

**FACULDADE PITÁGORAS – UNIDADE DIVINÓPOLIS
LEONARDO FARIA COELHO**

AUTOSIMULADO: aplicativo *web* para testes sobre legislação de trânsito

**DIVINÓPOLIS
2009**

LEONARDO FARIA COELHO

AUTOSIMULADO: aplicativo *web* para testes sobre legislação de trânsito

Monografia apresentada à Faculdade Pitágoras – Unidade Divinópolis, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação

Orientador: Prof. Daniel Moraes dos Reis

DIVINÓPOLIS
2009

Monografia intitulada “*AUTOSIMULADO: aplicativo web para testes sobre legislação de trânsito*”, de autoria do graduando Leonardo Faria Coelho, defendida em 5 de dezembro de 2009, avaliada pela banca examinadora constituída pelos seguintes professores:

Prof. Daniel Morais dos Reis - Orientador

Prof. Alexandre Dezem Bertozzi - Avaliador 1

Prof^a. Dayana B. Cruz Rodrigues - Avaliador 2

**DIVINÓPOLIS
DEZEMBRO, 2009**

Dedico esse trabalho a todos que apoiaram meus devaneios e anseios, que me inspiraram e me criticaram e que acima de tudo entenderam minha hiperatividade.

AGRADECIMENTOS

Agradeço àqueles que me apoiaram: minha mãe pelo apoio incondicional; o prof. Daniel Morais pelo apoio e orientação. Agradeço todo corpo docente da Faculdade Pitágoras por nortear meus conhecimentos e aos usuários do serviço aqui documentado.

“Innovation distinguishes between a leader and a follower” – Steve Jobs.

RESUMO

A evolução da tecnologia e a popularização da informática e da Internet permitem as pessoas novas formas de aprendizagem e a troca de conhecimento independentemente do lugar e da hora. Assim, qualquer assunto pode ser compartilhado das mais diversas maneiras - texto, imagem, vídeo, áudio - em uma estrutura não-hierárquica e onde conteúdo é a palavra-chave. O autosimulado é um aplicativo *Web 2.0* para auxiliar as pessoas na obtenção da carteira de motorista, oferecendo um espaço onde é possível fazer e discutir testes de legislação de trânsito. São centenas de questões de legislação de trânsito, sinalização, direção defensiva, primeiros socorros, meio ambiente e mecânica, disponíveis em uma interface minimalista e grátis para seus usuários. A missão do autosimulado não é substituir o ensino tradicional do trânsito mas sim oferecer uma alternativa para completar a aprendizagem do candidato a motorista. Essa monografia apresenta as inspirações, metodologias e conceitos do serviço. Em seguida é apresentado os métodos de desenvolvimento e o modelo de negócio por trás do aplicativo.

PALAVRAS-CHAVE: Aplicativo *web*, Simulado, *Ruby*, *Rails*, *Web 2.0*, *Getting Real*, Empreendedorismo.

ABSTRACT

The technology evolution and the computer and Internet popularization have allowed people to get new ways of learning and the knowledge exchange disregardless the place and time. So, any subject can be shared in many ways - text, image, video, audio - in a nonhierarchical structure where content is the main keyword. The autosimulado website is a web 2.0 application to help people to get their car's license, offering a place where it is possible to test and discuss traffics laws tests. There are hundreds of issues of traffic laws, signs, defensive driving, first aids, environment and mechanical, available in a free and minimalist interface to the users. The mission of Autosimulado is not to replace the traditional teaching way but to offer an alternative way to improve the learning of the prospective drivers. This monograph presents the insights, methodologies and the service concepts . Afterwards it is presented the development methods and business model over the application.

KEYWORDS: web application, Simulado, Ruby, Rails, Web 2.0, Getting Real, entrepreneurship.

LISTA DE ABREVIATURAS E SIGLAS

Ajax - Asynchronous Javascript And XML

API - Application Programming Interface

CNH - Carteira Nacional de Habilitação

CSS - Cascading Style Sheets

DETRAN - Departamento de Trânsito

DSL - Linguagem Específica de Domínio

FTP - File Transfer Protocol

HTML - HyperText Markup Language

MVC - Model, View, Controller

RoR - Ruby on Rails

RSS - Really Simple Syndication

SSH - Secure Shell

SQL - Structured Query Language

XHTML - eXtensible HyperText Markup Language

XML - eXtensible Markup Language

URL - Universal Resource Locator

LISTA DE ILUSTRAÇÕES

Figura 1 - Logomarca do autosimulado.....	13
Figura 2 - Estrutura de um <i>site</i>	16
Figura 3 - Olá mundo	18
Figura 4 - Olá mundo com <i>CSS</i>	19
Figura 5 - <i>Ajax</i>	21
Figura 6 - O padrão de projeto <i>MVC</i>	26
Figura 7 - Ciclo de Requisições no <i>Ruby on Rails</i>	28
Figura 8 - Estrutura de um projeto <i>Rails</i>	30
Figura 9 - <i>Screenshot</i> da <i>homepage</i> do autosimulado	35
Figura 10 - Diagrama de Entidade Relacionamento	36
Figura 11 - <i>Model exam.rb</i>	38
Figura 12 - Exemplo de <i>migration</i>	39
Figura 13 - Página de estatísticas.....	43

LISTA DE TABELAS

Tabela 1 - Comparativo das sintaxes de <i>Java</i> e <i>Ruby</i>	24
--	----

SUMÁRIO

1 INTRODUÇÃO	13
1.1 Objetivos.....	14
1.1.1 Objetivo Geral	14
1.1.2 Objetivos Específicos.....	14
1.2 Estrutura do trabalho	15
2 METODOLOGIAS	16
2.1 Funcionamento do aplicativo <i>web</i>	16
2.1.1 <i>XHTML</i>	17
2.1.2 <i>CSS</i>	18
2.1.3 <i>Javascript</i>	20
2.1.4 <i>Ajax</i>	20
2.2 <i>Ruby</i>	22
2.2.1 <i>Frameworks Ruby</i>	24
2.3 <i>Ruby on Rails</i>	25
2.3.1 <i>MVC</i>	25
2.3.2 Recursos e características.....	26
2.3.3 Estrutura de um aplicativo <i>Ruby on Rails</i>	30
2.4 <i>Web 2.0</i>	31
2.5 <i>MySQL</i>	32
2.6 Metodologia de desenvolvimento	32
3 DESENVOLVIMENTO	33
3.1 Produto	33
3.2 Levantamento de requisitos e funcionalidades.....	33
3.3 <i>Layout</i>	34
3.4 Base de dados.....	35
3.5 Codificação	37
3.6 Controle de versões.....	39
3.7 Testes	40
3.8 Implantação	40
3.9 Atualizações	41

	12
3.10 Estatísticas	42
3.10.1 SQL	42
4 CONSIDERAÇÕES FINAIS	44
APÊNDICE A - <i>SCREENSHOTS</i> DO APLICATIVO	47
APÊNDICE B - MODELO DE NEGÓCIO	51
ANEXO A - ESTATÍSTICAS COM <i>GOOGLE ANALYTICS</i>	52

1 INTRODUÇÃO

O Código Brasileiro de Trânsito determina que os candidatos a obtenção da Carteira Nacional de Habilitação devem passar por três avaliações. O primeiro teste - psicotécnico - avalia a saúde física e mental do candidato. O segundo teste - de legislação - consiste em uma prova escrita, com 30 questões, onde são avaliados os conhecimentos de legislação de trânsito, sinalização, direção defensiva, primeiros socorros, meio ambiente e noções de mecânica. O exame de direção é o último dos três testes, onde o candidato é avaliado se está apto ou não para conduzir veículos nas vias públicas.

O candidato a obtenção da Carteira de Motorista se prepara para os testes de legislação e direção em um Centro de Formação de Condutores, que é uma empresa de treinamentos autorizada pelo DETRAN - órgão competente de trânsito.

O ensino da legislação de trânsito nos Centros de Formação de Condutores consiste geralmente em explicações detalhadas e exemplificadas da lei 9503/97 – conhecido como Código Nacional de Trânsito. Alguns candidatos não se adaptam a essa forma de aprendizagem e para resolver essa questão são necessárias novas formas de ensino que complementam esse modelo de educação.



Figura 1 - logomarca do autosimulado
Fonte: Autor da monografia

Autosimulado, FIG. 1, foi o nome escolhido para o aplicativo que foi desenvolvido nesse trabalho de graduação. O serviço oferece ao usuário um novo meio de aprendizado da legislação de trânsito, onde será possível realizar simulados on-line gratuitamente e discutir a matéria com outros usuários do aplicativo.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral desse trabalho é criar uma aplicação *web* que permita ao usuário fazer provas simuladas de legislação de trânsito e discutir em um formato de fórum o tema com outros usuários. A aplicação *web* será construída com o *framework Ruby on Rails* e com a aplicação de tendências da *Web 2.0*.

1.1.2 Objetivos Específicos

1.1.2.1 Utilização do *framework Ruby on Rails*

O aplicativo foi desenvolvido utilizando o *framework web Ruby on Rails* e com uma metodologia de desenvolvimento ágil chamada *Getting Real*, criada pelos mesmos autores do *framework*.

1.1.2.2 Aplicação de tendências da *Web 2.0*

autosimulado é uma ferramenta que segue as tendências da *Web 2.0*. Isso significa foco no usuário e aplicação de recursos ricos de interface do usuário para garantir a melhor interação com o sistema. A aplicação desses recursos será explicada no tópico 3.4.

1.2 Estrutura do trabalho

Serão apresentadas no capítulo 2 as principais características e recursos das tecnologias utilizadas no desenvolvimento do autosimulado. Já o capítulo 3 descreve as funcionalidades do produto e como foi seu desenvolvimento. O último capítulo apresenta uma análise dos resultados desenvolvidos, estatísticas de acesso e sugestões para próximas modificações do aplicativo.

2 METODOLOGIAS

Para o desenvolvimento desse trabalho foi necessário um estudo sobre a estrutura dos aplicativos utilizados via Internet, da linguagem *Ruby*, do *framework Ruby on Rails*, das tendências *Web 2.0* e da metodologia *Getting Real*.

2.1 Funcionamento do aplicativo *web*

O desenvolvimento de um aplicativo *web* apresenta preocupações diferentes do desenvolvimento de aplicações *desktop*. Entre as preocupações, destaca-se o ambiente do usuário do sistema.

O usuário de um sistema *web* pode possuir diversas configurações de *hardware*, sistema operacional, navegador e conexão com a Internet. Isso significa que o aplicativo deve considerar as particularidades de cada navegador, mantendo a compatibilidade entre eles, e ser o mais leve e rápido possível, considerando a variedade da velocidade de acesso a Internet.



Figura 2 - composição de uma página *web*
Fonte: Autor da monografia

Uma página *web*, FIG. 2, é a reunião das camadas de conteúdo, formatação e comportamento. O conteúdo é editado usando *XHTML*. A formatação é aplicada ao conteúdo usando *CSS* e a interação usuário-máquina é feita com *Javascript*.

2.1.1 XHTML

XHTML é a evolução da linguagem de formatação *HTML*. *HTML* é a linguagem de marcação de hipertexto pela qual é criado o conteúdo das páginas transmitidas pelo protocolo *HTTP*. No *HTML*, um documento é criado através de etiquetas – conhecidas por tags – que determinam as características daquele bloco de conteúdo. As etiquetas são elementos entre parênteses angulares (sinais de maior e menor - < e >) e possuem sua correspondente de fechamento.

Páginas *HTML* podem ser criadas em editores de texto puro, como o Bloco de Notas do *Windows* ou o Editor de Texto do *Macintosh*. Um exemplo de página *HTML*:

```
<html>
  <head>
    <title>Minha primeira página web</title>
  </head>
  <body>
    <h1>Olá mundo</h1>
    <p>Essa é minha primeira página web. Legal, não?</p>
  </body>
</html>
```

Um documento *HTML* é dividido em duas seções: cabeçalho - determinado pelo conteúdo entre as etiquetas *<head>* e *</head>* - e corpo - determinado pelo que está entre as etiquetas *<body>* e *</body>*. A etiqueta *<h1>* cria um título de seção e a etiqueta *<p>* determina um bloco de parágrafo de texto.



Figura 3 - Documento criado usando HTML
Fonte: Autor da monografia

A FIG. 3 apresenta o resultado do documento criado no exemplo.

O *XHTML* é a evolução do *HTML*, onde foram estipuladas novas regras para algumas etiquetas do *HTML*, além de padronizar a linguagem baseando-se no *XML*.

2.1.2 CSS

O *CSS* é uma linguagem de estilo usada para formatar documentos escritos em uma linguagem de formatação, como *HTML*, *XHTML* ou *XML*. Ao invés de formatar os elementos dentro do documento, o desenvolvedor separa em um arquivo todos os estilos daquela página. Isso, além de manter o código mais legível e organizado, garante o reaproveitamento dos estilos em várias páginas e agiliza a manutenção do *layout* do documento.

No *CSS* é possível formatar as páginas de acordo com seu *tag*, ou por identificadores chamados classes. Um exemplo da aplicação do *CSS* é apresentado a seguir:

```
body {  
    background: gray;  
}  
h1 {  
    color: blue;  
    font-family: arial;  
}  
p {  
    font-family: tahoma;  
    font-size: 13px;  
}
```

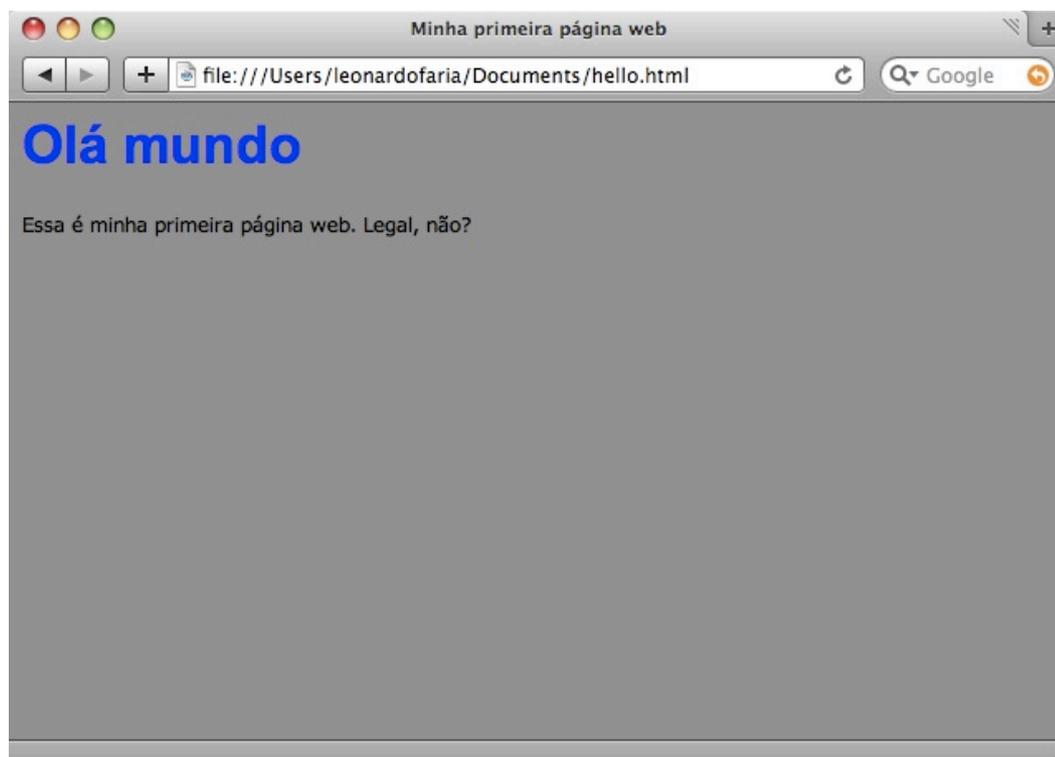


Figura 4 - Página com utilização de *CSS*
Fonte: Autor da monografia

Após a formatação utilizando *CSS*, a *FIG. 4* apresenta o documento criado no tópico 3.1.1.

2.1.3 *Javascript*

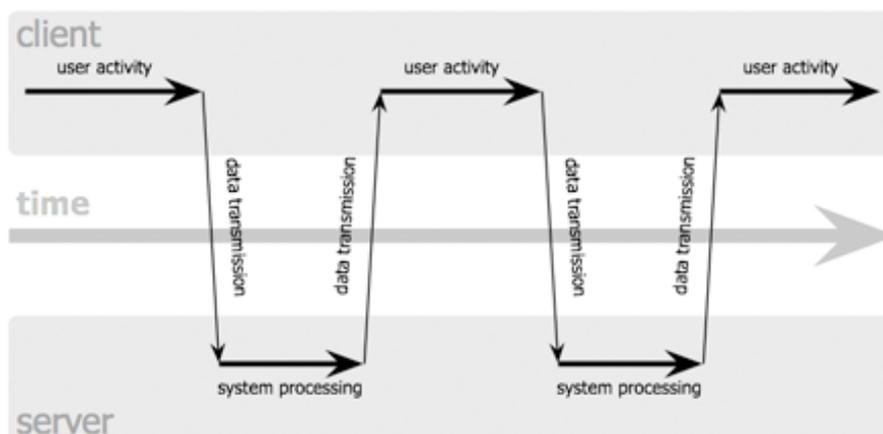
Javascript é uma linguagem de programação *client-side* - ou seja, interpretada pelo cliente (navegador) - criada pela Netscape em 1995 e que a princípio se chamava *Livescript*. O *Javascript* foi criado para fazer a interação do usuário com a página, como por exemplo a validação de dados em um formulário ou a classificação de dados em uma lista.

Existem várias bibliotecas que aprimoram e agilizam o desenvolvimento de soluções em *Javascript*. Entre as mais conhecidas, destacam-se a biblioteca *Prototype* em conjunto com a biblioteca *Script.aculo.us* e o conhecido *jQuery*. O *framework Rails* possui integração com a biblioteca *Prototype* e *Script.aculo.us*, conforme será explicado no tópico 3.3.

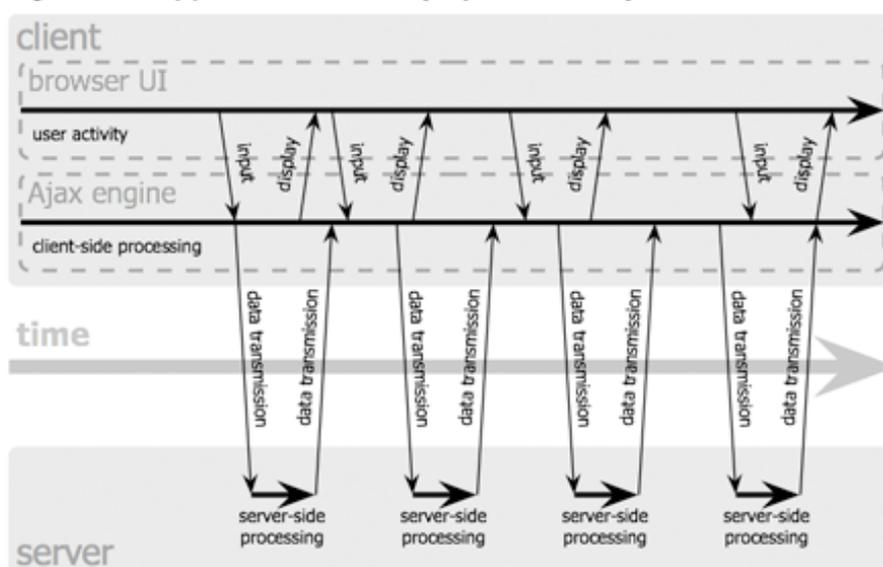
2.1.4 *Ajax*

Ajax é um nome de uma metodologia de desenvolvimento que utiliza as tecnologias acima descritas e o uso de requisições assíncronizadas, para garantir a melhor experiência do usuário.

classic web application model (synchronous)



Ajax web application model (asynchronous)



Jesse James Garrett / adaptivepath.com

Figura 5 - Modelos de aplicações web
 Fonte: Jesse J. Garrett <<http://www.adaptivepath.com>>

Conforme visto na FIG. 5, no modelo clássico de uma aplicação web, o usuário na interface dispara uma requisição *HTTP* para o servidor web e aguarda. O servidor web executa alguma operação - recupera dados, realiza algum cálculo, conversa com algum sistema legado ou banco de dados - e retorna uma página *HTML* para o cliente.

Durante o tempo entre requisições, o usuário espera indefinidamente o carregamento de uma página, inicialmente em branco, e não possui nenhuma perspectiva de quando obterá a resposta, o que obviamente não é uma boa experiência

de interface de um aplicativo. Desse modo, o modelo clássico não faz da *Web* uma boa plataforma para criação de *softwares*.

No modelo assíncrono, o usuário dispara alguma função e a página não é toda recarregada, apenas a parte dela necessária para a realização daquela função. Com as requisições assíncronas, o tempo de espera é menor e existe uma melhor experiência do usuário.

2.2 *Ruby*

A linguagem *Ruby* foi criada em 1993 pelo japonês Yukihiro “Matz” Matsumoto, com sua primeira versão pública lançada em 1995. Para Matz, o primeiro desejo é de *Ruby* tornar os programadores felizes, reduzindo o trabalho manual que precisasse ser feito. Segundo ele, o desenvolvimento de sistemas deveria enfatizar as necessidades do homem e não da máquina:

Muitas pessoas, especialmente engenheiros de computação, focam nas máquinas. Eles pensam, “Fazendo isso, a máquina será mais rápida. Fazendo isso, a máquina será mais eficiente. Fazendo isso, a máquina irá fazer determinada coisa melhor”. Eles estão focando nas máquinas. Mas de fato nós precisamos focar nos humanos, em como os humanos lidam com programação ou operação das aplicações das máquinas. Nós somos os mestres. Elas são as escravas. (VENNERS, 2003).

Ruby é uma linguagem orientada a objetos, ou seja, qualquer variável é um objeto, mesmo classes e tipos que em muitas linguagens são designadas como primitivos. Por exemplo:

```
puts "meu exemplo".upcase # imprimirá MEU EXEMPLO
```

Acima, aplica-se o método `upcase` da classe *String* no objeto “meu exemplo”. *Ruby* é extensível: um objeto pode receber melhorias e novos métodos em tempo de execução.

```
class Fixnum
  def +(numero)
    10
  end
end
```

No exemplo anterior, o método `+` da classe *Fixnum* foi sobrescrito e retornará sempre 10. A linguagem apresenta tipagem dinâmica, conforme no exemplo abaixo:

```
=> "meu exemplo"
>> minhavariavel.class
=> String
>> minhavariavel = 4 * 4
=> 16
>> minhavariavel.class
=> Fixnum
>> minhavariavel = 1000000 * 1000000
=> 1000000000000
>> minhavariavel.class
=> Bignum
```

No exemplo acima, ela recebeu alterações em sua tipagem em tempo de execução. São tipos de variáveis em *Ruby*:

- *Fixnum*: inteiros com até o tamanho da palavra binária do processador menos 1 bit. Exemplos: 1, 81, 6589, 100;
- *Bignum*: inteiros maiores que *Fixnum*. Exemplo: 1234567890;
- *Float*: números decimais. Exemplos: 1.41, 1.0;
- *String*: corresponde a uma cadeia de caracteres. Exemplo: "teste";
- *Range*: representa intervalos entre valores. Exemplos: 1..10 e a..z;
- Expressão regular: representa uma expressão regular. Exemplos: `/a/` ou `/^s*[a-z]/`.

Ruby é uma linguagem simples e elegante: nela não é obrigatório o uso de parênteses, colchetes e chaves e além disso a sintaxe da linguagem é humana e intuitiva, refletindo o minimalismo descrito por Matz. Em um comparativo entre as sintaxes das linguagens *Java* e *Ruby*:

TABELA 1
Comparativo das sintaxes de *Java* e *Ruby*

Java	Ruby
<code>l = list.get(list.size() - 1);</code>	<code>l = list.last</code>
<code>l = list.get(0);</code>	<code>l = list.first</code>
<code>for (int i=0; i<10; i++) {</code>	<code>10.times do i </code>
<code> System.out.println(i + "times");</code>	<code> puts "#{i} times"</code>
<code>}</code>	<code>end</code>

Fonte: Autor da monografia

No *Ruby*, também é possível criar *DSLs* - linguagens específicas de domínio - uma espécie de sub-linguagem onde o programador pode criar a sintaxe a sua necessidade. Desse modo, um algoritmo que fizesse uma receita de bolo poderia ser programado da seguinte forma:

```
receita "Bola de Fubá" do
  ingrediente "Farinha", "1 quilo"
  ingrediente "Açúcar", "200 gramas"
  ingrediente "Ovos", "2 unidades"
  preparo "Misture todos ingredientes"
  preparo "Leve ao forno"
  preparo "Sirva"
  tempo "2 horas"
  porcoes 3
end
```

Além disso, *Ruby* é portátil e livre. É possível executar *Ruby* em ambientes *Windows* e *Unix* e não é preciso pagar para usá-lo, copiá-lo, modificá-lo ou distribuí-lo.

2.2.1 Frameworks Ruby

Desenvolvedores de *software* constantemente se deparam com situações em que os problemas começam a se repetir em diversas partes do sistema. Para resolvê-los, rotinas são criadas e replicadas por todo o sistema, o que pode tornar o código facilmente suscetível a erros e demasiadamente replicado.

Os *frameworks* são soluções semi-prontas, para agilizar e tornar mais rápido o desenvolvimento de projetos. Eles seguem padrões de projeto bem definidos, que permitem que suas soluções sejam reutilizadas para problemas que outros desenvolvedores já enfrentaram. Dessa forma, os *frameworks* tornam-se recursos altamente confiáveis.

A linguagem *Ruby* possui vários *frameworks* web: *Merb*, *Ruby on Rails*, *Sinatra*. A adoção do *Ruby on Rails* deu-se a sua riqueza de funcionalidades e ampla documentação.

2.3 *Ruby on Rails*

O *framework Ruby on Rails* foi extraído de um sistema para gerenciamento de projetos chamado *Basecamp* (37Signals, 2009). A primeira versão do *framework* foi oficialmente lançada em 25 de Julho de 2004 e seu desenvolvimento conta com colaboradores em todo o mundo liderados pelo programador dinamarquês David Heinemeier Hansson.

2.3.1 MVC

Um padrão de projeto descreve e provê uma solução para um problema freqüente, sendo genérico e reusável. São criados a partir de problemas de problemas comuns enfrentados no desenvolvimento de projetos de *software*.

A criação de componentes reutilizáveis é uma das técnicas mais exploradas em Engenharia de *Software*. O uso de componentes diminui o tempo de desenvolvimento e a taxa de erros de codificação. Um padrão pode ser entendido como a abstração de detalhes sobre a implementação de um *software*.

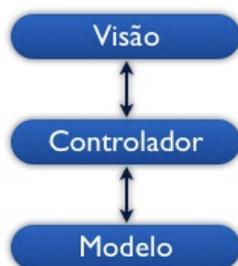


Figura 6 - O padrão de projeto MVC
Fonte: Autor da monografia

O padrão de projeto *MVC* divide o desenvolvimento de um aplicativo em três camadas: *View* (Visão), *Controller* (Controle) e *Model* (Modelo). A separação das camadas permite o aumento da flexibilidade e reuso do código. Sem essa separação, as funcionalidades podem ficar mescladas, o que acarreta um maior esforço para eventuais manutenções, pois as responsabilidades podem ser difundidas entre as camadas.

A camada Modelo representa o estado da aplicação. É responsável por fazer a interação da aplicação com a fonte de dados, muito frequentemente um banco de dados. Quando existe a necessidade de se guardar o estado da aplicação, é através dessa camada que as informações manipuladas pelo sistema podem ser armazenadas na base de dados. É na camada Modelo que são incorporadas as regras de negócio de um aplicativo.

A camada Controle é responsável por receber os dados do usuário e definir o fluxo da aplicação, invocando alterações dos dados da camada Modelo.

A camada Visão é responsável por apresentar a aplicação ao usuário. Nas aplicações *Web*, essa camada é construída com *HTML*, *CSS* e *Javascript*.

2.3.2 Recursos e características

DRY - Don't Repeat Yourself, em português, Não Se Repita é um conceito intrínseco do *Ruby on Rails*. Não é preciso copiar trechos de código por todo aplicativo. Para reaproveitamento de código, o desenvolvedor conta com os métodos

helpers e com as *partials*, arquivos que podem ser incluídos no seu aplicativo reduzindo redundâncias.

Helpers são uma importante funcionalidade do *framework*. Um *helper* é um módulo que contém funções para auxiliar a camada *View* do aplicativo, retirando lógicas complexas do código da apresentação. *Ruby on Rails* conta com *helpers* para diversas funções, incluindo formatação de datas, moedas, números, formulários *HTML*, *Javascript*, entre outros. Alguns exemplos da aplicação de *helpers*:

```
<%= distance_of_time_in_words(Time.now, Time.now + 33, false) %>
  1 minute

<%= human_size(123_456) %>
  120.6 KB

<%= number_to_currency(234.56, :unit => "R$ ", :precision => 2) %>
  R$ 235.56

<%= number_to_percentage(33.66666) %>
  33.667%

<%= number_to_phone(2125551212, :area_code => true, :delimiter => " ") %>
  (212) 555 1212

<%= truncate(@text, 9) %>
  Hello wor...

<%= pluralize(1, "person") %> and <%= pluralize(2, "person") %>
  1 person and 2 people
```

Além dos *helpers*, existem ainda os *plugins*, pequenas bibliotecas que adicionam novas funcionalidades ao *framework* sem que haja necessidade da alteração do seu núcleo e que permitem ser reaproveitados em diferentes projetos.

O *framework Ruby on Rails* possui suporte aos mais comuns banco de dados do mercado, como *IBM DB2*, *Microsoft SQL Server*, *Oracle OCL8*, *PostgreSQL*, *MySQL* e *SQLite*. Para definir qual banco utilizar, é necessário apenas especificar os dados de conexão em um único arquivo.

Outra característica interessante são os perfis de execução do aplicativo: desenvolvimento, produção e teste. No perfil desenvolvimento, o programador encontra mensagens de *log* detalhadas para cada requisição no aplicativo. Além disso, erros são apresentados com mensagens claras das inconsistências. O modo teste é

utilizado para testes funcionais do aplicativo e produção é o modo utilizado para quando o aplicativo estiver pronto e estável. Para cada um desses perfis é possível utilizar um banco de dados diferente, evitando conflitos entre dados de testes, desenvolvimento e produção.

Além disso, o *framework* já foi criado com suporte integrado a duas bibliotecas *Javascript* bastante conhecidas: *Prototype*, que manipula as interações dos objetos do documento, e *Script.aculo.us*, responsável por criação de efeitos visuais com *Javascript*.

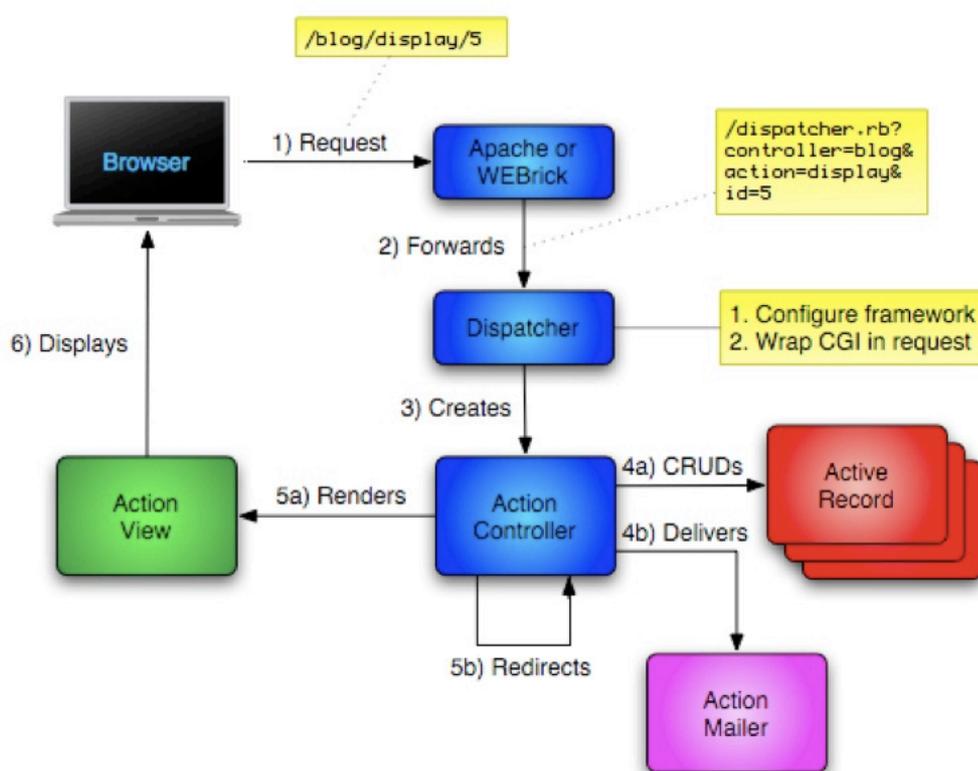


Figura 7 - Ciclo de requisições no *Ruby on Rails*
 Fonte: <<http://www.rubyonrails.com>>

A FIG. 7 exemplifica o ciclo de requisições no *Ruby on Rails*. Ao fazer uma requisição de uma página, o navegador faz uma chamada ao servidor - normalmente *Apache* - que é encaminhada para o *dispatcher*. O *dispatcher* é o responsável por transformar a *URL* do navegador em uma *URL* que é entendida pelo *framework*. Em seguida, o *Action Controller* é invocado para decidir o que fazer com a requisição. Caso seja necessária alguma interação com o banco de dados, o *Active Record* entra em ação. Também é possível que o *Action Mailer* seja invocado, caso

seja necessário enviar algum email e a resposta da requisição é processada pelo *Action View*.

Ruby on Rails é composto por 5 módulos independentes:

- O módulo *Active Record* conecta objetos de negócio com tabelas do banco de dados para criar um modelo de domínio onde lógica e dados se encontram presentes em conjunto. Trata-se portanto de uma implementação de um padrão de mapeamento objeto-relacional (*ORM*), baseado em convenções. Por exemplo, para uma classe *Book* espera-se a existência de uma tabela *Books*. Cada linha dessa tabela corresponde a um objeto da classe *Book*. Atributos da classe representarão as colunas da tabela *Books*, com os mesmos nomes, por padrão.
- O módulo *Action Pack* compreende o *Action Controller* e o *Action View*. O *Action Controller* coordena a interação entre o usuário, as visões e o modelo. Ele é responsável por rotear *URLs* para ações internas dos *controllers*, gerenciando *URLs* de fácil leitura para as pessoas; por responder o usuário exibindo uma *view* ou um arquivo qualquer e por gerenciar *cookies* e sessões. Já o *Action View* é responsável por compor toda a funcionalidade necessária para renderizar *views*, mais comumente gerando código *HTML* e *XML* para o usuário.
- O módulo *Action Support* agrupa várias classes úteis e extensões de bibliotecas padrão, que foram consideradas relevantes para aplicações com *Ruby on Rails*.
- O módulo *Action Mailer* é um *framework* poderoso para serviços de entrega e recebimento de emails.
- O módulo *Action WebServices* provê uma maneira de criar *APIs* interoperáveis com *Rails*. Na versão 2.0 do *framework* esse módulo foi retirado, devido a implementação do modelo *Rest* no *Rails*.

2.3.3 Estrutura de um aplicativo *Ruby on Rails*

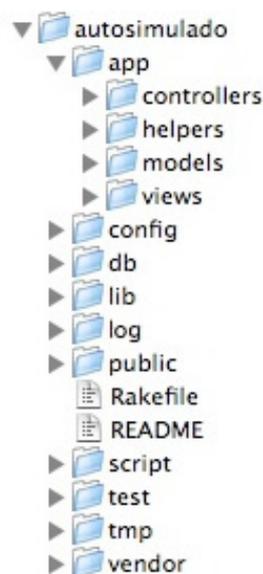


Figura 8 - Estrutura de um projeto
Fonte: Autor da monografia

A estrutura de um projeto *Rails* compreende a criação dos diretórios especificados na FIG. 8. A pasta “*app*” possui subdiretórios que armazenam *controllers*, *helpers*, *models* e *views*.

A pasta “*config*” abriga diversos arquivos de configuração, incluindo o responsável por determinar qual o banco de dados será utilizado.

A pasta “*db*” possui as *migrations*. *Migrations* são um recurso do *Rails* onde o desenvolvedor escreve arquivos *Ruby* em uma *DSL* exclusiva para manipular o banco de dados. Desse modo, é possível criar tabelas, campos e modificar a estrutura de banco de dados sem precisar escrever uma linha de *SQL* ou utilizar outro aplicativo.

A pasta “*lib*” é usada para que o desenvolvedor salve nesse local suas bibliotecas externas do projeto.

A pasta “*log*” é utilizada para salvar os *logs* gerados em tempo de execução do aplicativo. As ações executadas no aplicativos são salvas nos arquivos de *log* dessa pasta.

A pasta “*public*” é usada para armazenar as imagens, folhas de estilo e documentos *Javascript* do aplicativo.

A pasta “*script*” possui *scripts* usados para auxiliar o desenvolvedor. Com esses *scripts*, o programador pode gerar novos *controllers* e *models* pela linha de comando. Além disso, essa pasta possui um programa *server*, que é um servidor *web* para desenvolvimento.

A pasta “*test*” possui arquivos para realização de testes unitários, funcionais e de integração.

A pasta “*vendor*” é usada para armazenar os *plugins* do projeto e em alguns casos, armazenar o próprio *framework*.

2.4 Web 2.0

O termo *Web 2.0* foi proposto por O’Reilly (2005), na conferência de mesmo nome, que discutia os rumos da Internet após o fenômeno do “estouro da bolha das empresas de Internet”, ocorrido em 2001. Esse termo foi usado para definir tendências, serviços e aplicativos do que foi chamada de segunda geração de páginas e empresas de Internet.

O usuário é o protagonista da *Web 2.0*. Ele passa de consumidor passivo de tecnologia a editor [hiper]ativo de conteúdo. Os aplicativos dessa nova Internet são marcados pela colaboração de conteúdo feita em comunidade por usuários espalhados no mundo inteiro e unidos pela afinidade de um tema. Portanto, *Web 2.0* significa foco no usuário.

A interação do conteúdo via *RSS*, que permite ao usuário decidir qual a melhor forma de ler o conteúdo de seu aplicativo, e a criação de experiências ricas usando *Ajax* são características da *Web 2.0* que estão presentes no autosimulado.

2.5 MySQL

O *MySQL* é um banco de dados relacional baseado na linguagem *SQL*, criado na Suécia e atualmente mantido pela *Sun Microsystems*. Entre suas características principais destacam-se a portabilidade, sua bem estruturada arquitetura e a facilidade de uso.

O banco de dados *MySQL* é escrito em C, o que o torna portátil para a maioria dos sistemas operacionais existentes. A arquitetura do *MySQL* é organizada de acordo com a função da tabela. Por exemplo, tabelas que possuem muitos acessos concorrentes são do tipo *InnoDB*. Tabelas que precisam ser ágeis em buscas, por sua vez, são do tipo *MyISAM*.

O *MySQL* é um banco de dados *open-source*, o que amplia ainda mais sua escolha no desenvolvimento de software.

2.6 Metodologia de desenvolvimento

Getting Real (37Signals, 2006) é uma metodologia de desenvolvimento ágil criada pelos mesmos autores do *framework Ruby on Rails*. A ideia do *Getting Real* é simples: menos é mais. Deve-se evitar ao máximo a construção do que representa o aplicativo (gráficos, diagramas, especificações e *wireframes*) e sim construí-lo, com foco em prazos menores e escrevendo apenas as funcionalidades indispensáveis e essenciais.

A interface é a primeira parte do trabalho no desenvolvimento *Getting Real*, concentrando na experiência do usuário e codificando a partir disso. Em lugar de software em uma caixa cuja atualizações aparecerem anualmente, os aplicativos *web* evoluem rapidamente, com a participação do usuário e com atualizações cada vez mais rápidas, em um “beta eterno”.

3 DESENVOLVIMENTO

3.1 Produto

O aplicativo desenvolvido nesse trabalho foi criado utilizando os recursos apresentados no capítulo 3. As etapas de desenvolvimento seguiram o planejamento a seguir:

- levantamento de requisitos e funcionalidades;
- criação do *layout*;
- criação do diagrama entidade-relacionamento;
- criação das tabelas do *MySQL*;
- codificação;
- implantação de um sistema de controle de versões;
- testes;
- implantação;
- atualizações;
- estatísticas.

3.2 Levantamento de requisitos e funcionalidades

Definida as principais características do serviço, o levantamento de requisitos norteou quais as funcionalidades o autosimulado deveria ter. Dentre os requisitos apontados, destacam-se:

- o aplicativo deve ser simples e minimalista. O serviço não deve ter nenhuma poluição visual demasiada, animações ou quaisquer outros elementos que possam desconcentrar o usuário;

- os usuários devem ser cadastrados para realizar os testes. Com seu cadastro no serviço, ele pode acompanhar seu desempenho nos testes e participar do fórum;
- para evitar cadastros desnecessários e irrelevantes, o *site* deve ter um tour. O tour consiste na apresentação de um pequeno simulado, onde o usuário pode conhecer a interface do aplicativo e testar seus conhecimentos;
- o cadastro no *site* deve ser simples e funcional. Não devem ser perguntadas questões indesejáveis tão pouco dados irrelevantes;
- todas as páginas do serviço devem ser *print-friendly*, ou seja, prontas para impressão. Desse modo o usuário pode imprimir os testes do serviço e levar para qualquer lugar, em uma página otimizada para impressão;
- as *URLs* do site devem ser auto-explicativas, trazendo o título do simulado em questão, como por exemplo:
<<http://www.autosimulado.com.br/exams/1-simulado-detran-da-bahia>>. Esse requisito é necessário principalmente para a indexação do aplicativo em sistemas de busca, como por exemplo o *Google*;
- o site deve ser compatível com a maioria dos navegadores existentes;
- é necessário a criação de um *blog* para que os usuários do aplicativo acompanhem as novidades do serviço.

3.3 Layout

Definidos os requisitos do serviço, a próxima etapa do desenvolvimento foi a criação do *layout* do aplicativo. O *layout* realizado foi criado com cores sóbrias e seguindo os requisitos minimalistas propostos.

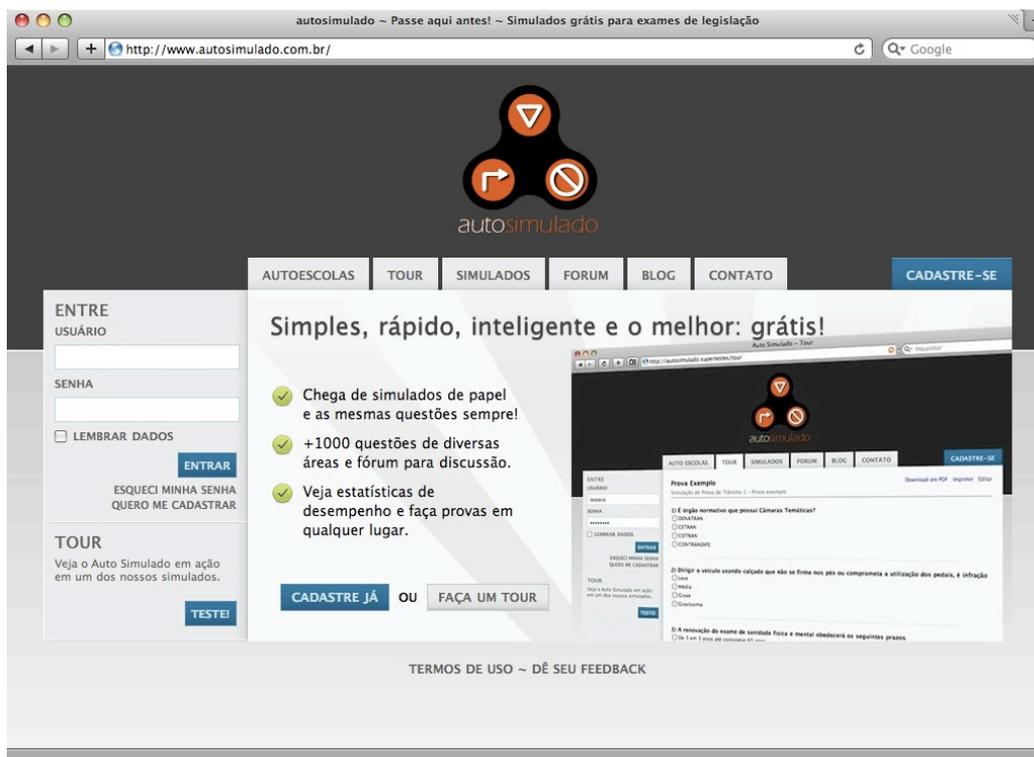


Figura 9 - Screenshot da homepage do autosimulado
Fonte: Autor da monografia

A FIG. 9 apresenta a página inicial do aplicativo desenvolvido. As outras screenshots do serviço estão no Apêndice A.

3.4 Base de dados

Durante o levantamento de requisitos e a criação do *layout* inicial, ficaram claras as tabelas que deveriam ser criadas e os dados que deveriam ser armazenados no banco de dados *MySQL* (MySQL, 2009). A versão final da base de dados do aplicativo é apresentada a seguir através do Diagrama de Entidade-Relacionamento, gerado com o aplicativo *OmniGraffle Pro 4.2*, disponível em <<http://www.omnigroup.com/applications/OmniGraffle/>>. Acesso em 20 abril 2009.

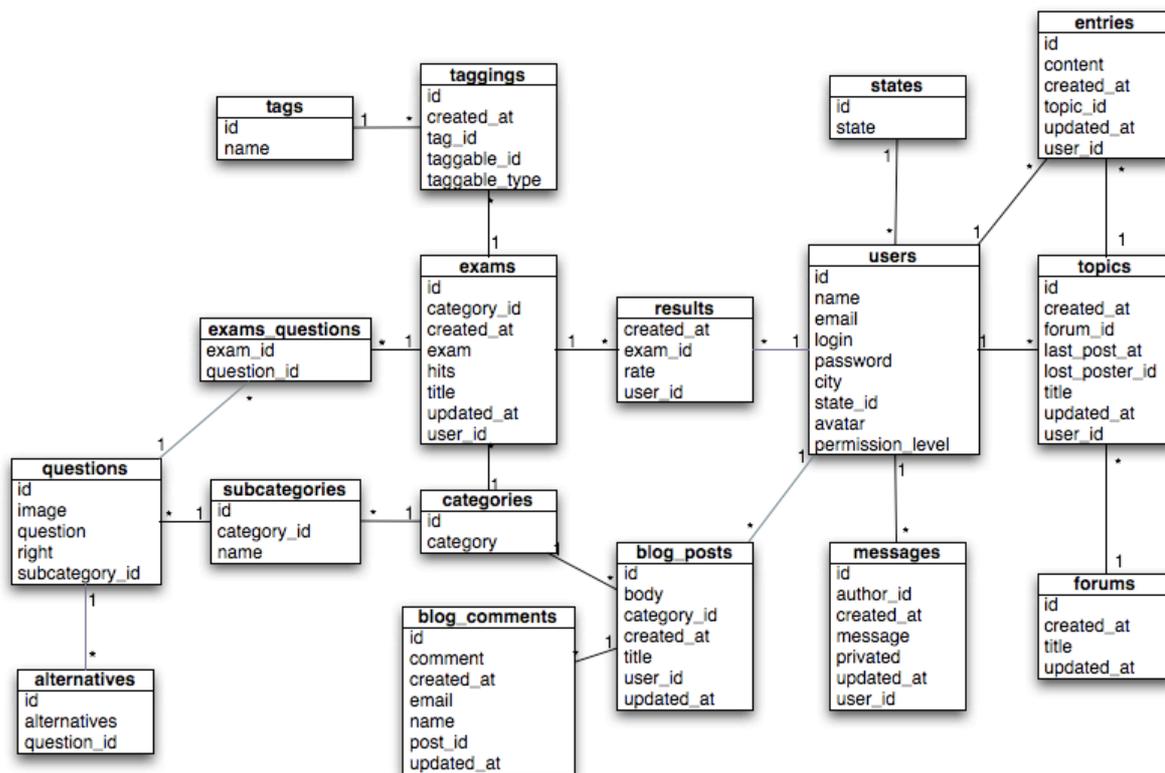


Figura 10 - Diagrama de Entidade Relacionamento
Fonte: Autor da monografia

As principais entidades do aplicativo são "*Exam*" e "*User*". A entidade "*Exam*" se relaciona com:

- *Category*, em um relacionamento N-1. A única categoria do aplicativo é a categoria autosimulado. Esse relacionamento pode ser lido como "uma categoria possui N exames";
- *Taggings*, em um relacionamento N-N. "*taggings*" é a tabela de quebra utilizada no relacionamento entre as entidades *Exam* e *Tag*. Esse relacionamento pode ser entendido como "um ou mais exames possuem uma ou mais etiquetas através da tabela *taggings*";
- *Exam_questions*, em um relacionamento N-N. "*exam_questions*" é a tabela de quebra utilizada no relacionamento entre as entidades *Exam* e *Question*. Esse relacionamento pode ser lido como "um ou mais exames possuem uma ou mais questões através da tabela *exam_questions*".

A entidade *User* é responsável por armazenar os dados dos usuários do serviço. Essa entidade se relaciona com as entidades *State*, em um relacionamento N-1; e com *Result*, em um relacionamento N-N. "*results*" é a tabela de quebra utili-

zada no relacionamento entre as entidades *User* e *Exam*. Esse relacionamento pode ser lido como "um ou mais usuários fazer um ou mais exames através da tabela *results*".

A entidade *User* também interage com o recurso de fórum do autosimulado. Uma entrada do *fórum* - tabela *entries* - ou um tópico do fórum - tabela *entries* - pertencem a um usuário.

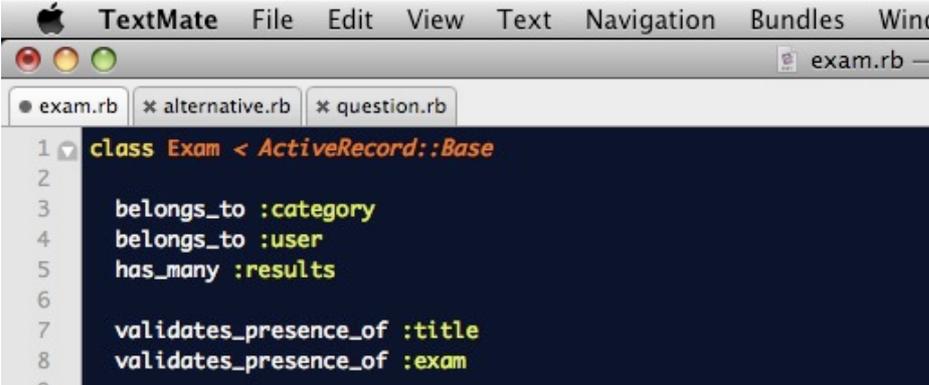
A entidade *messages* é responsável para armazenar recados que podem ser feitos por qualquer usuário do sistema. Essa funcionalidade só foi criada após a implantação da primeira versão do autosimulado, conforme será visto no capítulo 5.1.

O *framework Ruby on Rails* estabelece algumas convenções para o funcionamento de um banco de dados de um sistema. Entre essas convenções, está a utilização do nome das tabelas em inglês e no plural e que todas as chaves primárias do aplicativo chamem-se *id*. Os campos que são chave estrangeira de outra tabela devem ser representados pelo nome da tabela no singular seguido por *_id*.

3.5 Codificação

Definidas as funcionalidades e projetado o banco de dados, a próxima etapa do projeto foi a codificação do aplicativo.

Foi durante essa etapa que as funcionalidades, *layout* e regras de negócio foram implementadas. Uma parte importante do desenvolvimento foi o estabelecimento do relacionamento entre as tabelas.



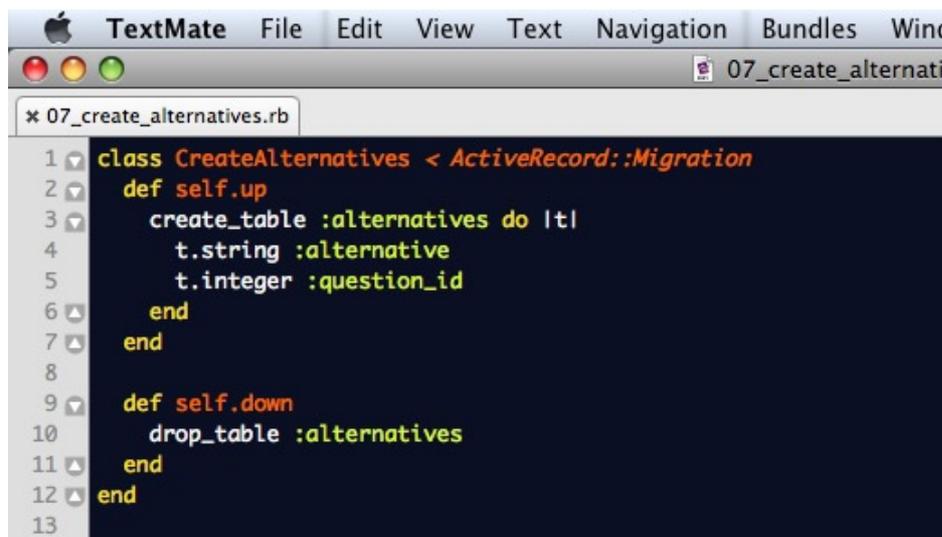
```
1 class Exam < ActiveRecord::Base
2
3   belongs_to :category
4   belongs_to :user
5   has_many :results
6
7   validates_presence_of :title
8   validates_presence_of :exam
9
```

Figura 11 - *Model exam.rb*
Fonte: Autor da monografia

É na camada de modelo do aplicativo que são especificados os relacionamentos entre as entidades. O método *belongs_to* estabelece um relacionamento N-1 entre duas entidades. Na imagem da FIG. 11, a classe *Exam* possui dois relacionamentos desse tipo, com as entidades *Category* e *User*. O código pode ser lido da seguinte forma: 1 ou mais exames pertence (*belongs_to*) a 1 categoria e a 1 usuário.

Já o método *has_many* estabelece um relacionamento 1-N entre duas classes. Existe esse relacionamento entre as classes *Exam* e *Result*. Nesse caso, lê-se que 1 exame tem muitos (*has_many*) resultados.

É também no *Model* que são estabelecidos as regras de validação de uma entidade. Na FIG. 11 é invocado o método *validates_presence_of*. Esse método verifica a presença de duas colunas da classe *Exam*: *title* e *exam*. Assim, não será possível criar um novo exame sem um título (*title*) e uma descrição (*exam*).



```
1 class CreateAlternatives < ActiveRecord::Migration
2   def self.up
3     create_table :alternatives do |t|
4       t.string :alternative
5       t.integer :question_id
6     end
7   end
8
9   def self.down
10    drop_table :alternatives
11  end
12 end
13
```

Figura 12 - Exemplo de *migration*
Fonte: Autor da monografia

Para gerar a estrutura do banco de dados como o previsto no Diagrama Entidade-Relacionamento, foram criadas diversas *Migrations*. *Migrations*, já conceitualizado no tópico 3.3.3, são arquivos *Ruby* escritos em uma linguagem específica de domínio que, quando executadas, geram sentenças *SQL* para modificar a estrutura do banco de dados. A vantagem da adoção desses arquivos é que a estrutura do banco de dados é criada junto com o aplicativo sem a necessidade da utilização de um *software* para gerenciamento do banco de dados.

A FIG. 12 apresenta um dos arquivos de migração criados durante o desenvolvimento do aplicativo. Esse arquivo foi responsável por criar a tabela “*alternatives*”. Essa tabela possui 2 colunas: “*alternative*” e “*question_id*”.

3.6 Controle de versões

O desenvolvimento de software envolve um processo contínuo de evolução de código e baseado nesse paradigma, surgiu a necessidade do desenvolvimento de uma solução que gerenciasse o controle de versões dos códigos-fonte, da documentação e do compartilhamento de trabalho. Assim, nasceram os softwares responsáveis de controle de versão.

Entre suas funções, os Sistemas de Controle de Versão se destacam por possibilitar o controle do histórico do desenvolvimento, possibilitar o trabalho em equipe e permitir a marcação e resgate de versões estáveis. Caso aconteça algum eventual problema, o projeto pode ser revertido a algum estado anterior do desenvolvimento.

No desenvolvimento do projeto a ferramenta utilizada foi o Git. O Git é um sistema de versão criado pelo finlandês Linus Torvalds, criador do sistema operacional Linux. Periodicamente após as alterações dos arquivos do aplicativo foi realizado a sincronização dos arquivos com o servidor.

3.7 Testes

Paralelamente a todas as etapas de desenvolvimento, o aplicativo foi testado pelo desenvolvedor e por alguns usuários convidados. Os testes efetuados serviram para verificar se as funcionalidades foram implementadas conforme planejadas e se não existiam páginas quebradas ou informações desencontradas.

Os testes com usuários convidados foram muito importantes e reafirmaram a necessidade de algumas funcionalidades. O tour do aplicativo, por exemplo, foi uma funcionalidade importante percebida da observação de um dos usuários convidados para os testes iniciais.

3.8 Implantação

O *deployment* do autosimulado foi realizado enviando os arquivos do aplicativo da máquina de desenvolvimento para o servidor de produção. O envio dos arquivos foi realizado via *FTP*.

A outra tarefa realizada no *deployment* foi a configuração do banco de dados de dados do servidor de implantação. A configuração envolvida consistia na

criação das tabelas utilizadas pelo aplicativo e na inserção dos simulados já cadastrados. Para ambas as tarefas, a configuração foi realizada através de uma conexão *SSH*, onde o desenvolvedor tem acesso ao servidor de produção e pode executar comandos como se estivesse em seu próprio computador.

A criação das tabelas do banco de dados foi efetivada através do programa *'rake'*. O *rake* do *Ruby* corresponde ao comando *Make* de algum aplicativo cujo código-fonte deve ser compilado. No *deployment*, executou-se a tarefa *'rake db:migrate'* para que fosse criadas as tabelas do banco de dados de acordo com o especificado nos arquivos de *migrations* do aplicativo. Os arquivos de *migrations* possuem as informações sobre as colunas das tabelas do banco de dados.

A inserção dos simulados já cadastrados deu-se a partir da importação de um arquivo *SQL* da máquina de desenvolvimento para o servidor de produção. Nessa tarefa foi utilizada via terminal o aplicativo *mysql* para a migração de dados.

3.9 Atualizações

Após o lançamento da versão inicial do autosimulado foram estudadas a implantação de novas funcionalidades. Um exemplo disso foi a criação da página de resultados do usuário, não prevista inicialmente. Nessa página, o usuário do aplicativo pode verificar quais simulados realizou e seu respectivo desempenho.

Também foi realizado o desenvolvimento de uma página de contato, para que os usuários do serviço pudessem entrar em contato e tirar as suas dúvidas. As mensagens são encaminhadas para o email <contato@autosimulado.com.br> e respondidas em tempo hábil. Outra página criada que não estava prevista na versão inicial do serviço era a página de Perguntas e Respostas (*FAQ*). Essa página de Perguntas e Respostas apresenta dúvidas comuns sobre o autosimulado.

Outra funcionalidade implementada que não fora previsto inicialmente foi a criação de um perfil público do usuário. Nesse perfil, outros membros do site podem verificar quais testes aquele usuário realizou e quais as suas participações no fórum. Além disso, a página de perfil público possui um livro de recados, semelhante

ao *orkut*. Nesse livro de recados, usuários podem deixar uma mensagem e assim se interagir em comunidade.

3.10 Estatísticas

O acompanhamento de sistemas de estatísticas é necessário e altamente relevante para aprimoramentos no modelo de negócio e na divulgação do aplicativo.

No autosimulado, foram utilizadas duas ferramentas para mensurar o crescimento do aplicativo: sentenças *SQL* em uma página de acesso exclusivo do administrador e *Google Analytics* - disponível no anexo A.

3.10.1 SQL

As sentenças *SQL* são comandos que, quando executados no servidor, retornam informações sobre os registros do banco de dados. O autosimulado possui uma página, de acesso exclusivo ao administrador, que executa algumas sentenças responsáveis por dizer estatísticas da utilização do serviço.

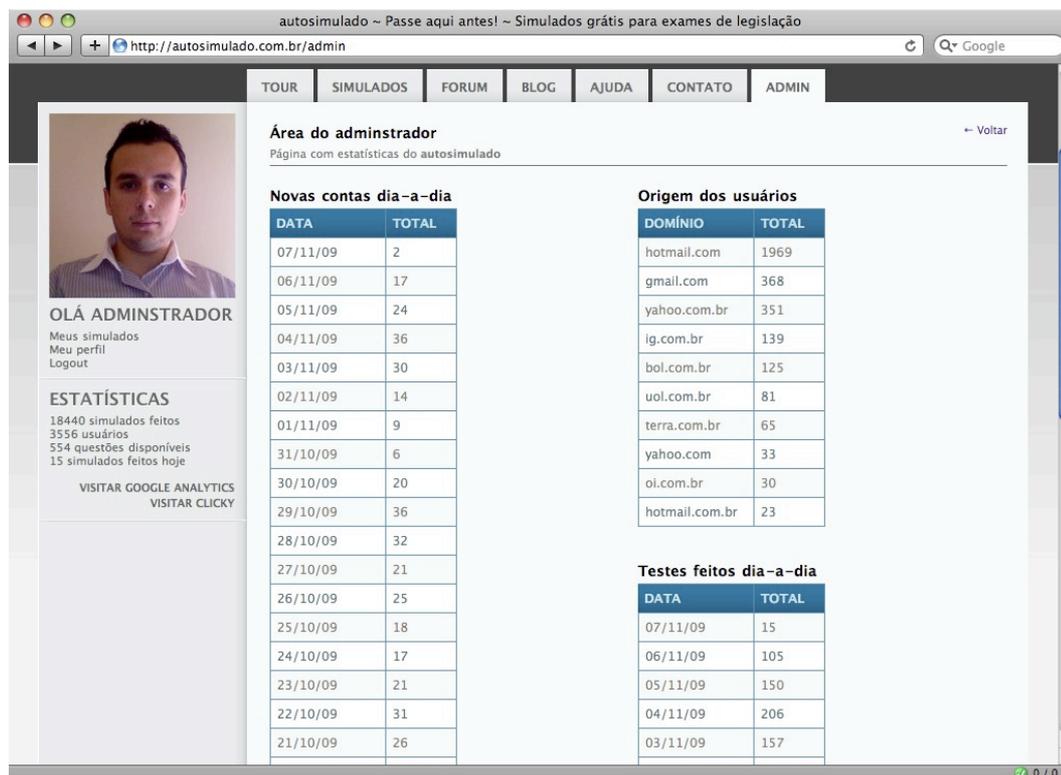


Figura 13 - Página de estatísticas

Fonte: Autor da monografia

A página de estatísticas do autosimulado apresenta os números de:

- Total de simulados feitos;
- Total de usuários cadastrados no sistema;
- Total de questões cadastradas;
- Número de usuários criados dia-a-dia;
- Número de testes realizados dia-a-dia;
- Principais serviços de email utilizados pelos usuários.

Com o acompanhamento desses dados, pode-se conhecer um pouco mais sobre os usuários que utilizam o aplicativo. Entre duas curiosidades, observa-se que mais da metade dos usuários do autosimulado usam o Hotmail como serviço de email e que o número de testes realizados e usuários criados diminui nos fins-de-semana.

4 CONSIDERAÇÕES FINAIS

O desenvolvimento do aplicativo autosimulado trouxe uma alternativa para o ensino da legislação de trânsito, oferecendo aos usuários testes e fóruns para discussão do tema. Com esse objetivo, o aplicativo foi concebido com todas as funcionalidades documentadas nesse documento e já em execução, tem auxiliado os usuários de forma eficiente e funcional.

A utilização de um framework que foca o desenvolvimento ágil mostrou-se interessante porque confirmou as expectativas da adoção de uma ferramenta produtiva e com uma pequena curva de aprendizado.

Para trabalhos futuros, é sugerido a execução de um modelo de negócio que traga novas oportunidades de ampliação do serviço e novos recursos para aplicação.

Para concluir, pode-se afirmar que o autosimulado realiza satisfatoriamente o objetivo de fornecer uma plataforma para execução de testes.

REFERÊNCIAS

37SIGNALS. Basecamp. Disponível em: <<http://www.basecamphq.com/>>. Acesso em: 20 abril 2009.

37SIGNALS. Getting Real - The smarter, faster, easier way to build a successful web application. Disponível em <http://gettingreal.37signals.com/GR_por.php> . Acesso em: 25 maio 2009.

AKITA, Fabio. *Repensando a web com Rails*. São Paulo: Braspost, 2006.

FERRAZ, Ronaldo. Rails para sua Diversão e Lucro. Disponível em <<http://kb.reflectivesurface.com/br/tutoriais/railsDiversaoLucro/>>. Acesso em: 4 outubro 2009.

GOOGLE. Google Adsense. Disponível em <<http://adsense.google.com>>. Acesso em: 7 novembro 2009.

GOOGLE. Google Analytics. Disponível em <<http://analytics.google.com>>. Acesso em: 25 maio 2009.

GOOGLE. orkut. Disponível em <<http://www.orkut.com>>. Acesso em: 25 maio 2009.

HANSSON, David Heinemeier. Ruby on Rails. Disponível em <<http://www.rubyonrails.com>>. Acesso em: 25 maio 2009.

MYSQL AB. MySQL. Disponível em <<http://www.mysql.com>>. Acesso em: 25 maio 2009.

O'REILLY, T. What is web 2.0. O'Reilly Media, setembro 2005. Disponível em <<http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>>. Acesso em: 20 abril 2009

VENNERS, Bill, 2003. The Philosophy of Ruby. Disponível em: <<http://www.artima.com/intv/ruby4.html>>. Acesso em: 20 abril 2009.

THOMAS, Dave; HANSSON, David Heinemeier. *Desenvolvimento web ágil com Rails*. São Paulo: Bookman, 2008.

GLOSSÁRIO

Ajax: Uso metodológico de tecnologias como Javascript, XML e CSS para tornar as páginas *Web* mais interativas com o usuário.

API: É um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por outros programas.

Deployment: Implantação de um sistema em um ambiente de produção.

FTP: Protocolo que permite ao usuário a transferência de arquivos entre dois computadores em rede.

REST: Técnica de engenharia de software para sistemas distribuídos como a *World Wide Web*.

RSS: Tecnologia que permite agregar conteúdo de um *website* em um formato *XML* padronizado, fazendo com que os usuários possam reunir diversas fontes *RSS* em um aplicativo específico.

SQL: uma linguagem de pesquisa declarativa para banco de dados, baseada em consulta estruturada e que é atualmente grande padrão para banco de dados.

Open-source: Tipo de licença de *software* que dá ao usuário a liberdade de alterar o código-fonte e distribuir as modificações para a comunidade, mantendo o trabalho sobre a mesma licença.

XML: formato de arquivo texto para a criação de documentos com dados organizados de forma hierárquica.

URL: Endereço de um documento na Internet.

APÊNDICE A - SCREENSHOTS DO APLICATIVO

Imagens capturadas do aplicativo:

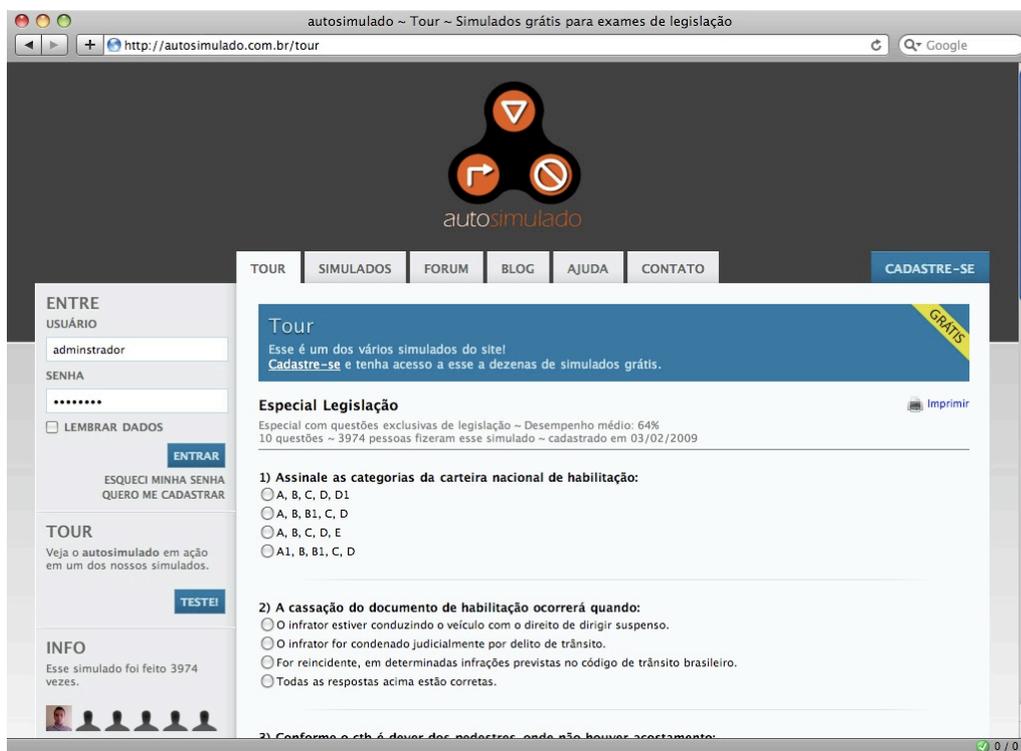


FIGURA A.1 - Página do *tour* do aplicativo. Nela, é mostrado um teste onde os visitantes do *site* podem conhecer a *interface* do serviço sem se cadastrar.

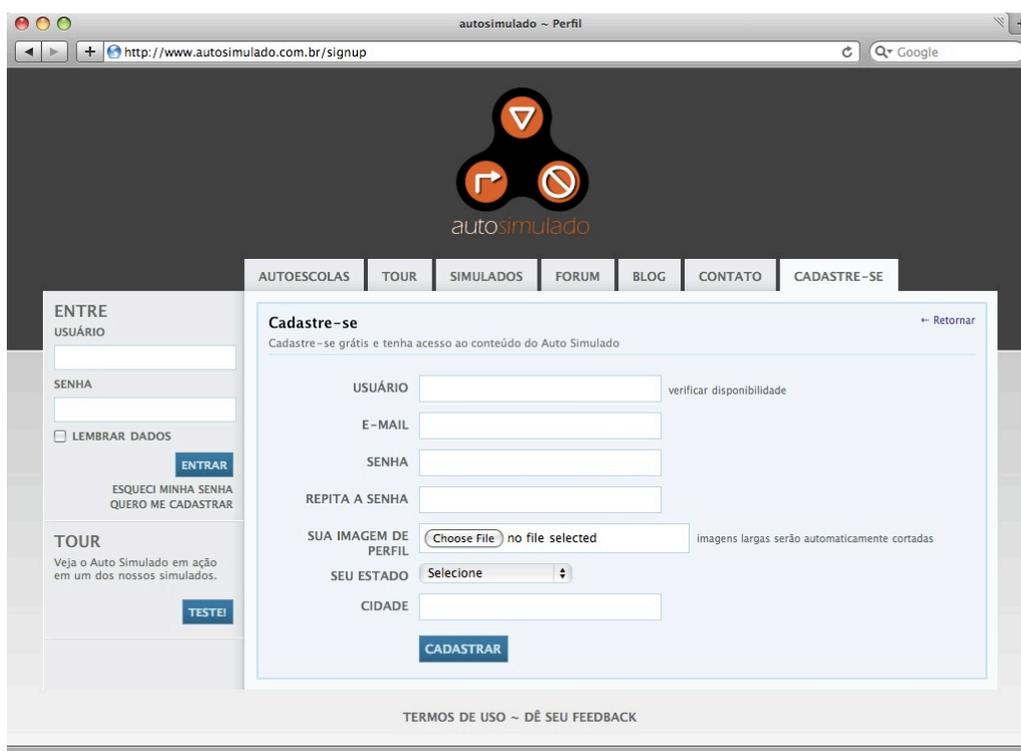


Figura A.2 - Tela de cadastro de usuários do autosimulado.

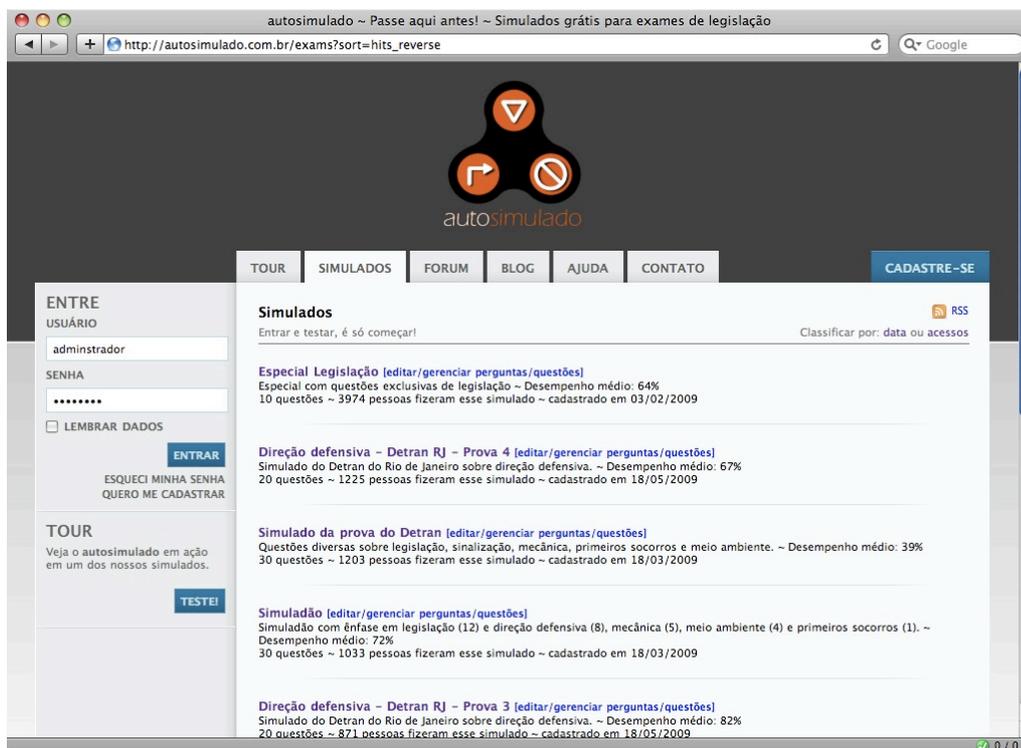


Figura A.3 - Tela de listagem de simulados.

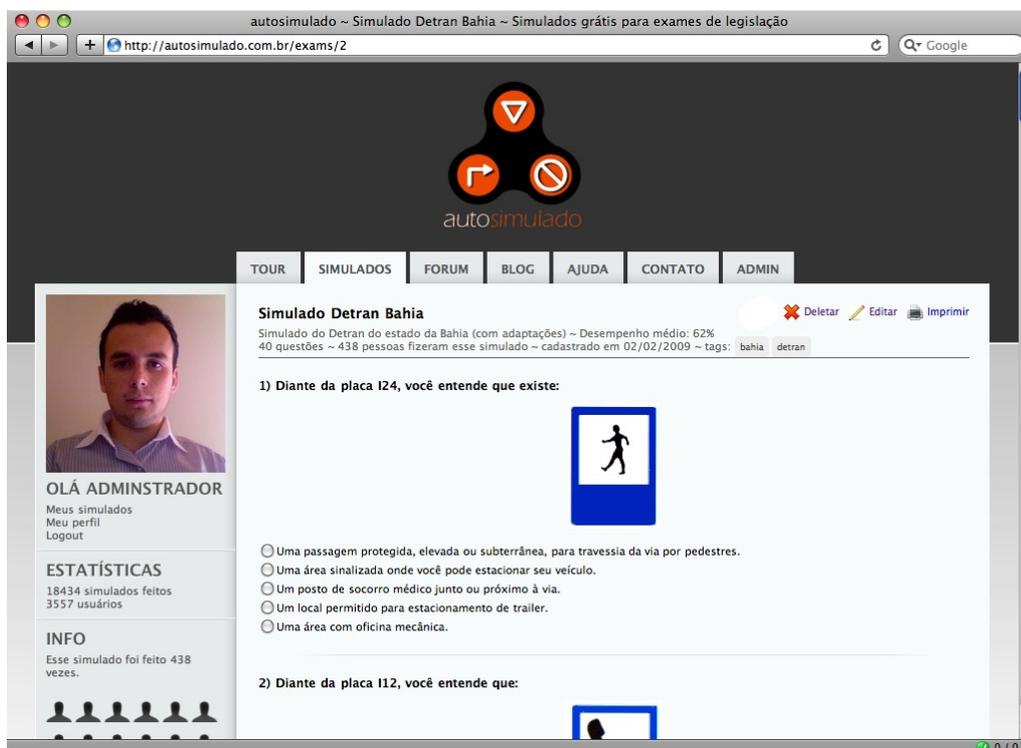


Figura A.4 - Simulado em execução.

autosimulado ~ Simulado Detran Bahia ~ Simulados grátis para exames de legislação

http://autosimulado.com.br/exams/results/2

Google

auto**simulado**

TOUR | SIMULADOS | FORUM | BLOG | AJUDA | CONTATO | ADMIN

Resultados: Simulado Detran Bahia [Imprimir](#)

Aproveitamento: 10%

Diante da placa I24, você entende que existe:



Uma passagem protegida, elevada ou subterrânea, para travessia da via por pedestres.
 Uma área sinalizada onde você pode estacionar seu veículo.
 Um posto de socorro médico junto ou próximo à via.
 Um local permitido para estacionamento de trailer.
 Uma área com oficina mecânica.

Diante da placa I12, você entende que:



0 / 33

Figura A.5 - Página com os resultados do simulado realizado.

autosimulado ~ Meus simulados

http://autosimulado.com.br/profile/exams

Google

auto**simulado**

TOUR | SIMULADOS | FORUM | BLOG | AJUDA | CONTATO | ADMIN

Meus simulados

Você já fez 7 testes. Classificar por: data ou aproveitamento

SEU DESEMPENHO POR EXAME	APROVEITAMENTO
Especial Primeiros Socorros ~ Feito em 11/02/2009 ~ Gráfico	80%
Direção defensiva - Detran RJ - Prova 3 ~ Feito em 18/05/2009 ~ Gráfico	60%
Simulado da prova do Detran ~ Feito em 18/03/2009 ~ Gráfico	90%
Simulado ~ Feito em 18/03/2009 ~ Gráfico	60%
Direção defensiva - Detran RJ - Prova 4 ~ Feito em 18/05/2009 ~ Gráfico	75%
Especial Legislação ~ Feito em 03/02/2009 ~ Gráfico	85%
Simulado Detran Alagoas ~ Feito em 23/03/2009 ~ Gráfico	90%

Questões certas — Questões erradas

0 / 8

Figura A.6 - Página com os resultados dos testes do usuário.

The screenshot shows the user profile page for 'adminstrador' on the website 'autosimulado.com.br'. The browser address bar shows 'http://autosimulado.com.br/user/adminstrador'. The page has a navigation menu with 'TOUR', 'SIMULADOS', 'FORUM', 'BLOG', 'AJUDA', 'CONTATO', and 'ADMIN'. The profile section includes a photo of a man, the name 'OLÁ ADMINSTRADOR', and statistics: '18434 simulados feitos' and '3557 usuários'. The 'Últimos testes' section lists several exams: 'Especial Primeiros Socorros, em 15/10/2009', 'Direção defensiva - Detran RJ - Prova 3, em 06/10/2009', 'Simulado da prova do Detran, em 12/08/2009', 'Simuladão, em 08/08/2009', 'Direção defensiva - Detran RJ - Prova 4, em 04/06/2009', 'Especial Legislação, em 24/05/2009', and 'Simulado Detran Alagoas, em 03/04/2009'. The 'Últimas discussões' section lists 'SIMULADO DA PROVA DO DETRAN' and 'Ciclistas e Motociclistas'. The 'Scrapbook' section contains a message from 'josephcharles' dated 27/10/09 01:36, discussing the difficulty of some questions.

Figura A.7 - Página do perfil público do usuário. Essa página mostra os exames feitos pelo usuário, suas mensagens do fórum e o *scrapbook*.

The screenshot shows the forum page on the website 'autosimulado.com.br'. The browser address bar shows 'http://autosimulado.com.br/forum'. The page has a navigation menu with 'TOUR', 'SIMULADOS', 'FORUM', 'BLOG', 'AJUDA', 'CONTATO', and 'ADMIN'. The forum section is titled 'Forum autosimulado' and shows 4 topics. The topics listed are: 'Simulado da prova do Detran ~ Feito em 18/03/2009 ~ [editar] [deletar]' with 2 responses; 'SIMULADO DA PROVA DO DETRAN [editar] [deletar]' with 2 responses; 'Ciclistas e Motociclistas [editar] [deletar]' with 1 response; and 'De que cidade você é? [editar] [deletar]' with 6 responses. The 'auto simulado' logo is visible at the top of the forum section.

Figura A.8 - Mensagens do fórum do autosimulado.

APÊNDICE B - MODELO DE NEGÓCIO

O autosimulado é um serviço que, como a maioria das aplicações web 2.0, não possui custo para sua utilização. Durante o planejamento do serviço foram avaliados alguns modelos de negócio para tornar rentável o serviço. Um dos caminhos seria a vinculação de publicidade através do *Google Adsense* (Google, 2009). O *Google Adsense* é um serviço que exibe banners contextualizados nas *homepages* de seus editores e que os paga por cliques em seus anúncios. O *Google Adsense* foi testado no aplicativo entretanto não se mostrou rentável, sendo desconsiderado em um primeiro momento.

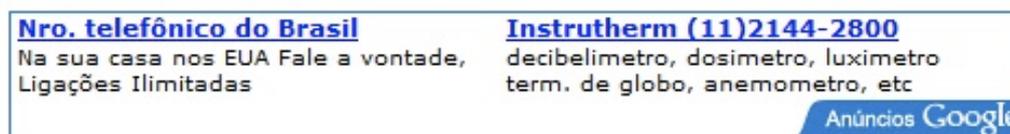


Figura B.1 - *Banner do Google Adsense*
Fonte: *Google* <<http://adsense.google.com>>

Outra fonte de rentabilização do autosimulado seria através de parcerias com Centros de Formação de Condutores ou editoras de material didático sobre o assunto. Alguns contatos com essas empresas foram realizados entretanto não foram concluídas nenhuma negociação. Esse modelo de negócio será aplicado futuramente.

Além disso, uma outra alternativa para rentabilizar o aplicativo seria através da venda de conteúdo exclusivo direto ao usuário. Essa oferta seria feita através da venda de conteúdo exclusivo por uma pequena assinatura de, por exemplo, R\$9,90. Essa forma de rentabilização pode parecer baixa a curto prazo mas revela-se uma receita com grande possibilidade de crescimento. Esse modelo de negócio também será aplicado futuramente.

ANEXO A - ESTATÍSTICAS COM *GOOGLE ANALYTICS*

O *Google Analytics* (Google, 2009) é um aplicativo que fornece as mais variadas estatísticas ao editor de conteúdo de um *website*. Entre os principais dados coletados pelo *Google Analytics*, destacam-se:

- *Visits*: apresenta o número de visitas únicas ao aplicativo.
- *Pageviews*: apresenta o número de páginas visualizadas.
- *Pages/visit*: apresenta a relação número de páginas vistas por visita.
- *Bounce Rate*: é a taxa de visitantes que deixam o seu site sem visitar outras páginas (além daquela que ele entrou) dividida pelo número total de visitantes dessa página antes que ocorra o tempo da sessão.
- *Time on Site*: apresenta o tempo médio do usuário no site.
- *New visits*: apresenta o porcentual de novos visitantes.
- *Traffic Sources Overview*: apresenta qual a origem das visitas ao serviço. Esse é um dado importante para verificar qual a eficiência do site diante aos sistemas de buscas. Ele também apresenta quais palavras buscadas nos sistemas de busca são usadas para encontrar o aplicativo.

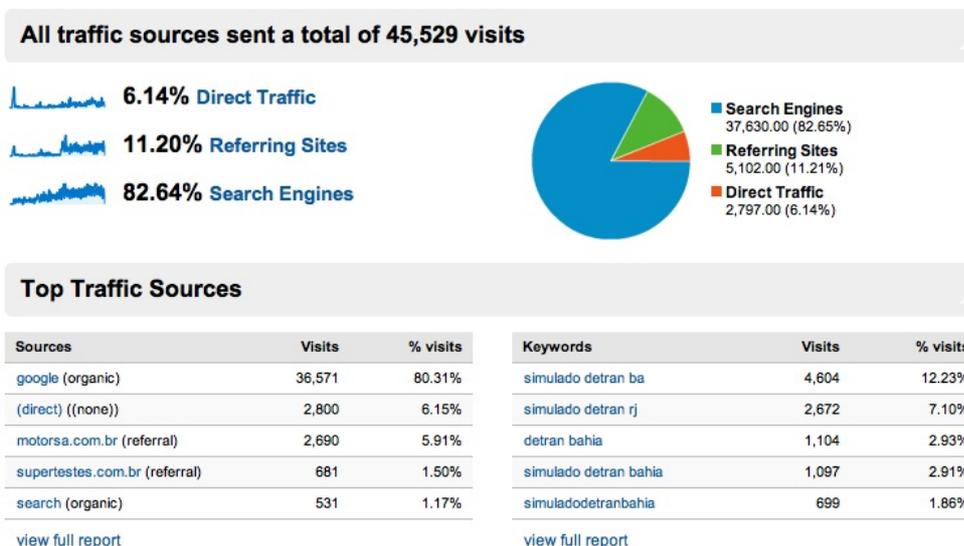


Figura A.1 - *Google Analytics*
Fonte: *Google* (2009)

- *Content Overview*: apresenta estatísticas das páginas mais visualizadas de um site. Para cada uma dessas páginas, o *Google Analytics* apresenta

o número de visitantes únicos, o tempo médio da visita e o percentual de saída do site.

- *Technical Profile*: apresenta relatórios com as características do equipamento do internauta: sistema operacional, navegador, tipo de conexão com a Internet, localização do usuário e resolução da tela.

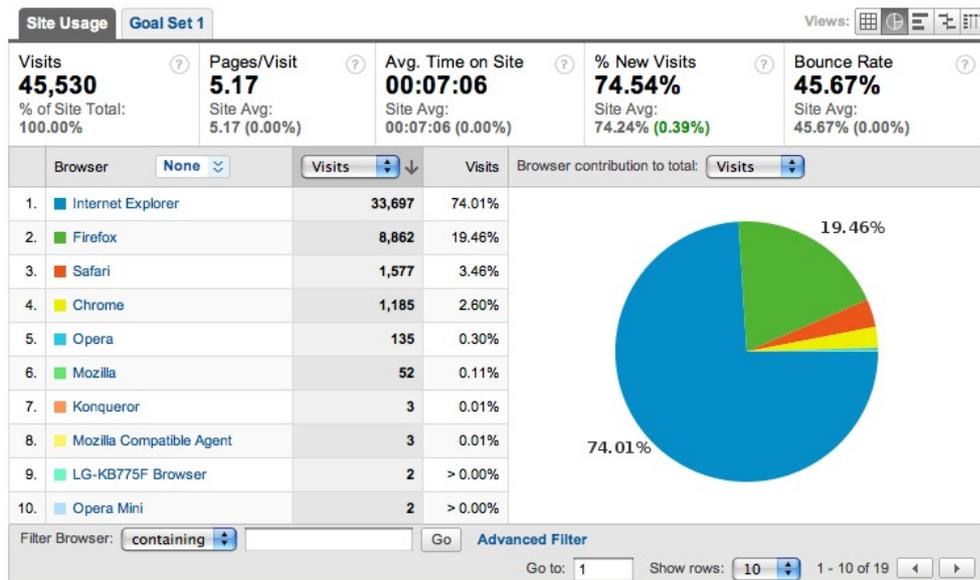


Figura A.2 - Google Analytics
Fonte: Google (2009)

Através das estatísticas do *Google Analytics*, é possível definir o perfil do usuário do autosimulado, o computador que ele usa e o mais importante: o que ele procura.