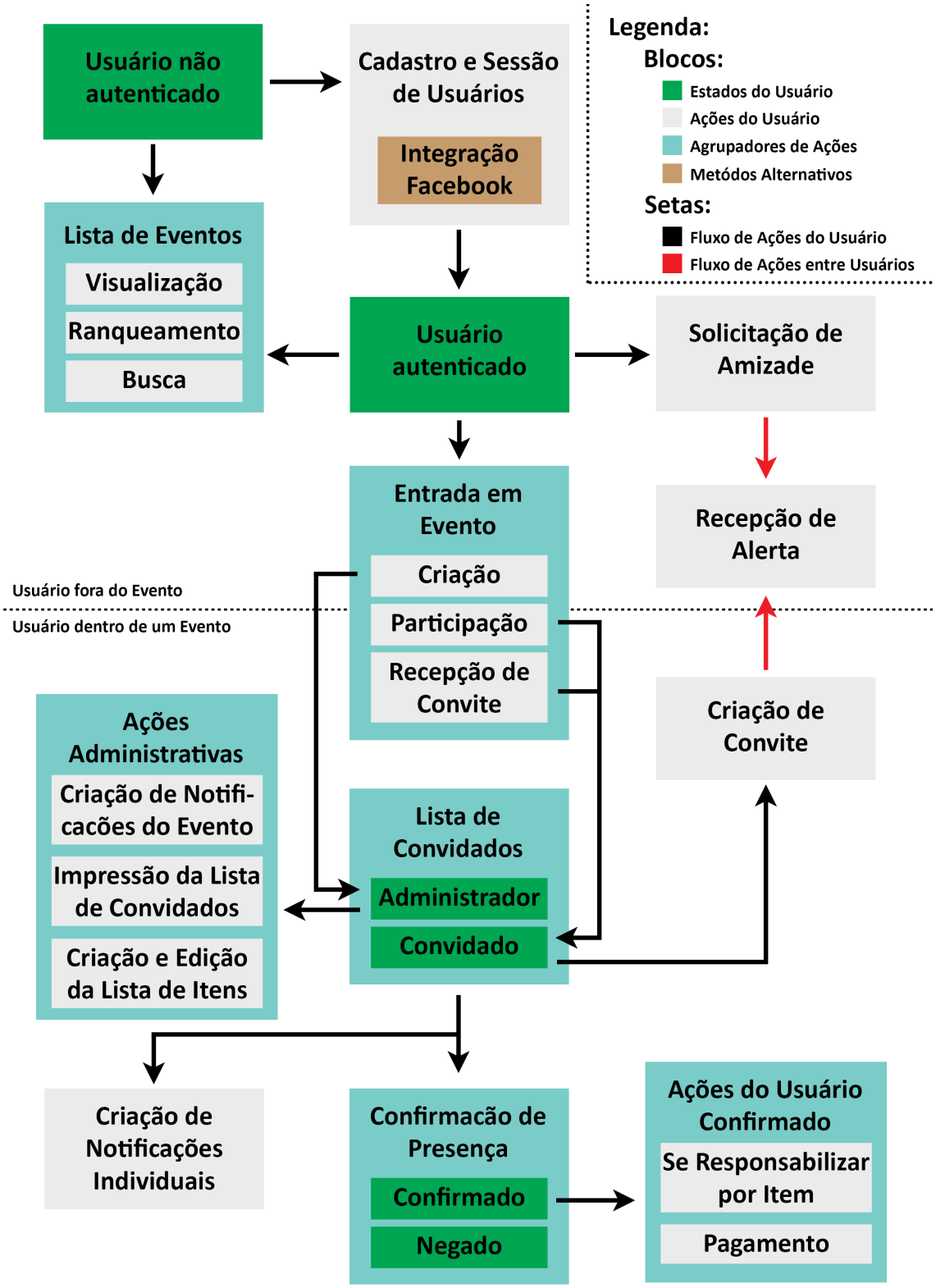

Figura 3.1: Fluxo de ações do usuário no sistema.

3.1.1 Cadastro e Sessão de Usuários

O projeto tem como objetivo prover funcionalidades aos usuários, tornando-os parte essencial para o bom funcionamento do sistema. Para poder identificá-los e realizar tarefas relacionadas a usuários específicos, é necessário existir sistemas de cadastro e sessão. Basicamente um sistema de cadastro consiste em um formulário que viabiliza a criação de um novo usuário no banco de dados. Um sistema de sessão, por sua vez, permite que ao retornar, esse usuário use um *login* (nome ou e-mail) e uma senha, cadastrados anteriormente, para se identificar.

Por ser um módulo comum em diversos sistemas, existem muitas bibliotecas que tentam facilitar o seu desenvolvimento. No caso do Rails, a gema mais difundida, que também foi utilizada no projeto, se chama Devise [21].

Devise

O Devise é uma solução bastante completa para toda a parte de autenticação de usuários. Primeiramente, é muito fácil de ser configurado e é completamente personalizável, garantindo sua popularidade. Vale ressaltar que o recurso gerado pelo Devise (o usuário) é construído seguindo o padrão RESTful, tornando-o ainda mais atrativo. Dentre suas vantagens e funcionalidades estão:

- Segurança

  Senhas criptografadas, com o custo de criptografia personalizável.

- Histórico

  Aliado também à segurança, o Devise já traz informações de data do último acesso, IP de acesso, entre outras.

- Validação

  O e-mail (campo padrão para *login*) e a senha são validados automaticamente.

- Confirmação por E-mail

  Opção que faz com que o usuário receba um e-mail após o início do cadastro, onde o mesmo é finalizado, garantindo que o usuário é, de fato, dono do e-mail.

- Manter Logado

  Módulo que implementa a criação de um cookie no computador do usuário para que sua sessão seja criada automaticamente na próxima vez que retornar ao site.

- Tempo Limite

  O Devise permite o uso de um tempo máximo de inatividade para a sessão, terminando-a automaticamente ao final do prazo.

- Trancamento de Conta

  Outra funcionalidade implementada é a de trancamento de conta, para o caso de muitas tentativas sucessivas mal sucedidas de acesso em sequência. Um e-mail com instruções para destrancar é enviado automaticamente ao e-mail cadastrado, no momento do trancamento.

Filepicker

Uma parte muitas vezes complicada de sistemas é a utilização de fotos do perfil, sendo necessária a criação de uma parte de upload, hospedagem de arquivos e muitas vezes processamento da imagem como cortes e lógicas para encaixá-la ao layout. Para isso, utilizamos uma ferramenta bastante interessante chamada Filepicker.io [22]. Todo o processo fica por conta da ferramenta, sendo cobrada uma taxa para tráfegos acima de 5000 fotos/mês. É bastante personalizável e permite além de upload de fotos do próprio computador, acesso a fotos de outros lugares como o Facebook, Google+, Dropbox, entre outros.

3.1.2 Integração com Redes Sociais

Para melhorar ainda mais a experiência do usuário em um sistema web, é muito comum usar contas compartilhadas, evitando que o usuário tenha que lembrar de mais um e-mail e senha. Dessa forma, foi implementada a possibilidade de cadastro e sessão utilizando a conta do Facebook do usuário. Uma gema muito útil e conhecida para isso é o Omniauth-facebook [23], atrelado ao Omniauth [24]. É necessária a criação de um aplicativo no Facebook Developer (https://developers.facebook.com/) para configurar o Omniauth com uma chave de identificação e senha. Além da utilidade de cadastro e abertura de sessão, o Facebook também provê informações sobre a conta do usuário, mediante autorização, que podem ser úteis na aplicação. Por ter se tornado popular, até mesmo o Devise implementou uma funcionalidade de integração com o Omniauth, gerando assim uma solução muito robusta e completa.

3.1.3 Eventos e Lista de Convidados

Os eventos são a parte mais importante do projeto, já que tudo gira ao redor deles. Basicamente são instâncias que devem conter informações sobre local, data, hora, preço e privacidade. Além disso, usuários devem ser capazes de criar eventos, confirmar ou negar presença nos mesmos e convidar outros usuários a participarem deles.

É importante também a ferramenta de impressão da lista de confirmados para o organizador, de forma a facilitar sua vida no dia do evento.

Privacidade

Foram implementados três tipos de privacidade para os eventos:

- Públicos

  Eventos vistos e encontrados por qualquer usuário. Qualquer um pode confirmar presença e convidar pessoas.

- Privados

  Eventos onde somente pessoas convidadas pelo organizador podem ver e confirmar presença, porém, convidados não podem convidar.

- Convidativos

  Parecido com o privado, porém, neste caso os convidados podem convidar mais usuários.

Preço

O preço do evento depende basicamente de três informações.

- Preço Fixo

  Valor constante, que deve ser pago por todos os confirmados.

- Preço Compartilhado

  Valor constante a ser dividido entre todos os confirmados.

- Preço de Itens

  Valor que representa o somatório dos preços dos itens. Este caso será explicado com mais detalhes na próxima seção.

3.1.4 Lista de Itens

Essa funcionalidade é um dos principais diferencias do sistema desenvolvido com relação ao que existe no mercado. Em suma, trata-se de uma lista de itens (como refrigerantes, bola de futebol ou aparelho de som) que são necessários para a realização do evento. Uma vez criada, usuários podem se tornar responsáveis por itens, isto é, se comprometem a levar o item no dia do evento. Os itens podem ter quantidades específicas, possibilitando que a responsabilidade sobre um mesmo item seja dividida entre mais de um usuário.

Além disso, itens podem possuir preço. Neste caso, o valor somado da lista também será dividido entre os confirmados no evento. O valor de um item, atrelado à quantidade adquirida por um responsável, será descontado do valor a ser pago por este usuário.

3.1.5 Pagamento

Outro diferencial do sistema é a possibilidade de pagamento através do mesmo, dando uma garantia muito maior ao organizador da presença de seus convidados e suporte ao planejamento financeiro do evento. Para que não ocorram problemas, é importante notar que eventos com custo compartilhado ou lista de itens pagos, não poderá ter pagamentos através do sistema, uma vez que, nestes casos, o valor se altera à medida que novos usuários confirmam presença.

O pagamento ocorrerá através do Paypal [25], uma ferramenta simples e segura de realizar pagamentos na Internet. Eles possuem uma API fácil de utilizar e bastante completa, incluindo até mesmo um ambiente para testes. Sua única desvantagem é que o usuário será redirecionado para um site externo ao sistema na hora de pagar, piorando a experiência do usuário.

O dinheiro irá para a conta do EVNTs, e, após a realização do evento, será transferido para a conta bancária cadastrada pelo usuário. Essa transferência, em um primeiro momento, ocorrerá de forma manual, porém em trabalhos futuros será implementado um sistema automático para isso. Além disso, diversos casos de segurança (principalmente fraudes) ainda não estão sendo contemplados e serão resolvidos também no futuro.

3.1.6 Amizades

Para facilitar a parte de convites para o evento, foi implementado um módulo de amizades, que permite um relacionamento entre dois usuários. Basicamente, um usuário envia uma solicitação de amizade a outro usuário que pode aceitá-la, recusá-la ou até mesmo recusá-la para sempre. Uma vez que a solicitação foi aceita, a amizade é criada. Desta forma, torna-se possível convidar o amigo para eventos de forma simples e direta.

Como já existe a integração com o Facebook no momento do cadastro, a lista de amigos da rede social será aproveitada de forma a automaticamente recriar as amizades dentro do sistema EVNTs, reduzindo o trabalho do usuário.

3.1.7 Notificações e Alertas

Para que o evento não entre no esquecimento, foi implementado um módulo de notificações. A ideia é que o organizador possa criar notificações a serem enviadas para todos os confirmados no seu evento e que cada usuário possa criar notificações individuais para si. Porém, o usuário pode se subscrever ou não às notificações gerais, de forma a evitar casos não desejáveis.

A notificação será na forma de um e-mail que será enviado na data desejada para o usuário indicando a data, horário, local do evento e os itens que aquele usuário deverá levar. Para que ela seja programada para uma data posterior, utilizamos uma gema chamada Delayed Jobs [26], que permite de forma bastante simples, executar ações assíncronas no sistema, seja por um sistema de fila, isto é, várias ações para serem executadas em sequência, ou por um sistema de data marcada. Seu funcionamento se dá através de uma tabela no BD onde as ações ficam guardadas e um sistema que verifica a existência de tarefas a serem executadas nessa tabela.

Além disso, há um sistema de alertas para convites de eventos e solicitações de amizades pendentes. Os alertas serão visíveis apenas na aplicação.

3.1.8 Ranqueamento de Eventos

Com o objetivo de divulgar eventos públicos, foi gerado uma lógica para ranqueamento de eventos, com o intuito de, na página inicial do sistema, exibir listas de eventos organizadas de forma interessante ao usuário. Foram implementados dois tipos de listas:

- Selecionada por distância em relação ao usuário e número de confirmados no evento.

  Neste caso, o cálculo da distância é feito através do uso da gema Geocoder [27], que permite desde a captura da localização até mesmo o cálculo da distância para um certo ponto geográfico. Utilizamos também a API do Google Maps [28] que permite visualizações de mapas e um "auto completar", utilizado no cadastro de eventos que permite adquirir uma latitude e longitude baseados no endereço.

- Selecionada pela quantidade de amigos confirmados no evento.

Nos dois casos, as listas estarão ordenadas pela data de realização do evento, eliminando eventos ocorridos, proporcionando uma visualização clara e objetiva dos eventos que estão por vir.

3.2 Modelos

Nesta seção, será mostrado em maior detalhe os principais modelos do projeto. Os modelos foram a primeira parte a ser construída do sistema, juntamente com seus testes, já que foi utilizada a metodologia TDD.

Para ajudar no entendimento desta seção, a figura 3.2 pode ser utilizada.

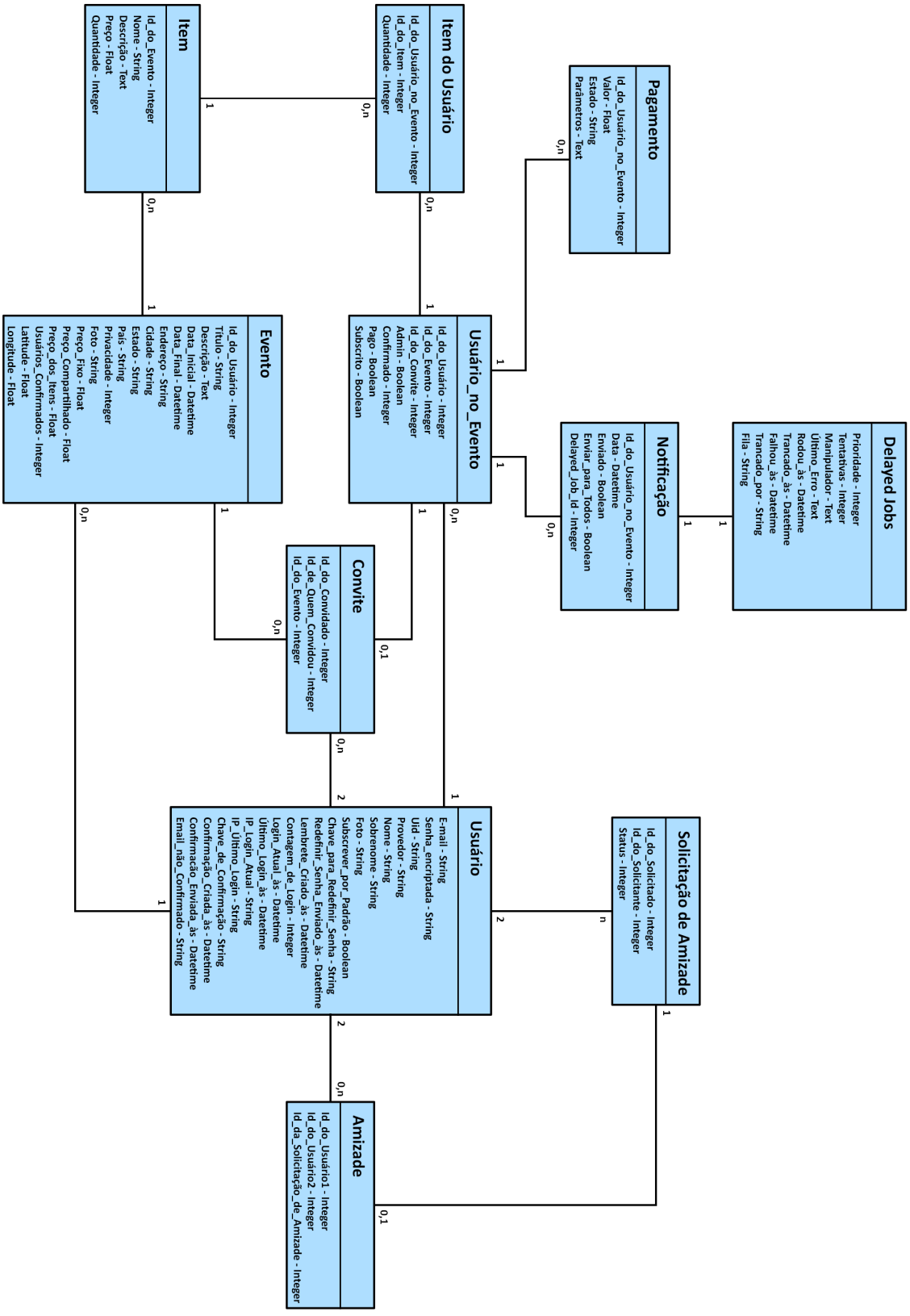

Figura 3.2: Modelo de Dados - Tabelas e Relacionamentos.

3.2.1 Usuário

A tabela de usuário possui diversos campos que foram criados devido às funcionalidades do Devise. Primeiramente, o e-mail e senha, já que são os campos padrão do Devise para identificação do usuário. Além disso, por ter sido utilizada a funcionalidade de confirmação por e-mail, os campos *chave_de_confirmação*, *confirmação_criada_às*, *confirmação_enviada_às* e *email_não_confirmado* foram criados pela gema. Os campos *contagem_de_login*, *login_atual_às*, *último_login_às*, *ip_login_atual* e *ip_último_login* são utilizados pela biblioteca para manter o histórico e segurança. Por último, os campos *chave_para_redefinir_senha* e *redefinir_senha_enviado_às* são utilizadas para, com segurança, ter a funcionalidade de "esqueci minha senha".

Para o bom funcionamento da integração com o *login* do Facebook, são criados os campos *uid* e o *provedor*. O *provedor* simplesmente guarda por qual provedor estamos fazendo o acesso, para o caso de existir o *login* por outros meios além do Facebook. Já o campo *uid* guarda a identificação do usuário dentro da rede a qual ele pertence.

Os outros campos são bastante intuitivos. O único que vale a atenção é o *subscrever_por_padrão*, que tem como objetivo definir se, por padrão, ao entrar num evento o usuário estará subscrito ou não às notificações gerais do mesmo.

3.2.2 Evento

A tabela de eventos é um ponto central do sistema. Como buscas de eventos são bastante comuns, algumas variáveis foram adicionadas à tabela para agilizar esse processo. Entre elas, *cidade*, *estado* e *país* que têm por objetivo restringir os eventos por localidade, *preço_dos_itens* e o número de confirmados para evitar o acesso a outras tabelas em funções muito requisitadas e a *latitude* e *longitude*, para rapidamente poder calcular a distância do usuário até o local do evento.

A validação dos campos de eventos é um ponto importante a ser discutido. Como existem tipos de privacidade e de pagamento, é importante que sempre exista a garantia de que esses dados estão consistentes entre si. Além disso, por possuir informações de valores relacionados a outras tabelas, é sempre importante garantir que ao editar um evento, esses valores estejam sempre corretos.

3.2.3 Usuário no Evento

Essa tabela é a terceira tabela essencial para o funcionamento do sistema, já que é ela que relaciona as duas entidades mais importantes (o evento e o usuário). Todas as tabelas que são relacionadas a um usuário no caso dele estar em um evento, se relacionam diretamente com esta tabela, facilitando a implementação de funções

relacionadas a esta situação, como um pagamento ou uma notificação.

Além dos campos que identificam seus relacionamentos, essa tabela possui alguns campos importantes:

- *Admin*

  Define se esse usuário tem direitos administrativos sobre o evento, como no caso de ser o organizador.

- *Confirmado*

  Deve ter um valor que indique se o usuário vai, não vai, ou ainda não respondeu se vai ao evento.

- *Pago*

  Evita o acesso a todos os pagamentos (ou tentativas mal sucedidas de pagamento) na hora de saber se usuário já pagou. É importante notar que essa variável possui uma validação importante, já que, além de depender da tabela de pagamentos, é uma parte sensível do fluxo do sistema.

- *Subscrito*

  Define se o usuário receberá as notificações gerais do evento ou não. Vale ressaltar que, por padrão, essa variável assume um valor relativo à variável *subscrever_por_padrão* do usuário em questão.

3.2.4 Notificação

A tabela de notificação possui algumas singularidades. Primeiramente, o campo *Enviar_para_Todos* que é o que diferencia uma notificação geral de um evento para uma notificação personalizada do usuário. É importante notar, que mesmo a notificação geral pertence a um *Usuário_no_Evento*, porém, no caso de ser geral, pertencerá ao *Usuário_no_Evento* referente ao organizador do evento.

Além disso, para se integrar com a gema Delayed Jobs citada anteriormente, é necessário que exista um campo que relacione a tabela de notificação com um registro na tabela de *Delayed Jobs*. No momento que este registro na tabela de *Delayed Jobs* disparar, ele vai verificar a notificação correspondente e de fato notificar os usuários. Caso o *Enviar_para_Todos* seja verdadeiro, essa tarefa irá enviar um e-mail para todos os participantes do evento.

3.2.5 Outras Tabelas

Por serem tabelas mais simples e óbvias, não serão explicitados detalhes. Vale notar que as tabelas *Convite*, *Solicitação_de_Amizade* e *Amizade* relacionam dois usuários

diferentes entre si. Cada uma com objetivos diferentes.

3.2.6 Extras

Um ponto importante a ser levantado é o relacionamento entre as tabelas. Alguns registros devem ser automaticamente criados a medida que determinadas ações são disparadas. Por exemplo, uma amizade deve ser criada automaticamente quando uma solicitação de amizade é registrada como aceita. Dessa forma, a lógica geral de integração entre os modelos já estará praticamente pronta, deixando para os controladores apenas as ações que de fato ocorrem com a interação de um usuário com o sistema.

3.3 Controladores

Uma vez tendo os modelos prontos e sendo validados, o projeto já está bem encaminhado e então torna-se necessária a criação dos controladores, que recebem as requisições feitas pelos usuários e definem o que deve ser feito. Como citado anteriormente, durante a implementação do projeto usou-se a metodologia RESTful, tornando os controladores bem simples e bem intuitivos, já que quase sempre possuem apenas ações de *Index*, *Show*, *New*, *Create*, *Edit*, *Update* e *Destroy*. Apenas em dois casos essa lógica não se mantém:

- Busca de evento

  Foi criado uma ação especial *Search* para a busca de eventos, deixando o código mais limpo e organizado.

- Alteração de subscrição

  Por ser uma tarefa bem simples e não necessitar de validação por conta do modelo, foi definida uma ação para alterar a subscrição de um usuário às notificações do evento ao invés de usar o método padrão *Update*.

Um ponto a ser lembrado é que, por padrão, controladores respondem com HTML, porém em casos de requisições AJAX, essa resposta deve ser dada de forma diferente, seja por um objeto JSON, um XML ou até mesmo Javascript diretamente. Inclusive, é possível configurar um controlador para responder a chamadas diferentes com formatos distintos, ou seja, responder tanto para requisições AJAX quanto a requisições HTTP padrões.

3.4 Visões

Para agilizar o desenvolvimento do *frontend* do sistema, foi utilizado o Bootstrap [29], um *framework* de CSS e Javascript muito intuitivo, de fácil utilização e poderoso. Basicamente, o Bootstrap é um conjunto de classes de estilo (CSS) e funções Javascript a serem utilizadas. Um exemplo de um resultado alcançado rapidamente utilizando bootstrap pode ser observado na figura 3.3. Trata-se da página de visualização de um evento dentro do sistema implementado. As barras de navegação (no topo e final da página), estilo dos botões, posicionamento em colunas e linhas, campos de preenchimento, entre outros elementos, estão seguindo o padrão do Bootstrap, sendo perceptível o poder da ferramenta e o motivo do seu uso.

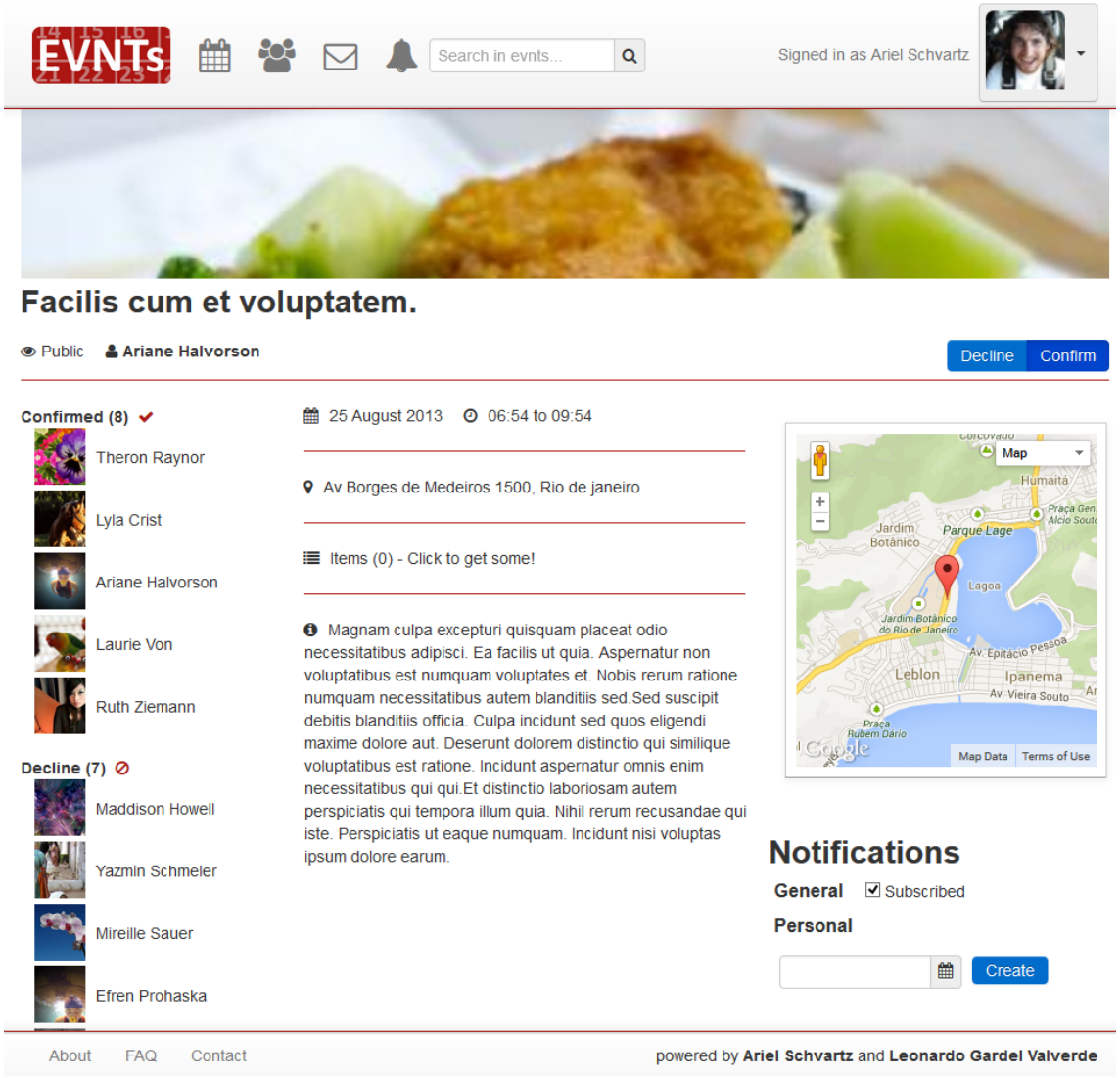

Figura 3.3: Página de Visualização de Evento.

Além disso, para gerar essa visualização interativa do mapa de localização, foi

utilizada a API do Google Maps. Sua utilização é gratuita enquanto o tráfego for pequeno e sua implementação, que se dá através de uma biblioteca em Javascript, é muito fácil e bem documentada.

Outra ferramenta importante para o layout do projeto, foi o Font Awesome [30], que basicamente é biblioteca em CSS que fornece uma fonte que ao invés de conter letras, contém ícones. Dessa forma, os ícones têm qualidade de vetor e são muito leves, já que não são imagens. Seu uso é totalmente gratuito, tornando-o ainda mais atrativo. Ao olhar a figura 3.3, da página do evento, percebe-se que o uso de ícones torna o site ainda mais agradável e bonito.

Outra questão importante sobre as visões foi o alto uso de AJAX e Javascript. O seu uso evita o carregamento repetidas vezes de páginas completas, reduzindo a quantidade de conteúdo carregado sem necessidade. Dessa forma, obtém-se uma melhora na experiência do usuário, que tem uma resposta mais rápida de suas ações. Outro aspecto bom do uso de Javascript e JQuery é a utilização de animações, criando um sistema mais fluido e agradável. Porém, em contrapartida, usuários podem desativar o Javascript de seus navegadores (principalmente em celulares), apesar de não ser muito comum.

3.5 Teste

Foram implementados testes unitários e funcionais, que representam os testes dos modelos e controladores, como explicado anteriormente.

3.5.1 Teste dos Modelos

Os testes de modelo tem como foco:

- Validações

  Verificar se casos não consistentes de modelos realmente estão sendo validados e negados, enquanto casos consistentes são aceitos pelo sistema. Esses testes garantem registros sempre consistentes no BD.

- Chamadas de retorno

  Diversos modelos devem realizar ações antes ou após serem salvos. Por exemplo, um item, ao ser salvo, deve alterar a variável *Preço_dos_Itens* do evento a qual esse item pertence. Neste caso, o valor final da variável deve ser testado para garantir que todas as chamadas estão acontecendo corretamente.

- Valores padrões

Algumas variáveis devem assumir valores padrões quando o registro é inicializado. Por exemplo, um convite deve gerar um *Usuário_no_Evento* com a variável confirmado em um valor padrão para o caso do usuário ainda não ter dado a resposta.

Para realizar os testes de modelo e simular dados consistentes e inconsistentes, foram usadas algumas gemas:

Factory Girl [31]

Permite de forma clara e organizada a criação de registros no banco de dados. Permite a criação de instâncias salvas, não salvas e até mesmo a criação de registros em cadeia, no caso da existência de relacionamentos. A ideia é basicamente criar "fábricas" de registros em um arquivo e nos testes apenas chama-las, criando assim um código limpo e DRY (ver seção 2.4)

FFaker [32]

O FFaker, é uma segunda versão da gema Faker, porém reescrita para rodar de forma mais rápida. A ideia da gema é gerar informação básica aleatória, como por exemplo nomes, e-mails, endereços, textos, entre outros. Desta forma, é possível simular informações pseudoaleatórias para testar o sistema.

Timecop [33]

Essa gema basicamente permite "viajar no tempo" e "parar no tempo". Na verdade, o que acontece é que a gema simula horários possibilitando os testes que são dependentes do tempo, como no caso dos testes das notificações. Dessa forma, é possível "viajar" até a data onde a notificação deveria ser enviada e verificar se tudo correu bem.

3.5.2 Teste dos Controladores

Os testes dos controladores têm como foco:

- Rotas e Redirecionamentos

  Testes que verificam se as rotas existem e estão retornando sucesso. Além disso, devem verificar que em certos casos, o usuário será redirecionado. Por exemplo, ao criar um evento, o usuário deve ser redirecionado para a página de visualização do evento que acabou de ser criado.

- Respostas

Testes que avaliam a resposta dada pela rota, isto é, se a requisição foi por AJAX, a resposta não deve ser HTML. Além disso, é importante verificar as informações que são respondidas. Por exemplo, ao criar um novo item por AJAX, deseja-se como retorno o novo item criado, para adicioná-lo à página sem recarregá-la.

- Permissões

  Testes que avaliam casos de permissões de acesso a páginas específicas. Por exemplo, um usuário que não iniciou sua sessão ou que não faz parte de um evento privado não deveria ser capaz de acessar esse evento.

Para ajudar nestes testes, além das gemas já mencionadas, foi utilizada a gema Mocha [34].

Mocha

Esta gema permite "enganar" o BD, simulando que qualquer instância a ser salva vai ser válida ou não. Isso ajuda muito na hora de testar rotas como o *create* e *update*, simulando casos de sucesso e de falha de forma fácil e rápida.

3.6 Ambientes

Existem três ambientes básicos a serem utilizados pelo sistema. O de desenvolvimento, de testes e de produção. Como o de testes é análogo ao ambiente de desenvolvimento, este não será explicado em detalhes.

3.6.1 Desenvolvimento

Para o ambiente de desenvolvimento, o SGBD utilizado é o SQLite, principalmente por ser muito leve e não precisar de nenhum tipo de configuração para funcionar. Este SGBD se torna suficiente em desenvolvimento, já que a quantidade de dados é baixa e não é necessária uma grande eficiência a nível de BD. Para utilizar o SGBD é necessário instalar a gema referente ao banco.

Neste ambiente, é muito comum o uso do IRB (*Interactive Ruby Shell*) , um prompt de comando do seu sistema, onde você pode acessar o BD, chamar funções, entre outros. Seu uso se dá principalmente pela necessidade de depurar o código.

As gemas utilizadas para o ambiente de desenvolvimento foram:

Pry [35]

O Pry é um substituto do prompt de comando padrão do Rails, o IRB. Suas principais vantagens são a coloração de sintaxe Ruby, a velocidade (rápido comparado ao

IRB) e a possibilidade de invoca-lo em tempo de execução. Esta última funcionalidade se torna extremamente necessária e importante para depurar código, já que a qualquer momento do fluxo de execução é possível travar o sistema e ter comandos Ruby a sua disposição.

Letter Opener [36]

Essa gema permite visualizar diretamente no browser, um e-mail que seria enviado. Isso evita a necessidade de configurar uma conta de e-mail no ambiente de desenvolvimento. Além de agilizar o processo de configuração do sistema e de visualização dos e-mails, a gema ainda evita que diversos e-mails sejam mandados para sua caixa de entrada.

Better Errors

Essa gema faz um trabalho bastante simples mas que ajuda muito na depuração de erros. Basicamente, a ideia é melhorar as páginas de erros. Ao invés de simplesmente uma mensagem jogada na tela, outras informações como inspeção de variáveis também são mostradas e a visualização do erro é muito mais agradável e de fácil leitura.

3.6.2 Produção

Para produção, utilizamos o Heroku [37], uma plataforma de aplicativos na nuvem que permite aos desenvolvedores se focarem somente no desenvolvimento de código, sem ter de se preocupar com gerenciamento de servidores, implantação ou escala. Além disso, existe um plano gratuito, obviamente limitado, porém completo e bastante útil para testar com usuários reais em um primeiro momento. Para garantir a facilidade no momento de implantação, esta se dá através do próprio GIT se tornando rápida e fácil.

Por padrão, o Heroku usa como SGBD o Postgres, que é um banco de dados robusto e eficiente quando comparado a outros SGBDs. Portanto, foi instalada a gema do Postgres para se adequar aos requisitos do sistema.

Capítulo 4

Modelo de Negócios

O modelo de negócios de uma empresa basicamente define onde uma empresa executa e vende seus produtos e (ou) serviços. Além disso, deve mostrar o valor que o produto (ou serviço) oferece para os seus clientes, definindo de que forma a empresa lucrará ao entregar esse valor para seus consumidores. Em outra palavras, a empresa irá definir o que fazer, porque fazer e como fazer, traçando uma linha de objetivos e caminhos a serem seguidos pela empresa. O modelo é útil tanto para empresas grandes que precisam reduzir custos, redefinir estratégias, entre outros, quanto para empresas iniciantes, que precisam definir suas diretrizes e, muitas vezes, de um grande documento que apresente os recursos necessários para dar início à empresa [38].

Existem alguns métodos para se definir um modelo de negócios, sendo que em sua maioria, são técnicas complementares. A mais conservadora é a criação de um plano de negócios, técnica muito difundida e utilizada. Porém, nos dias de hoje, uma outra técnica vem ganhando muito espaço, o BMC (*Business Model Canvas*). A seguir serão apresentadas vantagens e desvantagens de cada um destes métodos, visando permitir uma melhor análise comparativa entre eles [39] [40].

4.1 Plano de Negócios

O plano de negócios tem como objetivo especificar em detalhes o negócio que está sendo analisado. Por esse motivo, ao final do processo é gerado um documento muito extenso e explicativo relativo a todas as áreas da empresa. Além de uma explicação detalhada da empresa, seus objetivos e propostas de valor, sua produção pressupõe a criação de um plano financeiro, plano de marketing, plano de recursos humanos e um cronograma completo.

4.1.1 Vantagens

- Facilidade na aquisição de investimentos para o negócio, já que, além de mais detalhes, apresenta um plano financeiro, explicitando gastos e previsões sobre o tempo de retorno.
- Ajuda a prever todas as áreas da empresa, tendo uma visão macro do negócio e aumentando o entendimento dos sócios em cada área.

4.1.2 Desvantagens

- É um processo muito demorado e requer muita pesquisa. Pode demorar semanas ou até meses para ser criado.
- Muitas vezes, os sócios ainda não têm conhecimento sobre algumas hipóteses, podendo levar à criação de um plano inconsistente e irreal.
- Diversas variáveis mudam com o tempo e são imprevisíveis, de forma que o plano pode se desatualizar rapidamente.
- Os sócios, em geral, acabam utilizando informações incorretas ou não embasadas em diversos pontos do plano, gerando um documento muitas vezes inconsistente.
- Se ocorre uma mudança na empresa, é necessário mudar todo o plano, tendo um custo de atualização altíssimo.

4.2 BMC

Inicialmente proposto por Alexander Osterwalder, o BMC visa produzir o modelo de negócios rapidamente e de forma dinâmica. O documento final é de apenas uma folha pré-formatada com nove blocos que definem o modelo de negócios. São eles: principais atividades, principais recursos, rede de parceiros, proposição de valor, segmentos de clientes, canais, relacionamento com o cliente, estrutura de custos e fluxos de receita. A ideia sugere a utilização de *post-its* para que definições possam ser alteradas com facilidade. [41]

4.2.1 Vantagens

- Velocidade de produção, já que contém apenas uma folha com um padrão claro de preenchimento.

- Modelo preparado para mudanças ao longo do processo, considerando que um empreendedor nem sempre já tem de início o conhecimento necessário na área onde estará atuando.
- Cria um documento enxuto, isto é, focado apenas no que é mais importante, evitando perda de tempo com detalhes que têm grandes chances de serem alterados.
- Por estar sempre sendo revisado, propicia a inovação e melhoria das ideias sobre a empresa.

4.2.2 Desvantagens

- Por ser um documento de criação simples e rápida, muito vezes acaba sendo superficial.
- Pode conter dados inconsistentes, visto que os criadores nem sempre estudam e pesquisam a respeito da área de atuação.
- Ainda não é visto por investidores como um substituto do plano de negócios, tornando necessário o desenvolvimento do plano mais completo na hora de angariar recursos.

4.3 Blocos do BMC

4.3.1 Principais Atividades

Este bloco visa definir todas as atividades que são necessárias para executar o modelo de negócios da empresa. Basicamente, estas atividades fazem com que todas as informações descritas nos outros blocos sejam alcançadas.

4.3.2 Principais Recursos

Esta área do documento tem como objetivo explicitar todos os recursos necessários para entregar o devido valor para seus clientes e ter uma renda em cima disso. Os itens dessa seção devem incluir recursos tangíveis e até mesmo os intangíveis. Em geral existem quatro categorias de recursos:

- Recursos Físicos

  Envolvem máquinas, escritórios, instalações, entre outros.

- Recursos Intelectuais

  Neste são consideradas marcas, patentes ou informações (em um BD, por exemplo). Apesar da dificuldade de se gerar recursos intelectuais, uma vez gerados, são muito valiosos.

- Recursos Humanos

  Como o próprio nome já diz, visa definir os empregados necessários para o funcionamento da empresa.

- Recursos Financeiros

  Definem valores como investimentos ou linhas de crédito, necessários para tirar a empresa do papel e colocá-la para funcionar.

4.3.3 Rede de Parceiros

A rede de parceiros tem como objetivo definir quem são fornecedores e parceiros que fazem o modelo funcionar melhor. Isso pode envolver possíveis parcerias com não-concorrentes, concorrentes, fornecedores e compradores. Como motivação para as parcerias, pode-se considerar:

- Otmização e escala devido à melhor alocação de recursos e redução de custos.
- Para tentar minimizar o risco e as incertezas do projeto, é comum a parceria entre empresas concorrentes.
- Ao invés de desenvolver novas atividades na empresa, muitas vezes é melhor focar no que se sabe fazer, e criar uma parceria com quem o complemente. Então, ao criar parcerias, estendem-se a capacidade e as atividades da empresa.

4.3.4 Proposição de Valor

Esta seção é alma do documento. Nela está descrita a proposta de valor entregue ao cliente através do produto, ou seja, é o valor observado pelo cliente no produto ou serviço oferecido. Consequentemente, é determinante na aquisição de novos clientes. Diversos elementos podem contribuir à proposta de valor, como: inovação, design, preço, usabilidade, entre outros.

4.3.5 Segmentos de Clientes

Essa área visa definir quem são os consumidores e clientes da empresa. Numa situação de um crescimento muito grande da base de clientes, se torna interessante dividí-los em segmentos de forma a atendê-los melhor. Os segmentos podem ser de

massa (não há distinção de clientes), de nicho (focado em uma área de atuação), diversificados (que atendem diversas áreas) ou até mesmo mercados multilaterais (que trabalham com mais de um segmentos para funcionar).

4.3.6 Canais

Como o próprio nome já diz, visa definir os canais de comunicação, distribuição e venda do produto ou serviço. Eles são muito importantes para saber com clareza por onde o consumidor irá conhecer o produto, onde irá adquirí-lo e por onde poderá dar um feedback. É importante avaliar por onde aumentar a base de clientes, como de fato passar a proposta de valor ao cliente, como entregar esse valor e como dar suporte ao consumidor. Dessa forma, todo o canal de comunicação estará definido.

4.3.7 Relacionamento com o Cliente

Este bloco define que tipo de relacionamento com o cliente ocorrerá. Existem diversas possibilidades, como uma assistência pessoal, comunidades (consumidores se ajudam) ou até mesmo por meio de auto ajuda, onde a empresa oferece meios ao consumidor para descobrir sozinho o que precisa.

4.3.8 Estrutura de Custos

Esta parte do BMC tenta definir quais são os gastos necessários para o bom funcionamento da empresa. Além disso, tenta encontrar recursos e atividades que são muito caros. Se o resto do documento já estiver preenchido, se torna mais fácil definir a estrutura de custos. Basicamente, existe em empresas focadas em reduzir custos e empresas focadas em agregar valor ao seu produto ou serviço. As duas formas visam aumentar o lucro, porém de maneira oposta.

É importante também dar o devido valor ao tipo de custo, isto é, se é um custo fixo, onde os custos se mantém ao longo do tempo, ou variável, que variam de acordo com o volume de produto ou serviço.

4.3.9 Fluxos de Receita

Fluxos de receita definem a renda gerada por cada segmento de clientes, definindo quanto cada um deles está disposto a pagar pela proposta de valor. As fontes de receita podem ser geradas de diversas formas, desde uma venda específica, venda de assinaturas, venda de anúncios ou até mesmo licenciando o produto ou serviço da empresa.

4.4 Modelo de Negócios do EVNTs

Neste capítulo, será apresentado o modelo de negócios do projeto. Primeiramente, analisando as vantagens e desvantagens, foi escolhido aplicar o BMC, já que o trabalho é focado em desenvolvimento ágil. Além disso, o BMC é interessante por facilitar modificações futuras, que certamente ocorrerão. O BMC produzido pode ser visualizado na 4.1.

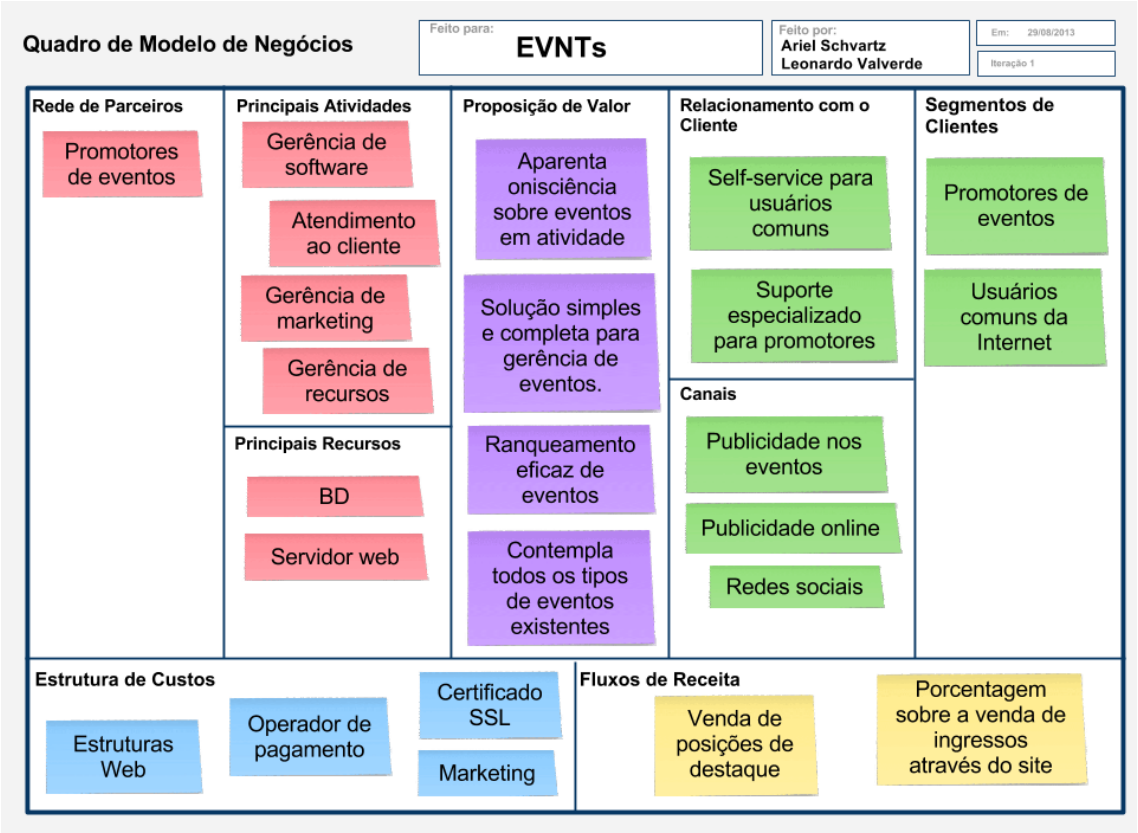

Figura 4.1: BMC do projeto

Como o BMC é uma representação muito sucinta do negócio, será realizado a seguir um detalhamento de cada tópico incluído no mesmo.

- Proposições de Valor
  - Aparenta onisciência sobre os eventos em atividade

    Obviamente, é essencial para o sucesso da aplicação que nela estejam cadastrados muitos eventos. De forma ideal, pode-se imaginar o caso extremo, onde todos os eventos existentes estão mapeados na aplicação.
  - Ranqueamento eficaz de eventos

De nada adianta mapear boa parte dos eventos existentes sem um ranqueamento eficiente. É humanamente impossível detectar eventos interessantes dentre uma lista não ordenada contendo todos os eventos em atividade. A ferramenta de ranqueamento é vital para o sucesso da aplicação.

  - Contempla todos os tipos de eventos existentes

    Como citado anteriormente, uma das grandes deficiências dos sistemas vigentes que auxiliam na gerência de eventos é a sua pouca versatilidade. A aplicação deve dar suporte aos diferentes tipos de eventos existentes, de forma a permitir manutebilidade ao seu gerenciamento.

  - Solução simples e completa para a gerência de eventos

    Um dos principais focos é oferecer todas as ferramentas demandadas pelo usuário, no que diz respeito à gerência de eventos. A ideia é que os usuários não precisem utilizar nenhuma outra plataforma para realizar tal tarefa.

- Principais Recursos

  - Banco de Dados

    O Banco de Dados é um recurso essencial para o funcionamento do sistema. Além disso, é responsável pelo armazenamento de informações, sendo um recurso intelectual de extrema importância.

  - Servidor Web

    Além do BD, o servidor é o outro pilar essencial para que um sistema web funcione plenamente. Dessa forma, se torna outro recurso essencial para o modelo de negócios. Sua escolha afeta a manutenção, custo e escalabilidade do projeto.

- Segmentos de Clientes

  - Usuários Comuns da Internet

    O aplicativo foca em prover funcionalidades úteis a qualquer pessoa. Ou seja, em um primeiro momento, qualquer um que esteja conectado à Internet poderá se interessar por usar o EVNTs, caracterizando-se assim, um segmento de massa.

  - Promotores de Eventos

    Os promotores de eventos são usuários organizadores de grandes eventos públicos e pagos, sendo assim extremamente valiosos para o projeto. Dessa forma, são os usuários que, provavelmente, criarão os eventos com a

maior quantidade de pessoas confirmadas dentro do sistema. Assim, além de serem grandes geradores de renda (ver a parte de fluxos de receita) serão atrativos para novos usuários.

- Rede de Parceiros

  - Promotores de Eventos

    Além de usuários do sistema, organizando eventos, os promotores podem ser considerados também parceiros. São pessoas de suma importância para a criação de uma grande base de eventos, pois através de uma relação de benefício mútuo, os promotores encontram no sistema um local para divulgação e venda de ingressos, enquanto atraem novos clientes a este.

- Canais

  - Publicidade nos Eventos

    Este canal visa fazer um contato *offline* com os clientes. A ideia é que em grandes festas, shows e outros eventos públicos, sejam colocadas propagandas (como cartazes e banners) de forma a mostrar aos usuários, por exemplo, que poderiam ter adquirido ingressos por meio da aplicação.

  - Publicidade *Online*

    Uma das formas mais utilizadas para divulgação de aplicativos web é a realização desta por meio de propagandas em sites. É um método extremamente eficiente, pois basta um clique para que o usuário acesse o sistema. Existem empresas muito populares que proveem este serviço, proporcionando ao contratante um grau de visibilidade proporcional ao pagamento realizado.

  - Redes Sociais

    As redes sociais vêm cada vez mais se tornando o principal meio de divulgação para micro e pequenas empresas, já que são baratas (ou até mesmo gratuitas) e, com dedicação, é possível uma boa alcançabilidade.

- Relacionamento com o Cliente

  - Auto ajuda para usuários comuns

    Em geral, como teremos uma gama muito grande de usuários, torna-se muito custosa a criação de um sistema de suporte pessoal. Porém, por dispor de diversas funcionalidades inéditas pelos usuários, é importante que se estabeleça um relacionamento através de um sistema auto ajuda, como um FAQ (*Frequently Asked Questions*) para auxiliá-los.

- Suporte Especializado para Promotores

  Por terem uma importância muito grande para o sistema, é essencial oferecer atendimento especial aos promotores, além de suporte personalizado, de forma a deixá-los a vontade com o sistema e sem dúvidas sobre o funcionamento do mesmo.

- Principais atividades

  - Gerência de *Software*

    Se faz necessária uma equipe de desenvolvimento e manutenção do *software*, para garantir o seu bom funcionamento, além de proporcionar melhorias ao mesmo.

  - Atendimento ao Cliente

    É necessária uma equipe voltada para o relacionamento com os clientes, citado anteriormente. Isso inclui esforços tanto para a manutenção do sistema de auto ajuda como para o contato direto com os promotores.

  - Gerência de Marketing

    Equipe especializada em estudar locais (*online* e *offline*) para divulgação do sistema, além de criar campanhas nos mesmos, inclusive em redes sociais.

  - Gerência de Recursos

    É essencial que o caixa da empresa seja gerenciado de forma inteligente. Para isso é necessário que haja uma equipe voltada para a gerência de recursos do projeto, distribuindo-os ao longo dos demais setores da empresa.

- Fluxos de Receita

  - Porcentagem sobre a Venda de Ingressos

    A principal fonte de receita da empresa, se baseia na venda de ingressos. Basicamente, seria cobrado um percentual do valor do ingresso pelos serviços sendo prestados. O organizador, pode optar por incluir esse valor ao preço do ingresso ou deixar que o usuário pague por ele.

  - Venda de Posições de Destaque

    Outra forma de gerar receita, seria através da venda de posições de destaque para eventos. Com isso, haveria uma área no site de "Eventos em Destaque" com os eventos que pagaram para ter aquela posição especial.

- Estrutura de Custos

– Estruturas Web

  Os gastos em estruturas web basicamente são representados com: servidor, banco de dados e domínio.

– Operador de Pagamento

  O operador de pagamento, em um primeiro momento, será o PayPal por seu selo de segurança e praticidade de implementação. Porém, é um custo muito alto para a empresa, já que o valor cobrado é de 5,4% a 6,4% + R$0,60 por transação. É importante ter em mente que se faz necessária a pesquisa por outras soluções de pagamento, para não inviabilizar o projeto.

– Certificado SSL

  Como explicado na parte de segurança do Rails, é importante o uso de conexões SSL, para que informações se tornem criptografadas e, consequentemente, seguras. Para que o usuário saiba que um site é criptografado, existem empresas que vendem certificados SSL, atestando a segurança do mesmo.

– Marketing

  Marketing é um dos principais gastos de qualquer empresa que quer crescer rapidamente. Além dos gastos com uma equipe, existem diversos outros relacionados aos meios de divulgação e a produção de campanhas.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

O *framework* Ruby on Rails proporcionou o desenvolvimento da aplicação de forma extremamente ágil. O tempo para produção desta poderia ser ainda mais reduzido, caso não fosse preciso um tempo de aprendizado para utilização de diversas ferramentas e metodologias. Porém, vale ressaltar que ambos os programadores do projeto possuíam razoável experiência na utilização deste *framework*, tendo passado por um processo de aprendizado relativamente longo. Desta forma, este projeto razoavelmente complexo pôde ser produzido em um número reduzido de horas de trabalho.

Técnicas como o TDD contribuíram enormemente para a produção de um código claro e manutenível, por todas as razões elucidadas anteriormente. Foi percebida uma dificuldade inicial na adoção deste método, já que os desenvolvedores não estavam habituados à realização de testes de forma precoce. Entretanto, a sua utilização proporcionou uma experiência positiva, deixando claras as vantagens da utilização desta técnica.

Ao longo do curso de graduação, foi introduzido apenas o conceito de plano de negócio. Porém, após pesquisas, foi descoberto que o BMC é uma técnica mais atual e interessante para empresas iniciantes. O seu aprendizado e realização foi uma experiência extremamente valiosa, pois possibilita o empreendimento de forma ágil.

Pode-se concluir através dos resultados demonstrados pelo BMC que o projeto possui um bom potencial de rentabilidade. Desta forma, pretende-se dar continuidade a este empreendimento visando torná-lo público em breve. Entretanto, para que isso seja possível se fazem necessários alguns ajustes, que serão enumerados na seção a seguir.

5.2 Trabalhos Futuros

5.2.1 Pagamento

A funcionalidade de pagamento do sistema, apesar de implementada, não está otimizada, sendo necessárias melhorias:

- Transferência Automática de Verba para Organizadores

  Como citado anteriormente, um gargalo para a escalabilidade do projeto é a necessidade de transferência manual de verbas para os organizadores dos eventos. Devido à dificuldade de implementação, esta característica foi deixada para o futuro.

- Fraudes

  Existe uma série de pontos de falha que deve ser cuidadosamente coberta. Um deles consiste na criação de eventos falsos com a finalidade de angariar fundos dos usuários por meio da funcionalidade de pagamento. A princípio, este problema pode ser sanado através de uma verificação presencial a respeito da existência do evento, porém, este método prejudica enormemente a escalabilidade do projeto. Diversas possíveis soluções foram pensadas para evitar estes e outros tipos de fraudes, porém todas possuem brechas, sendo necessária uma análise de risco mais aprofundada.

- Sistema Próprio de Pagamento

  A utilização de sistemas de pagamento terceirizados (como o PayPal) são muito custosos ao empreendimento. A complexidade da implementação de um sistema de pagamentos "online" inviabilizou a sua execução até o momento, porém, é sabido que este é um ponto chave para o sucesso do projeto.

5.2.2 Segurança

A aplicação ainda possui diversas brechas de segurança, que precisam ser identificadas e combatidas. É necessário um longo trabalho de aprimoramento até que tenhamos uma versão de fato segura.

5.2.3 Parcerias e Sociedade

- Promotores

  Como já dito anteriormente, as parcerias ajudam no alavancamento do site. Então, para um crescimento acelerado, um dos próximos passos seria a busca por promotores interessados.

- Designers

  Uma parte que ainda está precária no sistema devido à falta de conhecimento por parte dos desenvolvedores, é a parte de design. Dessa forma, torna-se importante a parceria ou sociedade com algum designer ou empresa do ramo, visando a produção de um resultado ainda melhor.

- Investidores

  Outro foco que deve ser dado aos trabalhos futuros, é a busca por recursos financeiros, já que, apesar de baixos, o empreendimento tem custos que precisam ser sanados. Além disso, seria interessante que os criadores pudessem dar atenção total ao projeto, sendo necessário um salário para mantê-los.

Além destas, também são necessárias outras parcerias, como para gerência administrativa e de marketing do projeto. Porém estas só são necessárias em uma fase mais avançada do empreendimento, já que inicialmente não existe fluxo de caixa que torne-as indispensáveis.

Referências Bibliográficas

[1] EVANS, M. "The Evolution of the Web - From Web 1.0 to Web 4.0". . http://www.cscan.org/presentations/08-11-06-MikeEvans-Web.pdf. Acessado em agosto de 2013.

[2] YOUTUBE. "Evolution Web 1.0, Web 2.0 to Web 3.0". . http://www.youtube.com/watch?v=bsNcjya56v8, . Acessado em agosto de 2013.

[3] UOL ECONOMIA - COTAÇÕES. "Ações do Facebook saltam quase 30milhões". . http://economia.uol.com.br/cotacoes/noticias/redacao/2013/07/25/acoes-do-facebook-saltam-quase-30-apos-lucro-de-us-333-milhoes.htm, jul. 2013. Acessado em agosto de 2013.

[4] WIKIPEDIA. "Software framework". . http://en.wikipedia.org/wiki/Software_framework, aug. 2013. Acessado em agosto de 2013.

[5] DOCFORGE - SOFTWARE DEVELOPMENT RESOURCES. "Web application framework". . http://docforge.com/wiki/Web_application_framework, jul. 2013. Acessado em agosto de 2013.

[6] WIKIPEDIA - THE FREE ENCYCLOPEDIA. "Hypertext Transfer Protocol". . http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, ago. 2013. Acessado em agosto de 2013.

[7] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Relatório técnico, 2000.

[8] WIKIPEDIA - THE FREE ENCYCLOPEDIA. "HTML". . http://en.wikipedia.org/wiki/HTML, ago. 2013. Acessado em agosto de 2013.

[9] YOUTUBE. "What is HTML5?" . http://www.youtube.com/watch?v=mzPxo7Y6JyA, . Acessado em agosto de 2013.

[10] FERNÁNDEZ-VILLAMOR, J. I., DIÁS-CASILLAS, L., IGLESIAS, C. "A Comparison Model for Agile Web Frameworks". .

http://www.gsi.dit.upm.es/administrator/components/com_jresearch/files/publications/frameworks.pdf.

[11] GOOGLE. "Google Trends - Web Search interest". . http://www.google.com/trends/explore#q=symfony,ruby+on+rails,cakephp,django+python,asp.net+mvc, . Acessado em agosto de 2013.

[12] MOECK, G. "Sproutcore MVC VS Rails MVC". . http://gmoeck.github.io/2011/03/10/sproutcore-mvc-vs-rails-mvc.html. Acessado em agosto de 2013.

[13] OLSEN, R. *Eloquent Ruby*. Addison-Wesley Professional, 2011. ISBN: 9780321584106.

[14] RAILS GUIDES. "Rails Guides - Ruby on Rails Security Guide". . http://guides.rubyonrails.org/security.html. Acessado em agosto de 2013.

[15] RSPEC. "RSpec". . http://rspec.info/. Acessado em agosto de 2013.

[16] METZ, S. *Practical Object-Oriented Design in Ruby - An Agile Primer*. Addison-Wesley Professional, 2012. ISBN: 9780321721334.

[17] GIT. "Git –everything-is-local". . http://git-scm.com/. Acessado em agosto de 2013.

[18] BITBUCKET. "Bitbucket". . https://bitbucket.org/. Acessado em agosto de 2013.

[19] SKYPE. "Skype". . http://www.skype.com/. Acessado em agosto de 2013.

[20] LINDSAY, JEFF. "localtunnel". . http://progrium.com/localtunnel/. Acessado em agosto de 2013.

[21] PLATAFORMATEC. "Github - Devise". . https://github.com/plataformatec/devise. Acessado em agosto de 2013.

[22] INK. "Filepicker". . https://www.inkfilepicker.com/. Acessado em agosto de 2013.

[23] DODWELL, M. "Omniauth-Facebook". . https://github.com/mkdynamic/omniauth-facebook. Acessado em agosto de 2013.

[24] INTRIDEA, INC. "Omniauth". . https://github.com/intridea/omniauth. Acessado em agosto de 2013.

[25] PAYPAL. “Ferramenta de Pagamento”. . https://www.paypal.com/br/webapps/mpp/home. Acessado em agosto de 2013.

[26] COLLECTIVE IDEA. “Delayed Jobs”. . https://github.com/collectiveidea/delayed_job. Acessado em agosto de 2013.

[27] REISNER, A. “Geocoder”. . https://github.com/alexreisner/geocoder. Acessado em agosto de 2013.

[28] GOOGLE. “Google Maps API”. . https://developers.google.com/maps/?hl=pt-br, . Acessado em agosto de 2013.

[29] OTTO, M., THORNTON, J. “Bootstrap”. . http://getbootstrap.com/2.3.2/. Acessado em agosto de 2013.

[30] GANDY, D. “Font Awesome”. . http://fortawesome.github.io/Font-Awesome/. Acessado em agosto de 2013.

[31] THOUGHTBOT, INC. “Factory Girl”. . https://github.com/thoughtbot/factory_girl_rails. Acessado em agosto de 2013.

[32] OGA, E. “FFaker”. . https://github.com/EmmanuelOga/ffaker. Acessado em agosto de 2013.

[33] JEFFERY, T. “Timecop”. . https://github.com/travisjeffery/timecop. Acessado em agosto de 2013.

[34] FREERANGE. “Mocha”. . https://github.com/freerange/mocha. Acessado em agosto de 2013.

[35] MAIR, J. “Pry”. . http://pryrepl.org/. Acessado em agosto de 2013.

[36] BATES, R. “Letter Opener”. . https://github.com/ryanb/letter_opener. Acessado em agosto de 2013.

[37] HEROKU. “Heroku”. . https://www.heroku.com/. Acessado em agosto de 2013.

[38] REBOUÇAS, F. “Modelo de Negócio”. . http://www.infoescola.com/administracao_/modelo-de-negocio/. Acessado em agosto de 2013.

[39] GHISI, J. “Business Model Canvas x Plano de negócios”. . http://catarinasdesign.com.br/blog/2013/08/business-model-canvas-um-atalho-antes-do-seu-plano-de-negocio/. Acessado em agosto de 2013.

[40] DORNELAS, J. "Modelo de negócios Canvas ou plano de negócios". . http://economia.uol.com.br/ultimas-noticias/colunistas/jose-dornelas/2013/07/08/modelo-de-negocios-canvas-ou-plano-de-negocios.htm. Acessado em agosto de 2013.

[41] SUA IDEIA VALE UM MILHÃO. "Elaborando seu Modelo de Negócios: Business Model Canvas – Parte 1". . http://suaideiavale1milhao.com.br/saiba-tudo-sobre-business-model-canvas-parte-1/. Acessado em agosto de 2013.